

IOT Based Smart Home



Chanon Watchratornsuk
Rungruj Vivatthanamukda
Qiwen Zhou

Bachelor of Science in
Engineering and Technology Management
Faculty of Engineering
King Mongkut's Institute of Technology Ladkrabang
Academic Year 2019

COPYRIGHT 2019
FACULTY OF ENGINEERING
KING MONGKUTS INSTITUTE TECHNOLOGY LADKRABANG

Abstract

The rising level of device connectivity in homes and buildings provides a wide variety of opportunities for building managers, device manufacturers and solution providers. The home automation technology gradually becomes mature. The aim of this report is to present the design and implementation of this advanced technology that allows users to control household electrical appliances via mobile devices, so-called “Homuto”, for making life easy and safe. The design is based on Apple’s mobile operation systems for both “IOS” and “iPad OS”. Home automation is aimed to build home automation for a home. There are three distinct levels of home automation which are monitoring, controlling and automation. By applying the Homuto system, lighting, climate, power, entertainment devices and other electrical appliances will be under control and monitored through a smartphone. The system also provides home security by controlling of access.

Table of Contents

Chapter 1 Introduction	1
1.1 Motivation	1
1.2 Objectives and Scope.....	2
Chapter 2 Literature Review.....	3
2.1 Apple HomeKit	3
2.2 Google Home.....	3
2.3 Amazon Echo	4
2.4 Samsung SmartThings.....	5
2.5 Wink Hub 2	5
Chapter 3 Background Knowledge.....	7
3.1 Wireless Communication	7
3.1.1 The Transmitter	8
3.1.2 The Channel	8
3.1.3 The Receiver	8
3.2 Remote Control	8
3.3 Operating Standards	9
3.3.1 Wi-Fi.....	10
3.3.2 Z-Wave	10
3.3.3 Zigbee	10
3.3.4 Bluetooth Low Energy	10
Chapter 4 Requirement Analysis / Design	12
4.1 Requirement Analysis	12
4.1.1 Device Managing.....	12

4.1.2 Classification Managing	12
4.1.3 Energy Monitoring	13
4.1.4 Home Temperature Monitoring.....	13
4.1.5 Scene and Trigger.....	13
4.1.6 Security System	13
4.1.7 Voice Control	14
4.1.8 GPS.....	14
4.2 Use Case Diagram	14
4.2.1 Brief Use Case	15
4.3 System Architecture	16
4.2.1 Firebase authentication	16
4.2.2 Firebase real-time database	16
4.2.3 WIFI	17
4.2.4 Microcontroller.....	17
4.2.5 Relay.....	17
4.2.6 Sensor	17
4.4 Domain Class Diagram.....	18
Chapter 5 Implementation.....	19
5.1 Hardware Tools	19
5.1.1 ESP8266 ESP-01S.....	19
5.1.2 ESP-01S Relay Module	20
5.1.3 DHT11 Digital Temperature & Humidity Sensor Module.....	20
5.1.4 Ultra-compact power module HLK-PM01	21
5.1.5 YHDC SCT013 100A SCT-013-000.....	21
5.2 Software Tools.....	21
5.2.1 Arduino IDE	22

5.2.2 Xcode.....	22
5.2.3 Swift	22
5.2.4 UIKit.....	22
5.2.5 CocoaPods	23
5.2.6 Firebase Authentication.....	23
5.2.7 Firebase Realtime Database	23
Chapter 6 Preliminary Result	24
6.1 Mobile Application.....	24
6.1.1 Start Page.....	24
6.1.2 Login Page.....	25
6.1.3 Sign Up Page	26
6.1.4 Home Page.....	27
6.1.5 Add Accessory Page.....	28
6.1.6 Device Information Page	29
6.1.7 Monitor Page	30
6.1.8 Scene & Trigger Page.....	31
6.1.9 My Home Page	32
6.1.10 Create Scene Page	33
6.1.11 Add Trigger Page	34
6.1.12 Timer Trigger Page.....	35
6.1.13 Characteristic Trigger Pages.....	36
6.1.14 Location Trigger Page	37
6.1.15 Setting Page	38
Chapter 7 Conclusion.....	39
7.1 Future Work.....	39
7.1.1 Security System	39

7.1.2 User Experience.....	40
References	41
Appendix	42

List of Figures

Figure 2.1: Google Home	4
Figure 2.2: Literature review summary table	6
Figure 3.1: Block diagram of wireless communication system	7
Figure 3.2: Home Automation Protocols.....	9
Figure 4.1: Use case diagram	14
Figure 4.2: System architecture	16
Figure 4.3: Domain class diagram	17
Figure 5.1: ESP8266 module.....	18
Figure 5.2: ESP-01S relay module	19
Figure 5.3: DHT11	19
Figure 5.4: HLK-PM01 module	20
Figure 5.5: YHDC SCT013 module	20
Figure 6.1: Start page	24
Figure 6.2: Login page	25
Figure 6.3: Sign up page.....	26
Figure 6.4: Home page	27
Figure 6.5: Add accessory page.....	28
Figure 6.6: Device information page.....	29
Figure 6.7: Monitor page.....	30
Figure 6.8: Scene & trigger page.....	31

Figure 6.9: My home page.....	32
Figure 6.10: Create scene page.....	33
Figure 6.11: Add trigger page	34
Figure 6.12: Timer trigger page.....	35
Figure 6.13: Characteristic trigger page(a).....	36
Figure 6.14: Characteristic trigger page(b)	36
Figure 6.15: Location trigger page	37
Figure 6.16: Setting page	38

List of Tables

Table 3.1: Comparison of Home Automation Protocols 11

Chapter 1

Introduction

1.1 Motivation

In the past decade, the home environment has seen a rapid development of network and digital technology. This advanced technology offers new opportunities to improve the connectivity of devices which increase the availability of home automation. Moreover, as the Internet of Things seeping deeply into people's lives, automation has made lives easier, homes smarter, and often more cost efficient. Smart home is the foundation of smart cities and a safer way of lifestyle.

A home automation system typically connects electrical appliances wirelessly to be a home hub or 'gateway'. It comprises many sub-systems to better control all-round house from different aspects such as power, environment, and security. There are lots of devices and appliances in people's houses, each of which owns a totally different switch. These conventional switches are in different places of the house. They are difficult to operate for the users and especially for disable people as they cannot approach, much less operate. In addition, if people live in a two-storey or even higher villa, they will have no way to control all of them, which may cause severe safety problems. To better and easily control home devices and appliances gradually becomes a prevalent topic among people.

In this project, we want to cope with this issue by implementing home automation technology. Therefore, our aim is to create an easy-to-use and multifunctional application which provides people with safety, security, efficiency, and convenience. Homuto can be installed in a smartphone acting as a central remote. With one smartphone, people can seamlessly control and monitor all devices and appliances over all aspects of your home and more.

1.2 Objectives and Scope

- To develop an all-in-one platform for managing all of home device function on smartphones and tablets.
- To develop a system which provides flexibility for connecting devices and appliances.
- To develop a system which provides temperature monitoring by installing temperature sensors.
- To develop a system which maximizes home security by controlling access.
- To develop an access control system and alarm system for protecting property.
- To develop a control monitor that helps manage energy by calculating precise usage.
- To develop a home tool that operates user's home combining with daily habits and lifestyles.

Chapter 2

Literature Review

Some several existing software are playing a similar role as the proposed project on the market. This chapter primarily describes the comparison and evaluation between software related to home automation, which are Apple HomeKit, Google Home, and Amazon Echo, Samsung SmartThings and Wink Hub 2.

2.1 Apple HomeKit

HomeKit is an application developed by Apple that lets users configure, control smart-home appliances and customize functions on Apple brand devices. HomeKit uses Bluetooth and Wi-Fi protocols. Users can design automatic actions through HomeKit service in the house and easily control them with a simple voice command to Siri or directly through the HomeKit application. iPads, HomePods, and the newer Apple TVs can be used as a home hub. HomeKit primarily competes against Amazon and Google regarding smart home standards. As of October 2019, Apple lists 450 compatible devices that are available on Homekit. On contrast, there are over 10,000 devices compatible with Google and 85,000 with Amazon [1].

2.2 Google Home

In appearance, Google Home resembles one of our most common smart Home boxes, with four colors of LED lights hidden on the tilting, touchable surface at the top, and a magnetically connected scroll wheel at the bottom of the lower speaker grille. It exists in every part of the smart home, with places built in processing system and Google Assistant. Google Home can use Google Assistant with increasing power to recognize the voice control of user and transform it into an information source to control smart home

which is fully covered by voice control. The appearance of Google Home is shown in the figure2.1.



Figure 2.1: Google Home

Google Home will play the music and do lots of tasks when user gives a voice order. Furthermore, users can have a two-way conversation with Google Home. Functionally, through voice control, Google Home can control audio electronics and other devices to be a home hub.

2.3 Amazon Echo

The Amazon Echo is a leading hub for home automation, using Amazon's proprietary Alexa voice-powered technology. Alexa is the virtual assistant AI technology developed by Amazon. With it, Echo could do lots of tasks such as voice interaction, music playback, to-do lists making, and provide information like weather, traffic for daily life and news. Like Google Home, it can also control several smart devices using itself as a home automation system. Echo devices connect to Alexa, which will respond when you say "Alexa".

2.4 Samsung SmartThings

The Samsung SmartThings aims to integrate smart home systems with a single home automation hub, which is operated through a mobile application. Samsung also makes several peripherals that work directly with smart home systems, including motion sensors, multipurpose sensors and smart buttons. Samsung's SmartThings hub also works well with most third-party smart devices, though it doesn't have as many connectivity options as some devices and is largely limited to z-wave and Zigbee protocols.

2.5 Wink Hub 2

Wink Hub 2 is one of the most versatile hubs for home automation. The hub can be set up wirelessly or connected to the router via an Ethernet cable. Then, you can download the Wink app on your mobile device and control everything from the app. Here, you can select other smart devices you want to connect to and control their Settings accordingly. This means that commands are usually executed through a visual interface. It all works with devices you might carry, like your smartphone or even a wearable smartwatch. However, it doesn't have voice command by default, but you can use any of the existing built-in voice assistants, such as Google Home, Siri, or even Alexa.

Figure 2.2 indicates the comparison of some features of these 4 home automation products.

	Apple HomeKit	Google Home	Amazon Echo	Samsung SmartThings	Wink Hub 2
Voice Command	Yes	Yes	Yes		
Security Concern	Yes	Yes	Yes	Yes	Yes
Customisation	Yes	Yes	Yes	Yes	Yes
Compatibility Level	Low	Medium	High	Low	Low
Protocol	HomeKit Accessory Protocol (HAP)	Z-wave, Nest	Z-wave, Zigbee	Z-wave, Zigbee	Bluetooth LE, Zigbee, Z-wave, Kiddle, Lutron Clear Connect

Figure 2.2: Literature review summary table

Chapter 3

Background Knowledge

This chapter describes relevant background knowledge that might be needed to get a better understanding of the project, which can be divided into three parts.

3.1 Wireless Communication

Wireless communication is the fastest growing and one of the most vibrant areas in the communication field nowadays. Unlike the wired communication, wireless communication is the method for transmitting information between two or more points without using any physical medium like wires and cables but air. Wireless systems allow users interconnect freely within the limited distance. These systems are using radio waves as a transmission/receiving medium. The basis of radio communication is the use of antennas to transmit and receive electromagnetic waves [2].

A typical Wireless Communication System can be divided into three elements: The Transmitter, the Channel and the Receiver. Figure 3.1 shows the relationship of these three elements and how they work.

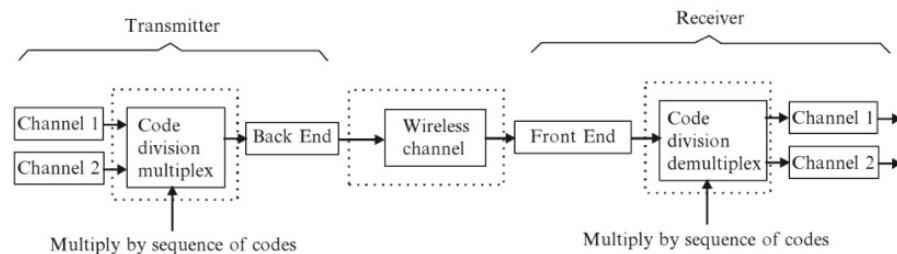


Figure 3.1: Block diagram of wireless communication system [2]

3.1.1 The Transmitter

The transmitter side handles encode, encryption, modulation, and multiplexing. The signal from the source first is transmitted through the Source Encoder, which converts the signal into an appropriate form which is suitable for signal processing techniques. It is then encrypted to ensure the security of the signal. After that, the signal will be modulated so that the signal can be easily transmitted through the antenna. And the modulated signal will be multiplexed with other signals, using different multiplexing techniques to share valuable bandwidth.

3.1.2 The Channel

In wireless communication, the channel represents the medium of transmission of the signal. A wireless channel is unpredictable, highly variable, and random in nature. There is interference inside the channel. The interference can occur in different situations like between transmitters communicating with a common receiver, between signals from a single transmitter to multiple receivers or between different transmitter-receiver pairs [2].

3.1.3 The Receiver

The composition of the receiver side is the opposite of that transmitter. The receiving path of wireless communication system consists of multiplexing, demodulation, channel decoding, decryption, and source decoding. The receiver collects the signal from the channel and reproduce it to the original transmitted message or signal.

Wireless home automation is an advance application of wireless system which uses wireless technologies like Zigbee, Wi-Fi, GSM, Bluetooth, etc., for achieving remote operation.

3.2 Remote Control

The other main characteristic of home automation is remote monitoring and access. A remote control is an electronic device used to control another device from a given distance, usually wirelessly. Remote control is mainly to provide a convenient function for

users, and to allow easy operation of devices that are out of convenient reach for direct operation of controls. In today's consumer market, television remote controls are most widely used. A remote control for television has buttons for each numeral from zero to nine, as well as other control function such as controls of volume, channel, brightness, color intensity and others according to type of TV sets.

In home automation technology, a mobile device can present a home hub, which is acting an all-in-one remote control and monitor to all appliances in home in a short distance. The home appliances can be switched on/off using remote control without actually going near the switch boards or regulators. By integrating wireless sensors into the system, remote systems can be controlled in a more flexible way [3].

3.3 Operating Standards

There are several common home automation operating standards that allow different devices to speak a common language. The primary standards for home automation technology are these four: Wi-Fi, Z-Wave, Zigbee, and Bluetooth Low Energy (BLE). The following are the information of these four protocols.

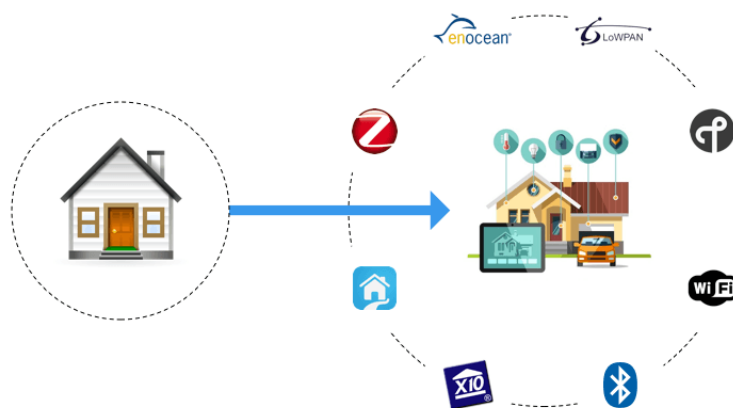


Figure 3.2: Home Automation Protocols [4]

3.3.1 Wi-Fi

Wi-Fi is a wireless network technology that uses radio waves to provide wireless high-speed Internet and network connections. It is a widely available network for people daily used. It allows devices such as computers, mobile devices, and other equipment to interface with the Internet [5]. It is also taking an important part in home automation where lots of smart devices on the market connect to smartphones or hubs via Wi-Fi.

3.3.2 Z-Wave

Z-Wave is a wireless communications protocol used mainly for home automation. Z-Wave networks devices by sending messages via the 800-900 MHz (megahertz) frequency range. The Z-Wave compatible smart devices speak to each other on this frequency range and form a mesh network which is a chain that converts individual smart devices into nodes [6]. And these nodes transmit data packets from one device to another until they reach the ultimate destination.

3.3.3 Zigbee

Zigbee is like Z-Wave in that it is a wireless mesh network. Zigbee radios operate primarily on the 2.4 GHz band and many smart home devices use the frequency because of its long range. Zigbee is suitable for creating personal area networks for small, low-power digital radios, such as home automation. It has become popular among home automation manufacturers.

3.3.4 Bluetooth Low Energy

Bluetooth Low Energy is another well-known wireless personal area network technology. Bluetooth LE uses the same 2.4 GHz radio frequencies. It is used to transmit smaller packets at regular intervals compared to Bluetooth Classic. It always stays in sleep mode except when going through a data exchange, which reduces overall energy consumption [7]. Because of its low cost, long battery life, and ease of deployment, it is attractive to consumer electronics manufacturers.

Table 3.1 shows the differences between those mentioned operating standards.

Table 3.1: Comparison of Home Automation Protocols

Variable	Wi-Fi	Z-Wave	Zigbee	BLE
Area Coverage	Wide	Wide	Wide	Wide
Power Efficient	No	Yes	Yes	Yes
Data Bandwidth	High	Low	Low	High
Frequency Band	2.4GHz	900MHz	2.4GHz	2.4GHz
Topology	Star	Mesh	Mesh	Scatternet
Alliance	Wi-Fi Alliance	Z-Wave Alliance	Zigbee Alliance	Bluetooth SIG

Chapter 4

Requirement Analysis / Design

This chapter categorizes the requirements for developing the Smart Home application, and the basic structure of the system will be explained in detail.

4.1 Requirement Analysis

This project wants to provide a low-cost but versatile and flexible Home Automation System for use in private homes. Users use the system to control devices around the house in order to obtain the satisfaction of comfort, convenience, and security. The following provides an extensive list of requirements for composition of smart home applications from the user's perspective.

4.1.1 Device Managing

Device managing is the main function of the system. Users are able to add new appliance to the system by scanning the setup code embedded on the product packaging. Users can implement the remote control of appliance that was already added through the IOS mobile devices in any position around the house. Of course, users may want them to be removed out of the system.

4.1.2 Classification Managing

There are appliances which may counts from dozens to hundreds inside a flat or a building that would cause difficulties for users to operate although they are all in one system. Appliances can be classified into different rooms such as bedroom, living room, kitchen, restroom, etc. With classification, users manage appliances by room.

4.1.3 Energy Monitoring

A home energy monitor provides feedback on electrical energy use. The real-time energy information like daily consumption is shown on the screen of mobile device for making it more intuitive. Energy monitor is in demand for cost saving and energy efficiency.

4.1.4 Home Temperature Monitoring

Stable temperature is often an essential factor in building an ease and comfortable environment for home. With the home temperature monitor, users get to know the real-time temperature anytime and make adjustment the air-condition to keep warm and comfortable in the winter and cool and fresh in the summer.

4.1.5 Scene and Trigger

The function of Scene and Trigger helps reduce the complexity of routine tasks. One probably has many different routines of basic tasks that he goes through every time he goes to bed, wakes up, or leaves the house. Users can set up the scene based on time trigger, location trigger, and characteristic trigger according to their lifestyles and personal preferences. When the trigger is activated, like when it is at a designated time or when the host is being tracked by GPS that he arrives the house, a sequence of commands will take place automatically.

4.1.6 Security System

Home security is the most concerned and most indispensable issue in home automation technology. A home security system includes alarm system, home access control system, and all of the doors, windows, locks, smoke detectors, surveillance cameras and any other sensors that are linked to it. The system will detect whether the door of entryway is open or closed and send a notification to users.

4.1.7 Voice Control

For further convenience, users especially elderly and disabled people are able to control appliances by making voice command. This can be done with intelligent virtual assistant that built into the Apple mobile devices which is “Siri”. It will respond immediately when the user gives a command started with “Hey, Siri”.

4.1.8 GPS

The Global Positioning System (GPS) has to be applied in order to achieve the application of the Scene and Trigger like the scene of ‘Arrive home’ or ‘Leave home’. So that the system can track the location of users to know whether the triggers should be activated.

4.2 Use Case Diagram

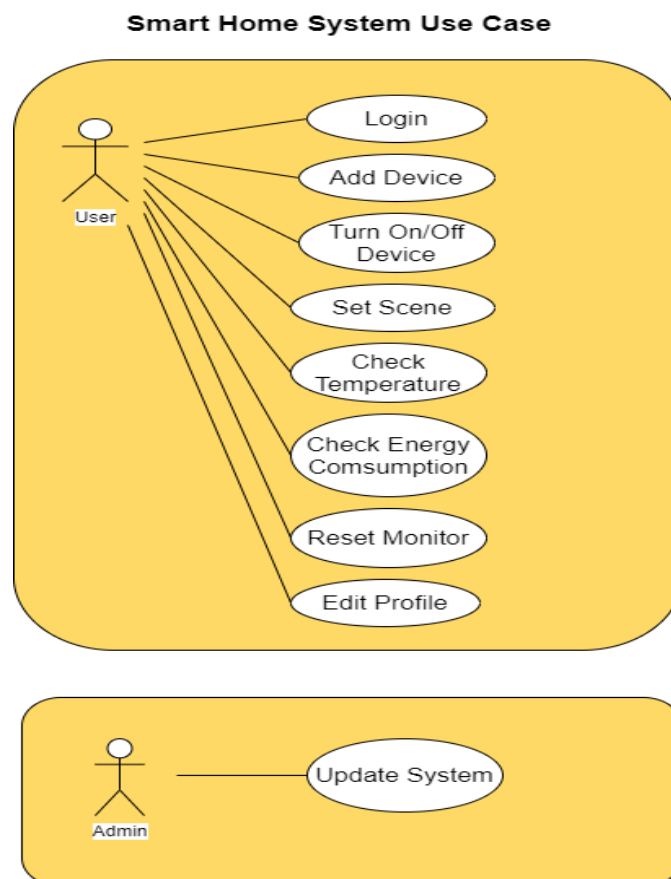


Figure 4.1: Use case diagram

4.2.1 Brief Use Case

The following are the description of use case in brief.

Login

User login to the mobile application with username and password.

Add Device

User can add more than one device to the system.

Turn On / Off Device

User can turn on / off the device up to the need of the user via mobile application.

Set Scene

User can set different scenes according to their lifestyles.

Check Temperature

User can check house temperature via the application anytime.

Check Energy Consumption

User can check the real-time energy consumption to help with cost saving.

Reset Monitor

User can reset the energy monitor to make the data begin with zero.

Edit Profile

User can change the profile.

Update System

The system will be updated at regular intervals based on feedbacks and R&D.

4.3 System Architecture

The following diagram shows how the user's request is handled behind the scenes. And the explanation of each term is given.

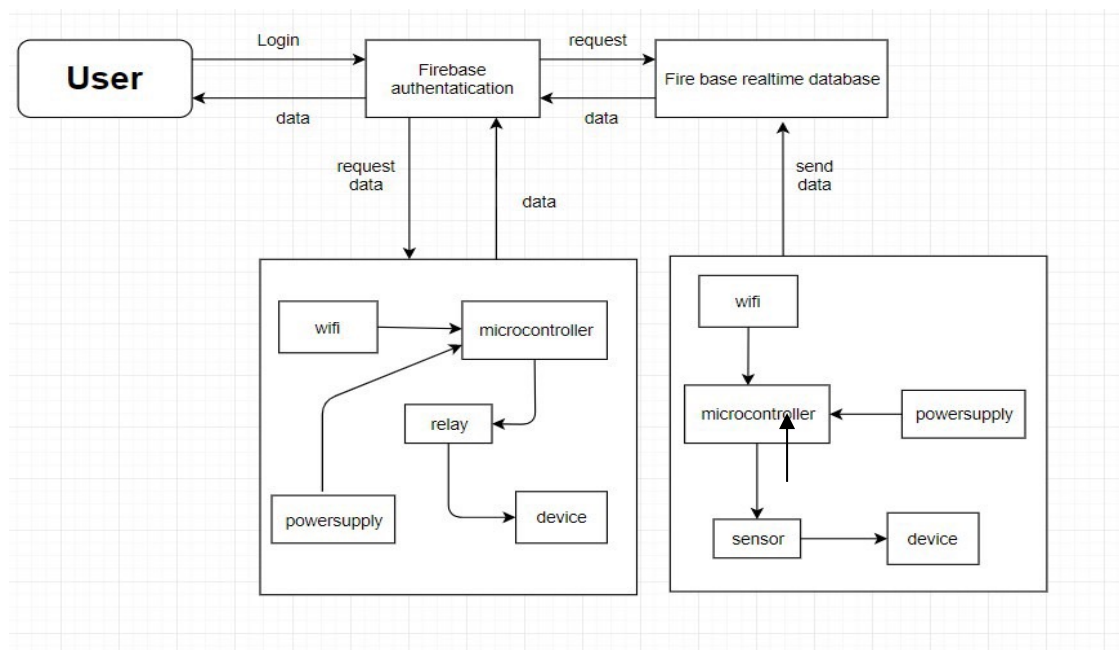


Figure 4.2: System architecture

4.2.1 Firebase authentication

The firebase authentication provides backend services, easy to use SDKs, and ready-made UI libraries to authentication users to the application.

4.2.2 Firebase real-time database

Firebase real time database use like a cloud hosted NoSQL database that let user store and sync data in real time.

4.2.3 WIFI

Users need to connect to the internet or WIFI to use application.

4.2.4 Microcontroller

We use microcontroller to control the relay and turn on and of the device.

4.2.5 Relay

We use relay to control the power supply and control electrical equipment via relay.

4.2.6 Sensor

Sensors use to measure the temperature with the electric current.

4.4 Domain Class Diagram

The following diagram shows the relationship between entities within the system.

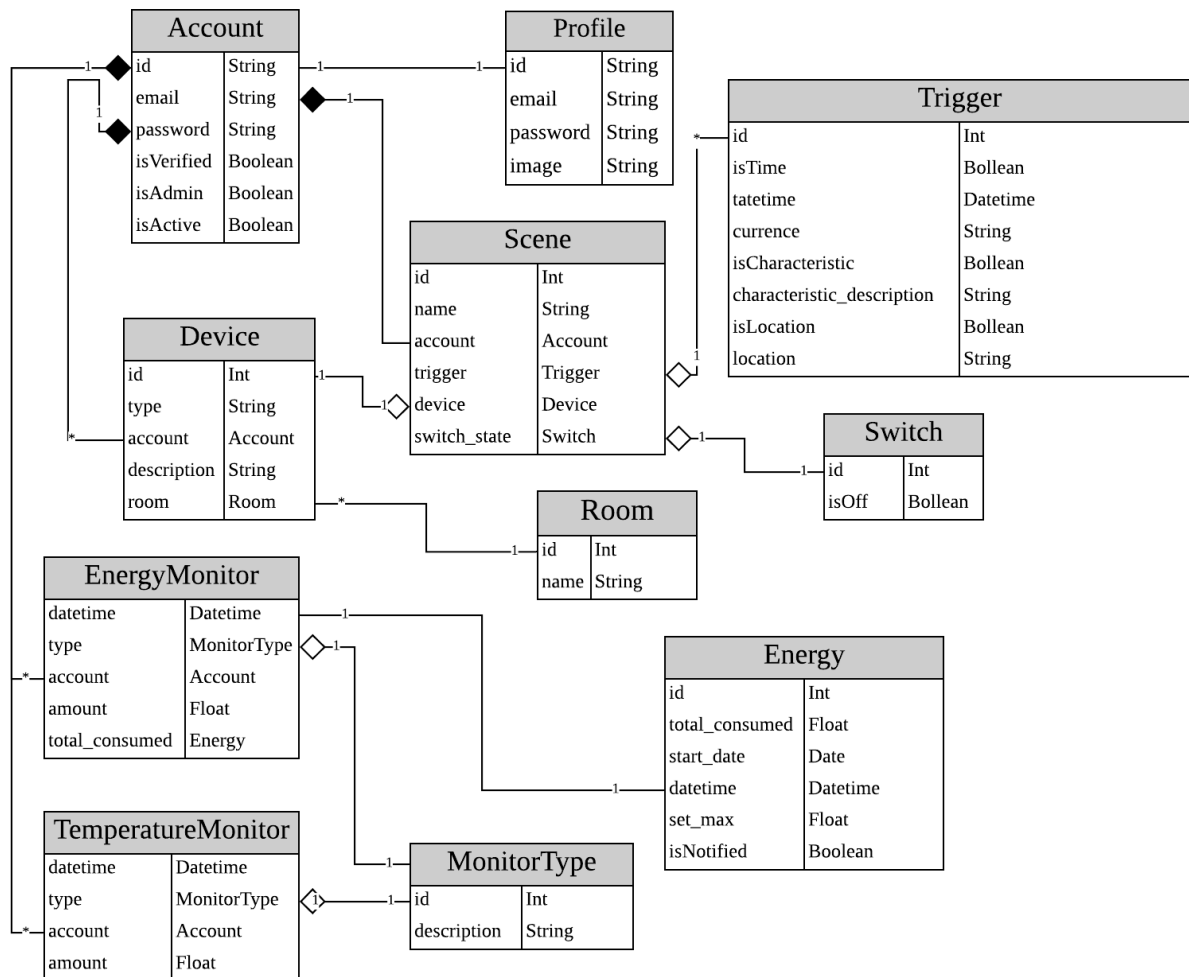


Figure 4.3: Domain class diagram

Chapter 5

Implementation

This chapter focuses on the actual development process of the proposed system and how the development tools worked together to function the system. Development tools are explained in two sections: hardware and software.

5.1 Hardware Tools

This project is using the ESP8266 Wi-Fi module with ESP-01S Relay Module and HLK-PM01 power module to build a home automation prototype as an Esp8266 module will receive commands from smartphone wirelessly through the internet. And DHT11 sensor module is used to detect temperature and humidity.

5.1.1 ESP8266 ESP-01S

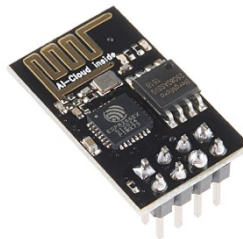


Figure 5.1: ESP8266 module

ESP-01 Wi-Fi module is a low-power and cost-effective embedded wireless network control module. It can meet the needs of Internet of Things applications such as smart grid, building automation, security, smart home, telemedicine and so on. The core processor ESP8266 integrates the industry-leading Tensilica L106 ultra-low power 32-bit micro MCU with 16-bit streamlined mode, main frequency of 80 MHz and 160 MHz, supports RTOS, integrates Wi-Fi MAC/BB/RF/PA/LNA, and on-board antenna. This module supports standard IEEE802.11 b/g/n protocol and complete TCP/IP protocol stack. Users can use this module to add networking functions to existing devices, or to build independent network controllers. This module has a powerful enough on-board processing

and storage capability that allows it to be integrated with the sensors and other application specific devices through its GPIOs with minimal development [8].

5.1.2 ESP-01S Relay Module



Figure 5.2: ESP-01S relay module

This ESP-01S Relay Module is based on the AI-Thinker ESP-01/01S WIFI module designed for smart home, internet of things. It uses the GPIO0 of ESP-01/ 01S to control the relay by low level.

5.1.3 DHT11 Digital Temperature & Humidity Sensor Module

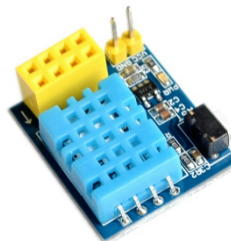


Figure 5.3: DHT11

DHT11 is a composite sensor that contains a calibrated digital signal output of temperature and humidity.

5.1.4 Ultra-compact power module HLK-PM01



Figure 5.4: HLK-PM01 module

HLK-PM01 module converts 240v AC to 5v DC to use as a power supply for ESP01S.

5.1.5 YHDC SCT013 100A SCT-013-000



Figure 5.5: YHDC SCT013 module

YHDC SCT013 module is Non-invasive AC current sensor.

5.2 Software Tools

To code the ESP8266, an Integrated Development Environment is needed and that is built by Arduino IDE software. As the proposed system is based on Apple's mobile operation systems, some relevant programming software are using for developing the application. We use Firebase Authentication who provides backend services to securely save user data in the cloud and provide the same personalized experience across all of the user's devices, and Firebase Realtime Database to store data in cloud.

5.2.1 Arduino IDE

The Arduino Integrated Development Environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in functions from C and C++. It is used to write and upload programs to Arduino compatible boards, but also, with the help of 3rd party cores, other vendor development boards.

5.2.2 Xcode

Xcode is an integrated development environment (IDE) for macOS containing a suite of software development tools developed by Apple for developing software for macOS, iOS, iPadOS, watchOS, and tvOS .Xcode supports source code for the programming languages C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, ResEdit (Rez), and Swift, with a variety of programming models, including but not limited to Cocoa, Carbon, and Java. Xcode can also be used to compile and debug applications for iOS that run on ARM architecture processors [9].

5.2.3 Swift

Swift is a general-purpose, multi-paradigm, compiled programming language developed by Apple Inc. for iOS, iPadOS, macOS, watchOS, tvOS, Linux, and iOS. Swift is designed to work with Apple's Cocoa and Cocoa Touch frameworks and the large body of existing Objective-C code written for Apple products. It is built with the open source LLVM compiler framework and has been included in Xcode since version 6, released in 2014. On Apple platforms, it uses the Objective-C runtime library which allows C, Objective-C, C++ and Swift code to run within one program [10].

5.2.4 UIKit

The UIKit framework provides the required infrastructure for your iOS or tvOS apps. It provides the window and view architecture for implementing your interface, the event handling infrastructure for delivering Multi-Touch and other types of input to your app, and the main run loop needed to manage interactions among the user, the system, and your app. Other features offered by the framework include animation support, document support, drawing and printing support, information about the current device, text management and display, search support, accessibility support, app extension support, and resource management.

5.2.5 CocoaPods

CocoaPods is an application level dependency manager for the Objective-C, Swift and any other languages that run on the Objective-C runtime, such as RubyMotion, that provides a standard format for managing external libraries.

5.2.6 Firebase Authentication

Firebase Authentication is a service that can authenticate users using only client-side code. It supports social login providers Facebook, GitHub, Twitter and Google as well as other service providers like Google Play Games, Apple, Yahoo, and Microsoft. Additionally, it includes a user management system whereby developers can enable user authentication with email and password login stored with Firebase [11].

5.2.7 Firebase Realtime Database

Firebase provides a real-time database and back-end as a service. The service provides application developers an API that allows application data to be synchronized across clients and stored in Firebase's cloud. The company provides client libraries that enable integration with Android, iOS, JavaScript, Java, Objective-C, Swift and Node.js applications. The database is also accessible through a REST API and bindings for several JavaScript frameworks such as AngularJS, React, Ember.js and Backbone.js. The REST API uses the Server-Sent Events protocol, which is an API for creating HTTP connections for receiving push notifications from a server. Developers using the realtime database can secure their data by using the company's server-side-enforced security rules.

Chapter 6

Preliminary Result

Every function of the proposed system will be shown in this chapter based on the on-page content. And explanation to the performance and functionality will be provided for a better understanding.

6.1 Mobile Application

6.1.1 Start Page

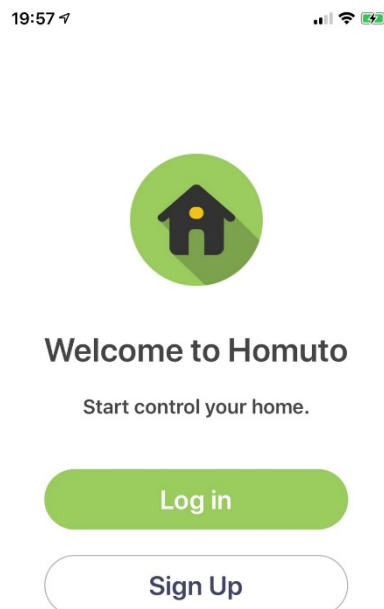


Figure 6.1: Start page

Start page will show when a user first opens the application and require the user to login if he already had an account or sign up for new user.

6.1.2 Login Page

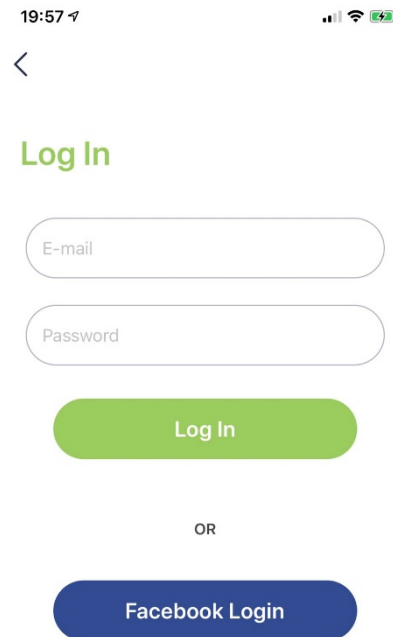
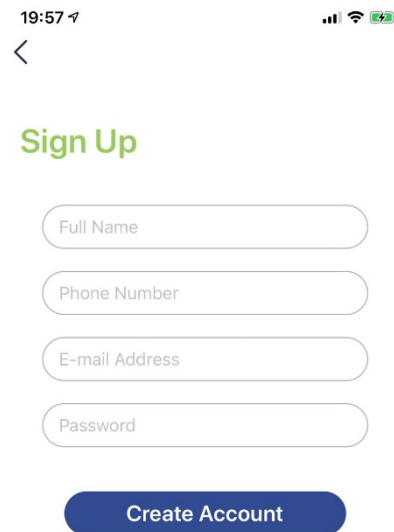
A mobile application login screen. At the top, the status bar shows the time 19:57, a back arrow, and icons for cellular signal, Wi-Fi, and a 43% battery level. Below the status bar is a back arrow icon. The title "Log In" is displayed in green. There are two input fields: "E-mail" and "Password". Below these fields is a green "Log In" button. Underneath the button is the text "OR". At the bottom is a blue "Facebook Login" button.

Figure 6.2: Login page

This is the login page that requires a user to login with email and password or connect to Facebook account to access the system and then the user will get the permission.

6.1.3 Sign Up Page



A mobile app interface for a sign-up page. At the top, the status bar shows the time 19:57, a back arrow, and signal/battery icons. Below the status bar is a green 'Sign Up' title. The form consists of four rounded rectangular input fields stacked vertically, labeled 'Full Name', 'Phone Number', 'E-mail Address', and 'Password'. Below these fields is a dark blue button with the text 'Create Account' in white.

Figure 6.3: Sign up page

This page requires the basic information of a user to create an account and the information will be stored into the system.

6.1.4 Home Page

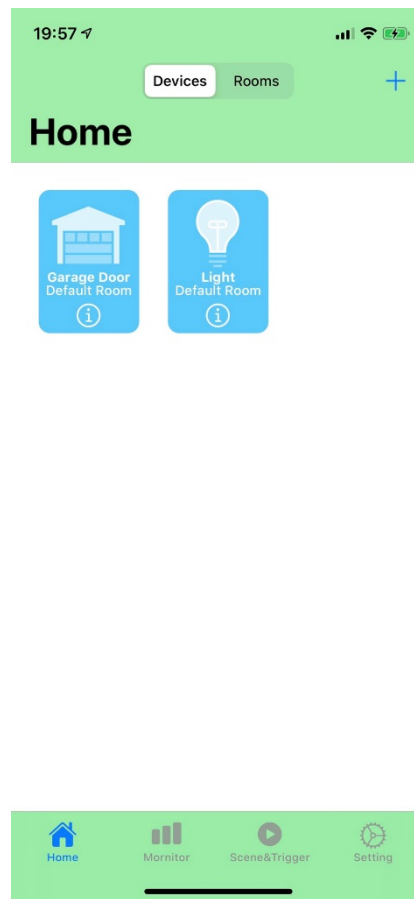


Figure 6.4: Home page

The application mainly consists of 4 parts: Home, Monitor, Scene & Trigger and Setting. The home page presents the primary function of the system which is to manage devices. Users can add devices to the system and all devices that have been added will be displayed succinctly and sequentially. Only a tap by the user, the state of the device will change. The user also allows to control them sorted by rooms.

6.1.5 Add Accessory Page

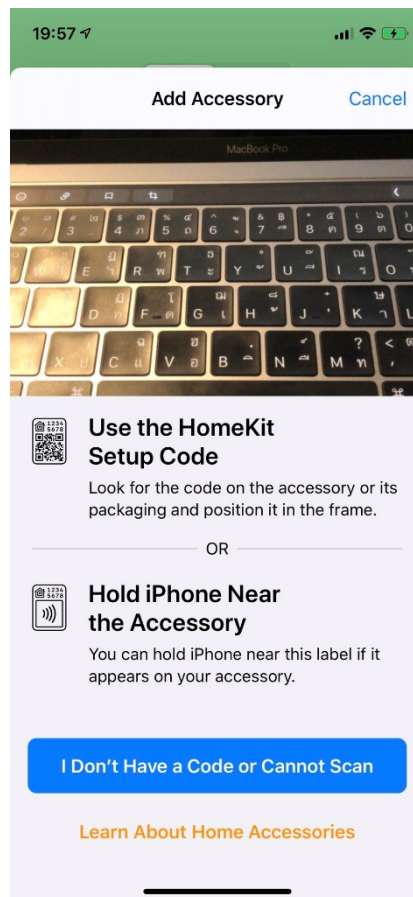


Figure 6.5: Add accessory page

This is the page that will show when a user is trying to add a new device to the system. The guidelines are clearly listed. The user can scan the HomeKit setup code via the camera or hold the mobile device near the new device.

6.1.6 Device Information Page

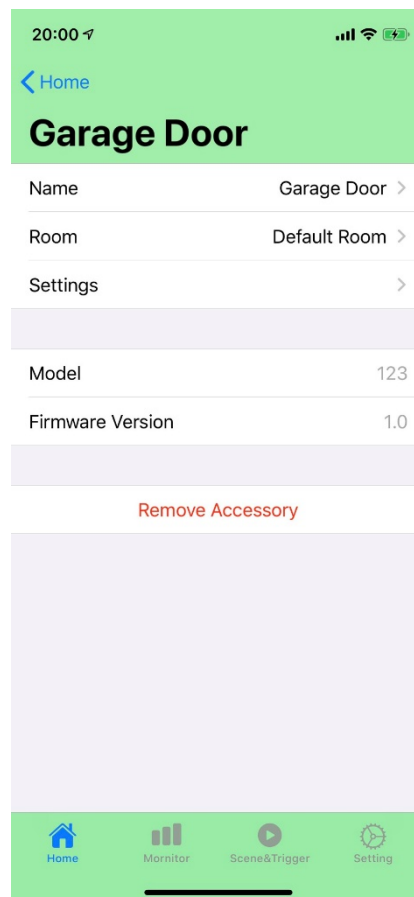


Figure 6.6: Device information page

After scanning, the basic information of the device will show up and users can make modifications except model and firmware version which are default. If the device is broken or replaced, users can just remove it from here directly.

6.1.7 Monitor Page

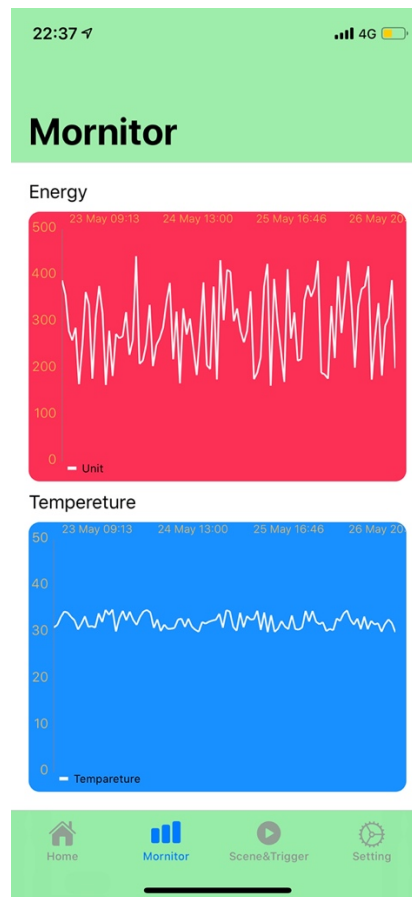


Figure 6.7: Monitor page

The monitor page shows the graph of energy consumption and the trend of temperature inhouse for users to monitor which are intuitive and easy to understand. The X-axis is time, and Y-axis is the amount unit of consumption or temperature degree.

6.1.8 Scene & Trigger Page

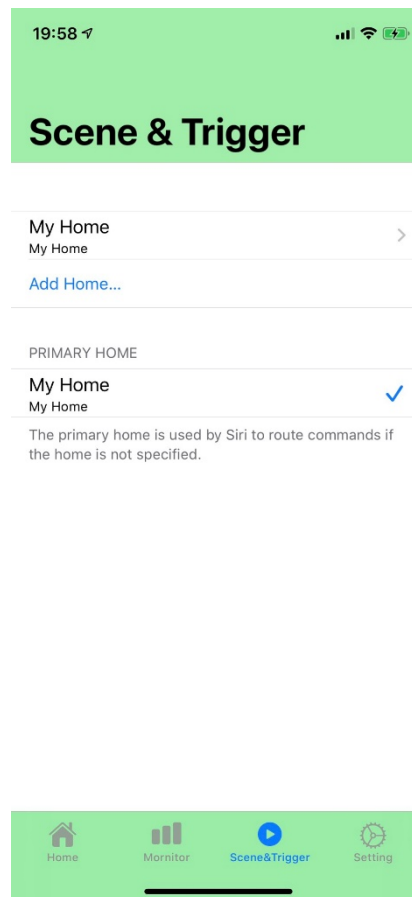


Figure 6.8: Scene & trigger page

The scene & trigger page shows scenes that users set, and they are sorted by home. If a user has more than one house, they can manage them separately. Users can assign one of them to be the primary home which is used by Siri to response the commands if the home is not specific

6.1.9 My Home Page

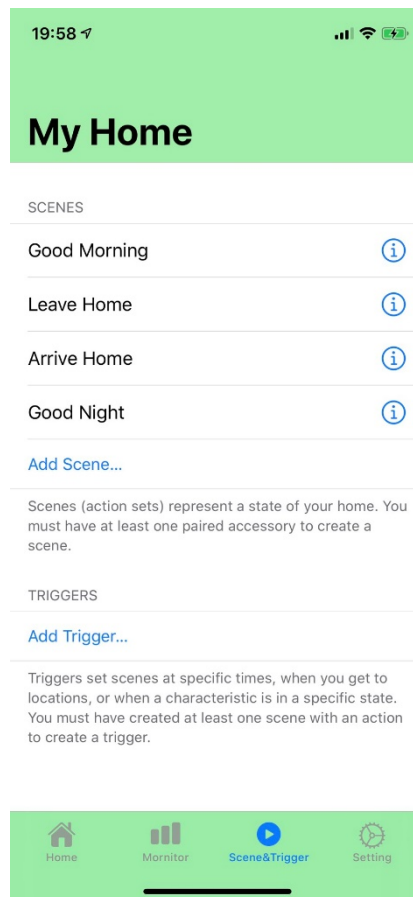


Figure 6.9: My home page

My home page is a sub-page of the scene & trigger page. All scenes are listed and can be managed. In this page, users can add scene and trigger. A scene represents the state of a combination of devices, and it's set by triggers.

6.1.10 Create Scene Page

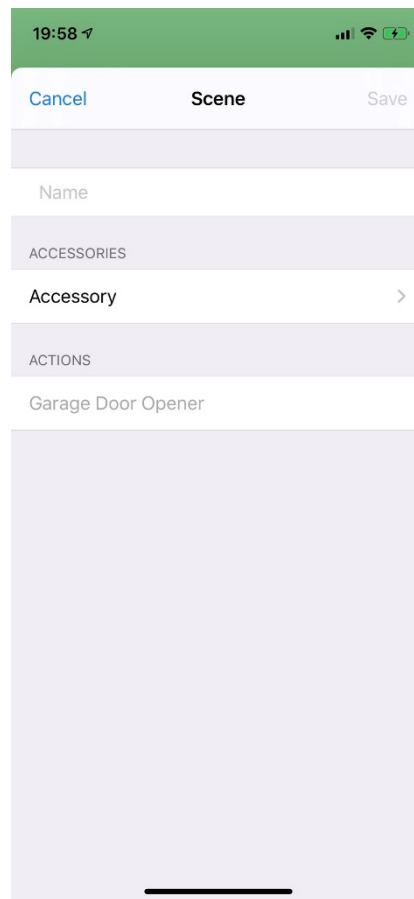


Figure 6.10: Create scene page

Users can create a scene with at least one paired devices and actions of changing the switch.

6.1.11 Add Trigger Page

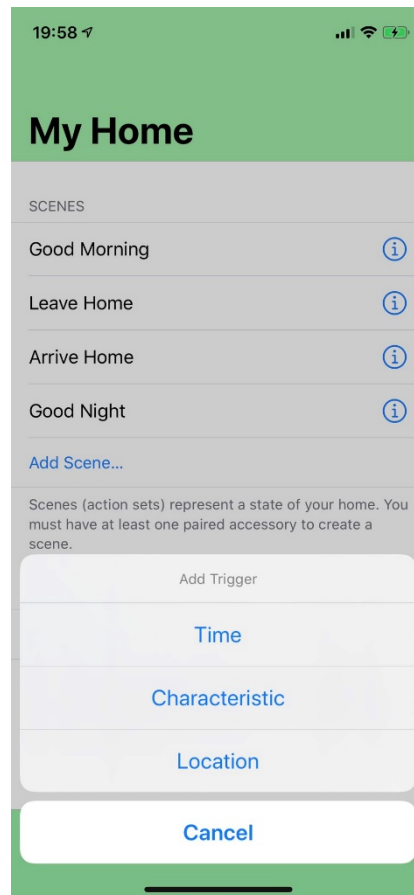


Figure 6.11: Add trigger page

There are 3 types of trigger that make the scene happen: time, characteristic, and location.

6.1.12 Timer Trigger Page

19:58

Cancel Timer Trigger Save

Name

Enabled ☒

This trigger will only activate if it is enabled. You can disable triggers to temporarily stop them from running.

DATE & TIME

DATE	TIME	TIME
Sun 3 May	16	55
Mon 4 May	17	56
Tue 5 May	18	57
Today	19	58
Thu 7 May	20	59
Fri 8 May	21	00
Sat 9 May	22	01

RECURRENCE

Every Hour

Every Day

Every Week

Figure 6.12: Timer trigger page

Users can set a specific time when a scene is triggered and its recurrence. A trigger will only activate if it is enabled. Users can turn it off to temporarily stop it from running.

6.1.13 Characteristic Trigger Pages

19:58

Cancel Characteristic Trigger Done

Name

Enabled ☒

This trigger will only activate if it is enabled. You can disable triggers to temporarily stop them from running.

CHARACTERISTIC EVENTS

[Add Characteristic...](#)

This trigger will activate when any of these characteristics change to their value. For example, 'run when the garage door is opened'.

CONDITIONS

[Add Condition...](#)

When a trigger is activated by an event, it checks these conditions. If all of them are true, it will set its scenes.

SCENES

Good Night

When this trigger is activated, it will set these scenes. You can only select scenes which have at least one action.

12:39

Cancel Time Condition Save

CONDITION TYPE

☒ Exact Time ☐ Special Event

Time conditions can relate to specific times or special events, like sunrise and sunset.

Before ☒

After ☐

At ☐

TIME

09	36
10	37
11	38
12	39
13	40
14	41
15	42

Figure 6.13: Characteristic trigger page(a) Figure 6.14: Characteristic trigger page(b)

To create a characteristic trigger, users customize a characteristic, for example, 'run when the garbage door is opened'. Users can set condition to the corresponding characteristic based on an exact time or a special event, for example, after 12:00. Then the scene will be activated if the garbage door is opened after 12:00.

6.1.14 Location Trigger Page

The screenshot shows a mobile app interface for setting a location trigger. At the top, there's a status bar with the time 19:58 and signal/battery icons. Below it is a header bar with a green background, containing 'Cancel', 'Location Trigger', and 'Done' buttons. The main content area has a light purple background and includes the following sections: a 'Name' input field, an 'Enabled' toggle switch (currently turned on), a descriptive text block about enabling triggers, a 'LOCATION' section with a 'Set Location' button and a chevron, a 'REGION STATUS' section, two radio button options for 'When I Enter The Area' (selected) and 'When I Leave The Area', another descriptive text block about trigger activation, a 'CONDITIONS' section with an 'Add Condition...' link, a final descriptive text block about conditions, and a 'SCENES' section at the bottom.

Figure 6.15: Location trigger page

Location trigger page allows users to set location where the trigger can activate when users enter or leave. For example, a user would like to turn on the lights and AC when he arrives at home.

6.1.15 Setting Page

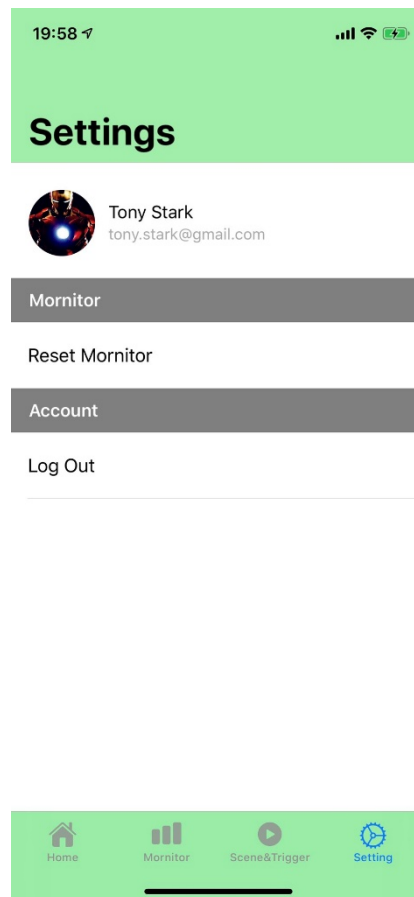


Figure 6.16: Setting page

The setting page shows the profile of the user and allows users to change their information such as photo and name.

Chapter 7

Conclusion

This thesis provides the whole idea and development process of the proposed system. The proposed system refers to smart home technology trying to help users manage their home. Now, almost everything in daily life is simplified. The Homuto system is no exception. It is a mobile application for Apple users to function all home devices together in one system. The system promises a futuristic home experience where all they need is a very simple voice command or a casual tap to the screen, which reduces the cumbersome management steps. The system achieves the goal of automation as well. Designated devices will automatically change their states at a specific time, or a special event assigned by users. Homuto not only allows users to better control over home devices functions and create a comfortable environment, but also helps with energy saving by monitoring energy consumption at any time. Security concerns are also considered. The system keeps the house safe by detecting if the door is opened or not to.

The proposed system is not far enough for a perfect smart home and users will require more and more for smart home in the future. There will always be new versions of the product. So, the system will be kept update to meet the needs of users and continually improved to become a better smart house technology.

7.1 Future Work

7.1.1 Security System

A useful security system allows users to program, monitor and control their home's security setup remotely via a mobile device, providing both safety and security for the home and a potential deterrent for criminals. However, security system plays only a minor role in the proposed system. To build a smart security system, a doorbell camera is considered to develop next. So, when someone rings the bell or approaches the door, the

doorbell sends an alert to the mobile device and activates the camera, letting user see a clear picture of who is there at any place. As well as smoke detector, alarm system, motion sensor, they will be added to the system one by one.

7.1.2 User Experience

The proposed system now meets the exact needs of the customer, without fuss or bother. A good experience of using the system can enhance users' loyalty to the brand. While developing various functions, we also need to improve the user's sense of use such as such as concise, pleasure, and efficiency.

References

- [1] Porter, J. (2019, October 28). *HomeKit might be fading, but Apple's not giving up yet*. The Verge. <https://www.theverge.com/2019/10/28/20936292/apple-homekit-hiring-engineers>
- [2] Leung, B. (2011). *VLSI for Wireless Communication* (2nd ed.). Springer Verlag.
- [3] Krishna, R., Devi, S., & RajaSneha, P. (2016). Home Automation Using Remote Control System. *International Journal for Research in Applied Science and Engineering Technology*, 4(4), 406–412.
<https://www.ijraset.com/files/serve.php?FID=5559>
- [4] Beal, V. (n.d.). *Wi-Fi (wireless networking)*. Webopedia. https://www.webopedia.com/TERM/W/Wi_Fi.html
- [5] Hardi, S. (n.d.). *Home Automation Protocols*. Simform. <https://www.simform.com/iot-home-automation-protocols/>
- [6] Daniel, S. (2020, January 22). *What is Z-Wave?* Lifewire. <https://www.lifewire.com/what-is-z-wave-4588924>
- [7] Yuliia, H. (2019, March 4). *How Secure Is the BLE Communication Standard?* Lemberg. <https://lebergsolutions.com/blog/how-secure-ble-communication-standard>
- [8] ESP8266. (2020, April 26). Retrieved May 15, 2020, from <https://en.wikipedia.org/wiki/ESP8266>
- [9] Xcode. (2020, May 12). Retrieved May 15, 2020, from <https://en.wikipedia.org/wiki/Xcode>
- [10] Swift (programming language). (2020, May 13). Retrieved May 15, 2020, from [https://en.wikipedia.org/wiki/Swift_\(programming_language\)](https://en.wikipedia.org/wiki/Swift_(programming_language))
- [11] Firebase. (2020, May 13). Retrieved May 15, 2020, from <https://en.wikipedia.org/wiki/Firebase>

Appendix

```
import Foundation
```

```
/// `AFError` is the error type returned by Alamofire. It encompasses a few different types  
of errors, each with
```

```
/// their own associated reasons.
```

```
public enum AFError: Error {
```

```
    /// The underlying reason the `.multipartEncodingFailed` error occurred.
```

```
    public enum MultipartEncodingFailureReason {
```

```
        /// The `fileURL` provided for reading an encodable body part isn't a file `URL`.
```

```
        case bodyPartURLInvalid(url: URL)
```

```
        /// The filename of the `fileURL` provided has either an empty `lastPathComponent`  
or `pathExtension`.
```

```
        case bodyPartFilenameInvalid(in: URL)
```

```
        /// The file at the `fileURL` provided was not reachable.
```

```
        case bodyPartFileNotReachable(at: URL)
```

```

    /// Attempting to check the reachability of the `fileURL` provided threw an error.

    case bodyPartFileNotReachableWithError(atURL: URL, error: Error)

    /// The file at the `fileURL` provided is actually a directory.

    case bodyPartFileIsDirectory(at: URL)

    /// The size of the file at the `fileURL` provided was not returned by the system.

    case bodyPartFileSizeNotAvailable(at: URL)

    /// The attempt to find the size of the file at the `fileURL` provided threw an error.

    case bodyPartFileSizeQueryFailedWithError(forURL: URL, error: Error)

    /// An `InputStream` could not be created for the provided `fileURL`.

    case bodyPartInputStreamCreationFailed(for: URL)

    /// An `OutputStream` could not be created when attempting to write the encoded data
    to disk.

    case outputStreamCreationFailed(for: URL)

    /// The encoded body data could not be written to disk because a file already exists at
    the provided `fileURL`.

    case outputStreamFileAlreadyExists(at: URL)

    /// The `fileURL` provided for writing the encoded body data to disk is not a file
    `URL`.

    case outputStreamURLInvalid(url: URL)

```

```

    /// The attempt to write the encoded body data to disk failed with an underlying error.

    case outputStreamWriteFailed(error: Error)

    /// The attempt to read an encoded body part `InputStream` failed with underlying
    system error.

    case inputStreamReadFailed(error: Error)

}

/// The underlying reason the `.parameterEncodingFailed` error occurred.

public enum ParameterEncodingFailureReason {

    /// The `URLRequest` did not have a `URL` to encode.

    case missingURL

    /// JSON serialization failed with an underlying system error during the encoding
    process.

    case jsonEncodingFailed(error: Error)

    /// Custom parameter encoding failed due to the associated `Error`.

    case customEncodingFailed(error: Error)

}

```

/// The underlying reason the `.parameterEncoderFailed` error occurred.

public enum ParameterEncoderFailureReason {

/// Possible missing components.

public enum RequiredComponent {

/// The `URL` was missing or unable to be extracted from the passed `URLRequest`
or during encoding.

case url

/// The `HTTPMethod` could not be extracted from the passed `URLRequest`.

case httpMethod(rawValue: String)

}

/// A `RequiredComponent` was missing during encoding.

case missingRequiredComponent(RequiredComponent)

/// The underlying encoder failed with the associated error.

case encoderFailed(error: Error)

}

/// The underlying reason the `.responseValidationFailed` error occurred.

```

public enum ResponseValidationFailureReason {

    /// The data file containing the server response did not exist.

    case dataFileNil

    /// The data file containing the server response at the associated `URL` could not be
    read.

    case dataFileReadFailed(at: URL)

    /// The response did not contain a `Content-Type` and the `acceptableContentTypes`
    provided did not contain a

    /// wildcard type.

    case missingContentType(acceptableContentTypes: [String])

    /// The response `Content-Type` did not match any type in the provided
    `acceptableContentTypes`.

    case unacceptableContentType(acceptableContentTypes: [String],
    responseContentType: String)

    /// The response status code was not acceptable.

    case unacceptableStatusCode(code: Int)

    /// Custom response validation failed due to the associated `Error`.

    case customValidationFailed(error: Error)

}

```



```

/// The underlying reason the response serialization error occurred.

public enum ResponseSerializationFailureReason {

    /// The server response contained no data or the data was zero length.

    case inputDataNilOrZeroLength

    /// The file containing the server response did not exist.

    case inputFileNil

    /// The file containing the server response could not be read from the associated `URL`.

    case inputFileReadFailed(at: URL)

    /// String serialization failed using the provided `String.Encoding`.

    case stringSerializationFailed(encoding: String.Encoding)

    /// JSON serialization failed with an underlying system error.

    case jsonSerializationFailed(error: Error)

    /// A `DataDecoder` failed to decode the response due to the associated `Error`.

    case decodingFailed(error: Error)

    /// A custom response serializer failed due to the associated `Error`.

    case customSerializationFailed(error: Error)

```

/// Generic serialization failed for an empty response that wasn't type `Empty` but instead the associated type.

```
case invalidEmptyResponse(type: String)

}
```

/// Underlying reason a server trust evaluation error occurred.

```
public enum ServerTrustFailureReason {
```

/// The output of a server trust evaluation.

```
public struct Output {
```

/// The host for which the evaluation was performed.

```
public let host: String
```

/// The `SecTrust` value which was evaluated.

```
public let trust: SecTrust
```

/// The `OSStatus` of evaluation operation.

```
public let status: OSStatus
```

/// The result of the evaluation operation.

```
public let result: SecTrustResultType
```

```

        /// Creates an `Output` value from the provided values.

        init(_ host: String, _ trust: SecTrust, _ status: OSStatus, _ result:
SecTrustResultType) {

            self.host = host

            self.trust = trust

            self.status = status

            self.result = result

        }

    }

    /// No `ServerTrustEvaluator` was found for the associated host.

    case noRequiredEvaluator(host: String)

    /// No certificates were found with which to perform the trust evaluation.

    case noCertificatesFound

    /// No public keys were found with which to perform the trust evaluation.

    case noPublicKeysFound

    /// During evaluation, application of the associated `SecPolicy` failed.

    case policyApplicationFailed(trust: SecTrust, policy: SecPolicy, status: OSStatus)

```

```

    /// During evaluation, setting the associated anchor certificates failed.

    case settingAnchorCertificatesFailed(status: OSStatus, certificates: [SecCertificate])

    /// During evaluation, creation of the revocation policy failed.

    case revocationPolicyCreationFailed

    /// `SecTrust` evaluation failed with the associated `Error`, if one was produced.

    case trustEvaluationFailed(error: Error?)

    /// Default evaluation failed with the associated `Output`.

    case defaultEvaluationFailed(output: Output)

    /// Host validation failed with the associated `Output`.

    case hostValidationFailed(output: Output)

    /// Revocation check failed with the associated `Output` and options.

    case revocationCheckFailed(output: Output, options:
    RevocationTrustEvaluator.Options)

    /// Certificate pinning failed.

    case certificatePinningFailed(host: String, trust: SecTrust, pinnedCertificates:
    [SecCertificate], serverCertificates: [SecCertificate])

    /// Public key pinning failed.

    case publicKeyPinningFailed(host: String, trust: SecTrust, pinnedKeys: [SecKey],
    serverKeys: [SecKey])

```

```
/// Custom server trust evaluation failed due to the associated `Error`.
```

```
case customEvaluationFailed(error: Error)
```

```
}
```

```
/// The underlying reason the `urlRequestValidationFailed`
```

```
public enum URLRequestValidationFailureReason {
```

```
/// URLRequest with GET method had body data.
```

```
case bodyDataInGETRequest(Data)
```

```
}
```

```
/// `UploadableConvertible` threw an error in `createUploadable()`.
```

```
case createUploadableFailed(error: Error)
```

```
/// `URLRequestConvertible` threw an error in `asURLRequest()`.
```

```
case createURLRequestFailed(error: Error)
```

```
/// `SessionDelegate` threw an error while attempting to move downloaded file to  
destination URL.
```

```
case downloadedFileMoveFailed(error: Error, source: URL, destination: URL)
```

```
/// `Request` was explicitly cancelled.
```

case explicitlyCancelled

/// `URLConvertible` type failed to create a valid `URL`.

case invalidURL(url: URLConvertible)

/// Multipart form encoding failed.

case multipartEncodingFailed(reason: MultipartEncodingFailureReason)

/// `ParameterEncoding` threw an error during the encoding process.

case parameterEncodingFailed(reason: ParameterEncodingFailureReason)

/// `ParameterEncoder` threw an error while running the encoder.

case parameterEncoderFailed(reason: ParameterEncoderFailureReason)

/// `RequestAdapter` threw an error during adaptation.

case requestAdaptationFailed(error: Error)

/// `RequestRetrier` threw an error during the request retry process.

case requestRetryFailed(retryError: Error, originalError: Error)

/// Response validation failed.

case responseValidationFailed(reason: ResponseValidationFailureReason)

/// Response serialization failed.

case responseSerializationFailed(reason: ResponseSerializationFailureReason)

```

    /// `ServerTrustEvaluating` instance threw an error during trust evaluation.

    case serverTrustEvaluationFailed(reason: ServerTrustFailureReason)

    /// `Session` which issued the `Request` was deinitialized, most likely because its
    reference went out of scope.

    case sessionDeinitialized

    /// `Session` was explicitly invalidated, possibly with the `Error` produced by the
    underlying `URLSession`.

    case sessionInvalidated(error: Error?)

    /// `URLSessionTask` completed with error.

    case sessionTaskFailed(error: Error)

    /// `URLRequest` failed validation.

    case urlRequestValidationFailed(reason: URLRequestValidationFailureReason)
}

extension Error {

    /// Returns the instance cast as an `AFError`.

    public var asAFError: AFError? {

        return self as? AFError
    }
}

```

```
}
```

/// Returns the instance cast as an `AFError`. If casting fails, a `fatalError` with the specified `message` is thrown.

```
public func asAFError(orFailWith message: @autoclosure () -> String, file: StaticString  
= #file, line: UInt = #line) -> AFError {
```

```
    guard let afError = self as? AFError else {
```

```
        fatalError(message(), file: file, line: line)
```

```
    }
```

```
    return afError
```

```
}
```

/// Casts the instance as `AFError` or returns `defaultAFError`

```
func asAFError(or defaultAFError: @autoclosure () -> AFError) -> AFError {
```

```
    return self as? AFError ?? defaultAFError()
```

```
}
```

```
}
```



```
// MARK: - Error Booleans
```

```
extension AFError {
```

```
    /// Returns whether the instance is `.sessionDeinitialized`.
```

```
    public var isSessionDeinitializedError: Bool {
```

```
        if case .sessionDeinitialized = self { return true }
```

```
        return false
```

```
    }
```

```
    /// Returns whether the instance is `.sessionInvalidated`.
```

```
    public var isSessionInvalidatedError: Bool {
```

```
        if case .sessionInvalidated = self { return true }
```

```
        return false
```

```
    }
```

```
    /// Returns whether the instance is `.explicitlyCancelled`.
```

```
    public var isExplicitlyCancelledError: Bool {
```

```

        if case .explicitlyCancelled = self { return true }

        return false

    }

```

/// Returns whether the instance is `.invalidURL`.

```

public var isInvalidURL: Bool {

    if case .invalidURL = self { return true }

    return false

}

```

/// Returns whether the instance is `.parameterEncodingFailed`. When `true`, the `underlyingError` property will

/// contain the associated value.

```

public var isParameterEncodingError: Bool {

    if case .parameterEncodingFailed = self { return true }

    return false

}

```

/// Returns whether the instance is `.parameterEncoderFailed`. When `true`, the `underlyingError` property will

/// contain the associated value.

```
public var isParameterEncoderError: Bool {  
  
    if case .parameterEncoderFailed = self { return true }  
  
    return false  
  
}
```

/// Returns whether the instance is `.multipartEncodingFailed`. When `true`, the `url` and `underlyingError`

/// properties will contain the associated values.

```
public var isMultipartEncodingError: Bool {  
  
    if case .multipartEncodingFailed = self { return true }  
  
    return false  
  
}
```

/// Returns whether the instance is `.requestAdaptationFailed`. When `true`, the `underlyingError` property will

/// contain the associated value.

```

public var isRequestAdaptationError: Bool {

    if case .requestAdaptationFailed = self { return true }

    return false

}

```

/// Returns whether the instance is `.responseValidationFailed`. When `true`, the `acceptableContentTypes`,

/// `responseContentType`, `responseCode`, and `underlyingError` properties will contain the associated values.

```

public var isResponseValidationError: Bool {

    if case .responseValidationFailed = self { return true }

    return false

}

```

/// Returns whether the instance is `.responseSerializationFailed`. When `true`, the `failedStringEncoding` and

/// `underlyingError` properties will contain the associated values.

```

public var isResponseSerializationError: Bool {

    if case .responseSerializationFailed = self { return true }

```

```
    return false  
  
}
```

/// Returns whether the instance is `.serverTrustEvaluationFailed`. When `true`, the `underlyingError` property will

/// contain the associated value.

```
public var isServerTrustEvaluationError: Bool {  
  
    if case .serverTrustEvaluationFailed = self { return true }  
  
    return false  
  
}
```

/// Returns whether the instance is `requestRetryFailed`. When `true`, the `underlyingError` property will

/// contain the associated value.

```
public var isRequestRetryError: Bool {  
  
    if case .requestRetryFailed = self { return true }  
  
    return false  
  
}
```

/// Returns whether the instance is `createUploadableFailed`. When `true`, the
`underlyingError` property will

/// contain the associated value.

```
public var isCreateUploadableError: Bool {  
  
    if case .createUploadableFailed = self { return true }  
  
    return false  
  
}
```

/// Returns whether the instance is `createURLRequestFailed`. When `true`, the
`underlyingError` property will

/// contain the associated value.

```
public var isCreateURLRequestError: Bool {  
  
    if case .createURLRequestFailed = self { return true }  
  
    return false  
  
}
```

/// Returns whether the instance is `downloadedFileMoveFailed`. When `true`, the
`destination` and `underlyingError` properties will

```
/// contain the associated values.
```

```
public var isDownloadedFileMoveError: Bool {  
  
    if case .downloadedFileMoveFailed = self { return true }  
  
    return false  
  
}
```

```
/// Returns whether the instance is `createURLRequestFailed`. When `true`, the  
`underlyingError` property will
```

```
/// contain the associated value.
```

```
public var isSessionTaskError: Bool {  
  
    if case .sessionTaskFailed = self { return true }  
  
    return false  
  
}  
  
}
```

```
// MARK: - Convenience Properties
```

```
extension AFError {
```

```
/// The `URLConvertible` associated with the error.
```

```
public var urlConvertible: URLConvertible? {  
  
    guard case let .invalidURL(url) = self else { return nil }  
  
    return url  
  
}
```

```
/// The `URL` associated with the error.
```

```
public var url: URL? {  
  
    guard case let .multipartEncodingFailed(reason) = self else { return nil }  
  
    return reason.url  
  
}
```

```
/// The underlying `Error` responsible for generating the failure associated with  
`.sessionInvalidated`,
```

```
/// `.parameterEncodingFailed`, `.parameterEncoderFailed`, `.multipartEncodingFailed`,  
`.requestAdaptationFailed`,
```

```
/// `.responseSerializationFailed`, `.requestRetryFailed` errors.
```

```
public var underlyingError: Error? {
```



```
switch self {  
  
case let .multipartEncodingFailed(reason):  
  
    return reason.underlyingError  
  
case let .parameterEncodingFailed(reason):  
  
    return reason.underlyingError  
  
case let .parameterEncoderFailed(reason):  
  
    return reason.underlyingError  
  
case let .requestAdaptationFailed(error):  
  
    return error  
  
case let .requestRetryFailed(retryError, _):  
  
    return retryError  
  
case let .responseValidationFailed(reason):  
  
    return reason.underlyingError  
  
case let .responseSerializationFailed(reason):  
  
    return reason.underlyingError  
  
case let .serverTrustEvaluationFailed(reason):  
  
    return reason.underlyingError
```

```

case let .sessionInvalidated(error):

    return error

case let .createUploadableFailed(error):

    return error

case let .createURLRequestFailed(error):

    return error

case let .downloadedFileMoveFailed(error, _, _):

    return error

case let .sessionTaskFailed(error):

    return error

case .explicitlyCancelled,

    .invalidURL,

    .sessionDeinitialized,

    .urlRequestValidationFailed:

    return nil

}

}

```

```

/// The acceptable `Content-Type`'s of a `.responseValidationFailed` error.

public var acceptableContentTypes: [String]? {

    guard case let .responseValidationFailed(reason) = self else { return nil }

    return reason.acceptableContentTypes

}

```

```

/// The response `Content-Type` of a `.responseValidationFailed` error.

public var responseContentType: String? {

    guard case let .responseValidationFailed(reason) = self else { return nil }

    return reason.responseContentType

}

```

```

/// The response code of a `.responseValidationFailed` error.

public var responseCode: Int? {

    guard case let .responseValidationFailed(reason) = self else { return nil }

    return reason.responseCode

}

```

```
}
```

```
/// The `String.Encoding` associated with a failed `.stringResponse()` call.
```

```
public var failedStringEncoding: String.Encoding? {
```

```
    guard case let .responseSerializationFailed(reason) = self else { return nil }
```

```
    return reason.failedStringEncoding
```

```
}
```

```
/// The `source` URL of a `.downloadedFileMoveFailed` error.
```

```
public var sourceURL: URL? {
```

```
    guard case let .downloadedFileMoveFailed(_, source, _) = self else { return nil }
```

```
    return source
```

```
}
```

```
/// The `destination` URL of a `.downloadedFileMoveFailed` error.
```

```
public var destinationURL: URL? {
```

```
    guard case let .downloadedFileMoveFailed(_, _, destination) = self else { return nil }
```

```

        return destination
    }
}

extension AFError.ParameterEncodingFailureReason {

    var underlyingError: Error? {

        switch self {

        case let .jsonEncodingFailed(error),

            let .customEncodingFailed(error):

            return error

        case .missingURL:

            return nil

        }

    }

}

```

```

extension AFError.ParameterEncoderFailureReason {

```

```

var underlyingError: Error? {

    switch self {

    case let .encoderFailed(error):

        return error

    case .missingRequiredComponent:

        return nil

    }

}

}

}

extension ALError.MultipartEncodingFailureReason {

    var url: URL? {

        switch self {

        case let .bodyPartURLInvalid(url),

            let .bodyPartFilenameInvalid(url),

            let .bodyPartFileNotReachable(url),

            let .bodyPartFileIsDirectory(url),

```

```

        let .bodyPartFileSizeNotAvailable(url),

        let .bodyPartInputStreamCreationFailed(url),

        let .outputStreamCreationFailed(url),

        let .outputStreamFileAlreadyExists(url),

        let .outputStreamURLInvalid(url),

        let .bodyPartFileNotReachableWithError(url, _),

        let .bodyPartFileSizeQueryFailedWithError(url, _):

        return url

    case .outputStreamWriteFailed,

        .inputStreamReadFailed:

        return nil

    }

}

var underlyingError: Error? {

    switch self {

        case let .bodyPartFileNotReachableWithError(_, error),

```

```

        let .bodyPartFileSizeQueryFailedWithError(_, error),

        let .outputStreamWriteFailed(error),

        let .inputStreamReadFailed(error):

        return error

    case .bodyPartURLInvalid,

        .bodyPartFilenameInvalid,

        .bodyPartFileNotReachable,

        .bodyPartFileIsDirectory,

        .bodyPartFileSizeNotAvailable,

        .bodyPartInputStreamCreationFailed,

        .outputStreamCreationFailed,

        .outputStreamFileAlreadyExists,

        .outputStreamURLInvalid:

        return nil

    }

}

}

```



```
extension ALError.ResponseValidationFailureReason {
```

```
    var acceptableContentTypes: [String]? {
```

```
        switch self {
```

```
            case let .missingContentType(types),
```

```
                 let .unacceptableContentType(types, _):
```

```
                return types
```

```
            case .dataFileNil,
```

```
                 .dataFileReadFailed,
```

```
                 .unacceptableStatusCode,
```

```
                 .customValidationFailed:
```

```
                return nil
```

```
        }
```

```
    }
```

```
    var responseContentType: String? {
```

```
        switch self {
```

```

case let .unacceptableContentType(_, responseType):

    return responseType

case .dataFileNil,

    .dataFileReadFailed,

    .missingContentType,

    .unacceptableStatusCode,

    .customValidationFailed:

    return nil

}

}

```

```

var responseCode: Int? {

    switch self {

        case let .unacceptableStatusCode(code):

            return code

        case .dataFileNil,

            .dataFileReadFailed,

```

```

        .missingContentType,

        .unacceptableContentType,

        .customValidationFailed:

    return nil

}

}

```

```

var underlyingError: Error? {

    switch self {

        case let .customValidationFailed(error):

            return error

        case .dataFileNil,

            .dataFileReadFailed,

            .missingContentType,

            .unacceptableContentType,

            .unacceptableStatusCode:

            return nil

```

```

    }

}

}

```

```

extension AFError.ResponseSerializationFailureReason {

```

```

    var failedStringEncoding: String.Encoding? {

```

```

        switch self {

```

```

            case let .stringSerializationFailed(encoding):

```

```

                return encoding

```

```

            case .inputDataNilOrZeroLength,

```

```

                .inputFileNil,

```

```

                .inputFileReadFailed(_),

```

```

                .jsonSerializationFailed(_),

```

```

                .decodingFailed(_),

```

```

                .customSerializationFailed(_),

```

```

                .invalidEmptyResponse:

```

```

                return nil

```

```

    }

}

var underlyingError: Error? {

    switch self {

        case let .jsonSerializationFailed(error),

            let .decodingFailed(error),

            let .customSerializationFailed(error):

            return error

        case .inputDataNilOrZeroLength,

            .inputFileNil,

            .inputFileReadFailed,

            .stringSerializationFailed,

            .invalidEmptyResponse:

            return nil

    }

}

```

```
}
```

```
extension ALError.ServerTrustFailureReason {  
  
    var output: ALError.ServerTrustFailureReason.Output? {  
  
        switch self {  
  
            case let .defaultEvaluationFailed(output),  
                 let .hostValidationFailed(output),  
                 let .revocationCheckFailed(output, _):  
  
                return output  
  
            case .noRequiredEvaluator,  
                 .noCertificatesFound,  
                 .noPublicKeysFound,  
                 .policyApplicationFailed,  
                 .settingAnchorCertificatesFailed,  
                 .revocationPolicyCreationFailed,  
                 .trustEvaluationFailed,  
                 .certificatePinningFailed,
```

```

        .publicKeyPinningFailed,

        .customEvaluationFailed:

        return nil

    }

}

var underlyingError: Error? {

    switch self {

    case let .customEvaluationFailed(error):

        return error

    case let .trustEvaluationFailed(error):

        return error

    case .noRequiredEvaluator,

        .noCertificatesFound,

        .noPublicKeysFound,

        .policyApplicationFailed,

        .settingAnchorCertificatesFailed,

```

```

        .revocationPolicyCreationFailed,

        .defaultEvaluationFailed,

        .hostValidationFailed,

        .revocationCheckFailed,

        .certificatePinningFailed,

        .publicKeyPinningFailed:

    return nil

}

}

}

```

// MARK: - Error Descriptions

```

extension AFError: LocalizedError {

    public var errorDescription: String? {

        switch self {

            case .explicitlyCancelled:

```



```

        return "Request explicitly cancelled."

    case let .invalidURL(url):

        return "URL is not valid: \(url)"

    case let .parameterEncodingFailed(reason):

        return reason.localizedDescription

    case let .parameterEncoderFailed(reason):

        return reason.localizedDescription

    case let .multipartEncodingFailed(reason):

        return reason.localizedDescription

    case let .requestAdaptationFailed(error):

        return "Request adaption failed with error: \(error.localizedDescription)"

    case let .responseValidationFailed(reason):

        return reason.localizedDescription

    case let .responseSerializationFailed(reason):

        return reason.localizedDescription

    case let .requestRetryFailed(retryError, originalError):

        return ""

```

Request retry failed with retry error: \(\retryError.localizedDescription), \

original error: \(\originalError.localizedDescription)

""

case .sessionDeinitialized:

return ""

Session was invalidated without error, so it was likely deinitialized unexpectedly. \

Be sure to retain a reference to your Session for the duration of your requests.

""

case let .sessionInvalidated(error):

return "Session was invalidated with error: \(\error?.localizedDescription ?? "No description.")"

case let .serverTrustEvaluationFailed(reason):

return "Server trust evaluation failed due to reason: \(\reason.localizedDescription)"

case let .urlRequestValidationFailed(reason):

return "URLRequest validation failed due to reason: \(\reason.localizedDescription)"

case let .createUploadableFailed(error):

return "Uploadable creation failed with error: \(\error.localizedDescription)"

```

    case let .createURLRequestFailed(error):

        return "URLRequest creation failed with error: \(error.localizedDescription)"

    case let .downloadedFileMoveFailed(error, source, destination):

        return "Moving downloaded file from: \(source) to: \(destination) failed with error:
\(error.localizedDescription)"

    case let .sessionTaskFailed(error):

        return "URLSessionTask failed with error: \(error.localizedDescription)"

    }

}

}

```

```

extension AFError.ParameterEncodingFailureReason {

    var localizedDescription: String {

        switch self {

            case .missingURL:

                return "URL request to encode was missing a URL"

            case let .jsonEncodingFailed(error):

```

```
        return "JSON could not be encoded because of  
error:\n\(error.localizedDescription)"
```

```
    case let .customEncodingFailed(error):
```

```
        return "Custom parameter encoder failed with error: \(error.localizedDescription)"
```

```
    }
```

```
}
```

```
}
```

```
extension AFError.ParameterEncoderFailureReason {
```

```
    var localizedDescription: String {
```

```
        switch self {
```

```
        case let .missingRequiredComponent(component):
```

```
            return "Encoding failed due to a missing request component: \(component)"
```

```
        case let .encoderFailed(error):
```

```
            return "The underlying encoder failed with the error: \(error)"
```

```
        }
```

```
    }
```

```
}
```

```

extension ALError.MultipartEncodingFailureReason {

    var localizedDescription: String {

        switch self {

            case let .bodyPartURLInvalid(url):

                return "The URL provided is not a file URL: \(url)"

            case let .bodyPartFilenameInvalid(url):

                return "The URL provided does not have a valid filename: \(url)"

            case let .bodyPartFileNotReachable(url):

                return "The URL provided is not reachable: \(url)"

            case let .bodyPartFileNotReachableWithError(url, error):

                return ""

                The system returned an error while checking the provided URL for reachability.

                URL: \(url)

                Error: \(error)

                ""

            case let .bodyPartFileIsDirectory(url):

```

```
return "The URL provided is a directory: \url)"
```

```
case let .bodyPartFileSizeNotAvailable(url):
```

```
    return "Could not fetch the file size from the provided URL: \url)"
```

```
case let .bodyPartFileSizeQueryFailedWithError(url, error):
```

```
    return ""
```

The system returned an error while attempting to fetch the file size from the provided URL.

```
    URL: \url)
```

```
    Error: \error)
```

```
    ""
```

```
case let .bodyPartInputStreamCreationFailed(url):
```

```
    return "Failed to create an InputStream for the provided URL: \url)"
```

```
case let .outputStreamCreationFailed(url):
```

```
    return "Failed to create an OutputStream for URL: \url)"
```

```
case let .outputStreamFileAlreadyExists(url):
```

```
    return "A file already exists at the provided URL: \url)"
```

```
case let .outputStreamURLInvalid(url):
```

```
    return "The provided OutputStream URL is invalid: \url)"
```

```

    case let .outputStreamWriteFailed(error):

        return "OutputStream write failed with error: \(error)"

    case let .inputStreamReadFailed(error):

        return "InputStream read failed with error: \(error)"

    }

}

}

```

```

extension AFError.ResponseSerializationFailureReason {

    var localizedDescription: String {

        switch self {

            case .inputDataNilOrZeroLength:

                return "Response could not be serialized, input data was nil or zero length."

            case .inputFileNil:

                return "Response could not be serialized, input file was nil."

            case let .inputFileReadFailed(url):

                return "Response could not be serialized, input file could not be read: \(url)."

```

```

case let .stringSerializationFailed(encoding):

    return "String could not be serialized with encoding: \(encoding)."

case let .jsonSerializationFailed(error):

    return "JSON could not be serialized because of
error:\n\(error.localizedDescription)"

case let .invalidEmptyResponse(type):

    return ""

    Empty response could not be serialized to type: \(type). \

    Use Empty as the expected type for such responses.

    ""

case let .decodingFailed(error):

    return "Response could not be decoded because of
error:\n\(error.localizedDescription)"

case let .customSerializationFailed(error):

    return "Custom response serializer failed with
error:\n\(error.localizedDescription)"

}

}

}

```



```

extension ALError.ResponseValidationFailureReason {

    var localizedDescription: String {

        switch self {

            case .dataFileNil:

                return "Response could not be validated, data file was nil."

            case let .dataFileReadFailed(url):

                return "Response could not be validated, data file could not be read: \(url)."

            case let .missingContentType(types):

                return ""

                Response Content-Type was missing and acceptable content types \

                \(types.joined(separator: ",")) do not match "*/*".

                ""

            case let .unacceptableContentType(acceptableTypes, responseType):

                return ""

                Response Content-Type "\(responseType)" does not match any acceptable types: \

                \(acceptableTypes.joined(separator: ",")).

```

```

        ""

    case let .unacceptableStatusCode(code):

        return "Response status code was unacceptable: \(code)."

    case let .customValidationFailed(error):

        return "Custom response validation failed with error: \(error.localizedDescription)"

    }

}

}

}

extension AFError.ServerTrustFailureReason {

    var localizedDescription: String {

        switch self {

        case let .noRequiredEvaluator(host):

            return "A ServerTrustEvaluating value is required for host \(host) but none was found."

        case .noCertificatesFound:

            return "No certificates were found or provided for evaluation."

        case .noPublicKeysFound:

```

```

        return "No public keys were found or provided for evaluation."

    case .policyApplicationFailed:

        return "Attempting to set a SecPolicy failed."

    case .settingAnchorCertificatesFailed:

        return "Attempting to set the provided certificates as anchor certificates failed."

    case .revocationPolicyCreationFailed:

        return "Attempting to create a revocation policy failed."

    case let .trustEvaluationFailed(error):

        return "SecTrust evaluation failed with error: \(error?.localizedDescription ??
"None")"

    case let .defaultEvaluationFailed(output):

        return "Default evaluation failed for host \(output.host)."

    case let .hostValidationFailed(output):

        return "Host validation failed for host \(output.host)."

    case let .revocationCheckFailed(output, _):

        return "Revocation check failed for host \(output.host)."

    case let .certificatePinningFailed(host, _, _, _):

        return "Certificate pinning failed for host \(host)."

```

```

case let .publicKeyPinningFailed(host, _, _, _):

    return "Public key pinning failed for host \(host)."

case let .customEvaluationFailed(error):

    return "Custom trust evaluation failed with error: \(error.localizedDescription)"

    }

    }

}

```

```

extension AFError.URLRequestValidationFailureReason {

    var localizedDescription: String {

        switch self {

        case let .bodyDataInGETRequest(data):

            return ""

            Invalid URLRequest: Requests with GET method cannot have body data:

            \(String(decoding: data, as: UTF8.self))

            ""

        }
    }
}

```

```

    }

}

//

// ATCCClassicLandingScreenViewController.swift

// DashboardApp

//

// Created by Florian Marcu on 8/8/18.

// Copyright © 2018 Instamobile. All rights reserved.

//

```

```
import UIKit
```

```
class ATCCClassicLandingScreenViewController: UIViewController {
```

```
    @IBOutlet var logoImageView: UIImageView!
```

```
    @IBOutlet var subtitleLabel: UILabel!
```

```
    @IBOutlet var titleLabel: UILabel!
```

```
    @IBOutlet var loginButton: UIButton!
```

```
@IBOutlet var signUpButton: UIButton!
```

```
private let backgroundColor: UIColor = .white
```

```
private let tint_color = UIColor(hexString: "#9BCC5F")
```

```
private let subtitleColor = UIColor(hexString: "#464646")
```

```
private let signUpButtonColor = UIColor(hexString: "#414665")
```

```
private let signUpButtonBorderColor = UIColor(hexString: "#B0B3C6")
```

```
private let titleFont = UIFont.boldSystemFont(ofSize: 30)
```

```
private let subtitleFont = UIFont.boldSystemFont(ofSize: 18)
```

```
private let buttonFont = UIFont.boldSystemFont(ofSize: 24)
```

```
override func viewDidLoad() {
```

```
    super.viewDidLoad()
```

```
    view.backgroundColor = HelperDarkMode.mainThemeBackgroundColor
```

```
    logoImageView.image = UIImage.localImage("Home-icon")
```

```
//logoImageView.tintColor = tintColor
```

```
titleLabel.font = titleFont
```

```
titleLabel.text = "Welcome to Homuto"
```

```
titleLabel.textColor = HelperDarkMode.mainTextColor
```

```
subtitleLabel.font = subtitleFont
```

```
subtitleLabel.text = "Start control your home."
```

```
subtitleLabel.textColor = HelperDarkMode.mainTextColor
```

```
loginButton.setTitle("Log in", for: .normal)
```

```
loginButton.addTarget(self, action: #selector(didTapLoginButton),  
for: .touchUpInside)
```

```
loginButton.configure(color: .white,  
  
font: buttonFont,  
  
cornerRadius: 55/2,  
  
backgroundColor: tintColor)
```

```
        loginButton.addTarget(self,          action:      #selector(didTapLoginButton),
for: .touchUpInside)
```

```
        signUpButton.setTitle("Sign Up", for: .normal)
```

```
        signUpButton.addTarget(self,          action:      #selector(didTapSignUpButton),
for: .touchUpInside)
```

```
        signUpButton.configure(color: signUpButtonColor,
```

```
                                font: buttonFont,
```

```
                                cornerRadius: 55/2,
```

```
                                borderColor: signUpButtonBorderColor,
```

```
                                backgroundColor: backgroundColor,
```

```
                                borderWidth: 1)
```

```
        signUpButton.addTarget(self,          action:      #selector(didTapSignUpButton),
for: .touchUpInside)
```

```
    }
```

```
    override func viewWillAppear(_ animated: Bool) {
```

```
        super.viewWillAppear(animated)
```

```
        self.navigationController?.setNavigationBarHidden(true, animated: false)
```



```
}
```

```
override func viewWillDisappear(_ animated: Bool) {
```

```
    self.navigationController?.setNavigationBarHidden(false, animated: false)
```

```
}
```

```
@objc private func didTapLoginButton() {
```

```
    let loginVC = ATCClassicLoginScreenViewController(nibName:  
"ATCClassicLoginScreenViewController", bundle: nil)
```

```
    self.navigationController?.pushViewController(loginVC, animated: true)
```

```
}
```

```
@objc private func didTapSignUpButton() {
```

```
    let signUpVC = ATCClassicSignUpViewController(nibName:  
"ATCClassicSignUpViewController", bundle: nil)
```

```
    self.navigationController?.pushViewController(signUpVC, animated: true)
```

```
}
```

```
}
```

```
//  
  
// ATCClassicLoginScreenViewController.swift  
  
// DashboardApp  
  
//  
  
// Created by Florian Marcu on 8/9/18.  
  
// Copyright © 2018 Instamobile. All rights reserved.  
  
//
```

```
import FBSDKCoreKit
```

```
import FBSDKLoginKit
```

```
import FirebaseAuth
```

```
import UIKit
```

```
class ATCClassicLoginScreenViewController: UIViewController {
```

```
    @IBOutlet var titleLabel: UILabel!
```

```
    @IBOutlet var passwordTextField: ATCTextField!
```

@IBOutlet var contactPointTextField: ATCTextField!

@IBOutlet var loginButton: UIButton!

@IBOutlet var separatorLabel: UILabel!

@IBOutlet var facebookButton: UIButton!

@IBOutlet var backButton: UIButton!

private let backgroundColor = HelperDarkMode.mainThemeBackgroundColor

private let tintColor = UIColor(hexString: "#9BCC5F")

private let titleFont = UIFont.boldSystemFont(ofSize: 30)

private let buttonFont = UIFont.boldSystemFont(ofSize: 20)

private let textFieldFont = UIFont.systemFont(ofSize: 16)

private let textFieldColor = UIColor(hexString: "#B0B3C6")

private let textFieldBorderColor = UIColor(hexString: "#B0B3C6")

private let separatorFont = UIFont.boldSystemFont(ofSize: 14)

```
private let separatorTextColor = UIColor(hexString: "#464646")

override func viewDidLoad() {

    super.viewDidLoad()

    view.backgroundColor = backgroundColor

    backButton.setImage(UIImage.localImage("arrow-back-icon",
for: .normal),
template: true),

    backButton.tintColor = UIColor(hexString: "#282E4F")

    backButton.addTarget(self, action: #selector(didTapBackButton), for: .touchUpInside)


    titleLabel.font = titleFont

    titleLabel.text = "Log In"

    titleLabel.textColor = tintColor


    contactPointTextField.configure(color: textFieldColor,

font: textFieldFont,

cornerRadius: 55/2,

borderColor: textFieldBorderColor,
```

```

        backgroundColor: backgroundColor,

        borderWidth: 1.0)

contactPointTextField.placeholder = "E-mail"

contactPointTextField.textContentType = .emailAddress

contactPointTextField.clipsToBounds = true


passwordTextField.configure(color: textFieldColor,

        font: textFieldFont,

        cornerRadius: 55/2,

        borderColor: textFieldBorderColor,

        backgroundColor: backgroundColor,

        borderWidth: 1.0)

passwordTextField.placeholder = "Password"

passwordTextField.isSecureTextEntry = true

passwordTextField.textContentType = .emailAddress

passwordTextField.clipsToBounds = true

```

```
separatorLabel.font = separatorFont
```

```
separatorLabel.textColor = separatorTextColor
```

```
separatorLabel.text = "OR"
```

```
loginButton.setTitle("Log In", for: .normal)
```

```
loginButton.addTarget(self, action: #selector(didTapLoginButton), for: .touchUpInside)
```

```
loginButton.configure(color: backgroundColor,
```

```
    font: buttonFont,
```

```
    cornerRadius: 55/2,
```

```
    backgroundColor: tintColor)
```

```
facebookButton.setTitle("Facebook Login", for: .normal)
```

```
facebookButton.addTarget(self,    action:    #selector(didTapFacebookButton),  
for: .touchUpInside)
```

```
facebookButton.configure(color: backgroundColor,
```

```
    font: buttonFont,
```

```
    cornerRadius: 55/2,
```

```
    backgroundColor: UIColor(hexString: "#334D92"))
```

```
self.hideKeyboardWhenTappedAround()

}
```

```
override func viewWillAppear(_ animated: Bool) {

    super.viewWillAppear(animated)

    self.navigationController?.setNavigationBarHidden(true, animated: false)

}
```

```
override func viewWillDisappear(_ animated: Bool) {

    self.navigationController?.setNavigationBarHidden(false, animated: false)

}
```

```
@objc func didTapBackButton() {

    self.navigationController?.popViewController(animated: true)

}
```

```
@objc func didTapLoginButton() {
```

```

let loginManager = FirebaseAuthManager()

guard let email = contactPointTextField.text, let password = passwordTextField.text
else { return }

loginManager.signIn(email: email, pass: password) {[weak self] (success) in

    guard let `self` = self else { return }

    var message: String = ""

    if (success) {

        let Storyboard = UIStoryboard(name: "Main", bundle: nil)

        let vc = Storyboard.instantiateViewController(withIdentifier: "view")

        vc.modalPresentationStyle = .fullScreen

        self.present(vc , animated: true , completion: nil)

    } else {

        message = "There was an error."

    }

    let alertController = UIAlertController(title: nil, message: message,
preferredStyle: .alert)

    alertController.addAction(UIAlertAction(title: "OK", style: .cancel, handler: nil))

    self.display(alertController: alertController)

```



```
}  
  
}
```

```
@objc func didTapFacebookButton() {
```

```
    let loginManager = LoginManager()
```

```
    loginManager.logIn(permissions: ["email"], from: self) { (result, error) in
```

```
        if error != nil {
```

```
            self.showPopup(isSuccess: false)
```

```
            return
```

```
        }
```

```
        guard let token = AccessToken.current else {
```

```
            print("Failed to get access token")
```

```
            self.showPopup(isSuccess: false)
```

```
            return
```

```
        }
```

```

        FirebaseAuthManager().login(credential:
FacebookAuthProvider.credential(withAccessToken: token.tokenString)) {[weak self]
(success) in

        self?.showPopup(isSuccess: true)

    }

}

}

}

@objc func pushToHome() {

let Storyboard = UIStoryboard(name: "Main", bundle: nil)

let vc = Storyboard.instantiateViewController(withIdentifier: "view")

vc.modalPresentationStyle = .fullScreen

present(vc , animated: true , completion: nil)

}

func display(alertController: UIAlertController) {

self.present(alertController, animated: true, completion: nil)

}

```

```

    }

    extension ATCClassicLoginScreenViewController {

        func showPopup(isSuccess: Bool) {

            let Storyboard = UIStoryboard(name: "Main", bundle: nil)

            let vc = Storyboard.instantiateViewController(withIdentifier: "view")

            vc.modalPresentationStyle = .fullScreen

            present(vc , animated: true , completion: nil)

        }

    }

    //

    // ATCClassicSignUpViewController.swift

    // DashboardApp

    //

    // Created by Florian Marcu on 8/10/18.

    // Copyright © 2018 Instamobile. All rights reserved.

```

```
//
```

```
import UIKit
```

```
class ATCClassicSignUpViewController: UIViewController {
```

```
    @IBOutlet var titleLabel: UILabel!
```

```
    @IBOutlet var containerView: UIView!
```

```
    @IBOutlet var backButton: UIButton!
```

```
    @IBOutlet var nameTextField: ATCTextField!
```

```
    @IBOutlet var phoneNumberTextField: ATCTextField!
```

```
    @IBOutlet var passwordTextField: ATCTextField!
```

```
    @IBOutlet var emailTextField: ATCTextField!
```

```
    @IBOutlet var errorLabel: UILabel!
```

```
    @IBOutlet var signUpButton: UIButton!
```

```
    private let tintColor = UIColor(hexString: "#9BCC5F")
```

```
    private let backgroundColor: UIColor =  
    HelperDarkMode.mainThemeBackgroundColor
```

```
private let textFieldColor = UIColor(hexString: "#B0B3C6")
```

```
private let textFieldBorderColor = UIColor(hexString: "#B0B3C6")
```

```
private let titleFont = UIFont.boldSystemFont(ofSize: 30)
```

```
private let textFieldFont = UIFont.systemFont(ofSize: 16)
```

```
private let buttonFont = UIFont.boldSystemFont(ofSize: 20)
```

```
override func viewDidLoad() {
```

```
    super.viewDidLoad()
```

```
    view.backgroundColor = backgroundColor
```

```
    let color = UIColor(hexString: "#282E4F")
```

```
    backButton.tintColor = color
```

```
    backButton.addTarget(self, action: #selector(didTapBackButton),  
for: .touchUpInside)
```

```
    titleLabel.font = titleFont
```

```
    titleLabel.text = "Sign Up"
```

```
titleLabel.textColor = tintColor
```

```
nameTextField.configure(color: textFieldColor,
```

```
    font: textFieldFont,
```

```
    cornerRadius: 40/2,
```

```
    borderColor: textFieldBorderColor,
```

```
    backgroundColor: backgroundColor,
```

```
    borderWidth: 1.0)
```

```
nameTextField.placeholder = "Full Name"
```

```
nameTextField.clipsToBounds = true
```

```
emailTextField.configure(color: textFieldColor,
```

```
    font: textFieldFont,
```

```
    cornerRadius: 40/2,
```

```
    borderColor: textFieldBorderColor,
```

```
    backgroundColor: backgroundColor,
```

```
    borderWidth: 1.0)
```

```
emailTextField.placeholder = "E-mail Address"
```

```
emailTextField.clipsToBounds = true
```

```
phoneNumberTextField.configure(color: textFieldColor,
```

```
    font: textFieldFont,
```

```
    cornerRadius: 40/2,
```

```
    borderColor: textFieldBorderColor,
```

```
    backgroundColor: backgroundColor,
```

```
    borderWidth: 1.0)
```

```
phoneNumberTextField.placeholder = "Phone Number"
```

```
phoneNumberTextField.clipsToBounds = true
```

```
passwordTextField.configure(color: textFieldColor,
```

```
    font: textFieldFont,
```

```
    cornerRadius: 40/2,
```

```
    borderColor: textFieldBorderColor,
```

```
    backgroundColor: backgroundColor,
```

```

        borderWidth: 1.0)

passwordTextField.placeholder = "Password"

passwordTextField.isSecureTextEntry = true

passwordTextField.clipsToBounds = true


signUpButton.setTitle("Create Account", for: .normal)

signUpButton.addTarget(self,          action:          #selector(didTapSignUpButton),
for: .touchUpInside)

signUpButton.configure(color: backgroundColor,

                        font: buttonFont,

                        cornerRadius: 40/2,

                        backgroundColor: UIColor(hexString: "#334D92"))


self.hideKeyboardWhenTappedAround()

}


override func viewWillAppear(_ animated: Bool) {

    super.viewWillAppear(animated)

```



```
        self.navigationController?.setNavigationBarHidden(true, animated: false)

    }
}
```

```
override func viewWillDisappear(_ animated: Bool) {

    self.navigationController?.setNavigationBarHidden(false, animated: false)

}
}
```

```
@objc func didTapBackButton() {

    self.navigationController?.popViewController(animated: true)

}
}
```

```
@objc func didTapSignUpButton() {

    let signUpManager = FirebaseAuthManager()

    if let email = emailTextField.text, let password = passwordTextField.text {

        signUpManager.createUser(email: email, password: password) {[weak self]
(success) in

            guard let `self` = self else { return }

            var message: String = ""


```

```

        if (success) {

            message = "User was sucessfully created."

            let loginVC = ATCClassicLoginScreenViewController(nibName:
"ATCClassicLoginScreenViewController", bundle: nil)

            self.navigationController?.pushViewController(loginVC, animated: true)

        } else {

            message = "There was an error."

        }

        let alertController = UIAlertController(title: nil, message: message,
preferredStyle: .alert)

        alertController.addAction(UIAlertAction(title: "OK", style: .cancel, handler: nil))

        self.display(alertController: alertController)

    }

}

}

```

```

func display(alertController: UIAlertController) {

    self.present(alertController, animated: true, completion: nil)

```

```

    }

}

//

// ATCTextField.swift

// FirebaseStarterApp

//

// Created by Florian Marcu on 2/21/19.

// Copyright © 2019 Instamobile. All rights reserved.

//

import UIKit

import UIKit

class ATCTextField: UITextField {

    let padding = UIEdgeInsets(top: 0, left: 15, bottom: 0, right: 5)

```

```

override open func textRect(forBounds bounds: CGRect) -> CGRect {

    return bounds.inset(by: padding)

}

override open func placeholderRect(forBounds bounds: CGRect) -> CGRect {

    return bounds.inset(by: padding)

}

override open func editingRect(forBounds bounds: CGRect) -> CGRect {

    return bounds.inset(by: padding)

}

}

//

//  MIT License

//

//  Copyright (c) 2017 Robert-Hein Hooijmans <rh.hooijmans@gmail.com>

```

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY

KIND, EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF

MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO

EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,

DAMAGES OR OTHER

```
//      LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR  
OTHERWISE, ARISING FROM,
```

```
//  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER  
DEALINGS IN
```

```
//  THE SOFTWARE.
```

```
import Foundation
```

```
#if os(OSX)
```

```
import AppKit
```

```
public typealias View = NSView
```

```
public typealias LayoutGuide = NSLayoutGuide
```

```
public typealias ConstraintAxis = NSLayoutConstraint.Orientation
```

```
public typealias LayoutPriority = NSLayoutConstraint.Priority
```

```
public typealias TinyEdgeInsets = UIEdgeInsets
```

```
public extension UIEdgeInsets {
```

```

        static var zero = NSEdgeInsetsZero

    }

#else

    import UIKit

    public typealias View = UIView

    public typealias LayoutGuide = UILayoutGuide

    public typealias ConstraintAxis = NSLayoutConstraint.Axis

    public typealias LayoutPriority = UILayoutPriority

    public typealias TinyEdgeInsets = UIEdgeInsets

#endif

// Copyright (c) 2016-present, Facebook, Inc. All rights reserved.

//

// You are hereby granted a non-exclusive, worldwide, royalty-free license to use,

// copy, modify, and distribute this software in source code or binary form for use

```

// in connection with the web services and APIs provided by Facebook.

//

// As with any software that integrates with the Facebook platform, your use of

// this software is subject to the Facebook Developer Principles and Policies

// [<http://developers.facebook.com/policy/>]. This copyright notice shall be

// included in all copies or substantial portions of the software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS

// FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR

// COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER

// IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN

// CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.


```
import Foundation
```

```
/**
```

```
    AccessToken Extension
```

```
*/
```

```
public extension AccessToken {
```

```
    /**
```

```
        Returns the known granted permissions.
```

```
    */
```

```
    var permissions: Set<Permission> {
```

```
        return Set(__permissions.map { Permission(stringLiteral: $0) })
```

```
    }
```

```
    /**
```

```
        Returns the known declined permissions.
```

```
    */
```

```
    var declinedPermissions: Set<Permission> {
```

```

return Set(__declinedPermissions.map { Permission(stringLiteral: $0) })

}

```

```

/**

```

Returns the known expired permissions.

```

*/

```

```

var expiredPermissions: Set<Permission> {

    return Set(__expiredPermissions.map { Permission(stringLiteral: $0) })

}

```

```

/**

```

Convenience getter to determine if a permission has been granted

- parameter permission: The permission to check

```

*/

```

```

func hasGranted(_ permission: Permission) -> Bool {

    return hasGranted(permission: permission.name)

}

```

```
}
```

```
/*
```

See LICENSE folder for this sample's licensing information.

Abstract:

Computed accessory properties.

```
*/
```

```
import HomeKit
```

```
extension HMAccessory {
```

```
    /// A description for presentation in the UI.
```

```
    var displayName: String {
```

```
        let string = category.categoryType == HMAccessoryCategoryTypeOther ?  
        "Accessory" : category.localizedDescription
```

```
        return (isBridged ? "Bridged " : "") + string
```

```
    }
```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `AccessoryBrowserViewController` displays new accessories and allows the user to pair with them.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
import ExternalAccessory
```

```
/// Represents an accessory type and encapsulated accessory.
```

```
enum AccessoryType: Equatable, Nameable {
```

```
    /// A HomeKit object
```

```
    case homeKit(accessory: HMAccessory)
```

```

/// An external, `EAWiFiUnconfiguredAccessory` object

case external(accessory: EAWiFiUnconfiguredAccessory)

/// The name of the accessory.

var name: String {

    return accessory.name

}

/// The accessory within the `AccessoryType`.

var accessory: AnyObject {

    switch self {

        case .homeKit(let accessory):

            return accessory

        case .external(let accessory):

            return accessory

```

```

    }

}

}

```

/// Comparison of `AccessoryType`s based on name.

```

func ==(lhs: AccessoryType, rhs: AccessoryType) -> Bool {

    return lhs.name == rhs.name

}

```

```

/**

```

A view controller that displays a list of nearby accessories and allows the user to add them to the provided HMHome.

```

*/

```

```

class      AccessoryBrowserViewController:      HMCatalogViewController,
ModifyAccessoryDelegate,      EAWiFiUnconfiguredAccessoryBrowserDelegate,
HMAccessoryBrowserDelegate {

```

```

    // MARK: Types

```

```

struct Identifiers {

    static let accessoryCell = "AccessoryCell"

    static let addedAccessoryCell = "AddedAccessoryCell"

    static let addAccessorySegue = "Add Accessory"

}


// MARK: Properties


var addedAccessories = [HMAccessory]()

var displayedAccessories = [AccessoryType]()

let accessoryBrowser = HMAccessoryBrowser()

var externalAccessoryBrowser: EAWiFiUnconfiguredAccessoryBrowser?


// MARK: View Methods


/// Configures the table view and initializes the accessory browsers.

override func viewDidLoad() {

```

```
tableView.estimatedRowHeight = 44.0
```

```
tableView.rowHeight = UITableView.automaticDimension
```

```
accessoryBrowser.delegate = self
```

```
#if arch(arm)
```

```
// We can't use the ExternalAccessory framework on the iPhone simulator.
```

```
externalAccessoryBrowser = EAWiFiUnconfiguredAccessoryBrowser(delegate: self,  
queue: DispatchQueue.main)
```

```
#endif
```

```
startBrowsing()
```

```
}
```

```
/// Reloads the view.
```

```
override func viewWillAppear(_ animated: Bool) {
```

```
    super.viewWillAppear(animated)
```

```
    reloadTable()
```

```
}
```



```
// MARK: IBAction Methods
```

```
/// Stops browsing and dismisses the view controller.
```

```
@IBAction func dismiss(_ sender: AnyObject) {  
  
    stopBrowsing()  
  
    self.dismiss(animated: true, completion: nil)  
  
}
```

```
/// Sets the accessory, home, and delegate of a ModifyAccessoryViewController.
```

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
  
    super.prepare(for: segue, sender: sender)  
  
    if let sender = sender as? HMAccessory , segue.identifier ==  
Identifiers.addAccessorySegue {  
  
        let modifyViewController = segue.intendedDestinationViewController as!  
ModifyAccessoryViewController  
  
        modifyViewController.accessory = sender  
    }  
}
```

```

        modifyViewController.delegate = self

    }

}

// MARK: Table View Methods

/**
    Generates the number of rows based on the number of displayed accessories.

    This method will also display a table view background message, if required.

    - returns: The number of rows based on the number of displayed accessories.
*/

override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {

    let rows = displayedAccessories.count

    if rows == 0 {

```

```
        let message = NSLocalizedString("No Discovered Accessories", comment: "No  
Discovered Accessories")
```

```
        setBackgroundMessage(message)
```

```
    }
```

```
    else {
```

```
        setBackgroundMessage(nil)
```

```
    }
```

```
    return rows
```

```
}
```

```
/**
```

- returns: Creates a cell that lists an accessory, and if it hasn't been added to the home,

shows a disclosure indicator instead of a checkmark.

```
*/
```

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath:  
IndexPath) -> UITableViewCell {
```

```
    let accessoryType = displayedAccessories[indexPath.row]
```

```

var reuseIdentifier = Identifiers.accessoryCell

        if case let .homeKit(hmAccessory) = accessoryType,
addedAccessories.contains(hmAccessory) {

            reuseIdentifier = Identifiers.addedAccessoryCell

        }

        let cell = tableView.dequeueReusableCell(withIdentifier: reuseIdentifier, for:
indexPath)

        cell.textLabel?.text = accessoryType.name

        if let accessory = accessoryType.accessory as? HMAccessory {

            cell.detailTextLabel?.text = accessory.category.localizedDescription

        }

        else {

            cell.detailTextLabel?.text = NSLocalizedString("External Accessory", comment:
"External Accessory")

        }

```

```

        return cell
    }

    /// Configures the accessory based on its type.

    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {

        switch displayedAccessories[indexPath.row] {

            case .homeKit(let accessory):

                configureAccessory(accessory)

            case .external(let accessory):

                externalAccessoryBrowser?.configureAccessory(accessory,
withConfigurationUIOn: self)

        }

    }

    // MARK: Helper Methods

```

```
/// Starts browsing on both HomeKit and External accessory browsers.
```

```
private func startBrowsing(){  
  
    accessoryBrowser.startSearchingForNewAccessories()  
  
    externalAccessoryBrowser?.startSearchingForUnconfiguredAccessories(matching:  
nil)  
  
}
```

```
/// Stops browsing on both HomeKit and External accessory browsers.
```

```
private func stopBrowsing(){  
  
    accessoryBrowser.stopSearchingForNewAccessories()  
  
    externalAccessoryBrowser?.stopSearchingForUnconfiguredAccessories()  
  
}
```

```
/**
```

Concatenates and sorts the discovered and added accessories.

- returns: A sorted list of all accessories involved with this

browser session.

*/

```
func allAccessories() -> [AccessoryType] {
```

```
    var accessories = [AccessoryType]()
```

```
    accessories += accessoryBrowser.discoveredAccessories.map { .homeKit(accessory: $0) }
```

```
    accessories += addedAccessories.compactMap { addedAccessory in
```

```
        let accessoryType = AccessoryType.homeKit(accessory: addedAccessory)
```

```
        return accessories.contains(accessoryType) ? nil : accessoryType
```

```
    }
```

```
    if let external = externalAccessoryBrowser?.unconfiguredAccessories {
```

```
        let unconfiguredAccessoriesArray = Array(external)
```

```
        accessories += unconfiguredAccessoriesArray.compactMap { addedAccessory in
```

```
            let accessoryType = AccessoryType.external(accessory: addedAccessory)
```

```

        return accessories.contains(accessoryType) ? nil : accessoryType

    }

}

return accessories.sortByLocalizedNames()

}

/// Updates the displayed accessories array and reloads the table view.

private func reloadData() {

    displayedAccessories = allAccessories()

    tableView.reloadData()

}

/// Sends the accessory to the next view.

func configureAccessory(_ accessory: HMAccessory) {

    if displayedAccessories.contains(.homeKit(accessory: accessory)) {

```



```

        performSegue(withIdentifier: Identifiers.addAccessorySegue, sender: accessory)

    }

}

/**

Finds an unconfigured accessory with a specified name.

- parameter name: The name string of the accessory.

- returns: An `HMAccessory?` from the search; `nil` if

        the accessory could not be found.

*/

func unconfiguredHomeKitAccessoryWithName(_ name: String) -> HMAccessory? {

    for type in displayedAccessories {

        if case let .homeKit(accessory) = type , accessory.name == name {

            return accessory

        }

    }

}

```

```

    }

    return nil

}

// MARK: ModifyAccessoryDelegate Methods

/// Adds the accessory to the internal array and reloads the views.

func      accessoryViewController(_      accessoryViewController:
ModifyAccessoryViewController, didSaveAccessory accessory: HMAccessory) {

    addedAccessories.append(accessory)

    reloadTable()

}

// MARK: EAWiFiUnconfiguredAccessoryBrowserDelegate Methods

// Any updates to the external accessory browser causes a reload in the table view.

```

```

func accessoryBrowser(_ browser: EAWiFiUnconfiguredAccessoryBrowser,
didFindUnconfiguredAccessories accessories: Set<EAWiFiUnconfiguredAccessory>) {

    reloadTable()

}

```

```

func accessoryBrowser(_ browser: EAWiFiUnconfiguredAccessoryBrowser,
didRemoveUnconfiguredAccessories accessories: Set<EAWiFiUnconfiguredAccessory>)
{

    reloadTable()

}

```

```

func accessoryBrowser(_ browser: EAWiFiUnconfiguredAccessoryBrowser,
didUpdate state: EAWiFiUnconfiguredAccessoryBrowserState) {

    reloadTable()

}

```

/// If the configuration was successful, presents the 'Add Accessory' view.

```

func accessoryBrowser(_ browser: EAWiFiUnconfiguredAccessoryBrowser,
didFinishConfiguringAccessory accessory: EAWiFiUnconfiguredAccessory, with status:
EAWiFiUnconfiguredAccessoryConfigurationStatus) {

```

```

        if status != .success {

            return

        }

        if let foundAccessory = unconfiguredHomeKitAccessoryWithName(accessory.name)
        {

            configureAccessory(foundAccessory)

        }

    }

    // MARK: HMAccessoryBrowserDelegate Methods

    /**

        Inserts the accessory into the internal array and inserts the

        row into the table view.

    */

    func accessoryBrowser(_ browser: HMAccessoryBrowser, didFindNewAccessory
    accessory: HMAccessory) {

```

```

let newAccessory = AccessoryType.homeKit(accessory: accessory)

if displayedAccessories.contains(newAccessory) {

    return

}

displayedAccessories.append(newAccessory)

displayedAccessories = displayedAccessories.sortByLocalizedName()


if let newIndex = displayedAccessories.firstIndex(of: newAccessory) {

    let newIndexPath = IndexPath(row: newIndex, section: 0)

    tableView.insertRows(at: [newIndexPath], with: .automatic)

}

}

/**

Removes the accessory from the internal array and deletes the

row from the table view.

*/

```

```

func accessoryBrowser(_ browser: HMAccessoryBrowser, didRemoveNewAccessory
accessory: HMAccessory) {

    let removedAccessory = AccessoryType.homeKit(accessory: accessory)

    if !displayedAccessories.contains(removedAccessory) {

        return

    }

    if let removedIndex = displayedAccessories.firstIndex(of: removedAccessory) {

        let removedIndexPath = IndexPath(row: removedIndex, section: 0)

        displayedAccessories.remove(at: removedIndex)

        tableView.deleteRows(at: [removedIndexPath], with: .automatic)

    }

}

}

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

The detailed view of a single accessory.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
class AccessoryDetail: UITableViewController {
```

```
    @IBOutlet weak var nameLabel: UILabel!
```

```
    @IBOutlet weak var roomLabel: UILabel!
```

```
    @IBOutlet weak var modelLabel: UILabel!
```

```
    @IBOutlet weak var firmwareLabel: UILabel!
```

```
    /// The service that this detail view shows. The UI refers to `HMService`
```

```
    /// instances as 'accessories' to be consistent with Home app terminology.
```

```
    var service: HMService?
```

```
    /// The home of which the service is a part, used for room picking and accessory removal.
```

```
var home: HMHome?
```

```
/// Resets the view data based on the current service.
```

```
func reloadData() {
```

```
    title = service?.name ?? "Accessory Detail"
```

```
    nameLabel.text = service?.name
```

```
    roomLabel.text = service?.accessory?.room?.name
```

```
    modelLabel.text = service?.accessory?.model
```

```
    firmwareLabel.text = service?.accessory?.firmwareVersion
```

```
}
```

```
/// Registers this view controller to receive various delegate callbacks.
```

```
override func viewDidLoad() {
```

```
    super.viewDidLoad()
```

```
    HomeStore.shared.addHomeDelegate(self)
```

```
    HomeStore.shared.addAccessoryDelegate(self)
```



```

        reloadData()

    }

    /// Deregisters this view controller as various kinds of delegate.

    deinit {

        HomeStore.shared.removeHomeDelegate(self)

        HomeStore.shared.removeAccessoryDelegate(self)

    }

    /// Prepares to show one of the child views of this view.

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

        if let controller = segue.destination as? NameEditor {

            controller.service = service

        } else if let controller = segue.destination as? RoomPicker {

            controller.service = service

```

```

        controller.home = home

    } else if let controller = segue.destination as? AccessorySettings {

        controller.service = service

    }

}

/// Handles table view cell taps.

override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {

    if indexPath == IndexPath(row: 0, section: 2) {

        let alert = UIAlertController(title: "Remove Accessory",

            message: "Are you sure you want to remove this accessory?",

            preferredStyle: .alert)

        alert.addAction(UIAlertAction(title: "Remove", style: .destructive) { _ in

            if let accessory = self.service?.accessory,

                let home = self.home {

                HomeStore.shared.remove(accessory, from: home)

```

```

    }

    })

    alert.addAction(UIAlertAction(title: "Cancel", style: .cancel) { _ in

        self.tableView.deselectRow(at: indexPath, animated: true)

    })

    present(alert, animated: true)

}

}

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

The accessory delegate for the accessory detail view.

```
*/
```

```
import HomeKit
```

```
/// Handle the accessory delegate callbacks.
```

```
extension AccessoryDetail: HMAccessoryDelegate {
```

```
    func accessory(_ accessory: HMAccessory, didUpdateNameFor service: HMService) {
```

```
        reloadData()
```

```
    }
```

```
}
```

```
/*
```

See LICENSE folder for this sample's licensing information.

Abstract:

The accessory delegate for the accessory detail view.

```
*/
```

```
import HomeKit
```

```
/// Handle the accessory delegate callbacks.
```

```

extension AccessoryDetail: HMAccessoryDelegate {

    func accessory(_ accessory: HMAccessory, didUpdateNameFor service: HMService) {

        reloadData()

    }

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

The home delegate for the accessory detail view.

```

*/

```

```

import HomeKit

```

```

/// Handle the home delegate callbacks.

```

```

extension AccessoryDetail: HMHomeDelegate {

    func home(_ home: HMHome, didUpdate room: HMRoom, for accessory:
HMAccessory) {

```

```

        reloadData()

    }

    func home(_ home: HMHome, didUpdateNameFor room: HMRoom) {

        reloadData()

    }

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

The home delegate for the accessory detail view.

```
*/
```

```
import HomeKit
```

```
/// Handle the home delegate callbacks.
```

```

extension AccessoryDetail: HMHomeDelegate {

    func home(_ home: HMHome, didUpdate room: HMRoom, for accessory:
HMAccessory) {

        reloadData()

    }

    func home(_ home: HMHome, didUpdateNameFor room: HMRoom) {

        reloadData()

    }

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

The detailed view of a single accessory.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
class AccessoryDetail: UITableViewController {
```

```
    @IBOutlet weak var nameLabel: UILabel!
```

```
    @IBOutlet weak var roomLabel: UILabel!
```

```
    @IBOutlet weak var modelLabel: UILabel!
```

```
    @IBOutlet weak var firmwareLabel: UILabel!
```

```
    /// The service that this detail view shows. The UI refers to `HMService`
```

```
    /// instances as 'accessories' to be consistent with Home app terminology.
```

```
    var service: HMService?
```

```
    /// The home of which the service is a part, used for room picking and accessory removal.
```

```
    var home: HMHome?
```

```
    /// Resets the view data based on the current service.
```



```

func reloadData() {

    title = service?.name ?? "Accessory Detail"

    nameLabel.text = service?.name

    roomLabel.text = service?.accessory?.room?.name

    modelLabel.text = service?.accessory?.model

    firmwareLabel.text = service?.accessory?.firmwareVersion

}

/// Registers this view controller to receive various delegate callbacks.

override func viewDidLoad() {

    super.viewDidLoad()

    HomeStore.shared.addHomeDelegate(self)

    HomeStore.shared.addAccessoryDelegate(self)

    reloadData()

}

```

```
/// Deregisters this view controller as various kinds of delegate.
```

```
deinit {
```

```
    HomeStore.shared.removeHomeDelegate(self)
```

```
    HomeStore.shared.removeAccessoryDelegate(self)
```

```
}
```

```
/// Prepares to show one of the child views of this view.
```

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
```

```
    if let controller = segue.destination as? NameEditor {
```

```
        controller.service = service
```

```
    } else if let controller = segue.destination as? RoomPicker {
```

```
        controller.service = service
```

```
        controller.home = home
```

```
    } else if let controller = segue.destination as? AccessorySettings {
```

```

        controller.service = service

    }

}

/// Handles table view cell taps.

override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {

    if indexPath == IndexPath(row: 0, section: 2) {

        let alert = UIAlertController(title: "Remove Accessory",

                                     message: "Are you sure you want to remove this accessory?",

                                     preferredStyle: .alert)

        alert.addAction(UIAlertAction(title: "Remove", style: .destructive) { _ in

            if let accessory = self.service?.accessory,

               let home = self.home {

                HomeStore.shared.remove(accessory, from: home)

            }

        })

        alert.addAction(UIAlertAction(title: "Cancel", style: .cancel) { _ in

```

```

        self.tableView.deselectRow(at: indexPath, animated: true)

    })

    present(alert, animated: true)

}

}

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

The accessory delegate for the accessory list.

```
*/
```

```
import HomeKit
```

```
extension AccessoryList: HMAccessoryDelegate {
```

```
    /// Handles characteristic value updates.
```

```

func accessory(_ accessory: HMAccessory, service: HMService, didUpdateValueFor
characteristic: HMCharacteristic) {

    if let item = kilgoServices.firstIndex(of: service) {

        let cell = collectionView.cellForItem(at: IndexPath(item: item, section: 0)) as?
DeviceCell

        cell?.redrawState()

    }

}

```

```

func accessory(_ accessory: HMAccessory, didUpdateNameFor service: HMService) {

    if let item = kilgoServices.firstIndex(of: service) {

        let cell = collectionView.cellForItem(at: IndexPath(item: item, section: 0)) as?
DeviceCell

        cell?.nameLabel.text = service.name

    }

}

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

The home delegate for the accessory list.

*/

import HomeKit

/// Handle the home delegate callbacks.

extension AccessoryList: HMHomeDelegate {

func homeDidUpdateName(_ home: HMHome) {

guard home == self.home else { return }

title = home.name

}

func home(_ home: HMHome, didAdd accessory: HMAccessory) {

```

guard home == self.home else { return }

// Make sure the new accessory generates callbacks to the home store.

accessory.delegate = HomeStore.shared

reloadData()

}

func home(_ home: HMHome, didUpdate room: HMRoom, for accessory:
HMAccessory) {

    for service in accessory.services {

        if let item = kilgoServices.firstIndex(of: service) {

            let cell = collectionView.cellForItem(at: IndexPath(item: item, section: 0)) as?
DeviceCell

            cell?.roomLabel.text = room.name

        }

    }

}

```

```

func home(_ home: HMHome, didRemove accessory: HMAccessory) {

    guard home == self.home else { return }

    navigationController?.popToRootViewController(animated: true)

    reloadData()

}

func home(_ home: HMHome, didUpdateNameFor room: HMRoom) {

    for cell in collectionView.visibleCells {

        (cell as? DeviceCell)?.roomLabel.text = room.name

    }

}

func home(_ home: HMHome, didEncounterError error: Error, for accessory:
HMAccessory) {

    print(error.localizedDescription)

}

}

```



```
/*
```

See LICENSE folder for this sample's licensing information.

Abstract:

The home manager delegate for the accessory list.

```
*/
```

```
import HomeKit
```

```
/// Handle the home manager delegate callbacks.
```

```
extension AccessoryList: HMHomeManagerDelegate {
```

```
    func homeManagerDidUpdateHomes(_ manager: HMHomeManager) {
```

```
        setOrAddHome(manager: manager)
```

```
    }
```

```
    func homeManagerDidUpdatePrimaryHome(_ manager: HMHomeManager) {
```

```
        setOrAddHome(manager: manager)
```

```
}
```

/// Sets the home to either the primary home, or the first home, or a home that the user creates.

```
func setOrAddHome(manager: HMHomeManager) {

    if manager.primaryHome != nil {

        home = manager.primaryHome

    } else if let firstHome = manager.homes.first {

        home = firstHome

    } else {

        let alert = UIAlertController(title: "Add a Home",

                                     message: "There aren't any homes in the database. Create a
home to work with.",

                                     preferredStyle: .alert)

        alert.addTextField { $0.placeholder = "Name" }

        alert.addAction(UIAlertAction(title: "Create", style: .default) { _ in

            if let name = alert.textFields?[0].text {

                manager.addHome(withName: name) { home, error in
```

```

        if let error = error {

            print("Error adding home: \(error)")

        } else {

            self.home = home

        }

    }

}

})

present(alert, animated: true)

}

}

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

A collection of known Kilgo accessories in the home.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
/// - Tag: AccessoryList
```

```
class AccessoryList: UICollectionViewController {
```

```
    /// The filtered list of services that the app displays.
```

```
    var kilgoServices = [HMService]() // These are called "accessories" in the UI.
```

```
    /// The home whose accessories the app displays.
```

```
    var home: HMHome? {
```

```
        didSet {
```

```
home?.delegate = HomeStore.shared
```

```
reloadData()
```

```
}
```

```
}
```

```
/// Resets the list of Kilgo services from the currently set home.
```

```
func reloadData() {
```

```
    kilgoServices = []
```

```
    guard let home = home else { return }
```

```
    for accessory in home.accessories.filter({ $0.manufacturer == "Kilgo Devices, Inc." }) {
```

```
        accessory.delegate = HomeStore.shared
```

```
        for service in accessory.services.filter({ $0.isUserInteractive }) {
```

```
            kilgoServices.append(service)
```

```

        // Ask for notifications from any characteristics that support them.

        for characteristic in service.characteristics.filter({

            $0.properties.contains(HMCharacteristicPropertySupportsEventNotification)

        }) {

            characteristic.enableNotification(true) { _ in }

        }

    }

}

collectionView.reloadData()

}

/// Registers this view controller to receive various delegate callbacks.

override func viewDidLoad() {

    super.viewDidLoad()

    if #available(iOS 13.0, *) {

```

```

        let navBarAppearance = UINavigationBarAppearance()

        navBarAppearance.configureWithOpaqueBackground()

        navBarAppearance.titleTextAttributes = [.foregroundColor: UIColor.white]

        navBarAppearance.largeTitleTextAttributes = [.foregroundColor: UIColor.black]

        navBarAppearance.backgroundColor = UIColor.init(hexString: "#a1edab")

        navigationController?.navigationBar.standardAppearance = navBarAppearance

        navigationController?.navigationBar.scrollEdgeAppearance = navBarAppearance

    }

    //navigationController?.navigationBar.isTranslucent = false

    HomeStore.shared.homeManager.delegate = self

    HomeStore.shared.addHomeDelegate(self)

    HomeStore.shared.addAccessoryDelegate(self)

    navigationController?.navigationBar.prefersLargeTitles = true

}

```

```
/// Deregisters this view controller as various kinds of delegate.
```

```
deinit {
```

```
    HomeStore.shared.homeManager.delegate = nil
```

```
    HomeStore.shared.removeHomeDelegate(self)
```

```
    HomeStore.shared.removeAccessoryDelegate(self)
```

```
}
```

```
/// Prepares to show the detail view for an accessory.
```

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
```

```
    if segue.identifier == "showDetail",
```

```
        let accessoryDetail = segue.destination as? AccessoryDetail,
```

```
        let button = sender as? UIButton {
```

```
            accessoryDetail.service = kilgoServices[button.tag]
```

```
            accessoryDetail.home = home
```

```
        }
```



```
}
```

```
/// Starts the add accessory flow.
```

```
@IBAction func tapAdd(_ sender: UIBarButtonItem) {
```

```
    home?.addAndSetupAccessories(completionHandler: { error in
```

```
        if let error = error {
```

```
            print(error)
```

```
        } else {
```

```
            // Make no assumption about changes; just reload everything.
```

```
            self.reloadData()
```

```
        }
```

```
    })
```

```
}
```

```
// MARK: UICollectionViewDataSource
```

```
/// Tells the collection view to create one cell for each service.
```

```

        override func collectionView(_ collectionView: UICollectionView,
numberOfItemsInSection section: Int) -> Int {

            return kilgoServices.count

        }

    /// Supplies the collection view with configured cells.

    override func collectionView(_ collectionView: UICollectionView, cellForItemAt
indexPath: IndexPath) -> UICollectionViewCell {

        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "DeviceCell",
for: indexPath)

        cell.layer.cornerRadius = 10

        if let deviceCell = cell as? DeviceCell {

            deviceCell.service = kilgoServices[indexPath.item]

            deviceCell.disclosureButton.tag = indexPath.item

        }

        return cell

    }

```

```

    /// Passes taps along to the cell.

    override func collectionView(_ collectionView: UICollectionView, didSelectItemAt
indexPath: IndexPath) {

        (collectionView.cellForItem(at: indexPath) as? DeviceCell)?.tap()

    }

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

A table of characteristics for a particular service.

```

*/

```

```

import UIKit

```

```

import HomeKit

```

```

class AccessorySettings: UITableViewController {

```

```

/// The list of characteristics to display.

var characteristics = [HMCharacteristic]()


/// The service that owns the characteristics to display.

var service: HMService? {

    didSet {

        // Capture a curated list of characteristics.

        characteristics = service?.displayableCharacteristics ?? []

    }

}


/// Registers this instance to receive accessory delegate callbacks.

override func viewDidLoad() {

    super.viewDidLoad()


    HomeStore.shared.addAccessoryDelegate(self)

```

```
}
```

```
/// Deregisters this instance as an accessory delegate.
```

```
deinit {
```

```
    HomeStore.shared.removeAccessoryDelegate(self)
```

```
}
```

```
// MARK: - Table view data source
```

```
/// Tells the table view to create one cell for each characteristic.
```

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section:  
Int) -> Int {
```

```
    return characteristics.count
```

```
}
```

```
/// Supplies the table view with configured cells.
```

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath:  
IndexPath) -> UITableViewCell {
```

```
        let cell = tableView.dequeueReusableCell(withIdentifier: "CharacteristicCell", for:
indexPath)
```

```
        if let characteristicCell = cell as? CharacteristicCell {
```

```
            characteristicCell.characteristic = characteristics[indexPath.row]
```

```
        }
```

```
        return cell
```

```
    }
```

```
}
```

```
/*
```

See LICENSE folder for this sample's licensing information.

Abstract:

The accessory delegate for the service detail view controller.

```
*/
```

```
import HomeKit
```

```

extension AccessorySettings: HMAccessoryDelegate {

    /// Handles characteristic updates.

    func accessory(_ accessory: HMAccessory, service: HMService, didUpdateValueFor
characteristic: HMCharacteristic) {

        guard service == self.service else { return }

        // Find the cell that displays this characteristic.

        guard let row = characteristics.firstIndex(of: characteristic),

            let cell = tableView.cellForRow(at: IndexPath(row: row, section: 0)) as?
CharacteristicCell

            else { return }

        // Tell the cell to refresh using the characteristic it already knows about.

        cell.redrawValueLabel()

        cell.redrawControls(animated: true)

    }

}

```

```
/*
```

See LICENSE folder for this sample's licensing information.

Abstract:

The accessory delegate for the service detail view controller.

```
*/
```

```
import HomeKit
```

```
extension AccessorySettings: HMAccessoryDelegate {
```

```
    /// Handles characteristic updates.
```

```
    func accessory(_ accessory: HMAccessory, service: HMService, didUpdateValueFor  
characteristic: HMCharacteristic) {
```

```
        guard service == self.service else { return }
```

```
        // Find the cell that displays this characteristic.
```

```
        guard let row = characteristics.firstIndex(of: characteristic),
```



```
        let cell = tableView.cellForRow(at: IndexPath(row: row, section: 0)) as?
        CharacterCell
```

```
        else { return }
```

```
        // Tell the cell to refresh using the characteristic it already knows about.
```

```
        cell.redrawValueLabel()
```

```
        cell.redrawControls(animated: true)
```

```
    }
```

```
}
```

```
/*
```

See LICENSE folder for this sample's licensing information.

Abstract:

A table of characteristics for a particular service.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```

class AccessorySettings: UITableViewController {

    /// The list of characteristics to display.

    var characteristics = [HMCharacteristic]()

    /// The service that owns the characteristics to display.

    var service: HMService? {

        didSet {

            // Capture a curated list of characteristics.

            characteristics = service?.displayableCharacteristics ?? []

        }

    }

    /// Registers this instance to receive accessory delegate callbacks.

    override func viewDidLoad() {

        super.viewDidLoad()

```

```

        HomeStore.shared.addAccessoryDelegate(self)

    }

    /// Deregisters this instance as an accessory delegate.

    deinit {

        HomeStore.shared.removeAccessoryDelegate(self)

    }

    // MARK: - Table view data source

    /// Tells the table view to create one cell for each characteristic.

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {

        return characteristics.count

    }

    /// Supplies the table view with configured cells.

```

```

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

        let cell = tableView.dequeueReusableCell(withIdentifier: "CharacterCell", for:
indexPath)

        if let characterCell = cell as? CharacterCell {

            characterCell.characteristic = characteristics[indexPath.row]

        }

        return cell

    }

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `AccessoryUpdateController` manages `CharacteristicCell` updates and buffers them up before sending them to HomeKit.

```
*/
```

```
import HomeKit
```

```
/// An object that responds to `CharacteristicCell` updates and notifies HomeKit of changes.
```

```
class AccessoryUpdateController: NSObject, CharacteristicCellDelegate {
```

```
    // MARK: Properties
```

```
    let updateQueue = DispatchQueue(label: "com.sample.HMCatalog.CharacteristicUpdateQueue", attributes: [])
```

```
    lazy var pendingWrites = [HMCharacteristic:CellValueType]()
```

```
    lazy var sentWrites = [HMCharacteristic:CellValueType]()
```

```
    // Implicitly unwrapped optional because we need `self` to initialize.
```

```
    var updateValueTimer: Timer!
```

```
/// Starts the update timer on creation.
```

```
override init() {
```

```
    super.init()
```

```
    startListeningForCellUpdates()
```

```
}
```

```
/// Responds to a cell change, and if the update was marked immediate, updates the characteristics.
```

```
func characteristicCell(_ cell: CharacteristicCell, didUpdateValue value: CellValueType, forCharacteristic characteristic: HMCharacteristic, immediate: Bool) {
```

```
    pendingWrites[characteristic] = value
```

```
    if immediate {
```

```
        updateCharacteristics()
```

```
    }
```

```
}
```

```
/**
```

```
    Reads the characteristic's value and calls the completion with the characteristic's value.
```

If there is a pending write request on the same characteristic, the read is ignored to prevent

"UI glitching".

*/

```
func characteristicCell(_ cell: CharacteristicCell, readInitialValueForCharacteristic
characteristic: HMCharacteristic, completion: @escaping (CellValueType?, Error?) ->
Void) {
```

```
    characteristic.readValue { error in
```

```
        self.updateQueue.sync {
```

```
            if let sentValue = self.sentWrites[characteristic] {
```

```
                completion(sentValue, nil)
```

```
                return
```

```
            }
```

```
        DispatchQueue.main.async {
```

```
            completion(characteristic.value as? CellValueType, error)
```

```
        }
```

```
    }
```

```

    }

}

/// Creates and starts the update value timer.

func startListeningForCellUpdates() {

    updateValueTimer = Timer.scheduledTimer(timeInterval: 0.1, target: self, selector:
#selector(self.updateCharacteristics), userInfo: nil, repeats: true)

}

/// Invalidates the update timer.

func stopListeningForCellUpdates() {

    updateValueTimer.invalidate()

}

/// Sends all pending requests in the array.

@objc func updateCharacteristics() {

    updateQueue.sync {

        for (characteristic, value) in self.pendingWrites {

```



```
self.sentWrites[characteristic] = value
```

```
characteristic.writeValue(value) { error in
```

```
    if let error = error {
```

```
        print("HomeKit: Could not change value: \(error.localizedDescription).")
```

```
    }
```

```
    self.didCompleteWrite(characteristic, value: value)
```

```
    }
```

```
}
```

```
self.pendingWrites.removeAll()
```

```
}
```

```
}
```

```
/**
```

Synchronously adds the characteristic-value pair into the `sentWrites` map.

- parameter characteristic: The `HMCharacteristic` to add.

- parameter value: The value of the `characteristic`.

*/

```
func didSendWrite(_ characteristic: HMCharacteristic, value: CellValueType) {
```

```
    updateQueue.sync {
```

```
        self.sentWrites[characteristic] = value
```

```
    }
```

```
}
```

/**

Synchronously removes the characteristic-value pair from the `sentWrites` map.

- parameter characteristic: The `HMCharacteristic` to remove.

- parameter value: The value of the `characteristic` (unused, but included for clarity).

*/

```
func didCompleteWrite(_ characteristic: HMCharacteristic, value: AnyObject) {
```

```

        updateQueue.sync {

            _ = self.sentWrites.removeValue(forKey: characteristic)

        }

    }

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `ActionCell` displays a characteristic and a target value.

```

*/

```

```

import UIKit

```

```

import HomeKit

```

```

/// A `UITableViewCell` subclass that displays a characteristic's 'target value'.

```

```

class ActionCell: UITableViewCell {

    /// Ignores the passed-in style and overrides it with `.Subtitle`.

    override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {

        super.init(style: .subtitle, reuseIdentifier: reuseIdentifier)

        selectionStyle = .none

        detailTextLabel?.textColor = UIColor.lightGray

        accessoryType = .none

    }

```

/// Required init.

```

required init?(coder aDecoder: NSCoder) {

    super.init(coder: aDecoder)

}

```

/**

Sets the cell's text to represent a characteristic and target value.

For example, "Brightness → 60%"

Sets the subtitle to the service and accessory that this characteristic represents.

- parameter characteristic: The characteristic this cell represents.

- parameter targetValue: The target value from this action.

*/

```
func setCharacteristic(_ characteristic: HMCharacteristic, targetValue: AnyObject) {

    let    targetDescription    =    "\(characteristic.localizedDescription)    →
\characteristic.localizedDescriptionForValue(targetValue))"

    textLabel?.text = targetDescription

    let contextDescription = NSLocalizedString("%@ in %@", comment: "Service in
Accessory")

    if let service = characteristic.service, let accessory = service.accessory {

        detailTextLabel?.text    =    String(format:    contextDescription,    service.name,
accessory.name)

    }

    else {

        detailTextLabel?.text = NSLocalizedString("Unknown Characteristic", comment:
"Unknown Characteristic")
    }
}
```

```
    }  
  
    }  
  
    }  
  
    /*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `ActionSetCreator` builds `HMActionSet`s.

```
*/
```

```
import HomeKit
```

```
/// A `CharacteristicCellDelegate` that builds an `HMActionSet` when it receives delegate  
callbacks.
```

```
class ActionSetCreator: CharacteristicCellDelegate {
```

```
    // MARK: Properties
```

```
var actionSet: HMActionSet?
```

```
var home: HMHome
```

```
var saveError: Error?
```

```
/// The structure we're going to use to hold the target values.
```

```
let          targetValueMap          =          NSMapTable<HMCharacteristic,  
CellValueType>.strongToStrongObjects()
```

```
/// A dispatch group to wait for all of the individual components of the saving process.
```

```
let saveActionSetGroup = DispatchGroup()
```

```
required init(actionSet: HMActionSet?, home: HMHome) {
```

```
    self.actionSet = actionSet
```

```
    self.home = home
```

```
}
```

```
/**
```

If there is an action set, saves the action set and then updates its name.

Otherwise creates a new action set and adds all actions to it.

- parameter name: The new name for the action set.

- parameter completionHandler: A closure to call once the action set has been completely saved.

```
*/
```

```
func saveActionSetWithName(_ name: String, completionHandler: @escaping (_ error: Error?) -> Void) {
```

```
    if let actionSet = actionSet {
```

```
        saveActionSet(actionSet)
```

```
        updateNameIfNecessary(name)
```

```
    }
```

```
    else {
```

```
        createActionSetWithName(name)
```

```
    }
```

```
    saveActionSetGroup.notify(queue: DispatchQueue.main) {
```



```

        completionHandler(self.saveError)

        self.saveError = nil

    }

}

```

```

/**

```

Adds all of the actions that have been requested to the Action Set, then runs a completion block.

- parameter completion: A closure to be called when all of the actions have been added.

```

*/

```

```

func saveActionSet(_ actionSet: HMActionSet) {

    let actions = actionsFromMapTable(targetValueMap)

    for action in actions {

        saveActionSetGroup.enter()

        addAction(action, toActionSet: actionSet) { error in

            if let error = error {

                print("HomeKit: Error adding action: \(error.localizedDescription)")
            }
        }
    }
}

```

```

        self.saveError = error

    }

    self.saveActionSetGroup.leave()

}

}

}

```

```
/**
```

Sets the name of an existing action set.

- parameter name: The new name for the action set.

```
*/
```

```

func updateNameIfNecessary(_ name: String) {

    if actionSet?.name == name {

        return

    }

    saveActionSetGroup.enter()

```

```

        actionSet?.updateName(name) { error in

            if let error = error {

                print("HomeKit: Error updating name: \(error.localizedDescription)")

                self.saveError = error

            }

            self.saveActionSetGroup.leave()

        }

    }

```

```

/**

```

Creates and saves an action set with the provided name.

- parameter name: The name for the new action set.

```

*/

```

```

func createActionSetWithName(_ name: String) {

    saveActionSetGroup.enter()

    home.addActionSet(withName: name) { actionSet, error in

```

```

    if let error = error {

        print("HomeKit: Error creating action set: \(error.localizedDescription)")

        self.saveError = error

    }

    else {

        // There is no error, so the action set has a value.

        self.saveActionSet(actionSet!)

    }

    self.saveActionSetGroup.leave()

}

}

```

/**

Checks to see if an action already exists to modify the same characteristic as the action passed in. If such an action exists, the method tells the existing action to update its target value. Otherwise, the new action is simply added to the action set.

- parameter action: The action to add or update.
- parameter actionSet: The action set to which to add the action.
- parameter completion: A closure to call when the addition has finished.

*/

```
func addAction(_ action: HMCharacteristicWriteAction<NSCopying>, toActionSet
actionSet: HMActionSet, completion: @escaping (Error?) -> Void) {
```

```
    if let existingAction = existingActionInActionSetMatchingAction(action) {
```

```
        existingAction.updateTargetValue(action.targetValue, completionHandler:
completion)
```

```
    }
```

```
    else {
```

```
        actionSet.addAction(action, completionHandler: completion)
```

```
    }
```

```
}
```

/**

Checks to see if there is already an HMCharacteristicWriteAction in

the action set that matches the provided action.

- parameter action: The action in question.

- returns: The existing action that matches the characteristic or nil if

there is no existing action.

*/

```
func existingActionInActionSetMatchingAction(_ action:
HMCharacteristicWriteAction<CellValueType>) ->
HMCharacteristicWriteAction<CellValueType>? {

    if let actionSet = actionSet {

        for case let existingAction as HMCharacteristicWriteAction<CellValueType> in
            actionSet.actions {

                if action.characteristic == existingAction.characteristic {

                    return existingAction

                }

            }

        }

    }

    return nil
```

```
}
```

```
/**
```

Iterates over a map table of HMCharacteristic -> id objects and creates

an array of HMCharacteristicWriteActions based on those targets.

- parameter table: An NSMapTable mapping HMCharacteristics to id's.

- returns: An array of HMCharacteristicWriteActions.

```
*/
```

```
func actionsFromMapTable(_ table: NSMapTable<HMCharacteristic, CellValueType>)  
-> [HMCharacteristicWriteAction<CellValueType>] {  
  
    return targetValueMap.keyEnumerator().allObjects.map { key in  
  
        let characteristic = key as! HMCharacteristic  
  
        let targetValue = targetValueMap.object(forKey: characteristic)!  
  
        return HMCharacteristicWriteAction(characteristic: characteristic, targetValue:  
targetValue)  
  
    }
```

```
}
```

```
/**
```

```
- returns: `true` if the characteristic count is greater than zero;
```

```
  `false` otherwise.
```

```
*/
```

```
var containsActions: Bool {
```

```
  return !allCharacteristics.isEmpty
```

```
}
```

```
/**
```

```
All existing characteristics within `HMCharacteristicWriteActions`
```

```
and target values in the target value map.
```

```
*/
```

```
var allCharacteristics: [HMCharacteristic] {
```

```
  var characteristics = Set<HMCharacteristic>()
```



```

if let actions = actionSet?.actions {

    let actionSetCharacteristics = actions.compactMap { action in

        return (action as?
HMCharacteristicWriteAction<CellValueType>)?.characteristic

    }

    characteristics.formUnion(actionSetCharacteristics)

}

characteristics.formUnion(targetValueMap.keyEnumerator().allObjects as!
[HMCharacteristic])

return Array(characteristics)

}

```

/**

Searches through the target value map and existing 'HMCharacteristicWriteActions'
to find the target value for the characteristic in question.

- parameter characteristic: The characteristic in question.

- returns: The target value for this characteristic, or nil if there is no target.

*/

```
func    targetValueForCharacteristic(_    characteristic:    HMCharacteristic)    ->
CellValueType? {

    if let value = targetValueMap.object(forKey: characteristic) {

        return value

    }

    else if let actions = actionSet?.actions {

        for case let writeAction as HMCharacteristicWriteAction<CellValueType> in
actions {

            if writeAction.characteristic == characteristic {

                return writeAction.targetValue

            }

        }

    }

}
```

```
return nil
```

```
}
```

```
/**
```

First removes the characteristic from the `targetValueMap`.

Then removes any `HMCharacteristicWriteAction`'s from the action set

which set the specified characteristic.

- parameter characteristic: The `HMCharacteristic` to remove.

- parameter completion: The closure to invoke when the characteristic has been removed.

```
*/
```

```
func    removeTargetValueForCharacteristic(_    characteristic:    HMCharacteristic,  
completion: @escaping () -> Void) {
```

```
/*
```

We need to create a dispatch group here, because in many cases

there will be one characteristic saved in the Action Set, and one

in the target value map. We want to run the completion closure only one time,

to ensure we've removed both.

```
*/
```

```
let group = DispatchGroup()
```

```
if targetValueMap.object(forKey: characteristic) != nil {
```

```
    // Remove the characteristic from the target value map.
```

```
    DispatchQueue.main.async(group: group) {
```

```
        self.targetValueMap.removeObject(forKey: characteristic)
```

```
    }
```

```
}
```

```
if let actions = actionSet?.actions {
```

```
    for case let action as HMCharacteristicWriteAction<CellValueType> in actions {
```

```
        if action.characteristic == characteristic {
```

```
            /*
```

```
                Also remove the action, and only relinquish the dispatch group
```

```
                once the action set has finished.
```

```
            */
```

```
            group.enter()
```

```
        actionSet?.removeAction(action) { error in
```

```
            if let error = error {
```

```
                print(error.localizedDescription)
```

```
            }
```

```
        group.leave()
```

```
    }
```

```
}
```

```
}
```

```
}
```

```
    // Once we're positive both have finished, run the completion closure on the main
    queue.
```

```
    group.notify(queue: DispatchQueue.main, execute: completion)
```

```
}
```

```
// MARK: Characteristic Cell Delegate
```

```
/**
```

```
    Receives a callback from a `CharacteristicCell` with a value change.
```

Adds this value change into the targetValueMap, overwriting other value changes.

*/

```
func characteristicCell(_ cell: CharacteristicCell, didUpdateValue newValue:
CellValueType, forCharacteristic characteristic: HMCharacteristic, immediate: Bool) {

    targetValueMap.setObject(newValue, forKey: characteristic)

}
```

/**

Receives a callback from a `CharacteristicCell`, requesting an initial value for
a given characteristic.

It checks to see if we have an action in this Action Set that matches the characteristic.

If so, calls the completion closure with the target value.

*/

```
func characteristicCell(_ cell: CharacteristicCell, readInitialValueForCharacteristic
characteristic: HMCharacteristic, completion: @escaping (CellValueType?, Error?) ->
Void) {

    if let value = targetValueForCharacteristic(characteristic) {

        completion(value, nil)

    }

}
```

```

        return

    }

characteristic.readValue { error in

    /*

        The user may have updated the cell value while the

        read was happening. We check the map one more time.

    */

    if let value = self.targetValueForCharacteristic(characteristic) {

        completion(value, nil)

    }

    else {

        completion(characteristic.value as? CellValueType, error)

    }

}

}

}

```

/*

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `ActionSetViewController` allows users to create and modify action sets.

*/

import UIKit

import HomeKit

/// Represents table view sections of the `ActionSetViewController`.

enum ActionSetTableViewSection: Int {

case name, actions, accessories

}

/**

A view controller that facilitates creation of Action Sets.

It contains a cell for a name, and lists accessories within a home.

If there are actions within the action set, it also displays a list of ActionCells displaying those actions.

It owns an `ActionSetCreator` and routes events to the creator as appropriate.

*/

```
class ActionSetViewController: HMCatalogViewController {
```

```
    // MARK: Types
```

```
    struct Identifiers {
```

```
        static let accessoryCell = "AccessoryCell"
```

```
        static let unreachableAccessoryCell = "UnreachableAccessoryCell"
```

```
        static let actionCell = "ActionCell"
```

```
        static let showServiceSegue = "Show Services"
```

```
    }
```

```
// MARK: Properties
```

```
@IBOutlet weak var nameField: UITextField!
```

```
@IBOutlet weak var saveButton: UIBarButtonItem!
```

```
var actionSet: HMActionSet?
```

```
var actionSetCreator: ActionSetCreator!
```

```
var displayedAccessories = [HMAccessory]()
```

```
// MARK: View Methods
```

```
/**
```

```
    Creates the action set creator, registers the appropriate reuse identifiers in the table,
```

```
    and sets the `nameField` if appropriate.
```

```
*/
```

```
override func viewDidLoad() {
```

```
    super.viewDidLoad()
```

```
actionSetCreator = ActionSetCreator(actionSet: actionSet, home: home)
```

```
displayedAccessories = home.sortedControlAccessories
```

```
tableView.register(UITableViewCell.self, forCellReuseIdentifier:
Identifiers.accessoryCell)
```

```
tableView.register(UITableViewCell.self, forCellReuseIdentifier:
Identifiers.unreachableAccessoryCell)
```

```
tableView.register(ActionCell.self, forCellReuseIdentifier: Identifiers.actionCell)
```

```
tableView.rowHeight = UITableView.automaticDimension
```

```
tableView.estimatedRowHeight = 44.0
```

```
if let actionSet = actionSet {
```

```
    nameField.text = actionSet.name
```

```
    nameFieldDidChange(nameField)
```

```
}
```

```

    if !home.isAdmin {

        nameField.isEnabled = false

    }

}

/// Reloads the data and view.

override func viewWillAppear(_ animated: Bool) {

    super.viewWillAppear(animated)

    tableView.reloadData()

    enableSaveButtonIfNecessary()

}

/// Dismisses the view controller if our data is invalid.

override func viewDidAppear(_ animated: Bool) {

    super.viewDidAppear(animated)

    if shouldPopViewController() {

        self.dismiss(animated: true, completion: nil)

    }

}

```

```
}  
  
}
```

```
/// Dismisses the keyboard when we dismiss.
```

```
override func viewWillDisappear(_ animated: Bool) {
```

```
    super.viewWillDisappear(animated)
```

```
    resignFirstResponder()
```

```
}
```

```
/// Passes our accessory into the `ServicesViewController`.
```

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
```

```
    super.prepare(for: segue, sender: sender)
```

```
    if segue.identifier == Identifiers.showServiceSegue {
```

```
        let servicesViewController = segue.intendedDestinationViewController as!  
ServicesViewController
```

```
        servicesViewController.onlyShowsControlServices = true
```

```
        servicesViewController.cellDelegate = actionSetCreator
```

```

        let index = tableView.indexPath(for: sender as! UITableViewCell)?.row

        servicesViewController.accessory = displayedAccessories[index]

        servicesViewController.cellDelegate = actionSetCreator

    }

}

// MARK: IBAction Methods

/// Dismisses the view controller.

@IBAction func dismiss() {

    self.dismiss(animated: true, completion: nil)

}

/// Saves the action set, adds it to the home, and dismisses the view.

@IBAction func saveAndDismiss() {

    saveButton.isEnabled = false

```

```

        actionSetCreator.saveActionSetWithName(trimmedName) { error in

            self.saveButton.isEnabled = true

            if let error = error {

                self.displayError(error)

            }

            else {

                self.dismiss()

            }

        }

    }

}

/// Prompts an update to the save button enabled state.

@IBAction func nameFieldDidChange(_ field: UITextField) {

    enableSaveButtonIfNecessary()

}

```

```
// MARK: Table View Methods
```

```
/// We do not allow the creation of action sets in a shared home.
```

```
override func numberOfSections(in tableView: UITableView) -> Int {
```

```
    return home.isAdmin ? 3 : 2
```

```
}
```

```
/**
```

- returns: In the Actions section: the number of actions this set will contain upon saving.

In the Accessories section: The number of accessories in the home.

```
*/
```

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
```

```
    switch ActionSetTableViewSection(rawValue: section) {
```

```
        case .name?:
```

```
            return super.tableView(tableView, numberOfRowsInSection: section)
```



```

    case .actions?:

        return max(actionSetCreator.allCharacteristics.count, 1)

    case .accessories?:

        return displayedAccessories.count

    case nil:

        fatalError("Unexpected `ActionSetTableViewSection` raw value.")

    }

}

/**

```

Required override to allow for a tableView with both static and dynamic content.

Basically, since the superclass's `indentationLevelForRowAtIndexPath` is only

expecting 1 row per section, just call the super class's implementation

for the first row.

```

*/

override func tableView(_ tableView: UITableView, indentationLevelForRowAt
indexPath: IndexPath) -> Int {

    return super.tableView(tableView, indentationLevelForRowAt: IndexPath(row: 0,
section: indexPath.section))

}

/// Removes the action associated with the index path.

override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {

    if editingStyle == .delete {

        let characteristic = actionSetCreator.allCharacteristics[indexPath.row]

        actionSetCreator.removeTargetValueForCharacteristic(characteristic) {

            if self.actionSetCreator.containsActions {

                tableView.deleteRows(at: [indexPath], with: .automatic)

            }

            else {

                tableView.reloadRows(at: [indexPath], with: .automatic)

            }

        }
    }
}

```

```

    }

    }

}

```

/// - returns: `true` for the Actions section; `false` otherwise.

```

override func tableView(_ tableView: UITableView, canEditRowAt indexPath:
IndexPath) -> Bool {

    return ActionSetTableViewSection(rawValue: indexPath.section) == .actions &&
home.isAdmin

}

```

/// - returns: `UITableViewAutomaticDimension` for dynamic sections, otherwise the superclass's implementation.

```

override func tableView(_ tableView: UITableView, heightForRowAt indexPath:
IndexPath) -> CGFloat {

    switch ActionSetTableViewSection(rawValue: indexPath.section) {

        case .name?:

            return super.tableView(tableView, heightForRowAt: indexPath)

    }
}

```

```
case .actions?, .accessories?:
```

```
    return UITableView.automaticDimension
```

```
case nil:
```

```
    fatalError("Unexpected `ActionSetTableViewSection` raw value.")
```

```
}
```

```
}
```

/// - returns: An action cell for the actions section, an accessory cell for the accessory section, or the superclass's implementation.

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
```

```
    switch ActionSetTableViewSection(rawValue: indexPath.section) {
```

```
    case .name?:
```

```
        return super.tableView(tableView, cellForRowAt indexPath)
```

```
    case .actions?:
```

```
        if actionSetCreator.containsActions {
```

```

        return self.tableView(tableView, actionCellForRowAtIndexPath: indexPath)

    }

    else {

        return super.tableView(tableView, cellForRowAt: indexPath)

    }

case .accessories?:

    return self.tableView(tableView, accessoryCellForRowAtIndexPath: indexPath)

case nil:

    fatalError("Unexpected `ActionSetTableViewSection` raw value.")

}

}

// MARK: Helper Methods

/// Enables the save button if there is a valid name and at least one action.

```

```

private func enableSaveButtonIfNecessary() {

    saveButton.isEnabled = home.isAdmin && trimmedName.count > 0 &&
actionSetCreator.containsActions

}

/// - returns: The contents of the nameField, with whitespace trimmed from the
beginning and end.

private var trimmedName: String {

    return nameField.text!.trimmingCharacters(in:
CharacterSet.whitespacesAndNewlines)

}

/**

- returns: `true` if there are no accessories in the home, we have no set action set,

or if our home no longer exists; `false` otherwise

*/

private func shouldPopViewController() -> Bool {

    if homeStore.home?.accessories.count == 0 && actionSet == nil {

        return true

```

```
}
```

```
return !homeStore.homeManager.homes.contains { $0 == homeStore.home }
```

```
}
```

/// - returns: An `ActionCell` instance with the target value for the characteristic at the specified index path.

```
private func tableView(_ tableView: UITableView, actionCellForRowAtIndexPath  
indexPath: IndexPath) -> UITableViewCell {
```

```
    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.actionCell, for:  
indexPath) as! ActionCell
```

```
    let characteristic = actionSetCreator.allCharacteristics[indexPath.row] as  
HMCharacteristic
```

```
    if let target = actionSetCreator.targetValueForCharacteristic(characteristic) {
```

```
        cell.setCharacteristic(characteristic, targetValue: target)
```

```
    }
```

```
    return cell
```

```
}
```

/// - returns: An Accessory cell that contains an accessory's name.

```
private func tableView(_ tableView: UITableView, accessoryCellForRowAtIndexPath  
indexPath: IndexPath) -> UITableViewCell {
```

```
/*
```

These cells are static, the identifiers are defined in the Storyboard,

but they're not recognized here. In viewDidLoad:, we're registering

```
`UITableViewCell` as the class for "AccessoryCell" and  
"UnreachableAccessoryCell".
```

We must configure these cells manually, the cells in the Storyboard

are just for reference.

```
*/
```

```
let accessory = displayedAccessories[indexPath.row]
```

```
let cellIdentifier = accessory.isReachable ? Identifiers.accessoryCell :  
Identifiers.unreachableAccessoryCell
```



```

        let cell = tableView.dequeueReusableCell(withIdentifier: cellIdentifier, for:
indexPath)

        cell.textLabel?.text = accessory.name

        if accessory.isReachable {

            cell.textLabel?.textColor = UIColor.darkText

            cell.accessoryType = .disclosureIndicator

            cell.selectionStyle = .default

        }

        else {

            cell.textLabel?.textColor = UIColor.lightGray

            cell.accessoryType = .none

            cell.selectionStyle = .none

        }

        return cell

    }

```

```
/// Shows the services in the selected accessory.
```

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
```

```
    let cell = tableView.cellForRow(at: indexPath)!
```

```
    if cell.selectionStyle == .none {
```

```
        return
```

```
    }
```

```
    if ActionSetTableViewSection(rawValue: indexPath.section) == .accessories {
```

```
        performSegue(withIdentifier: Identifiers.showServiceSegue, sender: cell)
```

```
    }
```

```
}
```

```
// MARK: HMHomeDelegate Methods
```

```
/**
```

```
    Pops the view controller if our configuration is invalid;
```

```
    reloads the view otherwise.
```

```
*/
```

```
func home(_ home: HMHome, didRemoveAccessory accessory: HMAccessory) {
```

```
    if shouldPopViewController() {
```

```
        self.dismiss(animated: true, completion: nil)
```

```
    }
```

```
    else {
```

```
        tableView.reloadData()
```

```
    }
```

```
}
```

```
/// Reloads the table view data.
```

```
func home(_ home: HMHome, didAddAccessory accessory: HMAccessory) {
```

```
    tableView.reloadData()
```

```
}
```

```
/// If our action set was removed, dismiss the view.
```

```
func home(_ home: HMHome, didRemoveActionSet actionSet: HMActionSet) {
```

```

    if actionSet == self.actionSet {

        self.dismiss(animated: true, completion: nil)

    }

}

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `AddRoomViewController` allows the user to add rooms to a zone.

```

*/

```

```

import UIKit

```

```

import HomeKit

```

```

/// A view controller that lists rooms within a home and allows the user to add the rooms
to a provided zone.

```

```

class AddRoomViewController: HMCatalogViewController {

    // MARK: Types

    struct Identifiers {

        static let roomCell = "RoomCell"

    }

    // MARK: Properties

    var homeZone: HMZone!

    lazy var displayedRooms = [HMRoom]()

    lazy var selectedRooms = [HMRoom]()

    // MARK: View Methods

    override func viewWillAppear(_ animated: Bool) {

```

```

super.viewWillAppear(animated)

title = homeZone.name

resetDisplayedRooms()

}

```

/// Adds the selected rooms to the zone and dismisses the view.

```

@IBAction func dismiss(_ sender: AnyObject) {

    addSelectedRoomsToZoneWithCompletionHandler {

        self.dismiss(animated: true, completion: nil)

    }

}

```

/**

Creates a dispatch group, adds all of the rooms to the zone,

and runs the provided completion once all rooms have been added.

- parameter completion: A closure to call once all rooms have been added.

```
*/
```

```
func addSelectedRoomsToZoneWithCompletionHandler(_ completion: @escaping () ->
Void) {

    let group = DispatchGroup()

    for room in selectedRooms {

        group.enter()

        homeZone.addRoom(room) { error in

            if let error = error {

                self.displayError(error)

            }

            group.leave()

        }

    }

    group.notify(queue: DispatchQueue.main, execute: completion)

}
```

```
// MARK: Table View Methods
```

/// - returns: The number of displayed rooms.

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {

    return displayedRooms.count

}
```

/// - returns: A cell that includes the name of a room and a checkmark if it's intended to be added to the zone.

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.roomCell, for:
indexPath)

    let room = displayedRooms[indexPath.row]

    cell.textLabel?.text = room.name

    cell.accessoryType = selectedRooms.contains(room) ? .checkmark : .none

    return cell
}
```



```
}
```

```
/// Adds the selected room to the selected rooms array and reloads that cell
```

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
```

```
    let room = displayedRooms[indexPath.row]
```

```
    if let index = selectedRooms.firstIndex(of: room) {
```

```
        selectedRooms.remove(at: index)
```

```
    }
```

```
    else {
```

```
        selectedRooms.append(room)
```

```
    }
```

```
    tableView.reloadRows(at: [indexPath], with: .automatic)
```

```
}
```

```
/// Resets the list of displayed rooms and reloads the table.
```

```

func resetDisplayedRooms() {

    displayedRooms = home.roomsNotAlreadyInZone(homeZone, includingRooms:
selectedRooms)

    if displayedRooms.isEmpty {

        self.dismiss(animated: true, completion: nil)

    }

    else {

        tableView.reloadData()

    }

}

```

// MARK: HMHomeDelegate Methods

/// If our zone was removed, dismiss this view.

```

func home(_ home: HMHome, didRemoveZone zone: HMZone) {

    if zone == homeZone {

        self.dismiss(animated: true, completion: nil)

    }

}

```

```
}
```

```
/// If our zone was renamed, reset our title.
```

```
func home(_ home: HMHome, didUpdateNameForZone zone: HMZone) {
```

```
    if zone == homeZone {
```

```
        title = zone.name
```

```
    }
```

```
}
```

```
// All home updates reset the displayed homes and reload the view.
```

```
func home(_ home: HMHome, didUpdateNameForRoom room: HMRoom) {
```

```
    resetDisplayedRooms()
```

```
}
```

```
func home(_ home: HMHome, didAddRoom room: HMRoom) {
```

```
    resetDisplayedRooms()
```

```
}
```

```
func home(_ home: HMHome, didRemoveRoom room: HMRoom) {
```

```
    resetDisplayedRooms()
```

```
}
```

```
func home(_ home: HMHome, didAddRoom room: HMRoom, toZone zone: HMZone)
```

```
{
```

```
    resetDisplayedRooms()
```

```
}
```

```
func home(_ home: HMHome, didRemoveRoom room: HMRoom, fromZone zone:  
HMZone) {
```

```
    resetDisplayedRooms()
```

```
}
```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `AddServicesViewController` allows users to add services to a service group.

*/

import UIKit

import HomeKit

/**

A view controller that provides a list of services and lets the user select services to be added to the provided Service Group.

The services are not added to the service group until the 'Done' button is pressed.

*/

class AddServicesViewController: HMCatalogViewController, HMAccessoryDelegate {

// MARK: Types

```

struct Identifiers {

    static let serviceCell = "ServiceCell"

}

// MARK: Properties

lazy private var displayedAccessories = [HMAccessory]()

lazy private var displayedServicesForAccessory = [HMAccessory: [HMService]]()

lazy private var selectedServices = [HMService]()

var serviceGroup: HMServiceGroup!

// MARK: View Methods

/// Reloads internal data and view.

override func viewWillAppear(_ animated: Bool) {

    super.viewWillAppear(animated)

```

```

        selectedServices = []

        reloadData()

    }

    /// Registers as the delegate for the home and all accessories.

    override func registerAsDelegate() {

        super.registerAsDelegate()

        for accessory in homeStore.home!.accessories {

            accessory.delegate = self

        }

    }

```

// MARK: Table View Methods

```

    /// - returns: The number of displayed accessories.

    override func numberOfSections(in tableView: UITableView) -> Int {

        return displayedAccessories.count
    }

```

```
}
```

```
/// - returns: The number of displayed services for the provided accessory.
```

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
```

```
    let accessory = displayedAccessories[section]
```

```
    return displayedServicesForAccessory[accessory]!.count
```

```
}
```

```
/// - returns: A configured 'ServiceCell'.
```

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
```

```
    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.serviceCell, for: indexPath) as! ServiceCell
```

```
    let service = serviceAtIndexPath(indexPath)
```

```
    cell.includeAccessoryText = false
```

```
    cell.service = service
```



```
cell.accessoryType = selectedServices.contains(service) ? .checkmark : .none
```

```
return cell
```

```
}
```

```
/**
```

When an indexPath is selected, this function either adds or removes the selected service from the

service group.

```
*/
```

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
```

```
// Get the service associated with this index.
```

```
let service = serviceAtIndexPath(indexPath)
```

```
// Call the appropriate add/remove operation with the closure from above.
```

```
if let index = selectedServices.firstIndex(of: service) {
```

```
selectedServices.remove(at: index)
```

```

    }

    else {

        selectedServices.append(service)

    }

    tableView.reloadRows(at: [indexPath], with: .automatic)

}

/// - returns: The name of the displayed accessory at the given section.

override func tableView(_ tableView: UITableView, titleForHeaderInSection section:
Int) -> String? {

    return displayedAccessories[section].name

}

// MARK: Helper Methods

/**

    Adds the selected services to the service group.

```

Calls the provided completion handler once all services have been added.

```
*/
```

```
func addSelectedServicesWithCompletionHandler(_ completion: @escaping () -> Void)
{
    // Create a dispatch group for each of the service additions.

    let addServicesGroup = DispatchGroup()

    for service in selectedServices {

        addServicesGroup.enter()

        serviceGroup.addService(service) { error in

            if let error = error {

                self.displayError(error)

            }

            addServicesGroup.leave()

        }

    }

    addServicesGroup.notify(queue: DispatchQueue.main, execute: completion)

}
```

```
/**
```

Finds the service at a specific index path.

- parameter indexPath: An `NSIndexPath`

- returns: The `HMService` at the given index path.

```
*/
```

```
private func serviceAtIndexPath(_ indexPath: IndexPath) -> HMService {
```

```
    let accessory = displayedAccessories[indexPath.section]
```

```
    let services = displayedServicesForAccessory[accessory]!
```

```
    return services[indexPath.row]
```

```
}
```

```
/**
```

Commits the changes to the service group

and dismisses the view.

```

*/

@IBAction func dismiss() {

    addSelectedServicesWithCompletionHandler {

        self.dismiss(animated: true, completion: nil)

    }

}

/// Resets internal data and view.

func reloadTable() {

    resetDisplayedServices()

    tableView.reloadData()

}

/**

Updates internal array of accessories and the mapping

of accessories to selected services.

*/

```

```

func resetDisplayedServices() {

    displayedAccessories = []

    let allAccessories = home.accessories.sortByLocalizedName()

    displayedServicesForAccessory = [:]

    for accessory in allAccessories {

        var displayedServices = [HMService]()

        for service in accessory.services {

            if !serviceGroup.services.contains(service)    &&    service.serviceType !=
HMServiceTypeAccessoryInformation {

                displayedServices.append(service)

            }

        }

    }

    // Only add the accessory if it has displayed services.

    if !displayedServices.isEmpty {

        displayedServicesForAccessory[accessory] =
displayedServices.sortByLocalizedName()

        displayedAccessories.append(accessory)
    }
}

```

```
    }  
  
    }  
  
}
```

```
// MARK: HMHomeDelegate Methods
```

```
/// Dismisses the view controller if our service group was removed.
```

```
func home(_ home: HMHome, didRemoveServiceGroup group: HMServiceGroup) {  
  
    if serviceGroup == group {  
  
        self.dismiss(animated: true, completion: nil)  
  
    }  
  
}
```

```
/// Reloads the view if an accessory was added to HomeKit.
```

```
func home(_ home: HMHome, didAddAccessory accessory: HMAccessory) {  
  
    reloadData()  
  
    accessory.delegate = self
```

```
}
```

```
/// Dismisses the view controller if we no longer have accesories.
```

```
func home(_ home: HMHome, didRemoveAccessory accessory: HMAccessory) {
```

```
    if home.accessories.isEmpty {
```

```
        navigationController?.dismiss(animated: true, completion: nil)
```

```
    }
```

```
    reloadData()
```

```
}
```

```
// MARK: HMAccessoryDelegate Methods
```

```
// Accessory changes reload the data and view.
```

```
func accessory(_ accessory: HMAccessory, didUpdateNameFor service: HMService) {
```

```
    reloadData()
```



```

    }

func accessoryDidUpdateServices(_ accessory: HMAccessory) {

    reloadData()

}

}

//

// Alamofire.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

```

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

/// Reference to `Session.default` for quick bootstrapping and examples.

```
public let AF = Session.default
```

```
/// Current Alamofire version. Necessary since SPM doesn't use dynamic libraries. Plus  
this will be more accurate.
```

```
let version = "5.0.5"
```

```
//
```

```
// AlamofireExtended.swift
```

```
//
```

```
// Copyright (c) 2019 Alamofire Software Foundation (http://alamofire.org/)
```

```
//
```

```
// Permission is hereby granted, free of charge, to any person obtaining a copy
```

```
// of this software and associated documentation files (the "Software"), to deal
```

```
// in the Software without restriction, including without limitation the rights
```

```
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
```

```
// copies of the Software, and to permit persons to whom the Software is
```

```
// furnished to do so, subject to the following conditions:
```

```
//
```

```
// The above copyright notice and this permission notice shall be included in
```

```
// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
// EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
// MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
// EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
// ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
// DEALINGS IN

// THE SOFTWARE.

//
```

```
/// Type that acts as a generic extension point for all `AlamofireExtended` types.
```

```
public struct AlamofireExtension<ExtendedType> {
```

```
    /// Stores the type or meta-type of any extended type.
```

```

public private(set) var type: ExtendedType

/// Create an instance from the provided value.

///

/// - Parameter type: Instance being extended.

public init(_ type: ExtendedType) {

    self.type = type

}

}

/// Protocol describing the `af` extension points for Alamofire extended types.

public protocol AlamofireExtended {

    /// Type being extended.

    associatedtype ExtendedType

    /// Static Alamofire extension point.

    static var af: AlamofireExtension<ExtendedType>.Type { get set }

```

```

    /// Instance Alamofire extension point.

    var af: AlamofireExtension<ExtendedType> { get set }

}

public extension AlamofireExtended {

    /// Static Alamofire extension point.

    static var af: AlamofireExtension<Self>.Type {

        get { return AlamofireExtension<Self>.self }

        set {}

    }

    /// Instance Alamofire extension point.

    var af: AlamofireExtension<Self> {

        get { return AlamofireExtension(self) }

        set {}

    }

}

```

```

//

// AnimatedMoveViewJob.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class AnimatedMoveViewJob: AnimatedViewPortJob

{

    internal override func animationUpdate()

```

```

{

    guard

        let viewPortHandler = viewPortHandler,

        let transformer = transformer,

        let view = view

    else { return }

    var pt = CGPoint(

        x: xOrigin + (CGFloat(xValue) - xOrigin) * phase,

        y: yOrigin + (CGFloat(yValue) - yOrigin) * phase

    )

    transformer.pointValueToPixel(&pt)

    viewPortHandler.centerViewPort(pt: pt, chart: view)

}

}

//

```



```
// AnimatedViewPortJob.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//
```

```
import Foundation
```

```
import CoreGraphics
```

```
import QuartzCore
```

```
open class AnimatedViewPortJob: ViewPortJob
```

```
{
```

```
    internal var phase: CGFloat = 1.0
```

```
internal var xOrigin: CGFloat = 0.0
```

```
internal var yOrigin: CGFloat = 0.0
```

```
private var _startTime: TimeInterval = 0.0
```

```
private var _displayLink: NSUIDisplayLink!
```

```
private var _duration: TimeInterval = 0.0
```

```
private var _endTime: TimeInterval = 0.0
```

```
private var _easing: ChartEasingFunctionBlock?
```

```
@objc public init(  

```

```
    viewportHandler: ViewPortHandler,  

```

```
    xValue: Double,  

```

```
    yValue: Double,  

```

```
    transformer: Transformer,  

```

```
    view: ChartViewBase,  

```

```
    xOrigin: CGFloat,  

```

```

    yOrigin: CGFloat,

    duration: TimeInterval,

    easing: ChartEasingFunctionBlock?)

{

    super.init(viewPortHandler: viewPortHandler,

        xValue: xValue,

        yValue: yValue,

        transformer: transformer,

        view: view)

    self.xOrigin = xOrigin

    self.yOrigin = yOrigin

    self._duration = duration

    self._easing = easing

}

deinit

```

```
{  
  
    stop(finish: false)  
  
}
```

open override func doJob()

```
{  
  
    start()  
  
}
```

@objc open func start()

```
{  
  
    _startTime = CACurrentMediaTime()  
  
    _endTime = _startTime + _duration  
  
    _endTime = _endTime > _endTime ? _endTime : _endTime  
  
    updateAnimationPhase(_startTime)
```

```
_displayLink = NSUIDisplayLink(target: self, selector: #selector(animationLoop))

_displayLink.add(to: .main, forMode: RunLoop.Mode.common)

}
```

```
@objc open func stop(finish: Bool)
```

```
{

    guard _displayLink != nil else { return }
```

```
_displayLink.remove(from: .main, forMode: RunLoop.Mode.common)
```

```
_displayLink = nil
```

```
if finish
```

```
{
```

```
    if phase != 1.0
```

```
    {
```

```
        phase = 1.0
```

```
        animationUpdate()
```

```
}
```

```
    animationEnd()
```

```
}
```

```
}
```

```
private func updateAnimationPhase(_ currentTime: TimeInterval)
```

```
{
```

```
    let elapsedTime = currentTime - _startTime
```

```
    let duration = _duration
```

```
    var elapsed = elapsedTime
```

```
    elapsed = min(elapsed, duration)
```

```
    phase = CGFloat(_easing?(elapsed, duration) ?? elapsed / duration)
```

```
}
```

```

@objc private func animationLoop()

{

    let currentTime: TimeInterval = CACurrentMediaTime()


    updateAnimationPhase(currentTime)


    animationUpdate()


    if currentTime >= _endTime

    {

        stop(finish: true)

    }

}


internal func animationUpdate()

{

    // Override this

```

```

    }

    internal func animationEnd()

    {

        // Override this

    }

}

//

// AnimatedZoomViewJob.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

```



```

import Foundation

import CoreGraphics

open class AnimatedZoomViewJob: AnimatedViewPortJob

{

    internal var yAxis: YAxis?

    internal var xAxisRange: Double = 0.0

    internal var scaleX: CGFloat = 0.0

    internal var scaleY: CGFloat = 0.0

    internal var zoomOriginX: CGFloat = 0.0

    internal var zoomOriginY: CGFloat = 0.0

    internal var zoomCenterX: CGFloat = 0.0

    internal var zoomCenterY: CGFloat = 0.0


    @objc public init(

        viewPortHandler: ViewPortHandler,

```

```

transformer: Transformer,

view: ChartViewBase,

yAxis: YAxis,

xAxisRange: Double,

scaleX: CGFloat,

scaleY: CGFloat,

xOrigin: CGFloat,

yOrigin: CGFloat,

zoomCenterX: CGFloat,

zoomCenterY: CGFloat,

zoomOriginX: CGFloat,

zoomOriginY: CGFloat,

duration: TimeInterval,

easing: ChartEasingFunctionBlock?)

{

    super.init(viewPortHandler: viewPortHandler,

        xValue: 0.0,

```

```
yValue: 0.0,  
  
transformer: transformer,  
  
view: view,  
  
xOrigin: xOrigin,  
  
yOrigin: yOrigin,  
  
duration: duration,  
  
easing: easing)
```

```
self.yAxis = yAxis
```

```
self.xAxisRange = xAxisRange
```

```
self.scaleX = scaleX
```

```
self.scaleY = scaleY
```

```
self.zoomCenterX = zoomCenterX
```

```
self.zoomCenterY = zoomCenterY
```

```
self.zoomOriginX = zoomOriginX
```

```
self.zoomOriginY = zoomOriginY
```

```
}
```

```

internal override func animationUpdate()

{

    guard

        let viewPortHandler = viewPortHandler,

        let transformer = transformer,

        let view = view

    else { return }


    let scaleX = xOrigin + (self.scaleX - xOrigin) * phase

    let scaleY = yOrigin + (self.scaleY - yOrigin) * phase


    var matrix = viewPortHandler.setZoom(scaleX: scaleX, scaleY: scaleY)

    viewPortHandler.refresh(newMatrix: matrix, chart: view, invalidate: false)


    let valsInView = CGFloat(yAxis?.axisRange ?? 0.0) / viewPortHandler.scaleY

    let xsInView = CGFloat(xAxisRange) / viewPortHandler.scaleX

```

```

var pt = CGPoint(

    x: zoomOriginX + ((zoomCenterX - xsInView / 2.0) - zoomOriginX) * phase,

    y: zoomOriginY + ((zoomCenterY + valsInView / 2.0) - zoomOriginY) * phase

)

transformer.pointValueToPixel(&pt)

matrix = viewPortHandler.translate(pt: pt)

viewPortHandler.refresh(newMatrix: matrix, chart: view, invalidate: true)

}

internal override func animationEnd()

{

    (view as? BarLineChartViewBase)?.calculateOffsets()

    view?.setNeedsDisplay()

}

```

```
}

//

// Animator.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

import QuartzCore

@objc(ChartAnimatorDelegate)
```

```

public protocol AnimatorDelegate

{

    /// Called when the Animator has stepped.

    func animatorUpdated(_ animator: Animator)


    /// Called when the Animator has stopped.

    func animatorStopped(_ animator: Animator)

}


@objc(ChartAnimator)

open class Animator: NSObject

{

    @objc open weak var delegate: AnimatorDelegate?

    @objc open var updateBlock: (() -> Void)?

    @objc open var stopBlock: (() -> Void)?


    /// the phase that is animated and influences the drawn values on the x-axis

```

```
@objc open var phaseX: Double = 1.0
```

```
/// the phase that is animated and influences the drawn values on the y-axis
```

```
@objc open var phaseY: Double = 1.0
```

```
private var _startTimeX: TimeInterval = 0.0
```

```
private var _startTimeY: TimeInterval = 0.0
```

```
private var _displayLink: NSUIDisplayLink?
```

```
private var _durationX: TimeInterval = 0.0
```

```
private var _durationY: TimeInterval = 0.0
```

```
private var _endTimeX: TimeInterval = 0.0
```

```
private var _endTimeY: TimeInterval = 0.0
```

```
private var _endTime: TimeInterval = 0.0
```

```
private var _enabledX: Bool = false
```



```
private var _enabledY: Bool = false
```

```
private var _easingX: ChartEasingFunctionBlock?
```

```
private var _easingY: ChartEasingFunctionBlock?
```

```
public override init()
```

```
{  
  
    super.init()  
  
}
```

```
deinit
```

```
{  
  
    stop()  
  
}
```

```
@objc open func stop()
```

```
{
```

```
guard _displayLink != nil else { return }

_displayLink?.remove(from: .main, forMode: RunLoop.Mode.common)

_displayLink = nil

_enabledX = false

_enabledY = false

// If we stopped an animation in the middle, we do not want to leave it like this

if phaseX != 1.0 || phaseY != 1.0

{

    phaseX = 1.0

    phaseY = 1.0

    delegate?.animatorUpdated(self)

    updateBlock?()

}
```

```

        delegate?.animatorStopped(self)

        stopBlock?()

    }

    private func updateAnimationPhases(_ currentTime: TimeInterval)

    {

        if _enabledX

        {

            let elapsedTime: TimeInterval = currentTime - _startTimeX

            let duration: TimeInterval = _durationX

            var elapsed: TimeInterval = elapsedTime

            if elapsed > duration

            {

                elapsed = duration

            }

        }

    }

```

```

        phaseX = _easingX?(elapsed, duration) ?? elapsed / duration
    }

    if _enabledY

    {

        let elapsedTime: TimeInterval = currentTime - _startTimeY

        let duration: TimeInterval = _durationY

        var elapsed: TimeInterval = elapsedTime

        if elapsed > duration

        {

            elapsed = duration

        }

        phaseY = _easingY?(elapsed, duration) ?? elapsed / duration

    }

}

```

```

@objc private func animationLoop()

{

    let currentTime: TimeInterval = CACurrentMediaTime()


    updateAnimationPhases(currentTime)


    delegate?.animatorUpdated(self)

    updateBlock?()


    if currentTime >= _endTime

    {

        stop()

    }

}

```

/// Animates the drawing / rendering of the chart on both x- and y-axis with the specified animation time.

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the chart.

///

/// - Parameters:

/// - xAxisDuration: duration for animating the x axis

/// - yAxisDuration: duration for animating the y axis

/// - easingX: an easing function for the animation on the x axis

/// - easingY: an easing function for the animation on the y axis

@objc open func animate(xAxisDuration: TimeInterval, yAxisDuration: TimeInterval, easingX: ChartEasingFunctionBlock?, easingY: ChartEasingFunctionBlock?)

{

stop()

_startTimeX = CACurrentMediaTime()

_startTimeY = _startTimeX

_durationX = xAxisDuration

_durationY = yAxisDuration

_endTimeX = _startTimeX + xAxisDuration

```

    _endTimeY = _startTimeY + yAxisDuration

    _endTime = _endTimeX > _endTimeY ? _endTimeX : _endTimeY

    _enabledX = xAxisDuration > 0.0

    _enabledY = yAxisDuration > 0.0

    _easingX = easingX

    _easingY = easingY

    // Take care of the first frame if rendering is already scheduled...

    updateAnimationPhases(_startTimeX)

    if _enabledX || _enabledY

    {

        _displayLink = NSUIDisplayLink(target: self, selector: #selector(animationLoop))

        _displayLink?.add(to: RunLoop.main, forMode: RunLoop.Mode.common)

    }

}

```

/// Animates the drawing / rendering of the chart on both x- and y-axis with the specified animation time.

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the chart.

///

/// - Parameters:

/// - xAxisDuration: duration for animating the x axis

/// - yAxisDuration: duration for animating the y axis

/// - easingOptionX: the easing function for the animation on the x axis

/// - easingOptionY: the easing function for the animation on the y axis

@objc open func animate(xAxisDuration: TimeInterval, yAxisDuration: TimeInterval, easingOptionX: ChartEasingOption, easingOptionY: ChartEasingOption)

{

animate(xAxisDuration: xAxisDuration, yAxisDuration: yAxisDuration, easingX: easingFunctionFromOption(easingOptionX), easingY: easingFunctionFromOption(easingOptionY))

}

/// Animates the drawing / rendering of the chart on both x- and y-axis with the specified animation time.

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the chart.

///

/// - Parameters:

/// - xAxisDuration: duration for animating the x axis

/// - yAxisDuration: duration for animating the y axis

/// - easing: an easing function for the animation

@objc open func animate(xAxisDuration: TimeInterval, yAxisDuration: TimeInterval, easing: ChartEasingFunctionBlock?)

{

animate(xAxisDuration: xAxisDuration, yAxisDuration: yAxisDuration, easingX: easing, easingY: easing)

}

/// Animates the drawing / rendering of the chart on both x- and y-axis with the specified animation time.

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the chart.

```

///

/// - Parameters:

/// - xAxisDuration: duration for animating the x axis

/// - yAxisDuration: duration for animating the y axis

/// - easingOption: the easing function for the animation

@objc open func animate(xAxisDuration: TimeInterval, yAxisDuration: TimeInterval,
easingOption: ChartEasingOption = .easeInOutSine)

{

    animate(xAxisDuration: xAxisDuration, yAxisDuration: yAxisDuration, easing:
easingFunctionFromOption(easingOption))

}

/// Animates the drawing / rendering of the chart the x-axis with the specified animation
time.

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the
chart.

///

/// - Parameters:

/// - xAxisDuration: duration for animating the x axis

```

```

/// - easing: an easing function for the animation

@objc open func animate(xAxisDuration: TimeInterval, easing:
ChartEasingFunctionBlock?)

{

    _startTimeX = CACurrentMediaTime()

    _durationX = xAxisDuration

    _endTimeX = _startTimeX + xAxisDuration

    _endTime = _endTimeX > _endTimeY ? _endTimeX : _endTimeY

    _enabledX = xAxisDuration > 0.0

    _easingX = easing

    // Take care of the first frame if rendering is already scheduled...

    updateAnimationPhases(_startTimeX)

    if _enabledX || _enabledY,

        _displayLink == nil

    {

```

```

        _displayLink = NSUIDisplayLink(target: self, selector: #selector(animationLoop))

        _displayLink?.add(to: .main, forMode: RunLoop.Mode.common)

    }

}

/// Animates the drawing / rendering of the chart the x-axis with the specified animation
time.

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the
chart.

///

/// - Parameters:

/// - xAxisDuration: duration for animating the x axis

/// - easingOption: the easing function for the animation

@objc open func animate(xAxisDuration: TimeInterval, easingOption:
ChartEasingOption = .easeInOutSine)

{

    animate(xAxisDuration:          xAxisDuration,          easing:
easingFunctionFromOption(easingOption))

}

```

```
/// Animates the drawing / rendering of the chart the y-axis with the specified animation time.
```

```
/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the chart.
```

```
///
```

```
/// - Parameters:
```

```
/// - yAxisDuration: duration for animating the y axis
```

```
/// - easing: an easing function for the animation
```

```
@objc open func animate(yAxisDuration: TimeInterval, easing: ChartEasingFunctionBlock?)
```

```
{
```

```
    _startTimeY = CACurrentMediaTime()
```

```
    _durationY = yAxisDuration
```

```
    _endTimeY = _startTimeY + yAxisDuration
```

```
    _endTime = _endTimeX > _endTimeY ? _endTimeX : _endTimeY
```

```
    _enabledY = yAxisDuration > 0.0
```

```
    _easingY = easing
```

```

// Take care of the first frame if rendering is already scheduled...

updateAnimationPhases(_startTimeY)

if _enabledX || _enabledY,

    _displayLink == nil

{

    _displayLink = NSUIDisplayLink(target: self, selector: #selector(animationLoop))

    _displayLink?.add(to: .main, forMode: RunLoop.Mode.common)

}

}

/// Animates the drawing / rendering of the chart the y-axis with the specified animation
time.

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the
chart.

///

/// - Parameters:

```

```

/// - yAxisDuration: duration for animating the y axis

/// - easingOption: the easing function for the animation

@objc open func animate(yAxisDuration: TimeInterval, easingOption:
ChartEasingOption = .easeInOutSine)

{

    animate(yAxisDuration:          yAxisDuration,          easing:
easingFunctionFromOption(easingOption))

}

}

//

// AppDelegate.swift

// FirebaseStarterApp

//

// Created by Florian Marcu on 2/21/19.

// Copyright © 2019 Instamobile. All rights reserved.

//

import Firebase

```

```
import UIKit
```

```
import FacebookCore
```

```
@UIApplicationMain
```

```
class AppDelegate: UIResponder, UIApplicationDelegate {
```

```
    var window: UIWindow?
```

```
    func application(_ application: UIApplication, didFinishLaunchingWithOptions  
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
```

```
        FirebaseApp.configure()
```

```
        var navigationBarAppearance = UINavigationController.appearance()
```

```
        // Window setup
```

```
        window = UIWindow(frame: UIScreen.main.bounds)
```

```
        window?.rootViewController = UINavigationController(rootViewController:  
ATCCClassicLandingScreenViewController(nibName:  
"ATCCClassicLandingScreenViewController", bundle: nil))
```

```
        window?.makeKeyAndVisible()
```



```

        return true
    }

    func application(_ app: UIApplication, open url: URL, options:
[UIApplication.OpenURLOptionsKey : Any] = [:]) -> Bool {

        return AppDelegate.shared.application(app, open: url, options: options)

    }

}

```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `Array+Sorting` extension allows for easy sorting of HomeKit objects.

```
*/
```

```
import HomeKit
```

```
/// A protocol for objects which have a property called `name`.
```

```
protocol Nameable {
```

```
    var name: String { get }
```

```
}
```

```
/*
```

```
    All of these HomeKit objects have names and can conform
```

```
    to this protocol without modification.
```

```
*/
```

```
extension HMHome: Nameable {}
```

```
extension HMAccessory: Nameable {}
```

```
extension HMRoom: Nameable {}
```

```
extension HMZone: Nameable {}
```

```
extension HMActionSet: Nameable {}
```

```
extension HMService: Nameable {}
```

```
extension HMServiceGroup: Nameable {}
```

```
extension HMTrigger: Nameable {}
```

```
extension Collection where Iterator.Element: Nameable {
```

```
  /**
```

```
    Generates a new array from the original collection,
```

```
    sorted by localized name.
```

```
    - returns: New array sorted by localized name.
```

```
  */
```

```
  func sortByLocalizedNames() -> [Iterator.Element] {
```

```
    return sorted { return $0.name.localizedCompare($1.name) == .orderedAscending }
```

```
  }
```

```
}
```

```
extension Collection where Iterator.Element: HMActionSet {
```

```
  /**
```

Generates a new array from the original collection,

sorted by built-in first, then user-defined sorted

by localized name.

- returns: New array sorted by localized name.

*/

```
func sortByTypeAndLocalizedName() -> [HMActionSet] {  
  
    return sorted { (actionSet1, actionSet2) -> Bool in  
  
        if actionSet1.isBuiltIn != actionSet2.isBuiltIn {  
  
            // If comparing a built-in and a user-defined, the built-in is ranked first.  
  
            return actionSet1.isBuiltIn  
  
        }  
  
        else if actionSet1.isBuiltIn && actionSet2.isBuiltIn {  
  
            // If comparing two built-ins, we follow a standard ranking  
  
            let firstIndex = HMActionSet.Constants.builtInActionSetTypes.firstIndex(of:  
actionSet1.actionSetType) ?? NSNotFound  
  
            let secondIndex = HMActionSet.Constants.builtInActionSetTypes.firstIndex(of:  
actionSet2.actionSetType) ?? NSNotFound
```

```

        return firstIndex < secondIndex

    }

    else {

        // If comparing two user-defines, sort by localized name.

        return          actionSet1.name.localizedCompare(actionSet2.name)
        == .orderedAscending

    }

}

}

}

}

//

// AxisBase.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

```

```

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

/// Base class for all axes

@objc(ChartAxisBase)

open class AxisBase: ComponentBase

{

    public override init()

    {

        super.init()

    }

}

/// Custom formatter that is used instead of the auto-formatter if set

private var _axisValueFormatter: IAxisValueFormatter?

```

```
@objc open var labelFont = NSUIFont.systemFont(ofSize: 10.0)
```

```
@objc open var labelTextColor = NSUIColor.labelOrBlack
```

```
@objc open var axisLineColor = NSUIColor.gray
```

```
@objc open var axisLineWidth = CGFloat(0.5)
```

```
@objc open var axisLineDashPhase = CGFloat(0.0)
```

```
@objc open var axisLineDashLengths: [CGFloat]!
```

```
@objc open var gridColor = NSUIColor.gray.withAlphaComponent(0.9)
```

```
@objc open var gridLineWidth = CGFloat(0.5)
```

```
@objc open var gridLineDashPhase = CGFloat(0.0)
```

```
@objc open var gridLineDashLengths: [CGFloat]!
```

```
@objc open var gridLineCap = CGLineCap.butt
```

```
@objc open var drawGridLinesEnabled = true
```

```
@objc open var drawAxisLineEnabled = true
```

```
/// flag that indicates of the labels of this axis should be drawn or not
```

```
@objc open var drawLabelsEnabled = true
```

```
private var _centerAxisLabelsEnabled = false
```

```
/// Centers the axis labels instead of drawing them at their original position.
```

```
/// This is useful especially for grouped BarChart.
```

```
@objc open var centerAxisLabelsEnabled: Bool
```

```
{
```

```
    get { return _centerAxisLabelsEnabled && entryCount > 0 }
```

```
    set { _centerAxisLabelsEnabled = newValue }
```

```
}
```

```
@objc open var isCenterAxisLabelsEnabled: Bool
```

```
{
```

```
    get { return centerAxisLabelsEnabled }
```



```
}
```

```
/// array of limitlines that can be set for the axis
```

```
private var _limitLines = [ChartLimitLine]()
```

```
/// Are the LimitLines drawn behind the data or in front of the data?
```

```
///
```

```
/// default: false
```

```
@objc open var drawLimitLinesBehindDataEnabled = false
```

```
/// Are the grid lines drawn behind the data or in front of the data?
```

```
///
```

```
/// default: true
```

```
@objc open var drawGridLinesBehindDataEnabled = true
```

```
/// the flag can be used to turn off the antialias for grid lines
```

```
@objc open var gridAntialiasEnabled = true
```

```
/// the actual array of entries
```

```
@objc open var entries = [Double]()
```

```
/// axis label entries only used for centered labels
```

```
@objc open var centeredEntries = [Double]()
```

```
/// the number of entries the legend contains
```

```
@objc open var entryCount: Int { return entries.count }
```

```
/// the number of label entries the axis should have
```

```
///
```

```
/// default: 6
```

```
private var _labelCount = Int(6)
```

```
/// the number of decimal digits to use (for the default formatter
```

```
@objc open var decimals: Int = 0
```

```

/// When true, axis labels are controlled by the `granularity` property.

/// When false, axis values could possibly be repeated.

/// This could happen if two adjacent axis values are rounded to same value.

/// If using granularity this could be avoided by having fewer axis values visible.

@objc open var granularityEnabled = false


private var _granularity = Double(1.0)


/// The minimum interval between axis values.

/// This can be used to avoid label duplicating when zooming in.

///

/// **default**: 1.0

@objc open var granularity: Double

{

    get

    {

```

```

        return _granularity

    }

    set

    {

        _granularity = newValue

        // set this to `true` if it was disabled, as it makes no sense to set this property with
granularity disabled

        granularityEnabled = true

    }

}

/// The minimum interval between axis values.

@objc open var isGranularityEnabled: Bool

{

    get

    {

        return granularityEnabled

```

```
}
```

```
}
```

```
/// if true, the set number of y-labels will be forced
```

```
@objc open var forceLabelsEnabled = false
```

```
@objc open func getLongestLabel() -> String
```

```
{
```

```
    var longest = ""
```

```
    for i in 0 ..< entries.count
```

```
    {
```

```
        let text = getFormattedLabel(i)
```

```
        if longest.count < text.count
```

```
        {
```

```
            longest = text
```

```

    }

}

return longest

}

```

/// - Returns: The formatted label at the specified index. This will either use the auto-formatter or the custom formatter (if one is set).

```

@objc open func getFormattedLabel(_ index: Int) -> String

{

    if index < 0 || index >= entries.count

    {

        return ""

    }

    return valueFormatter?.stringForValue(entries[index], axis: self) ?? ""

}

```

```
/// Sets the formatter to be used for formatting the axis labels.
```

```
/// If no formatter is set, the chart will automatically determine a reasonable formatting  
(concerning decimals) for all the values that are drawn inside the chart.
```

```
/// Use `nil` to use the formatter calculated by the chart.
```

```
@objc open var valueFormatter: IAxisValueFormatter?
```

```
{
```

```
    get
```

```
{
```

```
    if _axisValueFormatter == nil ||
```

```
        (_axisValueFormatter is DefaultAxisValueFormatter &&
```

```
            (_axisValueFormatter as! DefaultAxisValueFormatter).hasAutoDecimals &&
```

```
            (_axisValueFormatter as! DefaultAxisValueFormatter).decimals != decimals)
```

```
{
```

```
    _axisValueFormatter = DefaultAxisValueFormatter(decimals: decimals)
```

```
}
```

```
    return _axisValueFormatter
```

```
}
```

```

set

{

    _axisValueFormatter = newValue ?? DefaultAxisValueFormatter(decimals:
decimals)

}

}

```

```

@objc open var isDrawGridLinesEnabled: Bool { return drawGridLinesEnabled }

```

```

@objc open var isDrawAxisLineEnabled: Bool { return drawAxisLineEnabled }

```

```

@objc open var isDrawLabelsEnabled: Bool { return drawLabelsEnabled }

```

```

/// Are the LimitLines drawn behind the data or in front of the data?

```

```

///

```

```

/// default: false

```

```

@objc open var isDrawLimitLinesBehindDataEnabled: Bool { return
drawLimitLinesBehindDataEnabled }

```



```

/// Are the grid lines drawn behind the data or in front of the data?

///

/// **default**: true

@objc open var isDrawGridLinesBehindDataEnabled: Bool { return
drawGridLinesBehindDataEnabled }

/// Extra spacing for `axisMinimum` to be added to automatically calculated
`axisMinimum`

@objc open var spaceMin: Double = 0.0

/// Extra spacing for `axisMaximum` to be added to automatically calculated
`axisMaximum`

@objc open var spaceMax: Double = 0.0

/// Flag indicating that the axis-min value has been customized

internal var _customAxisMin: Bool = false

/// Flag indicating that the axis-max value has been customized

```

```

internal var _customAxisMax: Bool = false

/// Do not touch this directly, instead, use axisMinimum.

/// This is automatically calculated to represent the real min value,

/// and is used when calculating the effective minimum.

internal var _axisMinimum = Double(0)

/// Do not touch this directly, instead, use axisMaximum.

/// This is automatically calculated to represent the real max value,

/// and is used when calculating the effective maximum.

internal var _axisMaximum = Double(0)

/// the total range of values this axis covers

@objc open var axisRange = Double(0)

/// The minumum number of labels on the axis

@objc open var axisMinLabels = Int(2) {

```

```
    didSet { axisMinLabels = axisMinLabels > 0 ? axisMinLabels : oldValue }  
  
}
```

```
/// The maximum number of labels on the axis
```

```
@objc open var axisMaxLabels = Int(25) {  
  
    didSet { axisMaxLabels = axisMaxLabels > 0 ? axisMaxLabels : oldValue }  
  
}
```

```
/// the number of label entries the axis should have
```

```
/// max = 25,
```

```
/// min = 2,
```

```
/// default = 6,
```

```
/// be aware that this number is not fixed and can only be approximated
```

```
@objc open var labelCount: Int
```

```
{
```

```
    get
```

```
    {
```

```

        return _labelCount
    }

    set

    {

        let range = axisMinLabels...axisMaxLabels as ClosedRange

        _labelCount = newValue.clamped(to: range)

        forceLabelsEnabled = false

    }

}

@objc open func setLabelCount(_ count: Int, force: Bool)

{

    self.labelCount = count

    forceLabelsEnabled = force

}

```

```
/// `true` if forcing the y-label count is enabled. Default: false
```

```
@objc open var isForceLabelsEnabled: Bool { return forceLabelsEnabled }
```

```
/// Adds a new ChartLimitLine to this axis.
```

```
@objc open func addLimitLine(_ line: ChartLimitLine)
```

```
{  
  
    _limitLines.append(line)  
  
}
```

```
/// Removes the specified ChartLimitLine from the axis.
```

```
@objc open func removeLimitLine(_ line: ChartLimitLine)
```

```
{  
  
    guard let i = _limitLines.firstIndex(of: line) else { return }  
  
    _limitLines.remove(at: i)  
  
}
```

```
/// Removes all LimitLines from the axis.
```

```

@objc open func removeAllLimitLines()

{

    _limitLines.removeAll(keepingCapacity: false)

}

```

/// The LimitLines of this axis.

```

@objc open var limitLines : [ChartLimitLine]

{

    return _limitLines

}

```

// MARK: Custom axis ranges

/// By calling this method, any custom minimum value that has been previously set is reseted, and the calculation is done automatically.

```

@objc open func resetCustomAxisMin()

{

    _customAxisMin = false

}

```

```
}
```

```
@objc open var isAxisMinCustom: Bool { return _customAxisMin }
```

/// By calling this method, any custom maximum value that has been previously set is reseted, and the calculation is done automatically.

```
@objc open func resetCustomAxisMax()
```

```
{
```

```
    _customAxisMax = false
```

```
}
```

```
@objc open var isAxisMaxCustom: Bool { return _customAxisMax }
```

/// The minimum value for this axis.

/// If set, this value will not be calculated automatically depending on the provided data.

/// Use `resetCustomAxisMin()` to undo this.

```
@objc open var axisMinimum: Double
```

```
{
```

```

get

{

    return _axisMinimum

}

set

{

    _customAxisMin = true

    _axisMinimum = newValue

    axisRange = abs(_axisMaximum - newValue)

}

}

/// The maximum value for this axis.

/// If set, this value will not be calculated automatically depending on the provided data.

/// Use `resetCustomAxisMax()` to undo this.

@objc open var axisMaximum: Double

{

```



```

get

{

    return _axisMaximum

}

set

{

    _customAxisMax = true

    _axisMaximum = newValue

    axisRange = abs(newValue - _axisMinimum)

}

}

```

/// Calculates the minimum, maximum and range values of the YAxis with the given minimum and maximum values from the chart data.

///

/// - Parameters:

/// - dataMin: the y-min value according to chart data

/// - dataMax: the y-max value according to chart

```

@objc open func calculate(min dataMin: Double, max dataMax: Double)

{

    // if custom, use value as is, else use data value

    var min = _customAxisMin ? _axisMinimum : (dataMin - spaceMin)

    var max = _customAxisMax ? _axisMaximum : (dataMax + spaceMax)


    // temporary range (before calculations)

    let range = abs(max - min)


    // in case all values are equal

    if range == 0.0

    {

        max = max + 1.0

        min = min - 1.0

    }


    _axisMinimum = min

```

```

        _axisMaximum = max

        // actual range

        axisRange = abs(max - min)

    }

}

//

// AxisRendererBase.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

```

```
import Foundation
```

```
import CoreGraphics
```

```
@objc(CharAxisRendererBase)
```

```
open class AxisRendererBase: Renderer
```

```
{
```

```
    /// base axis this axis renderer works with
```

```
    @objc open var axis: AxisBase?
```

```
    /// transformer to transform values to screen pixels and return
```

```
    @objc open var transformer: Transformer?
```

```
    @objc public init(viewPortHandler: ViewPortHandler, transformer: Transformer?, axis:  
AxisBase?)
```

```
{
```

```
    super.init(viewPortHandler: viewPortHandler)
```

```
    self.transformer = transformer
```

```

        self.axis = axis

    }

    /// Draws the axis labels on the specified context

    @objc open func renderAxisLabels(context: CGContext)

    {

        fatalError("renderAxisLabels() cannot be called on AxisRendererBase")

    }

    /// Draws the grid lines belonging to the axis.

    @objc open func renderGridLines(context: CGContext)

    {

        fatalError("renderGridLines() cannot be called on AxisRendererBase")

    }

    /// Draws the line that goes alongside the axis.

    @objc open func renderAxisLine(context: CGContext)

```

```

{

    fatalError("renderAxisLine() cannot be called on AxisRendererBase")

}


/// Draws the LimitLines associated with this axis to the screen.

@objc open func renderLimitLines(context: CGContext)

{

    fatalError("renderLimitLines() cannot be called on AxisRendererBase")

}


/// Computes the axis values.

///

/// - Parameters:

/// - min: the minimum value in the data object for this axis

/// - max: the maximum value in the data object for this axis

@objc open func computeAxis(min: Double, max: Double, inverted: Bool)

{

```

```

var min = min, max = max

if let transformer = self.transformer

{

    // calculate the starting and entry point of the y-labels (depending on zoom /
contentrect bounds)

    if viewPortHandler.contentWidth > 10.0
    && !viewPortHandler.isFullyZoomedOutY

    {

        let p1 = transformer.valueForTouchPoint(CGPoint(x:
viewPortHandler.contentLeft, y: viewPortHandler.contentTop))

        let p2 = transformer.valueForTouchPoint(CGPoint(x:
viewPortHandler.contentLeft, y: viewPortHandler.contentBottom))

        if !inverted

        {

            min = Double(p2.y)

            max = Double(p1.y)

        }

```

```

        else

        {

            min = Double(p1.y)

            max = Double(p2.y)

        }

    }

}

computeAxisValues(min: min, max: max)

}

```

/// Sets up the axis values. Computes the desired number of labels between the two given extremes.

```

@objc open func computeAxisValues(min: Double, max: Double)

{

    guard let axis = self.axis else { return }

    let yMin = min

```



```
let yMax = max
```

```
let labelCount = axis.labelCount
```

```
let range = abs(yMax - yMin)
```

```
if labelCount == 0 || range <= 0 || range.isInfinite
```

```
{
```

```
    axis.entries = [Double]()
```

```
    axis.centeredEntries = [Double]()
```

```
    return
```

```
}
```

```
// Find out how much spacing (in y value space) between axis values
```

```
let rawInterval = range / Double(labelCount)
```

```
var interval = rawInterval.roundedToNextSignificant()
```

```
// If granularity is enabled, then do not allow the interval to go below specified  
granularity.
```

```

// This is used to avoid repeated values when rounding values for display.

if axis.granularityEnabled

{

    interval = interval < axis.granularity ? axis.granularity : interval

}


// Normalize interval

let intervalMagnitude = pow(10.0,
Double(Int(log10(interval))))).roundedToNextSignificant()

let intervalSigDigit = Int(interval / intervalMagnitude)

if intervalSigDigit > 5

{

    // Use one order of magnitude higher, to avoid intervals like 0.9 or 90

    // if it's 0.0 after floor(), we use the old value

    interval = floor(10.0 * intervalMagnitude) == 0.0 ? interval : floor(10.0 *
intervalMagnitude)

}

```

```

var n = axis.centerAxisLabelsEnabled ? 1 : 0

// force label count

if axis.isForceLabelsEnabled

{

    interval = Double(range) / Double(labelCount - 1)

    // Ensure stops contains at least n elements.

    axis.entries.removeAll(keepingCapacity: true)

    axis.entries.reserveCapacity(labelCount)

    var v = yMin

    for _ in 0 ..< labelCount

    {

        axis.entries.append(v)

        v += interval
    }
}

```

```

    }

    n = labelCount

}

else

{

    // no forced count

    var first = interval == 0.0 ? 0.0 : ceil(yMin / interval) * interval

    if axis.centerAxisLabelsEnabled

    {

        first -= interval

    }

    let last = interval == 0.0 ? 0.0 : (floor(yMax / interval) * interval).nextUp

```

```

if interval != 0.0 && last != first

{

    for _ in stride(from: first, through: last, by: interval)

    {

        n += 1

    }

}

else if last == first && n == 0

{

    n = 1

}


// Ensure stops contains at least n elements.

axis.entries.removeAll(keepingCapacity: true)

axis.entries.reserveCapacity(labelCount)


var f = first

```

```

var i = 0

while i < n

{

    if f == 0.0

    {

        // Fix for IEEE negative zero case (Where value == -0.0, and 0.0 == -0.0)

        f = 0.0

    }

    axis.entries.append(Double(f))

    f += interval

    i += 1

}

// set decimals

```

```

if interval < 1

{

    axis.decimals = Int(ceil(-log10(interval)))

}

else

{

    axis.decimals = 0

}


if axis.centerAxisLabelsEnabled

{

    axis.centeredEntries.reserveCapacity(n)

    axis.centeredEntries.removeAll()

    let offset: Double = interval / 2.0

    for i in 0 ..< n

```

```

        {

            axis.centeredEntries.append(axis.entries[i] + offset)

        }

    }

}

}

//

// BarChartData.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

```



```
import Foundation
```

```
import CoreGraphics
```

```
open class BarChartData: BarLineScatterCandleBubbleChartData
```

```
{
```

```
    public override init()
```

```
    {
```

```
        super.init()
```

```
    }
```

```
    public override init(dataSets: [IChartDataSet]?)
```

```
    {
```

```
        super.init(dataSets: dataSets)
```

```
    }
```

```
    /// The width of the bars on the x-axis, in values (not pixels)
```

```
    ///
```

```
/// **default**: 0.85
```

```
@objc open var barWidth = Double(0.85)
```

/// Groups all BarDataSet objects this data object holds together by modifying the x-value of their entries.

/// Previously set x-values of entries will be overwritten. Leaves space between bars and groups as specified by the parameters.

/// Do not forget to call notifyDataSetChanged() on your BarChart object after calling this method.

```
///
```

```
/// - Parameters:
```

```
/// - fromX: the starting point on the x-axis where the grouping should begin
```

```
/// - groupSpace: The space between groups of bars in values (not pixels) e.g. 0.8f for bar width 1f
```

```
/// - barSpace: The space between individual bars in values (not pixels) e.g. 0.1f for bar width 1f
```

```
@objc open func groupBars(fromX: Double, groupSpace: Double, barSpace: Double)
```

```
{
```

```
    let setCount = _dataSets.count
```

```
    if setCount <= 1
```

```

{

    print("BarData needs to hold at least 2 BarDataSets to allow grouping.", terminator:
"\n")

    return

}

```

```

let max = maxEntryCountSet

```

```

let maxEntryCount = max?.entryCount ?? 0

```

```

let groupSpaceWidthHalf = groupSpace / 2.0

```

```

let barSpaceHalf = barSpace / 2.0

```

```

let barWidthHalf = self.barWidth / 2.0

```

```

var fromX = fromX

```

```

let interval = groupWidth(groupSpace: groupSpace, barSpace: barSpace)

```

```

for i in stride(from: 0, to: maxEntryCount, by: 1)

```

```

{

    let start = fromX

    fromX += groupSpaceWidthHalf

    (_dataSets as? [IBarChartDataSet])?.forEach { set in

        fromX += barSpaceHalf

        fromX += barWidthHalf

        if i < set.entryCount

        {

            if let entry = set.entryForIndex(i)

            {

                entry.x = fromX

            }

        }

        fromX += barWidthHalf

```

```

        fromX += barSpaceHalf

    }

    fromX += groupSpaceWidthHalf

    let end = fromX

    let innerInterval = end - start

    let diff = interval - innerInterval

    // correct rounding errors

    if diff > 0 || diff < 0

    {

        fromX += diff

    }

}

notifyDataChanged()

```

```
}
```

/// In case of grouped bars, this method returns the space an individual group of bar needs on the x-axis.

```
///
```

```
/// - Parameters:
```

```
/// - groupSpace:
```

```
/// - barSpace:
```

```
@objc open func groupWidth(groupSpace: Double, barSpace: Double) -> Double
```

```
{
```

```
    return Double(_dataSets.count) * (self.barWidth + barSpace) + groupSpace
```

```
}
```

```
}
```

```
//
```

```
// BarChartDataEntry.swift
```

```
// Charts
```

```
//
```

```
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```
// A port of MPAndroidChart for iOS
```

```
// Licensed under Apache License 2.0
```

```
//
```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
open class BarChartDataEntry: ChartDataEntry
```

```
{
```

```
    /// the values the stacked barchart holds
```

```
    private var _yVals: [Double]?
```

```
    /// the ranges for the individual stack values - automatically calculated
```

```
    private var _ranges: [Range]?
```

```
/// the sum of all negative values this entry (if stacked) contains
```

```
private var _negativeSum: Double = 0.0
```

```
/// the sum of all positive values this entry (if stacked) contains
```

```
private var _positiveSum: Double = 0.0
```

```
public required init()
```

```
{
```

```
    super.init()
```

```
}
```

```
/// Constructor for normal bars (not stacked).
```

```
public override init(x: Double, y: Double)
```

```
{
```

```
    super.init(x: x, y: y)
```

```
}
```



```
/// Constructor for normal bars (not stacked).
```

```
public convenience init(x: Double, y: Double, data: Any?)
```

```
{
```

```
    self.init(x: x, y: y)
```

```
    self.data = data
```

```
}
```

```
/// Constructor for normal bars (not stacked).
```

```
public convenience init(x: Double, y: Double, icon: NSUIImage?)
```

```
{
```

```
    self.init(x: x, y: y)
```

```
    self.icon = icon
```

```
}
```

```
/// Constructor for normal bars (not stacked).
```

```
public convenience init(x: Double, y: Double, icon: NSUIImage?, data: Any?)
```

```
{
```

```

        self.init(x: x, y: y)

        self.icon = icon

        self.data = data
    }

```

/// Constructor for stacked bar entries.

```

@objc public init(x: Double, yValues: [Double])

{

    super.init(x: x, y: BarChartDataEntry.calcSum(values: yValues))

    self._yVals = yValues

    calcPosNegSum()

    calcRanges()

}

```

/// Constructor for stacked bar entries. One data object for whole stack

```

@objc public convenience init(x: Double, yValues: [Double], icon: NSUIImage?)

{

```

```

        self.init(x: x, yValues: yValues)

        self.icon = icon

    }

```

/// Constructor for stacked bar entries. One data object for whole stack

```

@objc public convenience init(x: Double, yValues: [Double], data: Any?)

{

    self.init(x: x, yValues: yValues)

    self.data = data

}

```

/// Constructor for stacked bar entries. One data object for whole stack

```

@objc public convenience init(x: Double, yValues: [Double], icon: NSUIImage?, data:
Any?)

{

    self.init(x: x, yValues: yValues)

    self.icon = icon

    self.data = data

```

```
}
```

```
@objc open func sumBelow(stackIndex :Int) -> Double
```

```
{
```

```
    guard let yVals = _yVals else
```

```
    {
```

```
        return 0
```

```
    }
```

```
    var remainder: Double = 0.0
```

```
    var index = yVals.count - 1
```

```
    while (index > stackIndex && index >= 0)
```

```
    {
```

```
        remainder += yVals[index]
```

```
        index -= 1
```

```
    }
```

```
return remainder
```

```
}
```

```
/// The sum of all negative values this entry (if stacked) contains. (this is a positive  
number)
```

```
@objc open var negativeSum: Double
```

```
{
```

```
    return _negativeSum
```

```
}
```

```
/// The sum of all positive values this entry (if stacked) contains.
```

```
@objc open var positiveSum: Double
```

```
{
```

```
    return _positiveSum
```

```
}
```

```
@objc open func calcPosNegSum()
```

```

{
    (_negativeSum, _positiveSum) = _yVals?.reduce(into: (0,0)) { (result, y) in
        if y < 0
        {
            result.0 += -y
        }
        else
        {
            result.1 += y
        }
    } ?? (0,0)
}

```

/// Splits up the stack-values of the given bar-entry into Range objects.

///

/// - Parameters:

/// - entry:

/// - Returns:

```
@objc open func calcRanges()
```

```
{
```

```
    guard let values = yValues, !values.isEmpty else { return }
```

```
    if _ranges == nil
```

```
    {
```

```
        _ranges = [Range]()
```

```
    }
```

```
    else
```

```
    {
```

```
        _ranges!.removeAll()
```

```
    }
```

```
    _ranges!.reserveCapacity(values.count)
```

```
    var negRemain = -negativeSum
```

```

var posRemain: Double = 0.0

for value in values
{
    if value < 0
    {
        _ranges!.append(Range(from: negRemain, to: negRemain - value))

        negRemain -= value
    }

    else
    {
        _ranges!.append(Range(from: posRemain, to: posRemain + value))

        posRemain += value
    }
}
}

```



```
// MARK: Accessors
```

```
/// the values the stacked barchart holds
```

```
@objc open var isStacked: Bool { return _yVals != nil }
```

```
/// the values the stacked barchart holds
```

```
@objc open var yValues: [Double]?
```

```
{
```

```
    get { return self._yVals }
```

```
    set
```

```
    {
```

```
        self.y = BarChartDataEntry.calcSum(values: newValue)
```

```
        self._yVals = newValue
```

```
        calcPosNegSum()
```

```
        calcRanges()
```

```
    }
```

```
}
```

```
/// The ranges of the individual stack-entries. Will return null if this entry is not stacked.
```

```
@objc open var ranges: [Range]?
```

```
{  
  
    return _ranges  
  
}
```

```
// MARK: NSCopying
```

```
open override func copy(with zone: NSZone? = nil) -> Any
```

```
{  
  
    let copy = super.copy(with: zone) as! BarChartDataEntry  
  
    copy._yVals = _yVals  
  
    copy.y = y  
  
    copy._negativeSum = _negativeSum  
  
    copy._positiveSum = _positiveSum  
  
    return copy  
}
```

```
}
```

```
/// Calculates the sum across all values of the given stack.
```

```
///
```

```
/// - Parameters:
```

```
/// - vals:
```

```
/// - Returns:
```

```
private static func calcSum(values: [Double]?) -> Double
```

```
{
```

```
    guard let values = values
```

```
    else { return 0.0 }
```

```
    var sum = 0.0
```

```
    for f in values
```

```
    {
```

```
        sum += f
```

```

    }

    return sum

}

}

//

// BarChartDataProvider.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

```

```
import CoreGraphics
```

```
@objc
```

```
public protocol BarChartDataProvider: BarLineScatterCandleBubbleChartDataProvider
```

```
{
```

```
    var barData: BarChartData? { get }
```

```
    var isDrawBarShadowEnabled: Bool { get }
```

```
    var isDrawValueAboveBarEnabled: Bool { get }
```

```
    var isHighlightFullBarEnabled: Bool { get }
```

```
}//
```

```
// BarChartDataSet.swift
```

```
// Charts
```

```
//
```

```
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```
// A port of MPAndroidChart for iOS
```

```
// Licensed under Apache License 2.0
```

```

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics


open class BarChartDataSet: BarLineScatterCandleBubbleChartDataSet,
IBarChartDataSet

{

    private func initialize()

    {

        self.highlightColor = NSUIColor.black


        self.calcStackSize(entries: entries as! [BarChartDataEntry])

        self.calcEntryCountIncludingStacks(entries: entries as! [BarChartDataEntry])

    }

```

```
public required init()
```

```
{
```

```
    super.init()
```

```
    initialize()
```

```
}
```

```
public override init(entries: [ChartDataEntry]?, label: String?)
```

```
{
```

```
    super.init(entries: entries, label: label)
```

```
    initialize()
```

```
}
```

```
// MARK: - Data functions and accessors
```

```
/// the maximum number of bars that are stacked upon each other, this value
```

```
/// is calculated from the Entries that are added to the DataSet
```

```

private var _stackSize = 1

/// the overall entry count, including counting each stack-value individually

private var _entryCountStacks = 0

/// Calculates the total number of entries this DataSet represents, including
/// stacks. All values belonging to a stack are calculated separately.

private func calcEntryCountIncludingStacks(entries: [BarChartDataEntry])

{

    _entryCountStacks = 0

    for i in 0 ..< entries.count

    {

        if let vals = entries[i].yValues

        {

            _entryCountStacks += vals.count

        }

    }

}

```



```

else

    {

        _entryCountStacks += 1

    }

}

}

```

/// calculates the maximum stacksize that occurs in the Entries array of this DataSet

```
private func calcStackSize(entries: [BarChartDataEntry])
```

```

{

    for i in 0 ..< entries.count

    {

        if let vals = entries[i].yValues

        {

            if vals.count > _stackSize

            {

                _stackSize = vals.count
            }
        }
    }
}

```

```

    }

    }

}

}

```

```

open override fun calcMinMax(entry e: ChartDataEntry)

```

```

{

    guard let e = e as? BarChartDataEntry

        else { return }

```

```

    if !e.y.isNaN

```

```

    {

        if e.yValues == nil

            {

                if e.y < _yMin

                    {

                        _yMin = e.y

```

```
}
```

```
if e.y > _yMax
```

```
{
```

```
    _yMax = e.y
```

```
}
```

```
}
```

```
else
```

```
{
```

```
    if -e.negativeSum < _yMin
```

```
    {
```

```
        _yMin = -e.negativeSum
```

```
    }
```

```
    if e.positiveSum > _yMax
```

```
    {
```

```
        _yMax = e.positiveSum
```

```

    }

}

calcMinMaxX(entry: e)

}

}

/// The maximum number of bars that can be stacked upon another in this DataSet.

open var stackSize: Int

{

    return _stackSize

}

/// `true` if this DataSet is stacked (stacksize > 1) or not.

open var isStacked: Bool

{

    return _stackSize > 1 ? true : false

```

```
}
```

```
/// The overall entry count, including counting each stack-value individually
```

```
@objc open var entryCountStacks: Int
```

```
{
```

```
    return _entryCountStacks
```

```
}
```

```
/// array of labels used to describe the different values of the stacked bars
```

```
open var stackLabels: [String] = []
```

```
// MARK: - Styling functions and accessors
```

```
/// the color used for drawing the bar-shadows. The bar shadows is a surface behind the  
bar that indicates the maximum value
```

```
open var barShadowColor = NSUIColor(red: 215.0/255.0, green: 215.0/255.0, blue:  
215.0/255.0, alpha: 1.0)
```

/// the width used for drawing borders around the bars. If borderWidth == 0, no border will be drawn.

```
open var barBorderWidth : CGFloat = 0.0
```

/// the color drawing borders around the bars.

```
open var barBorderColor = NSUIColor.black
```

/// the alpha value (transparency) that is used for drawing the highlight indicator bar. min = 0.0 (fully transparent), max = 1.0 (fully opaque)

```
open var highlightAlpha = CGFloat(120.0 / 255.0)
```

```
// MARK: - NSCopying
```

```
open override func copy(with zone: NSZone? = nil) -> Any
```

```
{
```

```
    let copy = super.copy(with: zone) as! BarChartDataSet
```

```
    copy._stackSize = _stackSize
```

```
    copy._entryCountStacks = _entryCountStacks
```

```

        copy.stackLabels = stackLabels

        copy.barShadowColor = barShadowColor

        copy.barBorderWidth = barBorderWidth

        copy.barBorderColor = barBorderColor

        copy.highlightAlpha = highlightAlpha

        return copy

    }

}

//

// BarChartRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
#if canImport(Uikit)
```

```
    import UIKit
```

```
#endif
```

```
#if canImport(Cocoa)
```

```
import Cocoa
```

```
#endif
```

```
open class BarChartRenderer: BarLineScatterCandleBubbleRenderer
```

```
{
```

```
    /// A nested array of elements ordered logically (i.e not in visual/drawing order) for use  
    with VoiceOver
```


///

/// Its use is apparent when there are multiple data sets, since we want to read bars in left to right order,

/// irrespective of dataset. However, drawing is done per dataset, so using this array and then flattening it prevents us from needing to

/// re-render for the sake of accessibility.

///

/// In practise, its structure is:

///

/// ```

/// [

/// [dataset1 element1, dataset2 element1],

/// [dataset1 element2, dataset2 element2],

/// [dataset1 element3, dataset2 element3]

/// ...

///]

/// ```

/// This is done to provide numerical inference across datasets to a screenreader user, in the same way that a sighted individual

```

    /// uses a multi-dataset bar chart.

    ///

    /// The ``internal`` specifier is to allow subclasses (HorizontalBar) to populate the
    same array

    internal lazy var accessibilityOrderedElements: [[NSUIAccessibilityElement]] =
    accessibilityCreateEmptyOrderedElements()


    private class Buffer

    {

        var rects = [CGRect]()

    }


    @objc open weak var dataProvider: BarChartDataProvider?


    @objc public init(dataProvider: BarChartDataProvider, animator: Animator,
    viewPortHandler: ViewPortHandler)

    {

        super.init(animator: animator, viewPortHandler: viewPortHandler)
    }

```

```

        self.dataProvider = dataProvider
    }

    // [CGRect] per dataset

    private var _buffers = [Buffer]()

    open override func initBuffers()
    {
        if let barData = dataProvider?.barData
        {
            // Matche buffers count to dataset count

            if _buffers.count != barData.dataSetCount
            {
                while _buffers.count < barData.dataSetCount
                {
                    _buffers.append(Buffer())
                }
            }
        }
    }

```

```

while _buffers.count > barData.dataSetCount

{

    _buffers.removeLast()

}

}

for i in stride(from: 0, to: barData.dataSetCount, by: 1)

{

    let set = barData.dataSets[i] as! IBarChartDataSet

    let size = set.entryCount * (set.isStacked ? set.stackSize : 1)

    if _buffers[i].rects.count != size

    {

        _buffers[i].rects = [CGRect](repeating: CGRect(), count: size)

    }

}

}

else

```

```

    {

        _buffers.removeAll()

    }

}

```

```
private func prepareBuffer(dataSet: IBarChartDataSet, index: Int)
```

```

{

    guard

        let dataProvider = dataProvider,

        let barData = dataProvider.barData

        else { return }


    let barWidthHalf = barData.barWidth / 2.0


    let buffer = _buffers[index]


    var bufferIndex = 0


    let containsStacks = dataSet.isStacked

```

```

let isInverted = dataProvider.isInverted(axis: dataSet.axisDependency)

let phaseY = animator.phaseY

var barRect = CGRect()

var x: Double

var y: Double


    for i in stride(from: 0, to: min(Int(ceil(Double(dataSet.entryCount) *
animator.phaseX)), dataSet.entryCount), by: 1)

    {

        guard let e = dataSet.entryForIndex(i) as? BarChartDataEntry else { continue }

        let vals = e.yValues


        x = e.x

        y = e.y

```

```

if !containsStacks || vals == nil

{

    let left = CGFloat(x - barWidthHalf)

    let right = CGFloat(x + barWidthHalf)

    var top = isInverted

        ? (y <= 0.0 ? CGFloat(y) : 0)

        : (y >= 0.0 ? CGFloat(y) : 0)

    var bottom = isInverted

        ? (y >= 0.0 ? CGFloat(y) : 0)

        : (y <= 0.0 ? CGFloat(y) : 0)

```

/* When drawing each bar, the renderer actually draws each bar from 0 to the required value.

* This drawn bar is then clipped to the visible chart rect in BarLineChartViewBase's draw(rect:) using clipDataToContent.

* While this works fine when calculating the bar rects for drawing, it causes the accessibilityFrames to be oversized in some cases.

* This offset attempts to undo that unnecessary drawing when calculating barRects


```

* |                                     |
* +-----+-----+-----+
-----+

* |      Situation 2: (!inverted && y < 0)          |      Situation 4: (inverted
&& y < 0)          |

* |                                     |
* |      0 -> ---+--+---+--+----- <- top          |      y ->      +--+      <-
top          |

* |          | | | //|          } topOffset = -max      |          //|          }
topOffset = min - y    |

* |      max -> +--+---+--+-----+ <- top - topOffset      |      min -> +-----
+--+-----+ <- top + topOffset      |

* |          | | | //| |          |          | +--+ //| |          |
* |          | +--+ //| |          |          | | | //| |          |
* |          | //| |          |          | | | //| |          |

* |      min -> +-----+--+-----+ <- bottom + bottomOffset |      max -> +--+
+-----+--+-----+ <- bottom - bottomOffset |

* |          //|          } bottomOffset = min - y |          | | //|          }
bottomOffset = -max    |

* |      y ->      +--+      <- bottom          |      0 -> ---+--+---+--+-----
<- bottom          |

```

$$\frac{\begin{array}{c} * | \qquad \qquad \qquad | \qquad \qquad \qquad | \\ * +-----+-----+ \\ +-----+ \end{array}}{+}$$

*/

var topOffset: CGFloat = 0.0

var bottomOffset: CGFloat = 0.0

if let offsetView = dataProvider as? BarChartView

{

let offsetAxis = offsetView.GetAxis(dataSet.axisDependency)

if y >= 0

{

// situation 1

if offsetAxis.axisMaximum < y

{

topOffset = CGFloat(y - offsetAxis.axisMaximum)

}

if offsetAxis.axisMinimum > 0

{

```

        bottomOffset = CGFloat(offsetAxis.axisMinimum)

    }

}

else // y < 0

{

    //situation 2

    if offsetAxis.axisMaximum < 0

    {

        topOffset = CGFloat(offsetAxis.axisMaximum * -1)

    }

    if offsetAxis.axisMinimum > y

    {

        bottomOffset = CGFloat(offsetAxis.axisMinimum - y)

    }

}

if isInverted

{

```

```

        // situation 3 and 4

        // exchange topOffset/bottomOffset based on 1 and 2

        // see diagram above

        (topOffset, bottomOffset) = (bottomOffset, topOffset)

    }

}

//apply offset

top = isInverted ? top + topOffset : top - topOffset

bottom = isInverted ? bottom - bottomOffset : bottom + bottomOffset


// multiply the height of the rect with the phase

// explicitly add 0 + topOffset to indicate this is changed after adding accessibility
support (#3650, #3520)

    if top > 0 + topOffset

    {

        top *= CGFloat(phaseY)

    }

    else

```

```

{

    bottom *= CGFloat(phaseY)

}


barRect.origin.x = left

barRect.origin.y = top

barRect.size.width = right - left

barRect.size.height = bottom - top

buffer.rects[bufferIndex] = barRect

bufferIndex += 1

}

else

{

    var posY = 0.0

    var negY = -e.negativeSum

    var yStart = 0.0

```

```

// fill the stack

for k in 0 ..< vals!.count

{

    let value = vals![k]


    if value == 0.0 && (posY == 0.0 || negY == 0.0)

    {

        // Take care of the situation of a 0.0 value, which overlaps a non-zero bar

        y = value

        yStart = y

    }

    else if value >= 0.0

    {

        y = posY

        yStart = posY + value

        posY = yStart

    }

```

```

else

{

    y = negY

    yStart = negY + abs(value)

    negY += abs(value)

}


let left = CGFloat(x - barWidthHalf)

let right = CGFloat(x + barWidthHalf)

var top = isInverted

    ? (y <= yStart ? CGFloat(y) : CGFloat(yStart))

    : (y >= yStart ? CGFloat(y) : CGFloat(yStart))

var bottom = isInverted

    ? (y >= yStart ? CGFloat(y) : CGFloat(yStart))

    : (y <= yStart ? CGFloat(y) : CGFloat(yStart))


// multiply the height of the rect with the phase

```

```
top *= CGFloat(phaseY)
```

```
bottom *= CGFloat(phaseY)
```

```
barRect.origin.x = left
```

```
barRect.size.width = right - left
```

```
barRect.origin.y = top
```

```
barRect.size.height = bottom - top
```

```
buffer.rects[bufferIndex] = barRect
```

```
bufferIndex += 1
```

```
}
```

```
}
```

```
}
```

```
}
```

```
open override func drawData(context: CGContext)
```

```
{
```



```
guard
```

```
    let dataProvider = dataProvider,
```

```
    let barData = dataProvider.barData
```

```
    else { return }
```

```
    // If we redraw the data, remove and repopulate accessible elements to update label  
    values and frames
```

```
    accessibleChartElements.removeAll()
```

```
    accessibilityOrderedElements = accessibilityCreateEmptyOrderedElements()
```

```
    // Make the chart header the first element in the accessible elements array
```

```
    if let chart = dataProvider as? BarChartView {
```

```
        let element = createAccessibleHeader(usingChart: chart,
```

```
            andData: barData,
```

```
            withDefaultDescription: "Bar Chart")
```

```
        accessibleChartElements.append(element)
```

```
    }
```

```

        // Populate logically ordered nested elements into accessibilityOrderedElements in
drawDataSet()

        for i in 0 ..< barData.dataSetCount

        {

            guard let set = barData.getDataSetByIndex(i) else { continue }

            if set.isVisible

            {

                if !(set is IBarChartDataSet)

                {

                    fatalError("Datasets for BarChartRenderer must conform to
IBarChartDataSet")

                }

                drawDataSet(context: context, dataSet: set as! IBarChartDataSet, index: i)

            }

        }

```

```

        // Merge nested ordered arrays into the single accessibleChartElements.

        accessibleChartElements.append(contentsOf: accessibilityOrderedElements.flatMap
{ $0 } )

        accessibilityPostLayoutChangedNotification()

    }

```

```

private var _barShadowRectBuffer: CGRect = CGRect()

```

```

@objc open func drawDataSet(context: CGContext, dataSet: IBarChartDataSet, index:
Int)

```

```

{

    guard let dataProvider = dataProvider else { return }

```

```

    let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)

```

```

    prepareBuffer(dataSet: dataSet, index: index)

```

```

    trans.rectValuesToPixel(&_buffers[index].rects)

```

```

let borderWidth = dataSet.barBorderWidth

let borderColor = dataSet.barBorderColor

let drawBorder = borderWidth > 0.0


context.saveGState()


// draw the bar shadow before the values

if dataProvider.isDrawBarShadowEnabled

{

    guard let barData = dataProvider.barData else { return }


    let barWidth = barData.barWidth

    let barWidthHalf = barWidth / 2.0

    var x: Double = 0.0


    for i in stride(from: 0, to: min(Int(ceil(Double(dataSet.entryCount) *
    animator.phaseX)), dataSet.entryCount), by: 1)

    {

```

```
guard let e = dataSet.entryForIndex(i) as? BarChartDataEntry else { continue }
```

```
x = e.x
```

```
_barShadowRectBuffer.origin.x = CGFloat(x - barWidthHalf)
```

```
_barShadowRectBuffer.size.width = CGFloat(barWidth)
```

```
trans.rectValueToPixel(&_barShadowRectBuffer)
```

```
if !viewPortHandler.isInBoundsLeft(_barShadowRectBuffer.origin.x +  
_barShadowRectBuffer.size.width)
```

```
{
```

```
    continue
```

```
}
```

```
if !viewPortHandler.isInBoundsRight(_barShadowRectBuffer.origin.x)
```

```
{
```

```
    break
```

```
}
```

```
_barShadowRectBuffer.origin.y = viewPortHandler.contentTop
```

```
_barShadowRectBuffer.size.height = viewPortHandler.contentHeight
```

```
context.setFillColor(dataSet.barShadowColor.cgColor)
```

```
context.fill(_barShadowRectBuffer)
```

```
}
```

```
}
```

```
let buffer = _buffers[index]
```

```
// draw the bar shadow before the values
```

```
if dataProvider.isDrawBarShadowEnabled
```

```
{
```

```
for j in stride(from: 0, to: buffer.rects.count, by: 1)
```

```
{
```

```
let barRect = buffer.rects[j]

if (!viewPortHandler.isInBoundsLeft(barRect.origin.x + barRect.size.width))

{

    continue

}

if (!viewPortHandler.isInBoundsRight(barRect.origin.x))

{

    break

}

context.setFillColor(dataSet.barShadowColor.cgColor)

context.fill(barRect)

}

}
```

```
let isSingleColor = dataSet.colors.count == 1
```

```
if isSingleColor
```

```
{
```

```
    context.setFillColors(dataSet.colors[0].cgColors)
```

```
}
```

```
// In case the chart is stacked, we need to accomodate individual bars within  
accessibilityOrderedElements
```

```
let isStacked = dataSet.isStacked
```

```
let stackSize = isStacked ? dataSet.stackSize : 1
```

```
for j in stride(from: 0, to: buffer.rects.count, by: 1)
```

```
{
```

```
    let barRect = buffer.rects[j]
```

```
    if (!viewPortHandler.isInBoundsLeft(barRect.origin.x + barRect.size.width))
```

```
{
```



```

        continue
    }

    if (!viewportHandler.isInBoundsRight(barRect.origin.x))
    {
        break
    }

    if !isSingleColor
    {
        // Set the color for the currently drawn value. If the index is out of bounds, reuse
colors.

        context.setFill_color(datasets.colors.at(index: j).color)
    }

    context.fill(barRect)

    if drawBorder

```

```

{

    context.setStrokeColor(borderColor.cgColor)

    context.setLineWidth(borderWidth)

    context.stroke(barRect)

}

```

```

// Create and append the corresponding accessibility element to
accessibilityOrderedElements

```

```

if let chart = dataProvider as? BarChartView

{

    let element = createAccessibleElement(withIndex: j,

                                         container: chart,

                                         dataSet: dataSet,

                                         dataSetIndex: index,

                                         stackSize: stackSize)

    { (element) in

        element.accessibilityFrame = barRect

    }

}

```

```

        accessibilityOrderedElements[j/stackSize].append(element)

    }

}

context.restoreGState()

}

open func prepareBarHighlight(

    x: Double,

    y1: Double,

    y2: Double,

    barWidthHalf: Double,

    trans: Transformer,

    rect: inout CGRect)

{

    let left = x - barWidthHalf

```

```
let right = x + barWidthHalf
```

```
let top = y1
```

```
let bottom = y2
```

```
rect.origin.x = CGFloat(left)
```

```
rect.origin.y = CGFloat(top)
```

```
rect.size.width = CGFloat(right - left)
```

```
rect.size.height = CGFloat(bottom - top)
```

```
trans.rectValueToPixel(&rect, phaseY: animator.phaseY )
```

```
}
```

```
open override func drawValues(context: CGContext)
```

```
{
```

```
    // if values are drawn
```

```
    if isDrawingValuesAllowed(dataProvider: dataProvider)
```

```
    {
```

guard

let dataProvider = dataProvider,

let barData = dataProvider.barData

else { return }

let dataSets = barData.dataSets

let valueOffsetPlus: CGFloat = 4.5

var posOffset: CGFloat

var negOffset: CGFloat

let drawValueAboveBar = dataProvider.isDrawValueAboveBarEnabled

for dataSetIndex in 0 ..< barData.dataSetCount

{

guard let

dataSet = dataSets[dataSetIndex] as? IBarChartDataSet,

shouldDrawValues(forDataSet: dataSet)

```
else { continue }
```

```
let isInverted = dataProvider.isInverted(axis: dataSet.axisDependency)
```

```
// calculate the correct offset depending on the draw position of the value
```

```
let valueFont = dataSet.valueFont
```

```
let valueTextHeight = valueFont.lineHeight
```

```
posOffset = (drawValueAboveBar ? -(valueTextHeight + valueOffsetPlus) :  
valueOffsetPlus)
```

```
negOffset = (drawValueAboveBar ? valueOffsetPlus : -(valueTextHeight +  
valueOffsetPlus))
```

```
if isInverted
```

```
{
```

```
    posOffset = -posOffset - valueTextHeight
```

```
    negOffset = -negOffset - valueTextHeight
```

```
}
```

```

let buffer = _buffers[dataSetIndex]

guard let formatter = dataSet.valueFormatter else { continue }

let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)

let phaseY = animator.phaseY

let iconsOffset = dataSet.iconsOffset

// if only single values are drawn (sum)

if !dataSet.isStacked

{

    for j in 0 ..< Int(ceil(Double(dataSet.entryCount) * animator.phaseX))

    {

        guard let e = dataSet.entryForIndex(j) as? BarChartDataEntry else
{ continue }

```

```
let rect = buffer.rects[j]
```

```
let x = rect.origin.x + rect.size.width / 2.0
```

```
if !viewPortHandler.isInBoundsRight(x)
```

```
{
```

```
    break
```

```
}
```

```
if !viewPortHandler.isInBoundsY(rect.origin.y)
```

```
    || !viewPortHandler.isInBoundsLeft(x)
```

```
{
```

```
    continue
```

```
}
```

```
let val = e.y
```



```

if dataSet.isDrawValuesEnabled

{

drawValue(

context: context,

value: formatter.stringForValue(

val,

entry: e,

dataSetIndex: dataSetIndex,

viewPortHandler: viewPortHandler),

xPos: x,

yPos: val >= 0.0

? (rect.origin.y + posOffset)

: (rect.origin.y + rect.size.height + negOffset),

font: valueFont,

align: .center,

color: dataSet.valueTextColorAt(j))

}

```

```

if let icon = e.icon, dataSet.isDrawIconsEnabled

{

    var px = x

    var py = val >= 0.0

        ? (rect.origin.y + posOffset)

        : (rect.origin.y + rect.size.height + negOffset)


    px += iconsOffset.x

    py += iconsOffset.y


    ChartUtils.drawImage(

        context: context,

        image: icon,

        x: px,

        y: py,

        size: icon.size)

```

```

        }

    }

}

else

{

    // if we have stacks

    var bufferIndex = 0

    for index in 0 ..< Int(ceil(Double(dataSet.entryCount) * animator.phaseX))

    {

        guard let e = dataSet.entryForIndex(index) as? BarChartDataEntry else
{ continue }

        let vals = e.yValues

        let rect = buffer.rects[bufferIndex]

```

```

let x = rect.origin.x + rect.size.width / 2.0

// we still draw stacked bars, but there is one non-stacked in between

if vals == nil

{

    if !viewPortHandler.isInBoundsRight(x)

    {

        break

    }

    if !viewPortHandler.isInBoundsY(rect.origin.y)

    || !viewPortHandler.isInBoundsLeft(x)

    {

        continue

    }

    if dataSet.isDrawValuesEnabled

```

```

{

drawValue(

    context: context,

    value: formatter.stringForValue(

        e.y,

        entry: e,

        dataSetIndex: dataSetIndex,

        viewPortHandler: viewPortHandler),

    xPos: x,

    yPos: rect.origin.y +

        (e.y >= 0 ? posOffset : negOffset),

    font: valueFont,

    align: .center,

    color: dataSet.valueTextColorAt(index))

}

```

```

if let icon = e.icon, dataSet.isDrawIconsEnabled

```

```

{

    var px = x

    var py = rect.origin.y +

        (e.y >= 0 ? posOffset : negOffset)


    px += iconsOffset.x

    py += iconsOffset.y


    ChartUtils.drawImage(

        context: context,

        image: icon,

        x: px,

        y: py,

        size: icon.size)

    }

}

else

```

```

{

    // draw stack values

    let vals = vals!

    var transformed = [CGPoint]()

    var posY = 0.0

    var negY = -e.negativeSum

    for k in 0 ..< vals.count

    {

        let value = vals[k]

        var y: Double

        if value == 0.0 && (posY == 0.0 || negY == 0.0)

        {

            // Take care of the situation of a 0.0 value, which overlaps a non-
zero bar

```

```

        y = value

    }

    else if value >= 0.0

    {

        posY += value

        y = posY

    }

    else

    {

        y = negY

        negY -= value

    }

    transformed.append(CGPoint(x: 0.0, y: CGFloat(y * phaseY)))

}

trans.pointValuesToPixel(&transformed)

```



```

for k in 0 ..< transformed.count

{

    let val = vals[k]

    let drawBelow = (val == 0.0 && negY == 0.0 && posY > 0.0) || val <
0.0

    let y = transformed[k].y + (drawBelow ? negOffset : posOffset)

    if !viewPortHandler.isInBoundsRight(x)

    {

        break

    }

    if !viewPortHandler.isInBoundsY(y)
|| !viewPortHandler.isInBoundsLeft(x)

    {

        continue

    }

```

```
if dataSet.isDrawValuesEnabled

{

    drawValue(

        context: context,

        value: formatter.stringForValue(

            vals[k],

            entry: e,

            dataSetIndex: dataSetIndex,

            viewPortHandler: viewPortHandler),

        xPos: x,

        yPos: y,

        font: valueFont,

        align: .center,

        color: dataSet.valueTextColorAt(index))

}
```

```

        if let icon = e.icon, dataSet.isDrawIconsEnabled
        {
            ChartUtils.drawImage(
                context: context,
                image: icon,
                x: x + iconsOffset.x,
                y: y + iconsOffset.y,
                size: icon.size)
        }
    }
}

bufferIndex = vals == nil ? (bufferIndex + 1) : (bufferIndex + vals!.count)
}

}

}

}

```

```

    }

    /// Draws a value at the specified x and y position.

    @objc open func drawValue(context: CGContext, value: String, xPos: CGFloat, yPos:
CGFloat, font: NSUIFont, align: NSTextAlignment, color: NSUIColor)

    {

        ChartUtils.drawText(context: context, text: value, point: CGPoint(x: xPos, y: yPos),
align:        align,        attributes:        [NSAttributedString.Key.font:        font,
NSAttributedString.Key.foregroundColor: color])

    }

    open override func drawExtras(context: CGContext)

    {

    }

    open override func drawHighlighted(context: CGContext, indices: [Highlight])

    {

        guard

```

```

        let dataProvider = dataProvider,

        let barData = dataProvider.barData

        else { return }

context.saveGState()

var barRect = CGRect()

for high in indices
{
    guard

        let set = barData.getDataSetByIndex(high.dataSetIndex) as? IBarChartDataSet,

        set.isHighlightEnabled

        else { continue }

    if let e = set.entryForXValue(high.x, closestToY: high.y) as? BarChartDataEntry
    {

```

```
if !isInBoundsX(entry: e, dataSet: set)
```

```
{
```

```
    continue
```

```
}
```

```
let trans = dataProvider.getTransformer(forAxis: set.axisDependency)
```

```
context.setFillColor(set.highlightColor.cgColor)
```

```
context.setAlpha(set.highlightAlpha)
```

```
let isStack = high.stackIndex >= 0 && e.isStacked
```

```
let y1: Double
```

```
let y2: Double
```

```
if isStack
```

```
{
```

```

if dataProvider.isHighlightFullBarEnabled

{

    y1 = e.positiveSum

    y2 = -e.negativeSum

}

else

{

    let range = e.ranges?[high.stackIndex]

    y1 = range?.from ?? 0.0

    y2 = range?.to ?? 0.0

}

}

else

{

    y1 = e.y

    y2 = 0.0

```

```
}
```

```
        prepareBarHighlight(x: e.x, y1: y1, y2: y2, barWidthHalf: barData.barWidth /  
2.0, trans: trans, rect: &barRect)
```

```
        setHighlightDrawPos(highlight: high, barRect: barRect)
```

```
        context.fill(barRect)
```

```
    }
```

```
}
```

```
context.restoreGState()
```

```
}
```

```
/// Sets the drawing position of the highlight object based on the given bar-rect.
```

```
internal fun setHighlightDrawPos(highlight high: Highlight, barRect: CGRect)
```

```
{
```

```
    high.setDraw(x: barRect.midX, y: barRect.origin.y)
```



```

    }

    /// Creates a nested array of empty subarrays each of which will be populated with
    NSUIAccessibilityElements.

    /// This is marked internal to support HorizontalBarChartRenderer as well.

    internal      func      accessibilityCreateEmptyOrderedElements()      ->
    [[NSUIAccessibilityElement]]

    {

        guard let chart = dataProvider as? BarChartView else { return [] }

        // Unlike Bubble & Line charts, here we use the maximum entry count to account for
        stacked bars

        let maxEntryCount = chart.data?.maxEntryCountSet?.entryCount ?? 0

        return Array(repeating: [NSUIAccessibilityElement](),

            count: maxEntryCount)

    }

```

```
/// Creates an NSUIAccessibleElement representing the smallest meaningful bar of the
chart
```

```
/// i.e. in case of a stacked chart, this returns each stack, not the combined bar.
```

```
/// Note that it is marked internal to support subclass modification in the
HorizontalBarChart.
```

```
internal fun createAccessibleElement(withIndex idx: Int,

                                     container: BarChartView,

                                     dataSet: IBarChartDataSet,

                                     dataSetIndex: Int,

                                     stackSize: Int,

                                     modifier: (NSUIAccessibilityElement) -> () ->
NSUIAccessibilityElement

    {

        let element = NSUIAccessibilityElement(accessibilityContainer: container)

        let xAxis = container.xAxis

        guard let e = dataSet.entryForIndex(idx/stackSize) as? BarChartDataEntry else
{ return element }

        guard let dataProvider = dataProvider else { return element }
```

// NOTE: The formatter can cause issues when the x-axis labels are consecutive ints.

// i.e. due to the Double conversion, if there are more than one data set that are grouped,

// there is the possibility of some labels being rounded up. A floor() might fix this, but seems to be a brute force solution.

```
let label = xAxis.valueFormatter?.stringForValue(e.x, axis: xAxis) ?? "\(e.x)"
```

```
var elementValueText = dataSet.valueFormatter?.stringForValue(
```

```
    e.y,
```

```
    entry: e,
```

```
    dataSetIndex: dataSetIndex,
```

```
    viewPortHandler: viewPortHandler) ?? "\(e.y)"
```

```
if dataSet.isStacked, let vals = e.yValues
```

```
{
```

```
    let labelCount = min(dataSet.colors.count, stackSize)
```

```
    let stackLabel: String?
```

```

if (dataSet.stackLabels.count > 0 && labelCount > 0) {

    let labelIndex = idx % labelCount

    stackLabel = dataSet.stackLabels.indices.contains(labelIndex) ?
dataSet.stackLabels[labelIndex] : nil

} else {

    stackLabel = nil

}

//Handles empty array of yValues

let yValue = vals.isEmpty ? 0.0 : vals[idx % vals.count]

elementValueText = dataSet.valueFormatter?.stringForValue(

    yValue,

    entry: e,

    dataSetIndex: dataSetIndex,

    viewPortHandler: viewPortHandler) ?? "\n(e.y)"

if let stackLabel = stackLabel {

```

```

        elementValueText = stackLabel + " \$(elementValueText)"

    } else {

        elementValueText = "\$(elementValueText)"

    }

}

let dataSetCount = dataProvider.barData?.dataSetCount ?? -1

let doesContainMultipleDataSets = dataSetCount > 1

    element.accessibilityLabel = "\$(doesContainMultipleDataSets ? (dataSet.label ?? "")
+ ", " : "") \$(label): \$(elementValueText)"

    modifier(element)

    return element

}

}

//

```

```

// BarChartView.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

/// Chart that draws bars.

open class BarChartView: BarLineChartViewBase, BarChartDataProvider

{

    /// if set to true, all values are drawn above their bars, instead of below their top

```

```
private var _drawValueAboveBarEnabled = true
```

```
/// if set to true, a grey area is drawn behind each bar that indicates the maximum value
```

```
private var _drawBarShadowEnabled = false
```

```
internal override func initialize()
```

```
{
```

```
    super.initialize()
```

```
    renderer = BarChartRenderer(dataProvider: self, animator: _animator,  
viewPortHandler: _viewPortHandler)
```

```
    self.highlighter = BarHighlighter(chart: self)
```

```
    self.xAxis.spaceMin = 0.5
```

```
    self.xAxis.spaceMax = 0.5
```

```
}
```

```

internal override func calcMinMax()

{

    guard let data = self.data as? BarChartData

        else { return }

    if fitBars

    {

        _xAxis.calculate(

            min: data.xMin - data.barWidth / 2.0,

            max: data.xMax + data.barWidth / 2.0)

    }

    else

    {

        _xAxis.calculate(min: data.xMin, max: data.xMax)

    }

    // calculate axis range (min / max) according to provided data

```



```

leftAxis.calculate(

    min: data.getYMin(axis: .left),

    max: data.getYMax(axis: .left))

rightAxis.calculate(

    min: data.getYMin(axis: .right),

    max: data.getYMax(axis: .right))

}

```

/// - Returns: The Highlight object (contains x-index and DataSet index) of the selected value at the given touch point inside the BarChart.

```

open override func getHighlightByTouchPoint(_ pt: CGPoint) -> Highlight?

{

    if _data === nil

    {

        Swift.print("Can't select by touch. No data set.")

        return nil

    }

}

```

```
guard let h = self.highlighter?.getHighlight(x: pt.x, y: pt.y)
```

```
else { return nil }
```

```
if !isHighlightFullBarEnabled { return h }
```

```
// For isHighlightFullBarEnabled, remove stackIndex
```

```
return Highlight(
```

```
    x: h.x, y: h.y,
```

```
    xPx: h.xPx, yPx: h.yPx,
```

```
    dataIndex: h.dataIndex,
```

```
    dataSetIndex: h.dataSetIndex,
```

```
    stackIndex: -1,
```

```
    axis: h.axis)
```

```
}
```

/// - Returns: The bounding box of the specified Entry in the specified DataSet. Returns null if the Entry could not be found in the charts data.

```
@objc open func getBarBounds(entry e: BarChartDataEntry) -> CGRect
```

```

{

guard let

    data = _data as? BarChartData,

    let set = data.getDataSetForEntry(e) as? IBarChartDataSet

    else { return CGRect.null }


let y = e.y

let x = e.x


let barWidth = data.barWidth


let left = x - barWidth / 2.0

let right = x + barWidth / 2.0

let top = y >= 0.0 ? y : 0.0

let bottom = y <= 0.0 ? y : 0.0


var bounds = CGRect(x: left, y: top, width: right - left, height: bottom - top)

```

```

        getTransformer(forAxis: set.axisDependency).rectValueToPixel(&bounds)

    return bounds

}

/// Groups all BarDataSet objects this data object holds together by modifying the x-
value of their entries.

/// Previously set x-values of entries will be overwritten. Leaves space between bars and
groups as specified by the parameters.

/// Calls `notifyDataSetChanged()` afterwards.

///

/// - Parameters:

/// - fromX: the starting point on the x-axis where the grouping should begin

/// - groupSpace: the space between groups of bars in values (not pixels) e.g. 0.8f for
bar width 1f

/// - barSpace: the space between individual bars in values (not pixels) e.g. 0.1f for bar
width 1f

@objc open func groupBars(fromX: Double, groupSpace: Double, barSpace: Double)

```

```

{

    guard let barData = self.barData

    else

    {

        Swift.print("You need to set data for the chart before grouping bars.", terminator:
"\n")

        return

    }

    barData.groupBars(fromX: fromX, groupSpace: groupSpace, barSpace: barSpace)

    notifyDataSetChanged()

}

/// Highlights the value at the given x-value in the given DataSet. Provide -1 as the
dataSetIndex to undo all highlighting.

///

/// - Parameters:

/// - x:

```

```
/// - dataSetIndex:
```

```
/// - stackIndex: the index inside the stack - only relevant for stacked entries
```

```
@objc open func highlightValue(x: Double, dataSetIndex: Int, stackIndex: Int)
```

```
{
```

```
    highlightValue(Highlight(x: x, dataSetIndex: dataSetIndex, stackIndex: stackIndex))
```

```
}
```

```
// MARK: Accessors
```

```
/// if set to true, all values are drawn above their bars, instead of below their top
```

```
@objc open var drawValueAboveBarEnabled: Bool
```

```
{
```

```
    get { return _drawValueAboveBarEnabled }
```

```
    set
```

```
    {
```

```
        _drawValueAboveBarEnabled = newValue
```

```
        setNeedsDisplay()
```

```
}  
  
}
```

/// if set to true, a grey area is drawn behind each bar that indicates the maximum value

@objc open var drawBarShadowEnabled: Bool

```
{  
  
    get { return _drawBarShadowEnabled }  
  
    set  
  
    {  
  
        _drawBarShadowEnabled = newValue  
  
        setNeedsDisplay()  
  
    }  
  
}
```

/// Adds half of the bar width to each side of the x-axis range in order to allow the bars of the barchart to be fully displayed.

/// **default**: false

@objc open var fitBars = false

/// Set this to `true` to make the highlight operation full-bar oriented, `false` to make it highlight single values (relevant only for stacked).

/// If enabled, highlighting operations will highlight the whole bar, even if only a single stack entry was tapped.

```
@objc open var highlightFullBarEnabled: Bool = false
```

/// `true` the highlight is be full-bar oriented, `false` if single-value

```
open var isHighlightFullBarEnabled: Bool { return highlightFullBarEnabled }
```

// MARK: - BarChartDataProvider

```
open var barData: BarChartData? { return _data as? BarChartData }
```

/// `true` if drawing values above bars is enabled, `false` if not

```
open var isDrawValueAboveBarEnabled: Bool { return drawValueAboveBarEnabled }
```

/// `true` if drawing shadows (maxvalue) for each bar is enabled, `false` if not


```

        open var isDrawBarShadowEnabled: Bool { return drawBarShadowEnabled }

    }

    //

    // BarHighlighter.swift

    // Charts

    //

    // Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

    // A port of MPAndroidChart for iOS

    // Licensed under Apache License 2.0

    //

    // https://github.com/danielgindi/Charts

    //

import Foundation

import CoreGraphics

@objc(BarChartHighlighter)

```

```

open class BarHighlighter: ChartHighlighter

{

    open override fun getHighlight(x: CGFloat, y: CGFloat) -> Highlight?

    {

        guard

            let barData = (self.chart as? BarChartDataProvider)?.barData,

            let high = super.getHighlight(x: x, y: y)

            else { return nil }

        let pos = getValsForTouch(x: x, y: y)

        if let set = barData.getDataSetByIndex(high.dataSetIndex) as? IBarChartDataSet,

            set.isStacked

        {

            return getStackedHighlight(high: high,

                set: set,

                xValue: Double(pos.x),

```

```
yValue: Double(pos.y))
```

```
}
```

```
else
```

```
{
```

```
    return high
```

```
}
```

```
}
```

```
internal override func getDistance(x1: CGFloat, y1: CGFloat, x2: CGFloat, y2: CGFloat)  
-> CGFloat
```

```
{
```

```
    return abs(x1 - x2)
```

```
}
```

```
internal override var data: ChartData?
```

```
{
```

```
    return (chart as? BarChartDataProvider)?.barData
```

```
}
```

/// This method creates the Highlight object that also indicates which value of a stacked BarEntry has been selected.

///

/// - Parameters:

/// - high: the Highlight to work with looking for stacked values

/// - set:

/// - xIndex:

/// - yValue:

/// - Returns:

@objc open func getStackedHighlight(high: Highlight,

set: IBarChartDataSet,

xValue: Double,

yValue: Double) -> Highlight?

{

guard

let chart = self.chart as? BarLineScatterCandleBubbleChartDataProvider,

```
        let entry = set.entryForXValue(xValue, closestToY: yValue) as?
        BarChartDataEntry
```

```
    else { return nil }
```

```
// Not stacked
```

```
if entry.yValues == nil
```

```
{
```

```
    return high
```

```
}
```

```
guard
```

```
    let ranges = entry.ranges,
```

```
    ranges.count > 0
```

```
    else { return nil }
```

```
let stackIndex = getClosestStackIndex(ranges: ranges, value: yValue)
```

```
let pixel = chart
```

```
.getTransformer(forAxis: set.axisDependency)
```

```
.pixelForValues(x: high.x, y: ranges[stackIndex].to)
```

```
return Highlight(x: entry.x,
```

```
    y: entry.y,
```

```
    xPx: pixel.x,
```

```
    yPx: pixel.y,
```

```
    dataSetIndex: high.dataSetIndex,
```

```
    stackIndex: stackIndex,
```

```
    axis: high.axis)
```

```
}
```

```
/// - Parameters:
```

```
/// - entry:
```

```
/// - value:
```

```
/// - Returns: The index of the closest value inside the values array / ranges (stacked  
barchart) to the value given as a parameter.
```

```
@objc open func getClosestStackIndex(ranges: [Range]?, value: Double) -> Int
```

```
{
```

```

guard let ranges = ranges else { return 0 }

if let stackIndex = ranges.firstIndex(where: { $0.contains(value) }) {

    return stackIndex

} else {

    let length = max(ranges.count - 1, 0)

    return (value > ranges[length].to) ? length : 0

}

}

}

//

// BarLineChartViewBase.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
#if canImport(Uikit)
```

```
    import UIKit
```

```
#endif
```

```
#if canImport(Cocoa)
```

```
import Cocoa
```

```
#endif
```

```
/// Base-class of LineChart, BarChart, ScatterChart and CandleStickChart.
```

```
open class BarLineChartViewBase: ChartViewBase,  
BarLineScatterCandleBubbleChartDataProvider, NSUIGestureRecognizerDelegate
```

```
{
```



```
/// the maximum number of entries to which values will be drawn

/// (entry numbers greater than this value will cause value-labels to disappear)

internal var _maxVisibleCount = 100


/// flag that indicates if auto scaling on the y axis is enabled

private var _autoScaleMinMaxEnabled = false


private var _pinchZoomEnabled = false

private var _doubleTapToZoomEnabled = true

private var _dragXEnabled = true

private var _dragYEnabled = true


private var _scaleXEnabled = true

private var _scaleYEnabled = true


/// the color for the background of the chart-drawing area (everything behind the grid
lines).
```

```
@objc open var gridBackgroundColor = NSUIColor(red: 240/255.0, green: 240/255.0,  
blue: 240/255.0, alpha: 1.0)
```

```
@objc open var borderColor = NSUIColor.black
```

```
@objc open var borderLineWidth: CGFloat = 1.0
```

```
/// flag indicating if the grid background should be drawn or not
```

```
@objc open var drawGridBackgroundEnabled = false
```

```
/// When enabled, the borders rectangle will be rendered.
```

```
/// If this is enabled, there is no point drawing the axis-lines of x- and y-axis.
```

```
@objc open var drawBordersEnabled = false
```

```
/// When enabled, the values will be clipped to contentRect, otherwise they can bleed  
outside the content rect.
```

```
@objc open var clipValuesToContentEnabled: Bool = false
```

```
/// When disabled, the data and/or highlights will not be clipped to contentRect.  
Disabling this option can
```

/// be useful, when the data lies fully within the content rect, but is drawn in such a way
(such as thick lines)

/// that there is unwanted clipping.

@objc open var clipDataToContentEnabled: Bool = true

/// Sets the minimum offset (padding) around the chart, defaults to 10

@objc open var minOffset = CGFloat(10.0)

/// Sets whether the chart should keep its position (zoom / scroll) after a rotation
(orientation change)

/// **default**: false

@objc open var keepPositionOnRotation: Bool = false

/// The left y-axis object. In the horizontal bar-chart, this is the

/// top axis.

@objc open internal(set) var leftAxis = YAxis(position: .left)

/// The right y-axis object. In the horizontal bar-chart, this is the

```
/// bottom axis.
```

```
@objc open internal(set) var rightAxis = YAxis(position: .right)
```

/// The left Y axis renderer. This is a read-write property so you can set your own custom renderer here.

```
/// default: An instance of YAxisRenderer
```

```
@objc open lazy var leftYAxisRenderer = YAxisRenderer(viewPortHandler:
_viewPortHandler, yAxis: leftAxis, transformer: _leftAxisTransformer)
```

/// The right Y axis renderer. This is a read-write property so you can set your own custom renderer here.

```
/// default: An instance of YAxisRenderer
```

```
@objc open lazy var rightYAxisRenderer = YAxisRenderer(viewPortHandler:
_viewPortHandler, yAxis: rightAxis, transformer: _rightAxisTransformer)
```

```
internal var _leftAxisTransformer: Transformer!
```

```
internal var _rightAxisTransformer: Transformer!
```

/// The X axis renderer. This is a read-write property so you can set your own custom renderer here.

```
/// **default**: An instance of XAxisRenderer
```

```
@objc open lazy var xAxisRenderer = XAxisRenderer(viewPortHandler:  
_viewPortHandler, xAxis: _xAxis, transformer: _leftAxisTransformer)
```

```
internal var _tapGestureRecognizer: NSUITapGestureRecognizer!
```

```
internal var _doubleTapGestureRecognizer: NSUITapGestureRecognizer!
```

```
#if !os(tvOS)
```

```
internal var _pinchGestureRecognizer: NSUIPinchGestureRecognizer!
```

```
#endif
```

```
internal var _panGestureRecognizer: NSUIPanGestureRecognizer!
```

```
/// flag that indicates if a custom viewport offset has been set
```

```
private var _customViewPortEnabled = false
```

```
public override init(frame: CGRect)
```

```
{
```

```
    super.init(frame: frame)
```

```
}
```

```
public required init?(coder aDecoder: NSCoder)
```

```
{
```

```
    super.init(coder: aDecoder)
```

```
}
```

```
deinit
```

```
{
```

```
    stopDeceleration()
```

```
}
```

```
internal override func initialize()
```

```
{
```

```
    super.initialize()
```

```
    _leftAxisTransformer = Transformer(viewPortHandler: _viewPortHandler)
```

```
    _rightAxisTransformer = Transformer(viewPortHandler: _viewPortHandler)
```

```
self.highlighter = ChartHighlighter(chart: self)
```

```
_tapGestureRecognizer = NSUITapGestureRecognizer(target: self, action:  
#selector(tapGestureRecognized(_:)))
```

```
_doubleTapGestureRecognizer = NSUITapGestureRecognizer(target: self, action:  
#selector(doubleTapGestureRecognized(_:)))
```

```
_doubleTapGestureRecognizer.nsuiNumberOfTapsRequired = 2
```

```
_panGestureRecognizer = NSUIPanGestureRecognizer(target: self, action:  
#selector(panGestureRecognized(_:)))
```

```
_panGestureRecognizer.delegate = self
```

```
self.addGestureRecognizer(_tapGestureRecognizer)
```

```
self.addGestureRecognizer(_doubleTapGestureRecognizer)
```

```
self.addGestureRecognizer(_panGestureRecognizer)
```

```
_doubleTapGestureRecognizer.isEnabled = _doubleTapToZoomEnabled
```

```
_panGestureRecognizer.isEnabled = _dragXEnabled || _dragYEnabled
```

```

#if !os(tvOS)

    _pinchGestureRecognizer = NSUIPinchGestureRecognizer(target: self, action:
#selector(BarLineChartViewBase.pinchGestureRecognized(_:)))

    _pinchGestureRecognizer.delegate = self

    self.addGestureRecognizer(_pinchGestureRecognizer)

    _pinchGestureRecognizer.isEnabled = _pinchZoomEnabled || _scaleXEnabled ||
_scaleYEnabled

#endif

}

```

```

open override fun observeValue(forKeyPath keyPath: String?, of object: Any?, change:
[NSKeyValueChangeKey : Any]?, context: UnsafeMutableRawPointer?)

{

    // Saving current position of chart.

    var oldPoint: CGPoint?

    if (keepPositionOnRotation && (keyPath == "frame" || keyPath == "bounds"))

    {

        oldPoint = viewPortHandler.contentRect.origin
    }
}

```



```

        getTransformer(forAxis: .left).pixelToValues(&oldPoint!)
    }

    // Superclass transforms chart.

    super.observeValue(forKeyPath: keyPath, of: object, change: change, context:
context)

    // Restoring old position of chart

    if var newPoint = oldPoint , keepPositionOnRotation
    {

        getTransformer(forAxis: .left).pointValueToPixel(&newPoint)

        viewPortHandler.centerViewPort(pt: newPoint, chart: self)

    }

    else

    {

        viewPortHandler.refresh(newMatrix: viewPortHandler.touchMatrix, chart: self,
invalidate: true)

    }

```

```
}
```

```
open override func draw(_ rect: CGRect)
```

```
{
```

```
    super.draw(rect)
```

```
    guard data != nil, let renderer = renderer else { return }
```

```
    let optionalContext = NSUIGraphicsGetCurrentContext()
```

```
    guard let context = optionalContext else { return }
```

```
    // execute all drawing commands
```

```
    drawGridBackground(context: context)
```

```
    if _autoScaleMinMaxEnabled
```

```
    {
```

```

        autoScale()

    }

    if leftAxis.isEnabled

    {

        leftYAxisRenderer.computeAxis(min:      leftAxis._axisMinimum,      max:
leftAxis._axisMaximum, inverted: leftAxis.isInverted)

    }

    if rightAxis.isEnabled

    {

        rightYAxisRenderer.computeAxis(min:      rightAxis._axisMinimum,      max:
rightAxis._axisMaximum, inverted: rightAxis.isInverted)

    }

    if _xAxis.isEnabled

    {

        xAxisRenderer.computeAxis(min:      _xAxis._axisMinimum,      max:
_xAxis._axisMaximum, inverted: false)

```

```
}
```

```
xAxisRenderer.renderAxisLine(context: context)
```

```
leftYAxisRenderer.renderAxisLine(context: context)
```

```
rightYAxisRenderer.renderAxisLine(context: context)
```

```
// The renderers are responsible for clipping, to account for line-width center etc.
```

```
if xAxis.drawGridLinesBehindDataEnabled
```

```
{
```

```
    xAxisRenderer.renderGridLines(context: context)
```

```
    leftYAxisRenderer.renderGridLines(context: context)
```

```
    rightYAxisRenderer.renderGridLines(context: context)
```

```
}
```

```
if _xAxis.isEnabled && _xAxis.isDrawLimitLinesBehindDataEnabled
```

```
{
```

```
    xAxisRenderer.renderLimitLines(context: context)
```

```
}
```

```
if leftAxis.isEnabled && leftAxis.isDrawLimitLinesBehindDataEnabled
```

```
{
```

```
    leftYAxisRenderer.renderLimitLines(context: context)
```

```
}
```

```
if rightAxis.isEnabled && rightAxis.isDrawLimitLinesBehindDataEnabled
```

```
{
```

```
    rightYAxisRenderer.renderLimitLines(context: context)
```

```
}
```

```
context.saveGState()
```

```
// make sure the data cannot be drawn outside the content-rect
```

```
if clipDataToContentEnabled {
```

```
    context.clip(to: _viewPortHandler.contentRect)
```

```
}
```

```

renderer.drawData(context: context)

// The renderers are responsible for clipping, to account for line-width center etc.

if !xAxis.drawGridLinesBehindDataEnabled

{

    xAxisRenderer.renderGridLines(context: context)

    leftYAxisRenderer.renderGridLines(context: context)

    rightYAxisRenderer.renderGridLines(context: context)

}

// if highlighting is enabled

if (valuesToHighlight())

{

    renderer.drawHighlighted(context: context, indices: _indicesToHighlight)

}

context.restoreGState()

```

```
renderer.drawExtras(context: context)
```

```
if _xAxis.isEnabled && !_xAxis.isDrawLimitLinesBehindDataEnabled
```

```
{
```

```
    xAxisRenderer.renderLimitLines(context: context)
```

```
}
```

```
if leftAxis.isEnabled && !leftAxis.isDrawLimitLinesBehindDataEnabled
```

```
{
```

```
    leftYAxisRenderer.renderLimitLines(context: context)
```

```
}
```

```
if rightAxis.isEnabled && !rightAxis.isDrawLimitLinesBehindDataEnabled
```

```
{
```

```
    rightYAxisRenderer.renderLimitLines(context: context)
```

```
}
```

```
xAxisRenderer.renderAxisLabels(context: context)
```

```
leftYAxisRenderer.renderAxisLabels(context: context)
```

```
rightYAxisRenderer.renderAxisLabels(context: context)
```

```
if clipValuesToContentEnabled
```

```
{
```

```
    context.saveGState()
```

```
    context.clip(to: _viewPortHandler.contentRect)
```

```
    renderer.drawValues(context: context)
```

```
    context.restoreGState()
```

```
}
```

```
else
```

```
{
```

```
    renderer.drawValues(context: context)
```



```
}
```

```
    _legendRenderer.renderLegend(context: context)
```

```
    drawDescription(context: context)
```

```
    drawMarkers(context: context)
```

```
}
```

```
private var _autoScaleLastLowestVisibleX: Double?
```

```
private var _autoScaleLastHighestVisibleX: Double?
```

```
    /// Performs auto scaling of the axis by recalculating the minimum and maximum y-  
    values based on the entries currently in view.
```

```
internal fun autoScale()
```

```
{
```

```
    guard let data = _data
```

```
        else { return }
```

```
data.calcMinMaxY(fromX: self.lowestVisibleX, toX: self.highestVisibleX)
```

```
_xAxis.calculate(min: data.xMin, max: data.xMax)
```

```
// calculate axis range (min / max) according to provided data
```

```
if leftAxis.isEnabled
```

```
{
```

```
    leftAxis.calculate(min: data.getYMin(axis: .left), max: data.getYMax(axis: .left))
```

```
}
```

```
if rightAxis.isEnabled
```

```
{
```

```
    rightAxis.calculate(min: data.getYMin(axis: .right), max:  
data.getYMax(axis: .right))
```

```
}
```

```

        calculateOffsets()

    }

    internal func prepareValuePxMatrix()

    {

        _rightAxisTransformer.prepareMatrixValuePx(chartXMin: _xAxis._axisMinimum,
        deltaX: CGFloat(xAxis.axisRange), deltaY: CGFloat(rightAxis.axisRange), chartYMin:
        rightAxis._axisMinimum)

        _leftAxisTransformer.prepareMatrixValuePx(chartXMin:    xAxis._axisMinimum,
        deltaX: CGFloat(xAxis.axisRange), deltaY: CGFloat(leftAxis.axisRange), chartYMin:
        leftAxis._axisMinimum)

    }

    internal func prepareOffsetMatrix()

    {

        _rightAxisTransformer.prepareMatrixOffset(inverted: rightAxis.isInverted)

        _leftAxisTransformer.prepareMatrixOffset(inverted: leftAxis.isInverted)

    }

```

```

open override func notifyDataSetChanged()

{

    renderer?.initBuffers()


    calcMinMax()


    leftYAxisRenderer.computeAxis(min:      leftAxis._axisMinimum,      max:
leftAxis._axisMaximum, inverted: leftAxis.isInverted)

    rightYAxisRenderer.computeAxis(min:      rightAxis._axisMinimum,      max:
rightAxis._axisMaximum, inverted: rightAxis.isInverted)


    if let data = _data

    {

        xAxisRenderer.computeAxis(

            min: _xAxis._axisMinimum,

            max: _xAxis._axisMaximum,

            inverted: false)

```

```

    if _legend !== nil

    {

        legendRenderer?.computeLegend(data: data)

    }

}

```

```

calculateOffsets()

```

```

setNeedsDisplay()

```

```

}

```

```

internal override func calcMinMax()

```

```

{

```

```

    // calculate / set x-axis range

```

```

    _xAxis.calculate(min: _data?.xMin ?? 0.0, max: _data?.xMax ?? 0.0)

```

```

    // calculate axis range (min / max) according to provided data

```

```

        leftAxis.calculate(min:    _data?.getYMin(axis:    .left)    ??    0.0,    max:
_data?.getYMax(axis: .left) ?? 0.0)

```

```

        rightAxis.calculate(min:    _data?.getYMin(axis:    .right)    ??    0.0,    max:
_data?.getYMax(axis: .right) ?? 0.0)

```

```

    }

```

```

    internal func calculateLegendOffsets(offsetLeft: inout CGFloat, offsetTop: inout
CGFloat, offsetRight: inout CGFloat, offsetBottom: inout CGFloat)

```

```

    {

```

```

        // setup offsets for legend

```

```

        if _legend !== nil && _legend.isEnabled && !_legend.drawInside

```

```

        {

```

```

            switch _legend.orientation

```

```

            {

```

```

                case .vertical:

```

```

                    switch _legend.horizontalAlignment

```

```

                    {

```

```

                        case .left:

```

```
        offsetLeft += min(_legend.neededWidth, _viewportHandler.chartWidth *
        _legend.maxSizePercent) + _legend.xOffset
```

```
    case .right:
```

```
        offsetRight += min(_legend.neededWidth, _viewportHandler.chartWidth *
        _legend.maxSizePercent) + _legend.xOffset
```

```
    case .center:
```

```
        switch _legend.verticalAlignment
```

```
        {
```

```
        case .top:
```

```
            offsetTop += min(_legend.neededHeight, _viewportHandler.chartHeight *
            _legend.maxSizePercent) + _legend.yOffset
```

```
        case .bottom:
```

```
            offsetBottom += min(_legend.neededHeight,
            _viewportHandler.chartHeight * _legend.maxSizePercent) + _legend.yOffset
```

```

        default:

            break

        }

    }

case .horizontal:

    switch _legend.verticalAlignment

    {

        case .top:

            offsetTop += min(_legend.neededHeight, _viewPortHandler.chartHeight *
                _legend.maxSizePercent) + _legend.yOffset

        case .bottom:

            offsetBottom += min(_legend.neededHeight, _viewPortHandler.chartHeight
                * _legend.maxSizePercent) + _legend.yOffset

        default:

```



```

        break
    }

}

}

}

```

internal override func calculateOffsets()

```

{

    if !_customViewPortEnabled

    {

        var offsetLeft = CGFloat(0.0)

        var offsetRight = CGFloat(0.0)

        var offsetTop = CGFloat(0.0)

        var offsetBottom = CGFloat(0.0)

        calculateLegendOffsets(offsetLeft: &offsetLeft,

                                offsetTop: &offsetTop,

```

```

        offsetRight: &offsetRight,

        offsetBottom: &offsetBottom)

// offsets for y-labels

if leftAxis.needsOffset

{

    offsetLeft += leftAxis.requiredSize().width

}

if rightAxis.needsOffset

{

    offsetRight += rightAxis.requiredSize().width

}

if xAxis.isEnabled && xAxis.isDrawLabelsEnabled

{

    let xlabelheight = xAxis.labelRotatedHeight + xAxis.yOffset

```

```

// offsets for x-labels

if xAxis.labelPosition == .bottom

{

    offsetBottom += xlabelheight

}

else if xAxis.labelPosition == .top

{

    offsetTop += xlabelheight

}

else if xAxis.labelPosition == .bothSided

{

    offsetBottom += xlabelheight

    offsetTop += xlabelheight

}

}

```

```
offsetTop += self.extraTopOffset
```

```
offsetRight += self.extraRightOffset
```

```
offsetBottom += self.extraBottomOffset
```

```
offsetLeft += self.extraLeftOffset
```

```
_viewPortHandler.restrainViewPort(
```

```
    offsetLeft: max(self.minOffset, offsetLeft),
```

```
    offsetTop: max(self.minOffset, offsetTop),
```

```
    offsetRight: max(self.minOffset, offsetRight),
```

```
    offsetBottom: max(self.minOffset, offsetBottom))
```

```
}
```

```
prepareOffsetMatrix()
```

```
prepareValuePxMatrix()
```

```
}
```

```
/// draws the grid background
```

```

internal fun drawGridBackground(context: CGContext)

{

    if drawGridBackgroundEnabled || drawBordersEnabled

    {

        context.saveGState()

    }

    if drawGridBackgroundEnabled

    {

        // draw the grid background

        context.setFillColor(gridBackgroundColor.cgColor)

        context.fill(_viewPortHandler.contentRect)

    }

    if drawBordersEnabled

    {

        context.setLineWidth(borderLineWidth)

```

```

        context.setStrokeColor(borderColor.cgColor)

        context.stroke(_viewPortHandler.contentRect)

    }

    if drawGridBackgroundEnabled || drawBordersEnabled

    {

        context.restoreGState()

    }

}

```

```
// MARK: - Gestures
```

```

private enum GestureScaleAxis

{

    case both

    case x

    case y

```

```
}
```

```
private var _isDragging = false
```

```
private var _isScaling = false
```

```
private var _gestureScaleAxis = GestureScaleAxis.both
```

```
private var _closestDataSetToTouch: IChartDataSet!
```

```
private var _panGestureReachedEdge: Bool = false
```

```
private weak var _outerScrollView: NSUIScrollView?
```

```
private var _lastPanPoint = CGPoint() /// This is to prevent using setTranslation which  
resets velocity
```

```
private var _decelerationLastTime: TimeInterval = 0.0
```

```
private var _decelerationDisplayLink: NSUIDisplayLink!
```

```
private var _decelerationVelocity = CGPoint()
```

```
@objc private func tapGestureRecognized(_ recognizer: NSUITapGestureRecognizer)
```

```
{
```

```

if _data === nil

{

    return

}


if recognizer.state == NSUIGestureRecognizerState.ended

{

    if !isHighlightPerTapEnabled { return }

    let h = getHighlightByTouchPoint(recognizer.location(in: self))

    if h === nil || h == self.lastHighlighted

    {

        lastHighlighted = nil

        highlightValue(nil, callDelegate: true)

    }

    else

```



```

    {

        lastHighlighted = h

        highlightValue(h, callDelegate: true)

    }

}

}

```

```

@objc private func doubleTapGestureRecognized(_ recognizer:
NSUITapGestureRecognizer)

```

```

{

    if _data === nil

    {

        return

    }

}

```

```

if recognizer.state == NSUIGestureRecognizerState.ended

```

```

{

    if _data !== nil && _doubleTapToZoomEnabled && (data?.entryCount ?? 0) > 0

```

```

{

    var location = recognizer.location(in: self)

    location.x = location.x - _viewPortHandler.offsetLeft


    if isTouchInverted()

    {

        location.y = -(location.y - _viewPortHandler.offsetTop)

    }

    else

    {

        location.y = -(self.bounds.size.height - location.y -
_viewPortHandler.offsetBottom)

    }


    let scaleX: CGFloat = isScaleXEnabled ? 1.4 : 1.0

    let scaleY: CGFloat = isScaleYEnabled ? 1.4 : 1.0


    self.zoom(scaleX: scaleX, scaleY: scaleY, x: location.x, y: location.y)

```

```

        delegate?.chartScaled?(self, scaleX: scaleX, scaleY: scaleY)

    }

}

}

#if !os(tvOS)

@objc private func pinchGestureRecognized(_ recognizer:
NSUIPinchGestureRecognizer)

{

    if recognizer.state == NSUIGestureRecognizerState.began

    {

        stopDeceleration()

        if _data != nil &&

            (_pinchZoomEnabled || _scaleXEnabled || _scaleYEnabled)

        {

            _isScaling = true

```

```

if _pinchZoomEnabled

{

    _gestureScaleAxis = .both

}

else

{

    let x = abs(recognizer.location(in: self).x - recognizer.nsuiLocationOfTouch(1,
inView: self).x)

    let y = abs(recognizer.location(in: self).y - recognizer.nsuiLocationOfTouch(1,
inView: self).y)


    if _scaleXEnabled != _scaleYEnabled

    {

        _gestureScaleAxis = _scaleXEnabled ? .x : .y

    }

    else

    {

        _gestureScaleAxis = x > y ? .x : .y

```

```

    }

    }

}

}

else if recognizer.state == NSUIGestureRecognizerState.ended ||

    recognizer.state == NSUIGestureRecognizerState.cancelled

{

    if _isScaling

    {

        _isScaling = false

        // Range might have changed, which means that Y-axis labels could have
        changed in size, affecting Y-axis size. So we need to recalculate offsets.

        calculateOffsets()

        setNeedsDisplay()

    }

}

else if recognizer.state == NSUIGestureRecognizerState.changed

```

```

{

    let isZoomingOut = (recognizer.nsuiScale < 1)

    var canZoomMoreX = isZoomingOut ? _viewPortHandler.canZoomOutMoreX :
    _viewPortHandler.canZoomInMoreX

    var canZoomMoreY = isZoomingOut ? _viewPortHandler.canZoomOutMoreY :
    _viewPortHandler.canZoomInMoreY

    if _isScaling

    {

        canZoomMoreX    =    canZoomMoreX    &&    _scaleXEnabled    &&
        (_gestureScaleAxis == .both || _gestureScaleAxis == .x)

        canZoomMoreY    =    canZoomMoreY    &&    _scaleYEnabled    &&
        (_gestureScaleAxis == .both || _gestureScaleAxis == .y)

        if canZoomMoreX || canZoomMoreY

        {

            var location = recognizer.location(in: self)

            location.x = location.x - _viewPortHandler.offsetLeft

            if isTouchInverted()

```

```

    {

        location.y = -(location.y - _viewPortHandler.offsetTop)

    }

    else

    {

        location.y    =    -(_viewPortHandler.chartHeight    -    location.y    -
        _viewPortHandler.offsetBottom)

    }


    let scaleX = canZoomMoreX ? recognizer.nsuiScale : 1.0

    let scaleY = canZoomMoreY ? recognizer.nsuiScale : 1.0


    var matrix = CGAffineTransform(translationX: location.x, y: location.y)

    matrix = matrix.scaledBy(x: scaleX, y: scaleY)

    matrix = matrix.translatedBy(x: -location.x, y: -location.y)


    matrix = _viewPortHandler.touchMatrix.concatenating(matrix)

```

```
_viewPortHandler.refresh(newMatrix: matrix, chart: self, invalidate: true)
```

```
if delegate !== nil
```

```
{
```

```
    delegate?.chartScaled?(self, scaleX: scaleX, scaleY: scaleY)
```

```
}
```

```
}
```

```
recognizer.nsuiScale = 1.0
```

```
}
```

```
}
```

```
}
```

```
#endif
```

```
@objc private func panGestureRecognized(_ recognizer: NSUIPanGestureRecognizer)
```

```
{
```

```
    if recognizer.state == NSUIGestureRecognizerState.began &&  
recognizer.nsuiNumberOfTouches() > 0
```



```

{

    stopDeceleration()

    if _data === nil || !self.isDragEnabled

    { // If we have no data, we have nothing to pan and no data to highlight

        return

    }

    // If drag is enabled and we are in a position where there's something to drag:

    // * If we're zoomed in, then obviously we have something to drag.

    // * If we have a drag offset - we always have something to drag

    if !self.hasNoDragOffset || !self.isFullyZoomedOut

    {

        _isDragging = true

        _closestDataSetToTouch = getDataSetByTouchPoint(point:
recognizer.nsuiLocationOfTouch(0, inView: self))

```

```

var translation = recognizer.translation(in: self)

if !self.dragXEnabled

{

    translation.x = 0.0

}

else if !self.dragYEnabled

{

    translation.y = 0.0

}

let didUserDrag = translation.x != 0.0 || translation.y != 0.0

// Check to see if user dragged at all and if so, can the chart be dragged by the
given amount

if didUserDrag && !performPanChange(translation: translation)

{

    if _outerScrollView != nil

    {

```

```

        // We can stop dragging right now, and let the scroll view take control

        _outerScrollView = nil

        _isDragging = false

    }

}

else

{

    if _outerScrollView !== nil

    {

        // Prevent the parent scroll view from scrolling

        _outerScrollView?.nsuiIsScrollEnabled = false

    }

}

_lastPanPoint = recognizer.translation(in: self)

}

else if self.isHighlightPerDragEnabled

```

```

    {

        // We will only handle highlights on NSUIGestureRecognizerState.Changed

        _isDragging = false

    }

}

else if recognizer.state == NSUIGestureRecognizerState.changed

{

    if _isDragging

    {

        let originalTranslation = recognizer.translation(in: self)

        var translation = CGPoint(x: originalTranslation.x - _lastPanPoint.x, y:
originalTranslation.y - _lastPanPoint.y)

        if !self.dragXEnabled

        {

            translation.x = 0.0

        }

    }

}

```

```

else if !self.dragYEnabled

{

    translation.y = 0.0

}

let _ = performPanChange(translation: translation)

_lastPanPoint = originalTranslation

}

else if isHighlightPerDragEnabled

{

    let h = getHighlightByTouchPoint(recognizer.location(in: self))

    let lastHighlighted = self.lastHighlighted

    if h != lastHighlighted

    {

```

```

        self.lastHighlighted = h

        self.highlightValue(h, callDelegate: true)

    }

}

}

else if recognizer.state == NSUIGestureRecognizerState.ended || recognizer.state ==
NSUIGestureRecognizerState.cancelled

{

    if _isDragging

    {

        if recognizer.state == NSUIGestureRecognizerState.ended &&
isDragDecelerationEnabled

        {

            stopDeceleration()

            _decelerationLastTime = CACurrentMediaTime()

            _decelerationVelocity = recognizer.velocity(in: self)

```

```
        _decelerationDisplayLink = NSUIDisplayLink(target: self, selector:
#selector(BarLineChartViewBase.decelerationLoop))
```

```
        _decelerationDisplayLink.add(to: RunLoop.main, forMode:
RunLoop.Mode.common)
```

```
    }
```

```
    _isDragging = false
```

```
    delegate?.chartViewDidEndPanning?(self)
```

```
    }
```

```
    if _outerScrollView !== nil
```

```
    {
```

```
        _outerScrollView?.nsuiIsScrollEnabled = true
```

```
        _outerScrollView = nil
```

```
    }
```

```
    }
```

```
}
```

```
private func performPanChange(translation: CGPoint) -> Bool

{

    var translation = translation


    if isTouchInverted()

    {

        if self is HorizontalBarChartView

        {

            translation.x = -translation.x

        }

        else

        {

            translation.y = -translation.y

        }

    }

}
```



```

let originalMatrix = _viewPortHandler.touchMatrix

var matrix = CGAffineTransform(translationX: translation.x, y: translation.y)

matrix = originalMatrix.concatenating(matrix)

matrix = _viewPortHandler.refresh(newMatrix: matrix, chart: self, invalidate: true)

if matrix != originalMatrix
{
    delegate?.chartTranslated?(self, dX: translation.x, dY: translation.y)
}

// Did we managed to actually drag or did we reach the edge?

return matrix.tx != originalMatrix.tx || matrix.ty != originalMatrix.ty
}

private func isTouchInverted() -> Bool

```

```

{

    return isAnyAxisInverted &&

        _closestDataSetToTouch !== nil &&

        getAxis(_closestDataSetToTouch.axisDependency).isInverted

}


@objc open func stopDeceleration()

{

    if _decelerationDisplayLink !== nil

    {

        _decelerationDisplayLink.remove(from: RunLoop.main, forMode:
RunLoop.Mode.common)

        _decelerationDisplayLink = nil

    }

}


@objc private func decelerationLoop()

{

```

```

let currentTime = CACurrentMediaTime()

_decelerationVelocity.x *= self.dragDecelerationFrictionCoef

_decelerationVelocity.y *= self.dragDecelerationFrictionCoef

let timeInterval = CGFloat(currentTime - _decelerationLastTime)

let distance = CGPoint(

    x: _decelerationVelocity.x * timeInterval,

    y: _decelerationVelocity.y * timeInterval

)

if !performPanChange(translation: distance)

{

    // We reached the edge, stop

    _decelerationVelocity.x = 0.0

    _decelerationVelocity.y = 0.0

```

```
}
```

```
_decelerationLastTime = currentTime
```

```
if abs(_decelerationVelocity.x) < 0.001 && abs(_decelerationVelocity.y) < 0.001
```

```
{
```

```
    stopDeceleration()
```

```
    // Range might have changed, which means that Y-axis labels could have changed  
    in size, affecting Y-axis size. So we need to recalculate offsets.
```

```
    calculateOffsets()
```

```
    setNeedsDisplay()
```

```
}
```

```
}
```

```
private func nsuiGestureRecognizerShouldBegin(_ gestureRecognizer:  
NSUIGestureRecognizer) -> Bool
```

```
{
```

```

if gestureRecognizer == _panGestureRecognizer

{

    let velocity = _panGestureRecognizer.velocity(in: self)

    if _data === nil || !isDragEnabled ||

        (self.hasNoDragOffset                &&                self.isFullyZoomedOut
        && !self.isHighlightPerDragEnabled) ||

        (!_dragYEnabled && abs(velocity.y) > abs(velocity.x)) ||

        (!_dragXEnabled && abs(velocity.y) < abs(velocity.x))

    {

        return false

    }

}

else

{

    #if !os(tvOS)

    if gestureRecognizer == _pinchGestureRecognizer

    {

```

```
        if _data === nil || (!_pinchZoomEnabled && !_scaleXEnabled  
&& !_scaleYEnabled)
```

```
        {
```

```
            return false
```

```
        }
```

```
    }
```

```
    #endif
```

```
}
```

```
return true
```

```
}
```

```
#if !os(OSX)
```

```
open override func gestureRecognizerShouldBegin(_ gestureRecognizer:  
UIGestureRecognizer) -> Bool
```

```
{
```

```
    if !super.gestureRecognizerShouldBegin(gestureRecognizer)
```

```
    {
```

```

        return false
    }

    return nsuiGestureRecognizerShouldBegin(gestureRecognizer)
}

#endif

#ifdef os(OSX)

public func gestureRecognizerShouldBegin(gestureRecognizer: NSGestureRecognizer)
-> Bool

{

    return nsuiGestureRecognizerShouldBegin(gestureRecognizer)

}

#endif

open func gestureRecognizer(_ gestureRecognizer: NSUIGestureRecognizer,
shouldRecognizeSimultaneouslyWith otherGestureRecognizer: NSUIGestureRecognizer)
-> Bool

{

```

```

#if !os(tvOS)

    if ((gestureRecognizer is NSUIPinchGestureRecognizer && otherGestureRecognizer
is NSUIPanGestureRecognizer) ||

        (gestureRecognizer is NSUIPanGestureRecognizer && otherGestureRecognizer is
NSUIPinchGestureRecognizer))

    {

        return true

    }

#endif

if gestureRecognizer is NSUIPanGestureRecognizer,

    otherGestureRecognizer is NSUIPanGestureRecognizer,

    gestureRecognizer == _panGestureRecognizer

    {

        var scrollView = self.superview

        while scrollView != nil && !(scrollView is UIScrollView)

        {

            scrollView = scrollView?.superview


```



```
}
```

```
// If there is two scrollview together, we pick the superview of the inner scrollview.
```

```
// In the case of UITableViewWrapperView, the superview will be UITableView
```

```
if let superViewOfScrollView = scrollView?.superview,
```

```
    superViewOfScrollView is UIScrollView
```

```
{
```

```
    scrollView = superViewOfScrollView
```

```
}
```

```
var foundScrollView = scrollView as? UIScrollView
```

```
if !(foundScrollView?.isScrollEnabled ?? true)
```

```
{
```

```
    foundScrollView = nil
```

```
}
```

```

        let                scrollViewPanGestureRecognizer                =
foundScrollView?.nsuiGestureRecognizers?.first {

        $0 is NSUIPanGestureRecognizer

    }

    if otherGestureRecognizer === scrollViewPanGestureRecognizer

    {

        _outerScrollView = foundScrollView

        return true

    }

}

return false

}

```

```

/// MARK: Viewport modifiers

```

```
/// Zooms in by 1.4, into the charts center.
```

```
@objc open func zoomIn()
```

```
{
```

```
    let center = _viewPortHandler.contentCenter
```

```
    let matrix = _viewPortHandler.zoomIn(x: center.x, y: -center.y)
```

```
    _viewPortHandler.refresh(newMatrix: matrix, chart: self, invalidate: false)
```

```
    // Range might have changed, which means that Y-axis labels could have changed in  
    // size, affecting Y-axis size. So we need to recalculate offsets.
```

```
    calculateOffsets()
```

```
    setNeedsDisplay()
```

```
}
```

```
/// Zooms out by 0.7, from the charts center.
```

```
@objc open func zoomOut()
```

```
{
```

```
    let center = _viewPortHandler.contentCenter
```

```
let matrix = _viewPortHandler.zoomOut(x: center.x, y: -center.y)
```

```
_viewPortHandler.refresh(newMatrix: matrix, chart: self, invalidate: false)
```

// Range might have changed, which means that Y-axis labels could have changed in size, affecting Y-axis size. So we need to recalculate offsets.

```
calculateOffsets()
```

```
setNeedsDisplay()
```

```
}
```

/// Zooms out to original size.

```
@objc open func resetZoom()
```

```
{
```

```
let matrix = _viewPortHandler.resetZoom()
```

```
_viewPortHandler.refresh(newMatrix: matrix, chart: self, invalidate: false)
```

// Range might have changed, which means that Y-axis labels could have changed in size, affecting Y-axis size. So we need to recalculate offsets.

```

        calculateOffsets()

        setNeedsDisplay()

    }

    /// Zooms in or out by the given scale factor. x and y are the coordinates
    /// (in pixels) of the zoom center.

    ///

    /// - Parameters:

    /// - scaleX: if < 1 --> zoom out, if > 1 --> zoom in

    /// - scaleY: if < 1 --> zoom out, if > 1 --> zoom in

    /// - x:

    /// - y:

    @objc open func zoom(

        scaleX: CGFloat,

        scaleY: CGFloat,

        x: CGFloat,

        y: CGFloat)

```

```
{

    let matrix = _viewPortHandler.zoom(scaleX: scaleX, scaleY: scaleY, x: x, y: -y)

    _viewPortHandler.refresh(newMatrix: matrix, chart: self, invalidate: false)
```

// Range might have changed, which means that Y-axis labels could have changed in size, affecting Y-axis size. So we need to recalculate offsets.

```
    calculateOffsets()

    setNeedsDisplay()

}
```

/// Zooms in or out by the given scale factor.

/// x and y are the values (**not pixels**) of the zoom center.

///

/// - Parameters:

/// - scaleX: if < 1 --> zoom out, if > 1 --> zoom in

/// - scaleY: if < 1 --> zoom out, if > 1 --> zoom in

/// - xValue:

/// - yValue:

```

/// - axis:

@objc open func zoom(

    scaleX: CGFloat,

    scaleY: CGFloat,

    xValue: Double,

    yValue: Double,

    axis: YAxis.AxisDependency)

{

    let job = ZoomViewJob(

        viewPortHandler: viewPortHandler,

        scaleX: scaleX,

        scaleY: scaleY,

        xValue: xValue,

        yValue: yValue,

        transformer: getTransformer(forAxis: axis),

        axis: axis,

        view: self)

```

```

        addViewportJob(job)

    }

    /// Zooms to the center of the chart with the given scale factor.

    ///

    /// - Parameters:

    ///   - scaleX: if < 1 --> zoom out, if > 1 --> zoom in

    ///   - scaleY: if < 1 --> zoom out, if > 1 --> zoom in

    ///   - xValue:

    ///   - yValue:

    ///   - axis:

    @objc open func zoomToCenter(

        scaleX: CGFloat,

        scaleY: CGFloat)

    {

        let center = centerOffsets

        let matrix = viewPortHandler.zoom(

```



```

        scaleX: scaleX,

        scaleY: scaleY,

        x: center.x,

        y: -center.y)

viewPortHandler.refresh(newMatrix: matrix, chart: self, invalidate: false)

}

/// Zooms by the specified scale factor to the specified values on the specified axis.

///

/// - Parameters:

///   - scaleX:

///   - scaleY:

///   - xValue:

///   - yValue:

///   - axis: which axis should be used as a reference for the y-axis

///   - duration: the duration of the animation in seconds

///   - easing:

```

```

@objc open func zoomAndCenterViewAnimated(

    scaleX: CGFloat,

    scaleY: CGFloat,

    xValue: Double,

    yValue: Double,

    axis: YAxis.AxisDependency,

    duration: TimeInterval,

    easing: ChartEasingFunctionBlock?)

{

    let origin = valueForTouchPoint(

        point: CGPoint(x: viewPortHandler.contentLeft, y: viewPortHandler.contentTop),

        axis: axis)

    let job = AnimatedZoomViewJob(

        viewPortHandler: viewPortHandler,

        transformer: getTransformer(forAxis: axis),

        view: self,

```

```
yAxis: getAxis(axis),

xAxisRange: _xAxis.axisRange,

scaleX: scaleX,

scaleY: scaleY,

xOrigin: viewPortHandler.scaleX,

yOrigin: viewPortHandler.scaleY,

zoomCenterX: CGFloat(xValue),

zoomCenterY: CGFloat(yValue),

zoomOriginX: origin.x,

zoomOriginY: origin.y,

duration: duration,

easing: easing)
```

```
addViewportJob(job)
```

```
}
```

/// Zooms by the specified scale factor to the specified values on the specified axis.

```

///

/// - Parameters:

/// - scaleX:

/// - scaleY:

/// - xValue:

/// - yValue:

/// - axis: which axis should be used as a reference for the y-axis

/// - duration: the duration of the animation in seconds

/// - easing:

@objc open func zoomAndCenterViewAnimated(

    scaleX: CGFloat,

    scaleY: CGFloat,

    xValue: Double,

    yValue: Double,

    axis: YAxis.AxisDependency,

    duration: TimeInterval,

    easingOption: ChartEasingOption)

```

```

{

    zoomAndCenterViewAnimated(scaleX: scaleX, scaleY: scaleY, xValue: xValue,
yValue:    yValue,    axis:    axis,    duration:    duration,    easing:
easingFunctionFromOption(easingOption))

}

```

/// Zooms by the specified scale factor to the specified values on the specified axis.

///

/// - Parameters:

/// - scaleX:

/// - scaleY:

/// - xValue:

/// - yValue:

/// - axis: which axis should be used as a reference for the y-axis

/// - duration: the duration of the animation in seconds

/// - easing:

@objc open func zoomAndCenterViewAnimated(

scaleX: CGFloat,

```

        scaleY: CGFloat,

        xValue: Double,

        yValue: Double,

        axis: YAxis.AxisDependency,

        duration: TimeInterval)

{

    zoomAndCenterViewAnimated(scaleX: scaleX, scaleY: scaleY, xValue: xValue,
yValue: yValue, axis: axis, duration: duration, easingOption: .easeInOutSine)

}

/// Resets all zooming and dragging and makes the chart fit exactly it's bounds.

@objc open func fitScreen()

{

    let matrix = _viewPortHandler.fitScreen()

    _viewPortHandler.refresh(newMatrix: matrix, chart: self, invalidate: false)

    calculateOffsets()

    setNeedsDisplay()

```

```
}
```

```
/// Sets the minimum scale value to which can be zoomed out. 1 = fitScreen
```

```
@objc open func setScaleMinima(_ scaleX: CGFloat, scaleY: CGFloat)
```

```
{
```

```
    _viewPortHandler.setMinimumScaleX(scaleX)
```

```
    _viewPortHandler.setMinimumScaleY(scaleY)
```

```
}
```

```
@objc open var visibleXRange: Double
```

```
{
```

```
    return abs(highestVisibleX - lowestVisibleX)
```

```
}
```

```
/// Sets the size of the area (range on the x-axis) that should be maximum visible at once  
(no further zooming out allowed).
```

```
///
```

/// If this is e.g. set to 10, no more than a range of 10 values on the x-axis can be viewed at once without scrolling.

///

/// If you call this method, chart must have data or it has no effect.

```
@objc open func setVisibleXRangeMaximum(_ maxXRange: Double)
```

```
{
```

```
    let xScale = _xAxis.axisRange / maxXRange
```

```
    _viewPortHandler.setMinimumScaleX(CGFloat(xScale))
```

```
}
```

/// Sets the size of the area (range on the x-axis) that should be minimum visible at once (no further zooming in allowed).

///

/// If this is e.g. set to 10, no less than a range of 10 values on the x-axis can be viewed at once without scrolling.

///

/// If you call this method, chart must have data or it has no effect.

```
@objc open func setVisibleXRangeMinimum(_ minXRange: Double)
```

```
{
```



```

    let xScale = _xAxis.axisRange / minXRange

    _viewPortHandler.setMaximumScaleX(CGFloat(xScale))

}

```

/// Limits the maximum and minimum value count that can be visible by pinching and zooming.

```
///
```

/// e.g. minRange=10, maxRange=100 no less than 10 values and no more that 100 values can be viewed

```
/// at once without scrolling.
```

```
///
```

/// If you call this method, chart must have data or it has no effect.

```
@objc open func setVisibleXRange(minXRange: Double, maxXRange: Double)
```

```
{
```

```
    let minScale = _xAxis.axisRange / maxXRange
```

```
    let maxScale = _xAxis.axisRange / minXRange
```

```
    _viewPortHandler.setMinMaxScaleX(
```

```
        minScaleX: CGFloat(minScale),
```

```

        maxScaleX: CGFloat(maxScale))

    }

    /// Sets the size of the area (range on the y-axis) that should be maximum visible at once.

    ///

    /// - Parameters:

    ///   - yRange:

    ///   - axis: - the axis for which this limit should apply

    @objc open func setVisibleYRangeMaximum(_ maxYRange: Double, axis:
YAxis.AxisDependency)

    {

        let yScale = getAxisRange(axis: axis) / maxYRange

        _viewPortHandler.setMinimumScaleY(CGFloat(yScale))

    }

    /// Sets the size of the area (range on the y-axis) that should be minimum visible at once,
    no further zooming in possible.

    ///

```

```

/// - Parameters:

/// - yRange:

/// - axis: - the axis for which this limit should apply

@objc open func setVisibleYRangeMinimum(_ minYRange: Double, axis:
YAxis.AxisDependency)

{

    let yScale = getAxisRange(axis: axis) / minYRange

    _viewPortHandler.setMaximumScaleY(CGFloat(yScale))

}

/// Limits the maximum and minimum y range that can be visible by pinching and
zooming.

///

/// - Parameters:

/// - minYRange:

/// - maxYRange:

/// - axis:

@objc open func setVisibleYRange(minYRange: Double, maxYRange: Double, axis:
YAxis.AxisDependency)

```

```

{

    let minScale = getAxisRange(axis: axis) / minYRange

    let maxScale = getAxisRange(axis: axis) / maxYRange

    _viewPortHandler.setMinMaxScaleY(minScaleY: CGFloat(minScale), maxScaleY:
CGFloat(maxScale))

}

```

/// Moves the left side of the current viewport to the specified x-value.

/// This also refreshes the chart by calling setNeedsDisplay().

@objc open func moveViewToX(_ xValue: Double)

```

{

    let job = MoveViewJob(

        viewPortHandler: viewPortHandler,

        xValue: xValue,

        yValue: 0.0,

        transformer: getTransformer(forAxis: .left),

        view: self)

```

```

        addViewportJob(job)

    }

    /// Centers the viewport to the specified y-value on the y-axis.

    /// This also refreshes the chart by calling setNeedsDisplay().

    ///

    /// - Parameters:

    /// - yValue:

    /// - axis: - which axis should be used as a reference for the y-axis

    @objc open func moveViewToY(_ yValue: Double, axis: YAxis.AxisDependency)

    {

        let yInView = getAxisRange(axis: axis) / Double(_viewPortHandler.scaleY)

        let job = MoveViewJob(

            viewPortHandler: viewPortHandler,

            xValue: 0.0,

            yValue: yValue + yInView / 2.0,

```

```

        transformer: getTransformer(forAxis: axis),

        view: self)

    addViewportJob(job)

}

/// This will move the left side of the current viewport to the specified x-value on the x-
axis, and center the viewport to the specified y-value on the y-axis.

/// This also refreshes the chart by calling setNeedsDisplay().

///

/// - Parameters:

/// - xValue:

/// - yValue:

/// - axis: - which axis should be used as a reference for the y-axis

@objc open func moveViewTo(xValue: Double, yValue: Double, axis:
YAxis.AxisDependency)

{

    let yInView = getAxisRange(axis: axis) / Double(_viewPortHandler.scaleY)

```

```

let job = MoveViewJob(

    viewportHandler: viewportHandler,

    xValue: xValue,

    yValue: yValue + yInView / 2.0,

    transformer: getTransformer(forAxis: axis),

    view: self)

addViewportJob(job)

}

```

/// This will move the left side of the current viewport to the specified x-position and center the viewport to the specified y-position animated.

/// This also refreshes the chart by calling setNeedsDisplay().

///

/// - Parameters:

/// - xValue:

/// - yValue:

```

/// - axis: which axis should be used as a reference for the y-axis

/// - duration: the duration of the animation in seconds

/// - easing:

@objc open func moveViewToAnimated(

    xValue: Double,

    yValue: Double,

    axis: YAxis.AxisDependency,

    duration: TimeInterval,

    easing: ChartEasingFunctionBlock?)

{

    let bounds = valueForTouchPoint(

        point: CGPoint(x: viewPortHandler.contentLeft, y: viewPortHandler.contentTop),

        axis: axis)

    let yInView = getAxisRange(axis: axis) / Double(_viewPortHandler.scaleY)

    let job = AnimatedMoveViewJob(

```



```

        viewportHandler: viewportHandler,

        xValue: xValue,

        yValue: yValue + yInView / 2.0,

        transformer: getTransformer(forAxis: axis),

        view: self,

        xOrigin: bounds.x,

        yOrigin: bounds.y,

        duration: duration,

        easing: easing)

    addViewportJob(job)

}

```

/// This will move the left side of the current viewport to the specified x-position and center the viewport to the specified y-position animated.

/// This also refreshes the chart by calling setNeedsDisplay().

///

/// - Parameters:

```

/// - xValue:

/// - yValue:

/// - axis: which axis should be used as a reference for the y-axis

/// - duration: the duration of the animation in seconds

/// - easing:

@objc open func moveViewToAnimated(

    xValue: Double,

    yValue: Double,

    axis: YAxis.AxisDependency,

    duration: TimeInterval,

    easingOption: ChartEasingOption)

{

    moveViewToAnimated(xValue: xValue, yValue: yValue, axis: axis, duration:
duration, easing: easingFunctionFromOption(easingOption))

}

/// This will move the left side of the current viewport to the specified x-position and
center the viewport to the specified y-position animated.

```

```

/// This also refreshes the chart by calling setNeedsDisplay().

///

/// - Parameters:

/// - xValue:

/// - yValue:

/// - axis: which axis should be used as a reference for the y-axis

/// - duration: the duration of the animation in seconds

/// - easing:

@objc open func moveViewToAnimated(

    xValue: Double,

    yValue: Double,

    axis: YAxis.AxisDependency,

    duration: TimeInterval)

{

    moveViewToAnimated(xValue: xValue, yValue: yValue, axis: axis, duration:
duration, easingOption: .easeInOutSine)

}

```

```
/// This will move the center of the current viewport to the specified x-value and y-value.
```

```
/// This also refreshes the chart by calling setNeedsDisplay().
```

```
///
```

```
/// - Parameters:
```

```
/// - xValue:
```

```
/// - yValue:
```

```
/// - axis: - which axis should be used as a reference for the y-axis
```

```
@objc open func centerViewTo(
```

```
    xValue: Double,
```

```
    yValue: Double,
```

```
    axis: YAxis.AxisDependency)
```

```
{
```

```
    let yInView = getAxisRange(axis: axis) / Double(_viewportHandler.scaleY)
```

```
    let xInView = xAxis.axisRange / Double(_viewportHandler.scaleX)
```

```
    let job = MoveViewJob(
```

```
        viewportHandler: viewportHandler,
```

```
xValue: xValue - xInView / 2.0,  
  
yValue: yValue + yInView / 2.0,  
  
transformer: getTransformer(forAxis: axis),  
  
view: self)
```

```
addViewportJob(job)
```

```
}
```

/// This will move the center of the current viewport to the specified x-value and y-value animated.

```
///
```

```
/// - Parameters:
```

```
/// - xValue:
```

```
/// - yValue:
```

```
/// - axis: which axis should be used as a reference for the y-axis
```

```
/// - duration: the duration of the animation in seconds
```

```
/// - easing:
```

```
@objc open func centerViewToAnimated(
```

```

xValue: Double,

yValue: Double,

axis: YAxis.AxisDependency,

duration: TimeInterval,

easing: ChartEasingFunctionBlock?)

{

let bounds = valueForTouchPoint(

    point: CGPoint(x: viewPortHandler.contentLeft, y: viewPortHandler.contentTop),

    axis: axis)

let yInView = getAxisRange(axis: axis) / Double(_viewPortHandler.scaleY)

let xInView = xAxis.axisRange / Double(_viewPortHandler.scaleX)

let job = AnimatedMoveViewJob(

    viewPortHandler: viewPortHandler,

    xValue: xValue - xInView / 2.0,

    yValue: yValue + yInView / 2.0,

```

```

        transformer: getTransformer(forAxis: axis),

        view: self,

        xOrigin: bounds.x,

        yOrigin: bounds.y,

        duration: duration,

        easing: easing)

    addViewportJob(job)
}

```

/// This will move the center of the current viewport to the specified x-value and y-value animated.

```
///
```

```
/// - Parameters:
```

```
/// - xValue:
```

```
/// - yValue:
```

```
/// - axis: which axis should be used as a reference for the y-axis
```

```
/// - duration: the duration of the animation in seconds
```

```

/// - easing:

@objc open func centerViewToAnimated(

    xValue: Double,

    yValue: Double,

    axis: YAxis.AxisDependency,

    duration: TimeInterval,

    easingOption: ChartEasingOption)

{

    centerViewToAnimated(xValue: xValue, yValue: yValue, axis: axis, duration:
duration, easing: easingFunctionFromOption(easingOption))

}

/// This will move the center of the current viewport to the specified x-value and y-value
animated.

///

/// - Parameters:

/// - xValue:

/// - yValue:

```



```

/// - axis: which axis should be used as a reference for the y-axis

/// - duration: the duration of the animation in seconds

/// - easing:

@objc open func centerViewToAnimated(

    xValue: Double,

    yValue: Double,

    axis: YAxis.AxisDependency,

    duration: TimeInterval)

{

    centerViewToAnimated(xValue: xValue, yValue: yValue, axis: axis, duration:
duration, easingOption: .easeInOutSine)

}

/// Sets custom offsets for the current `ChartViewPort` (the offsets on the sides of the
actual chart window). Setting this will prevent the chart from automatically calculating it's
offsets. Use `resetViewPortOffsets()` to undo this.

/// ONLY USE THIS WHEN YOU KNOW WHAT YOU ARE DOING, else use
`setExtraOffsets(...)`.

@objc open func setViewPortOffsets(left: CGFloat, top: CGFloat, right: CGFloat,
bottom: CGFloat)

```

```

{

    _customViewPortEnabled = true

    if Thread.isMainThread

    {

        self._viewPortHandler.restrainViewPort(offsetLeft: left, offsetTop: top,
offsetRight: right, offsetBottom: bottom)

        prepareOffsetMatrix()

        prepareValuePxMatrix()

    }

    else

    {

        DispatchQueue.main.async(execute: {

            self.setViewPortOffsets(left: left, top: top, right: right, bottom: bottom)

        })

    }

}

```

/// Resets all custom offsets set via `setViewPortOffsets(...)` method. Allows the chart to again calculate all offsets automatically.

```
@objc open func resetViewPortOffsets()
```

```
{
```

```
    _customViewPortEnabled = false
```

```
    calculateOffsets()
```

```
}
```

// MARK: - Accessors

/// - Returns: The range of the specified axis.

```
@objc open func getAxisRange(axis: YAxis.AxisDependency) -> Double
```

```
{
```

```
    if axis == .left
```

```
    {
```

```
        return leftAxis.axisRange
```

```
    }
```

```
    else
```

```

{

    return rightAxis.axisRange

}

}

```

/// - Returns: The position (in pixels) the provided Entry has inside the chart view

```

@objc open func getPosition(entry e: ChartDataEntry, axis: YAxis.AxisDependency) ->
CGPoint

```

```

{

    var vals = CGPoint(x: CGFloat(e.x), y: CGFloat(e.y))

    getTransformer(forAxis: axis).pointValueToPixel(&vals)

    return vals

}

```

/// is dragging enabled? (moving the chart with the finger) for the chart (this does not affect scaling).

```
@objc open var dragEnabled: Bool
```

```
{  
  
    get  
  
    {  
  
        return _dragXEnabled || _dragYEnabled  
  
    }  
  
    set  
  
    {  
  
        _dragYEnabled = newValue  
  
        _dragXEnabled = newValue  
  
    }  
  
}
```

/// is dragging enabled? (moving the chart with the finger) for the chart (this does not affect scaling).

```
@objc open var isDragEnabled: Bool
```

```
{  
  
    return dragEnabled  
  
}
```

```
}
```

```
/// is dragging on the X axis enabled?
```

```
@objc open var dragXEnabled: Bool
```

```
{
```

```
    get
```

```
    {
```

```
        return _dragXEnabled
```

```
    }
```

```
    set
```

```
    {
```

```
        _dragXEnabled = newValue
```

```
    }
```

```
}
```

```
/// is dragging on the Y axis enabled?
```

```
@objc open var dragYEnabled: Bool
```

```

{

    get

    {

        return _dragYEnabled

    }

    set

    {

        _dragYEnabled = newValue

    }

}

```

/// is scaling enabled? (zooming in and out by gesture) for the chart (this does not affect dragging).

```

@objc open func setScaleEnabled(_ enabled: Bool)

{

    if _scaleXEnabled != enabled || _scaleYEnabled != enabled

    {

        _scaleXEnabled = enabled

```

```

        _scaleYEnabled = enabled

        #if !os(tvOS)

            _pinchGestureRecognizer.isEnabled = _pinchZoomEnabled || _scaleXEnabled ||
            _scaleYEnabled

        #endif

    }

}

@objc open var scaleXEnabled: Bool

{

    get

    {

        return _scaleXEnabled

    }

    set

    {

        if _scaleXEnabled != newValue

        {

```



```

        _scaleXEnabled = newValue

        #if !os(tvOS)

            _pinchGestureRecognizer.isEnabled = _pinchZoomEnabled || _scaleXEnabled ||
            _scaleYEnabled

        #endif

    }

}

}

```

```

@objc open var scaleYEnabled: Bool

```

```

{

    get

    {

        return _scaleYEnabled

    }

    set

    {

        if _scaleYEnabled != newValue

```

```

    {

        _scaleYEnabled = newValue

        #if !os(tvOS)

            _pinchGestureRecognizer.isEnabled = _pinchZoomEnabled || _scaleXEnabled ||
            _scaleYEnabled

        #endif

    }

}

}

}

```

```

@objc open var isScaleXEnabled: Bool { return scaleXEnabled }

```

```

@objc open var isScaleYEnabled: Bool { return scaleYEnabled }

```

```

/// flag that indicates if double tap zoom is enabled or not

```

```

@objc open var doubleTapToZoomEnabled: Bool

```

```

{

```

```

    get

```

```

    {

```

```

        return _doubleTapToZoomEnabled

    }

    set

    {

        if _doubleTapToZoomEnabled != newValue

        {

            _doubleTapToZoomEnabled = newValue

            _doubleTapGestureRecognizer.isEnabled = _doubleTapToZoomEnabled

        }

    }

}

```

```

/// default: true

```

```

/// `true` if zooming via double-tap is enabled `false` if not.

```

```

@objc open var isDoubleTapToZoomEnabled: Bool

```

```

{

    return doubleTapToZoomEnabled
}

```

```
}
```

```
/// flag that indicates if highlighting per dragging over a fully zoomed out chart is enabled
```

```
@objc open var highlightPerDragEnabled = true
```

```
/// If set to true, highlighting per dragging over a fully zoomed out chart is enabled
```

```
/// You might want to disable this when using inside a `NSUIScrollView`
```

```
///
```

```
/// default: true
```

```
@objc open var isHighlightPerDragEnabled: Bool
```

```
{
```

```
    return highlightPerDragEnabled
```

```
}
```

```
/// default: true
```

```
/// `true` if drawing the grid background is enabled, `false` if not.
```

```
@objc open var isDrawGridBackgroundEnabled: Bool
```

```

{

    return drawGridBackgroundEnabled

}

/// **default**: false

/// `true` if drawing the borders rectangle is enabled, `false` if not.

@objc open var isDrawBordersEnabled: Bool

{

    return drawBordersEnabled

}

/// - Returns: The x and y values in the chart at the given touch point

/// (encapsulated in a `CGPoint`). This method transforms pixel coordinates to

/// coordinates / values in the chart. This is the opposite method to

/// `getPixelsForValues(...)`.

@objc open func valueForTouchPoint(point pt: CGPoint, axis: YAxis.AxisDependency)
-> CGPoint

{

```

```

        return getTransformer(forAxis: axis).valueForTouchPoint(pt)

    }

    /// Transforms the given chart values into pixels. This is the opposite

    /// method to `valueForTouchPoint(...)`.

    @objc open func pixelForValues(x: Double, y: Double, axis: YAxis.AxisDependency)
    -> CGPoint

    {

        return getTransformer(forAxis: axis).pixelForValues(x: x, y: y)

    }

    /// - Returns: The Entry object displayed at the touched position of the chart

    @objc open func getEntryByTouchPoint(point pt: CGPoint) -> ChartDataEntry!

    {

        if let h = getHighlightByTouchPoint(pt)

        {

            return _data!.entryForHighlight(h)

        }

    }

```

```

        return nil
    }

    /// - Returns: The DataSet object displayed at the touched position of the chart

    @objc open func getDataSetByTouchPoint(point pt: CGPoint) ->
    IBarLineScatterCandleBubbleChartDataSet?

    {

        let h = getHighlightByTouchPoint(pt)

        if h !== nil

        {

            return _data?.getDataSetByIndex(h!.dataSetIndex) as?
            IBarLineScatterCandleBubbleChartDataSet

        }

        return nil
    }

    /// The current x-scale factor

    @objc open var scaleX: CGFloat

```

```

{

    if _viewPortHandler === nil

    {

        return 1.0

    }

    return _viewPortHandler.scaleX

}

```

/// The current y-scale factor

```

@objc open var scaleY: CGFloat

{

    if _viewPortHandler === nil

    {

        return 1.0

    }

    return _viewPortHandler.scaleY

}

```



```
/// if the chart is fully zoomed out, return true
```

```
@objc open var isFullyZoomedOut: Bool { return  
_viewPortHandler.isFullyZoomedOut }
```

```
/// - Returns: The y-axis object to the corresponding AxisDependency. In the
```

```
/// horizontal bar-chart, LEFT == top, RIGHT == BOTTOM
```

```
@objc open func getAxis(_ axis: YAxis.AxisDependency) -> YAxis
```

```
{  
  
    if axis == .left  
  
    {  
  
        return leftAxis  
  
    }  
  
    else  
  
    {  
  
        return rightAxis  
  
    }  
  
}
```

/// flag that indicates if pinch-zoom is enabled. if true, both x and y axis can be scaled simultaneously with 2 fingers, if false, x and y axis can be scaled separately

```
@objc open var pinchZoomEnabled: Bool

{

    get

    {

        return _pinchZoomEnabled

    }

    set

    {

        if _pinchZoomEnabled != newValue

        {

            _pinchZoomEnabled = newValue

            #if !os(tvOS)

                _pinchGestureRecognizer.isEnabled = _pinchZoomEnabled || _scaleXEnabled ||
_scaleYEnabled

            #endif

        }

    }

}
```

```

    }

}

}

/// **default**: false

/// `true` if pinch-zoom is enabled, `false` if not

@objc open var isPinchZoomEnabled: Bool { return pinchZoomEnabled }

/// Set an offset in dp that allows the user to drag the chart over it's

/// bounds on the x-axis.

@objc open func setDragOffsetX(_ offset: CGFloat)

{

    _viewPortHandler.setDragOffsetX(offset)

}

/// Set an offset in dp that allows the user to drag the chart over it's

/// bounds on the y-axis.

```

```

@objc open func setDragOffsetY(_ offset: CGFloat)

{

    _viewPortHandler.setDragOffsetY(offset)

}


/// `true` if both drag offsets (x and y) are zero or smaller.

@objc open var hasNoDragOffset: Bool { return _viewPortHandler.hasNoDragOffset }


open override var chartYMax: Double

{

    return max(leftAxis._axisMaximum, rightAxis._axisMaximum)

}


open override var chartYMin: Double

{

    return min(leftAxis._axisMinimum, rightAxis._axisMinimum)

}

```

```
/// `true` if either the left or the right or both axes are inverted.
```

```
@objc open var isAnyAxisInverted: Bool
```

```
{
```

```
    return leftAxis.isInverted || rightAxis.isInverted
```

```
}
```

```
/// flag that indicates if auto scaling on the y axis is enabled.
```

```
/// if yes, the y axis automatically adjusts to the min and max y values of the current x  
axis range whenever the viewport changes
```

```
@objc open var autoScaleMinMaxEnabled: Bool
```

```
{
```

```
    get { return _autoScaleMinMaxEnabled }
```

```
    set { _autoScaleMinMaxEnabled = newValue }
```

```
}
```

```
/// default: false
```

```
/// `true` if auto scaling on the y axis is enabled.
```

```
@objc open var isAutoScaleMinMaxEnabled : Bool { return autoScaleMinMaxEnabled }
```

```
/// Sets a minimum width to the specified y axis.
```

```
@objc open func setYAxisMinWidth(_ axis: YAxis.AxisDependency, width: CGFloat)
```

```
{
```

```
    if axis == .left
```

```
    {
```

```
        leftAxis.minWidth = width
```

```
    }
```

```
    else
```

```
    {
```

```
        rightAxis.minWidth = width
```

```
    }
```

```
}
```

```
/// **default**: 0.0
```

```
///
```

/// - Returns: The (custom) minimum width of the specified Y axis.

@objc open func getYAxisMinWidth(_ axis: YAxis.AxisDependency) -> CGFloat

{

if axis == .left

{

return leftAxis.minWidth

}

else

{

return rightAxis.minWidth

}

}

/// Sets a maximum width to the specified y axis.

/// Zero (0.0) means there's no maximum width

@objc open func setYAxisMaxWidth(_ axis: YAxis.AxisDependency, width: CGFloat)

{

if axis == .left

```

{

    leftAxis.maxWidth = width

}

else

{

    rightAxis.maxWidth = width

}

}

```

/// Zero (0.0) means there's no maximum width

///

/// ****default****: 0.0 (no maximum specified)

///

/// - Returns: The (custom) maximum width of the specified Y axis.

@objc open func getYAxisMaxWidth(_ axis: YAxis.AxisDependency) -> CGFloat

```

{

    if axis == .left

```



```

{

    return leftAxis.maxWidth

}

else

{

    return rightAxis.maxWidth

}

}

```

/// - Returns the width of the specified y axis.

```

@objc open func getYAxisWidth(_ axis: YAxis.AxisDependency) -> CGFloat

{

    if axis == .left

    {

        return leftAxis.requiredSize().width

    }

    else

```

```

    {

        return rightAxis.requiredSize().width

    }

}

```

// MARK: - BarLineScatterCandleBubbleChartDataProvider

/// - Returns: The Transformer class that contains all matrices and is

/// responsible for transforming values into pixels on the screen and

/// backwards.

open func getTransformer(forAxis axis: YAxis.AxisDependency) -> Transformer

```

{

    if axis == .left

    {

        return _leftAxisTransformer

    }

    else

```

```
{  
  
    return _rightAxisTransformer  
  
}  
  
}
```

/// the number of maximum visible drawn values on the chart only active when
`drawValuesEnabled` is enabled

open override var maxVisibleCount: Int

```
{  
  
    get  
  
    {  
  
        return _maxVisibleCount  
  
    }  
  
    set  
  
    {  
  
        _maxVisibleCount = newValue  
  
    }  
  
}
```

```
open func isInverted(axis: YAxis.AxisDependency) -> Bool
```

```
{
```

```
    return getAxis(axis).isInverted
```

```
}
```

```
/// The lowest x-index (value on the x-axis) that is still visible on the chart.
```

```
open var lowestVisibleX: Double
```

```
{
```

```
    var pt = CGPoint(
```

```
        x: viewPortHandler.contentLeft,
```

```
        y: viewPortHandler.contentBottom)
```

```
    getTransformer(forAxis: .left).pixelToValues(&pt)
```

```
    return max(xAxis._axisMinimum, Double(pt.x))
```

```
}
```

```

    /// The highest x-index (value on the x-axis) that is still visible on the chart.

    open var highestVisibleX: Double

    {

        var pt = CGPoint(

            x: viewPortHandler.contentRight,

            y: viewPortHandler.contentBottom)

        getTransformer(forAxis: .left).pixelToValues(&pt)

        return min(xAxis._axisMaximum, Double(pt.x))

    }

}

//

// BarLineScatterCandleBubbleChartData.swift

// Charts

//

```

```
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```
// A port of MPAndroidChart for iOS
```

```
// Licensed under Apache License 2.0
```

```
//
```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
open class BarLineScatterCandleBubbleChartData: ChartData
```

```
{
```

```
    public override init()
```

```
    {
```

```
        super.init()
```

```
    }
```

```
    public override init(dataSets: [IChartDataSet]?)
```

```

    {

        super.init(dataSets: dataSets)

    }

}

//

// BarLineScatterCandleBubbleChartDataProvider.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

```

@objc

public protocol BarLineScatterCandleBubbleChartDataProvider: ChartDataProvider

{

func getTransformer(forAxis: YAxis.AxisDependency) -> Transformer

func isInverted(axis: YAxis.AxisDependency) -> Bool

var lowestVisibleX: Double { get }

var highestVisibleX: Double { get }

}

//

// BarLineScatterCandleBubbleChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0


```

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics


open class BarLineScatterCandleBubbleChartDataSet: ChartDataSet,
IBarLineScatterCandleBubbleChartDataSet

{

    // MARK: - Data functions and accessors


    // MARK: - Styling functions and accessors


    open var highlightColor = NSUIColor(red: 255.0/255.0, green: 187.0/255.0, blue:
115.0/255.0, alpha: 1.0)

    open var highlightLineWidth = CGFloat(0.5)

```

```

open var highlightLineDashPhase = CGFloat(0.0)

open var highlightLineDashLengths: [CGFloat]?

// MARK: - NSCopying

open override func copy(with zone: NSZone? = nil) -> Any
{
    let copy = super.copy(with: zone) as! BarLineScatterCandleBubbleChartDataSet

    copy.highlightColor = highlightColor

    copy.highlightLineWidth = highlightLineWidth

    copy.highlightLineDashPhase = highlightLineDashPhase

    copy.highlightLineDashLengths = highlightLineDashLengths

    return copy
}

}

//

// BarLineScatterCandleBubbleRenderer.swift

```

```

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc(BarLineScatterCandleBubbleChartRenderer)

open class BarLineScatterCandleBubbleRenderer: DataRenderer

{

    internal var _xBounds = XBounds() // Reusable XBounds object

```

```

public override init(animator: Animator, viewPortHandler: ViewPortHandler)

{

    super.init(animator: animator, viewPortHandler: viewPortHandler)

}

/// Checks if the provided entry object is in bounds for drawing considering the current
animation phase.

internal func isInBoundsX(entry e: ChartDataEntry, dataSet:
IBarLineScatterCandleBubbleChartDataSet) -> Bool

{

    let entryIndex = dataSet.entryIndex(entry: e)

    return Double(entryIndex) < Double(dataSet.entryCount) * animator.phaseX

}

/// Calculates and returns the x-bounds for the given DataSet in terms of index in their
values array.

/// This includes minimum and maximum visible x, as well as range.

internal func xBounds(chart: BarLineScatterCandleBubbleChartDataProvider,

    dataSet: IBarLineScatterCandleBubbleChartDataSet,

```

```
    animator: Animator?) -> XBounds
```

```
{
```

```
    return XBounds(chart: chart, dataSet: dataSet, animator: animator)
```

```
}
```

```
/// - Returns: `true` if the DataSet values should be drawn, `false` if not.
```

```
internal fun shouldDrawValues(forDataSet set: IChartDataSet) -> Bool
```

```
{
```

```
    return set.isVisible && (set.isDrawValuesEnabled || set.isDrawIconsEnabled)
```

```
}
```

```
/// Class representing the bounds of the current viewport in terms of indices in the values  
array of a DataSet.
```

```
open class XBounds
```

```
{
```

```
    /// minimum visible entry index
```

```
    open var min: Int = 0
```

```
/// maximum visible entry index
```

```
open var max: Int = 0
```

```
/// range of visible entry indices
```

```
open var range: Int = 0
```

```
public init()
```

```
{
```

```
}
```

```
public init(chart: BarLineScatterCandleBubbleChartDataProvider,
```

```
            dataSet: IBarLineScatterCandleBubbleChartDataSet,
```

```
            animator: Animator?)
```

```
{
```

```
    self.set(chart: chart, dataSet: dataSet, animator: animator)
```

```
}
```

/// Calculates the minimum and maximum x values as well as the range between them.

```
open func set(chart: BarLineScatterCandleBubbleChartDataProvider,  
  
             dataSet: IBarLineScatterCandleBubbleChartDataSet,  
  
             animator: Animator?)  
  
{  
  
    let phaseX = Swift.max(0.0, Swift.min(1.0, animator?.phaseX ?? 1.0))  
  
  
    let low = chart.lowestVisibleX  
  
    let high = chart.highestVisibleX  
  
  
  
    let entryFrom = dataSet.entryForXValue(low, closestToY: .nan, rounding: .down)  
  
    let entryTo = dataSet.entryForXValue(high, closestToY: .nan, rounding: .up)  
  
  
  
    self.min = entryFrom == nil ? 0 : dataSet.entryIndex(entry: entryFrom!)  
  
    self.max = entryTo == nil ? 0 : dataSet.entryIndex(entry: entryTo!)  
  
    range = Int(Double(self.max - self.min) * phaseX)
```

```

    }

}

}

```

```

extension BarLineScatterCandleBubbleRenderer.XBounds: RangeExpression {

```

```

    public func relative<C>(to collection: C) -> Swift.Range<Int>

```

```

        where C : Collection, Bound == C.Index

```

```

    {

```

```

        return Swift.Range<Int>(min...min + range)

```

```

    }

```

```

    public func contains(_ element: Int) -> Bool {

```

```

        return (min...min + range).contains(element)

```

```

    }

```

```

}

```

```

extension BarLineScatterCandleBubbleRenderer.XBounds: Sequence {

```



```

public struct Iterator: IteratorProtocol {

    private var iterator: IndexingIterator<ClosedRange<Int>>

    fileprivate init(min: Int, max: Int) {

        self.iterator = (min...max).makeIterator()

    }

    public mutating func next() -> Int? {

        return self.iterator.next()

    }

}

public func makeIterator() -> Iterator {

    return Iterator(min: self.min, max: self.min + self.range)

}

}

```

```

extension BarLineScatterCandleBubbleRenderer.XBounds:
CustomDebugStringConvertible

{

    public var debugDescription: String

    {

        return "min:\(self.min), max:\(self.max), range:\(self.range)"

    }

}

//

// BubbleChartData.swift

// Charts

//

// Bubble chart implementation:

// Copyright 2015 Pierre-Marc Airoldi

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

```

```
import Foundation
```

```
import CoreGraphics
```

```
open class BubbleChartData: BarLineScatterCandleBubbleChartData
```

```
{
```

```
    public override init()
```

```
    {
```

```
        super.init()
```

```
    }
```

```
    public override init(dataSets: [IChartDataSet]?)
```

```
    {
```

```
        super.init(dataSets: dataSets)
```

```
    }
```

```
    /// Sets the width of the circle that surrounds the bubble when highlighted for all DataSet  
    objects this data object contains
```

```

@objc open func setHighlightCircleWidth(_ width: CGFloat)

{

    (_dataSets as? [IBubbleChartDataSet])?.forEach { $0.highlightCircleWidth = width }

}

}

//

// BubbleDataEntry.swift

// Charts

//

// Bubble chart implementation:

// Copyright 2015 Pierre-Marc Airoldi

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

```

```

import CoreGraphics

open class BubbleChartDataEntry: ChartDataEntry

{

    /// The size of the bubble.

    @objc open var size = CGFloat(0.0)

    public required init()

    {

        super.init()

    }

    /// - Parameters:

    /// - x: The index on the x-axis.

    /// - y: The value on the y-axis.

    /// - size: The size of the bubble.

    @objc public init(x: Double, y: Double, size: CGFloat)

```

```

{

    super.init(x: x, y: y)

    self.size = size

}

/// - Parameters:

/// - x: The index on the x-axis.

/// - y: The value on the y-axis.

/// - size: The size of the bubble.

/// - data: Spot for additional data this Entry represents.

@objc public convenience init(x: Double, y: Double, size: CGFloat, data: Any?)

{

    self.init(x: x, y: y, size: size)

    self.data = data

}

```

```
/// - Parameters:
```

```
/// - x: The index on the x-axis.
```

```
/// - y: The value on the y-axis.
```

```
/// - size: The size of the bubble.
```

```
/// - icon: icon image
```

```
@objc public convenience init(x: Double, y: Double, size: CGFloat, icon: NSUIImage?)
```

```
{
```

```
    self.init(x: x, y: y, size: size)
```

```
    self.icon = icon
```

```
}
```

```
/// - Parameters:
```

```
/// - x: The index on the x-axis.
```

```
/// - y: The value on the y-axis.
```

```
/// - size: The size of the bubble.
```

```
/// - icon: icon image
```

```
/// - data: Spot for additional data this Entry represents.
```

```
@objc public convenience init(x: Double, y: Double, size: CGFloat, icon: NSUIImage?,
data: Any?)
```

```
{

    self.init(x: x, y: y, size: size)

    self.icon = icon

    self.data = data

}
```

```
// MARK: NSCopying
```

```
open override func copy(with zone: NSZone? = nil) -> Any
```

```
{

    let copy = super.copy(with: zone) as! BubbleChartDataEntry

    copy.size = size

    return copy

}

}

//
```



```

// BubbleChartDataProvider.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc

public protocol BubbleChartDataProvider:
BarLineScatterCandleBubbleChartDataProvider

{

    var bubbleData: BubbleChartData? { get }

```

```

} //

// BubbleChartDataSet.swift

// Charts

//

// Bubble chart implementation:

// Copyright 2015 Pierre-Marc Airoldi

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class BubbleChartDataSet: BarLineScatterCandleBubbleChartDataSet,
IBubbleChartDataSet

{

```

```
// MARK: - Data functions and accessors
```

```
internal var _maxSize = CGFloat(0.0)
```

```
open var maxSize: CGFloat { return _maxSize }
```

```
@objc open var normalizeSizeEnabled: Bool = true
```

```
open var isNormalizeSizeEnabled: Bool { return normalizeSizeEnabled }
```

```
open override func calcMinMax(entry e: ChartDataEntry)
```

```
{
```

```
    guard let e = e as? BubbleChartDataEntry
```

```
    else { return }
```

```
    super.calcMinMax(entry: e)
```

```
    let size = e.size
```

```

    if size > _maxSize

    {

        _maxSize = size

    }

}

```

// MARK: - Styling functions and accessors

/// Sets/gets the width of the circle that surrounds the bubble when highlighted

```

open var highlightCircleWidth: CGFloat = 2.5

```

// MARK: - NSCopying

```

open override func copy(with zone: NSZone? = nil) -> Any

```

```

{

    let copy = super.copy(with: zone) as! BubbleChartDataSet

    copy._xMin = _xMin

```

```

        copy._xMax = _xMax

        copy._maxSize = _maxSize

        copy.normalizeSizeEnabled = normalizeSizeEnabled

        copy.highlightCircleWidth = highlightCircleWidth

        return copy

    }

}

//

// BubbleChartRenderer.swift

// Charts

//

// Bubble chart implementation:

// Copyright 2015 Pierre-Marc Airoldi

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

```

```
import Foundation
```

```
import CoreGraphics
```

```
open class BubbleChartRenderer: BarLineScatterCandleBubbleRenderer
```

```
{
```

```
    /// A nested array of elements ordered logically (i.e not in visual/drawing order) for use  
    with VoiceOver.
```

```
    private lazy var accessibilityOrderedElements: [[NSUIAccessibilityElement]] =  
    accessibilityCreateEmptyOrderedElements()
```

```
    @objc open weak var dataProvider: BubbleChartDataProvider?
```

```
    @objc public init(dataProvider: BubbleChartDataProvider, animator: Animator,  
    viewPortHandler: ViewPortHandler)
```

```
{
```

```
    super.init(animator: animator, viewPortHandler: viewPortHandler)
```

```
    self.dataProvider = dataProvider
```

```
}
```

```
open override func drawData(context: CGContext)
```

```
{
```

```
    guard
```

```
        let dataProvider = dataProvider,
```

```
        let bubbleData = dataProvider.bubbleData
```

```
    else { return }
```

```
    // If we redraw the data, remove and repopulate accessible elements to update label  
    values and frames
```

```
    accessibleChartElements.removeAll()
```

```
    accessibilityOrderedElements = accessibilityCreateEmptyOrderedElements()
```

```
    // Make the chart header the first element in the accessible elements array
```

```
    if let chart = dataProvider as? BubbleChartView {
```

```
        let element = createAccessibleHeader(usingChart: chart,
```

```
            andData: bubbleData,
```

```

        withDefaultDescription: "Bubble Chart")

        accessibleChartElements.append(element)

    }

    for (i, set) in (bubbleData.dataSets as! [IBubbleChartDataSet]).enumerated() where
set.isVisible

    {

        drawDataSet(context: context, dataSet: set, dataSetIndex: i)

    }

    // Merge nested ordered arrays into the single accessibleChartElements.

    accessibleChartElements.append(contentsOf: accessibilityOrderedElements.flatMap
{ $0 } )

    accessibilityPostLayoutChangedNotification()

}

private func getShapeSize(

    entrySize: CGFloat,

```



```

    maxSize: CGFloat,

    reference: CGFloat,

    normalizeSize: Bool) -> CGFloat

{

    let factor: CGFloat = normalizeSize

        ? ((maxSize == 0.0) ? 1.0 : sqrt(entrySize / maxSize))

        : entrySize

    let shapeSize: CGFloat = reference * factor

    return shapeSize

}

```

```

private var _pointBuffer = CGPoint()

```

```

private var _sizeBuffer = [CGPoint](repeating: CGPoint(), count: 2)

```

```

@objc open func drawDataSet(context: CGContext, dataSet: IBubbleChartDataSet,
dataSetIndex: Int)

```

```

{

    guard let dataProvider = dataProvider else { return }

```

```
let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)
```

```
let phaseY = animator.phaseY
```

```
_xBounds.set(chart: dataProvider, dataSet: dataSet, animator: animator)
```

```
let valueToPixelMatrix = trans.valueToPixelMatrix
```

```
_sizeBuffer[0].x = 0.0
```

```
_sizeBuffer[0].y = 0.0
```

```
_sizeBuffer[1].x = 1.0
```

```
_sizeBuffer[1].y = 0.0
```

```
trans.pointValuesToPixel(&_sizeBuffer)
```

```
context.saveGState()
```

```

defer { context.restoreGState() }

let normalizeSize = dataSet.isNormalizeSizeEnabled

// calculate the full width of 1 step on the x-axis

let maxBubbleWidth: CGFloat = abs(_sizeBuffer[1].x - _sizeBuffer[0].x)

let maxBubbleHeight: CGFloat = abs(viewPortHandler.contentBottom -
viewPortHandler.contentTop)

let referenceSize: CGFloat = min(maxBubbleHeight, maxBubbleWidth)

for j in _xBounds

{

    guard let entry = dataSet.entryForIndex(j) as? BubbleChartDataEntry else
    { continue }

    _pointBuffer.x = CGFloat(entry.x)

    _pointBuffer.y = CGFloat(entry.y * phaseY)

    _pointBuffer = _pointBuffer.applying(valueToPixelMatrix)

```

```
let shapeSize = getShapeSize(entrySize: entry.size, maxSize: dataSet.maxSize,  
reference: referenceSize, normalizeSize: normalizeSize)
```

```
let shapeHalf = shapeSize / 2.0
```

```
guard
```

```
viewPortHandler.isInBoundsTop(_pointBuffer.y + shapeHalf),
```

```
viewPortHandler.isInBoundsBottom(_pointBuffer.y - shapeHalf),
```

```
viewPortHandler.isInBoundsLeft(_pointBuffer.x + shapeHalf)
```

```
else { continue }
```

```
guard viewPortHandler.isInBoundsRight(_pointBuffer.x - shapeHalf) else { break }
```

```
let color = dataSet.color(atIndex: j)
```

```
let rect = CGRect(
```

```
x: _pointBuffer.x - shapeHalf,
```

```
y: _pointBuffer.y - shapeHalf,
```

```
        width: shapeSize,  
  
        height: shapeSize  
    )
```

```
context.setFillColor(color.cgColor)
```

```
context.fillEllipse(in: rect)
```

```
        // Create and append the corresponding accessibility element to  
accessibilityOrderedElements
```

```
if let chart = dataProvider as? BubbleChartView
```

```
{
```

```
    let element = createAccessibleElement(withIndex: j,
```

```
        container: chart,
```

```
        dataSet: dataSet,
```

```
        dataSetIndex: dataSetIndex,
```

```
        shapeSize: shapeSize)
```

```
    { (element) in
```

```
        element.accessibilityFrame = rect
```

```
}
```

```
accessibilityOrderedElements[dataSetIndex].append(element)
```

```
}
```

```
}
```

```
}
```

```
open override func drawValues(context: CGContext)
```

```
{
```

```
    guard let
```

```
        dataProvider = dataProvider,
```

```
        let bubbleData = dataProvider.bubbleData,
```

```
        isDrawingValuesAllowed(dataProvider: dataProvider),
```

```
        let dataSets = bubbleData.dataSets as? [IBubbleChartDataSet]
```

```
    else { return }
```

```
    let phaseX = max(0.0, min(1.0, animator.phaseX))
```

```
let phaseY = animator.phaseY
```

```
var pt = CGPoint()
```

```
for i in 0..
```

```
{
```

```
    let dataSet = dataSets[i]
```

```
    guard
```

```
        shouldDrawValues(forDataSet: dataSet),
```

```
        let formatter = dataSet.valueFormatter
```

```
        else { continue }
```

```
    let alpha = phaseX == 1 ? phaseY : phaseX
```

```
    _xBounds.set(chart: dataProvider, dataSet: dataSet, animator: animator)
```

```

let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)

let valueToPixelMatrix = trans.valueToPixelMatrix


let iconsOffset = dataSet.iconsOffset


for j in _xBounds

{

    guard let e = dataSet.entryForIndex(j) as? BubbleChartDataEntry else { break }


    let                                     valueTextColor                                     =
dataSet.valueTextColorAt(j).withAlphaComponent(CGFloat(alpha))


    pt.x = CGFloat(e.x)

    pt.y = CGFloat(e.y * phaseY)

    pt = pt.applying(valueToPixelMatrix)


    guard viewPortHandler.isInBoundsRight(pt.x) else { break }

```



```
guard
```

```
viewPortHandler.isInBoundsLeft(pt.x),
```

```
viewPortHandler.isInBoundsY(pt.y)
```

```
else { continue }
```

```
let text = formatter.stringForValue(
```

```
    Double(e.size),
```

```
    entry: e,
```

```
    dataSetIndex: i,
```

```
    viewPortHandler: viewPortHandler)
```

```
// Larger font for larger bubbles?
```

```
let valueFont = dataSet.valueFont
```

```
let lineHeight = valueFont.lineHeight
```

```
if dataSet.isDrawValuesEnabled
```

```
{
```

```

ChartUtils.drawText(

    context: context,

    text: text,

    point: CGPoint(

        x: pt.x,

        y: pt.y - (0.5 * lineHeight)),

    align: .center,

    attributes: [NSAttributedString.Key.font: valueFont,
NSAttributedString.Key.foregroundColor: valueTextColor])

}

```

```

if let icon = e.icon, dataSet.isDrawIconsEnabled

```

```

{

```

```

    ChartUtils.drawImage(context: context,

        image: icon,

        x: pt.x + iconsOffset.x,

        y: pt.y + iconsOffset.y,

        size: icon.size)

```

```
    }  
  }  
}  
}
```

```
open override fun drawExtras(context: CGContext)
```

```
{  
  
}
```

```
open override fun drawHighlighted(context: CGContext, indices: [Highlight])
```

```
{
```

```
    guard
```

```
        let dataProvider = dataProvider,
```

```
        let bubbleData = dataProvider.bubbleData
```

```
    else { return }
```

```

context.saveGState()

defer { context.restoreGState() }

let phaseY = animator.phaseY

for high in indices
{

    guard

        let dataSet = bubbleData.getDataSetByIndex(high.dataSetIndex) as?
IBubbleChartDataSet,

        dataSet.isEnabled,

        let entry = dataSet.entryForXValue(high.x, closestToY: high.y) as?
BubbleChartDataEntry,

        isInBoundsX(entry: entry, dataSet: dataSet)

    else { continue }

let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)

```

```
_sizeBuffer[0].x = 0.0
```

```
_sizeBuffer[0].y = 0.0
```

```
_sizeBuffer[1].x = 1.0
```

```
_sizeBuffer[1].y = 0.0
```

```
trans.pointValuesToPixel(&_sizeBuffer)
```

```
let normalizeSize = dataSet.isNormalizeSizeEnabled
```

```
// calculate the full width of 1 step on the x-axis
```

```
let maxBubbleWidth: CGFloat = abs(_sizeBuffer[1].x - _sizeBuffer[0].x)
```

```
let maxBubbleHeight: CGFloat = abs(viewPortHandler.contentBottom -  
viewPortHandler.contentTop)
```

```
let referenceSize: CGFloat = min(maxBubbleHeight, maxBubbleWidth)
```

```
_pointBuffer.x = CGFloat(entry.x)
```

```
_pointBuffer.y = CGFloat(entry.y * phaseY)
```

```
trans.pointValueToPixel(&_pointBuffer)
```

```
let shapeSize = getShapeSize(entrySize: entry.size, maxSize: dataSet.maxSize,  
reference: referenceSize, normalizeSize: normalizeSize)
```

```
let shapeHalf = shapeSize / 2.0
```

```
guard
```

```
viewPortHandler.isInBoundsTop(_pointBuffer.y + shapeHalf),
```

```
viewPortHandler.isInBoundsBottom(_pointBuffer.y - shapeHalf),
```

```
viewPortHandler.isInBoundsLeft(_pointBuffer.x + shapeHalf)
```

```
else { continue }
```

```
guard viewPortHandler.isInBoundsRight(_pointBuffer.x - shapeHalf) else { break }
```

```
let originalColor = dataSet.color(atIndex: Int(entry.x))
```

```
var h: CGFloat = 0.0
```

```
var s: CGFloat = 0.0
```

```
var b: CGFloat = 0.0
```

```
var a: CGFloat = 0.0
```

```
originalColor.getHue(&h, saturation: &s, brightness: &b, alpha: &a)
```

```
let color = NSUIColor(hue: h, saturation: s, brightness: b * 0.5, alpha: a)
```

```
let rect = CGRect(
```

```
    x: _pointBuffer.x - shapeHalf,
```

```
    y: _pointBuffer.y - shapeHalf,
```

```
    width: shapeSize,
```

```
    height: shapeSize)
```

```
context.setLineWidth(dataSet.highlightCircleWidth)
```

```
context.setStrokeColor(color.cgColor)
```

```
context.strokeEllipse(in: rect)
```

```
high.setDraw(x: _pointBuffer.x, y: _pointBuffer.y)
```

```
}
```

```
}
```

/// Creates a nested array of empty subarrays each of which will be populated with NSUIAccessibilityElements.

```
private func accessibilityCreateEmptyOrderedElements() ->
[[NSUIAccessibilityElement]]
```

```
{
```

```
    guard let chart = dataProvider as? BubbleChartView else { return [] }
```

```
    let dataSetCount = chart.bubbleData?.dataSetCount ?? 0
```

```
    return Array(repeating: [NSUIAccessibilityElement](),
```

```
        count: dataSetCount)
```

```
}
```

/// Creates an NSUIAccessibleElement representing individual bubbles location and relative size.

```
private func createAccessibleElement(withIndex idx: Int,
```

```
    container: BubbleChartView,
```



```

        dataSet: IBubbleChartDataSet,

        dataSetIndex: Int,

        shapeSize: CGFloat,

        modifier: (NSUIAccessibilityElement) -> () ->
NSUIAccessibilityElement

{

    let element = NSUIAccessibilityElement(accessibilityContainer: container)

    let xAxis = container.xAxis

    guard let e = dataSet.entryForIndex(idx) else { return element }

    guard let dataProvider = dataProvider else { return element }

    // NOTE: The formatter can cause issues when the x-axis labels are consecutive ints.

    // i.e. due to the Double conversion, if there are more than one data set that are grouped,

    // there is the possibility of some labels being rounded up. A floor() might fix this, but
    seems to be a brute force solution.

    let label = xAxis.valueFormatter?.stringForValue(e.x, axis: xAxis) ?? "\(e.x)"

```

```

let elementValueText = dataSet.valueFormatter?.stringForValue(e.y,

                                entry: e,

                                dataSetIndex: dataSetIndex,

                                viewportHandler:    viewportHandler)    ??

"\(e.y)"

let dataSetCount = dataProvider.bubbleData?.dataSetCount ?? -1

let doesContainMultipleDataSets = dataSetCount > 1

    element.accessibilityLabel = "\(\(doesContainMultipleDataSets ? (dataSet.label ?? "")
+ ", " : "") \(\label): \(\elementValueText), bubble size: \(\String(format: "%.2f",
(shapeSize/dataSet.maxSize) * 100)) %"

    modifier(element)

    return element

}

}

//

```

```

// BubbleChartView.swift

// Charts

//

// Bubble chart implementation:

// Copyright 2015 Pierre-Marc Airoldi

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class BubbleChartView: BarLineChartViewBase, BubbleChartDataProvider

{

    open override func initialize()

    {

```

```
super.initialize()
```

```
        renderer = BubbleChartRenderer(dataProvider: self, animator: _animator,  
viewPortHandler: _viewPortHandler)  
  
    }
```

```
// MARK: - BubbleChartDataProvider
```

```
open var bubbleData: BubbleChartData? { return _data as? BubbleChartData }  
  
}  
  
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `'CNMutablePostalAddress+Convenience'` method creates
'CNMutablePostalAddress' from a 'CLPlacemark'.

```
*/
```

```
import MapKit
```

```
import Contacts
```

```
extension CNMutablePostalAddress {
```

```
    /// Constructs a `CNMutablePostalAddress` from a `CLPlacemark`
```

```
    convenience init(placemark: CLPlacemark) {
```

```
        self.init()
```

```
        self.street = (placemark.subThoroughfare ?? "") + " " + (placemark.thoroughfare ??  
        "")
```

```
        self.city = placemark.locality ?? ""
```

```
        self.state = placemark.administrativeArea ?? ""
```

```
        self.postalCode = placemark.postalCode ?? ""
```

```
        self.country = placemark.country ?? ""
```

```
    }
```

```
}
```

```
//
```

```
// CachedResponseHandler.swift
```

//

// Copyright (c) 2019 Alamofire Software Foundation (<http://alamofire.org/>)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

```
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,  
DAMAGES OR OTHER
```

```
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,  
ARISING FROM,
```

```
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER  
DEALINGS IN
```

```
// THE SOFTWARE.
```

```
//
```

```
import Foundation
```

```
/// A type that handles whether the data task should store the HTTP response in the cache.
```

```
public protocol CachedResponseHandler {
```

```
    /// Determines whether the HTTP response should be stored in the cache.
```

```
    ///
```

```
    /// The `completion` closure should be passed one of three possible options:
```

```
    ///
```

```
    /// 1. The cached response provided by the server (this is the most common use case).
```

/// 2. A modified version of the cached response (you may want to modify it in some way before caching).

/// 3. A `nil` value to prevent the cached response from being stored in the cache.

///

/// - Parameters:

/// - task: The data task whose request resulted in the cached response.

/// - response: The cached response to potentially store in the cache.

/// - completion: The closure to execute containing cached response, a modified response, or `nil`.

```
func dataTask(_ task: URLSessionDataTask,

              willCacheResponse response: CachedURLResponse,

              completion: @escaping (CachedURLResponse?) -> Void)

}
```

// MARK: -

/// `ResponseCacher` is a convenience `CachedResponseHandler` making it easy to cache, not cache, or modify a cached

/// response.


```

public struct ResponseCacher {

    /// Defines the behavior of the `ResponseCacher` type.

    public enum Behavior {

        /// Stores the cached response in the cache.

        case cache

        /// Prevents the cached response from being stored in the cache.

        case doNotCache

        /// Modifies the cached response before storing it in the cache.

        case modify((URLSessionDataTask, CachedURLResponse) ->
CachedURLResponse?)

    }

    /// Returns a `ResponseCacher` with a follow `Behavior`.

    public static let cache = ResponseCacher(behavior: .cache)

    /// Returns a `ResponseCacher` with a do not follow `Behavior`.

    public static let doNotCache = ResponseCacher(behavior: .doNotCache)

    /// The `Behavior` of the `ResponseCacher`.

```

```

public let behavior: Behavior

/// Creates a `ResponseCacher` instance from the `Behavior`.
///
/// - Parameter behavior: The `Behavior`.

public init(behavior: Behavior) {

    self.behavior = behavior

}

}

extension ResponseCacher: CachedResponseHandler {

    public func dataTask(_ task: URLSessionDataTask,

                        willCacheResponse response: CachedURLResponse,

                        completion: @escaping (CachedURLResponse?) -> Void) {

        switch behavior {

        case .cache:

            completion(response)

```

```
        case .doNotCache:

            completion(nil)

        case let .modify(closure):

            let response = closure(task, response)

            completion(response)

        }

    }

}

//

// CandleChartData.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts
```

```
//

import Foundation

open class CandleChartData: BarLineScatterCandleBubbleChartData

{

    public override init()

    {

        super.init()

    }

    public override init(dataSets: [IChartDataSet]?)

    {

        super.init(dataSets: dataSets)

    }

}

//
```

```
// CandleChartDataEntry.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//
```

```
import Foundation
```

```
open class CandleChartDataEntry: ChartDataEntry
```

```
{
```

```
    /// shadow-high value
```

```
    @objc open var high = Double(0.0)
```

```
/// shadow-low value
```

```
@objc open var low = Double(0.0)
```

```
/// close value
```

```
@objc open var close = Double(0.0)
```

```
/// open value
```

```
@objc open var open = Double(0.0)
```

```
public required init()
```

```
{
```

```
    super.init()
```

```
}
```

```
@objc public init(x: Double, shadowH: Double, shadowL: Double, open: Double, close:  
Double)
```

```
{
```

```
    super.init(x: x, y: (shadowH + shadowL) / 2.0)
```

```
self.high = shadowH

self.low = shadowL

self.open = open

self.close = close

}
```

```
@objc public convenience init(x: Double, shadowH: Double, shadowL: Double, open: Double, close: Double, icon: NSUIImage?)
```

```
{

    self.init(x: x, shadowH: shadowH, shadowL: shadowL, open: open, close: close)

    self.icon = icon

}
```

```
@objc public convenience init(x: Double, shadowH: Double, shadowL: Double, open: Double, close: Double, data: Any?)
```

```
{

    self.init(x: x, shadowH: shadowH, shadowL: shadowL, open: open, close: close)
```

```
        self.data = data

    }
```

```
@objc public convenience init(x: Double, shadowH: Double, shadowL: Double, open: Double, close: Double, icon: NSUIImage?, data: Any?)
```

```
{

    self.init(x: x, shadowH: shadowH, shadowL: shadowL, open: open, close: close)

    self.icon = icon

    self.data = data

}
```

```
/// The overall range (difference) between shadow-high and shadow-low.
```

```
@objc open var shadowRange: Double
```

```
{

    return abs(high - low)

}
```

```
/// The body size (difference between open and close).
```



```
@objc open var bodyRange: Double
```

```
{  
  
    return abs(open - close)  
  
}
```

```
/// the center value of the candle. (Middle value between high and low)
```

```
open override var y: Double
```

```
{  
  
    get  
  
    {  
  
        return super.y  
  
    }  
  
    set  
  
    {  
  
        super.y = (high + low) / 2.0  
  
    }  
  
}
```

```

// MARK: NSCopying

open override func copy(with zone: NSZone? = nil) -> Any

{

    let copy = super.copy(with: zone) as! CandleChartDataEntry

    copy.high = high

    copy.low = low

    copy.open = open

    copy.close = close

    return copy

}

}

//

// CandleChartDataProvider.swift

// Charts

//

```

```
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```
// A port of MPAndroidChart for iOS
```

```
// Licensed under Apache License 2.0
```

```
//
```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
@objc
```

```
public protocol  
BarLineScatterCandleBubbleChartDataProvider
```

```
CandleChartDataProvider:
```

```
{  
  
    var candleData: CandleChartData? { get }  
  
} //
```

```
// CandleChartDataSet.swift
```

```
// Charts
```

```
//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class CandleChartDataSet: LineScatterCandleRadarChartDataSet,
ICandleChartDataSet

{

    public required init()

    {
```

```

        super.init()

    }

    public override init(entries: [ChartDataEntry]?, label: String?)

    {

        super.init(entries: entries, label: label)

    }

    // MARK: - Data functions and accessors

    open override func calcMinMax(entry e: ChartDataEntry)

    {

        guard let e = e as? CandleChartDataEntry

        else { return }

        if e.low < _yMin

        {

```

```
    _yMin = e.low
```

```
}
```

```
if e.high > _yMax
```

```
{
```

```
    _yMax = e.high
```

```
}
```

```
calcMinMaxX(entry: e)
```

```
}
```

```
open override fun calcMinMaxY(entry e: ChartDataEntry)
```

```
{
```

```
    guard let e = e as? CandleChartDataEntry
```

```
        else { return }
```

```
    if e.high < _yMin
```

```
{  
  
    _yMin = e.high  
  
}  
  
if e.high > _yMax  
  
{  
  
    _yMax = e.high  
  
}  
  
if e.low < _yMin  
  
{  
  
    _yMin = e.low  
  
}  
  
if e.low > _yMax  
  
{  
  
    _yMax = e.low  
  
}  
  
}
```

```
// MARK: - Styling functions and accessors
```

```
/// the space between the candle entries
```

```
///
```

```
/// **default**: 0.1 (10%)
```

```
private var _barSpace = CGFloat(0.1)
```

```
/// the space that is left out on the left and right side of each candle,
```

```
/// **default**: 0.1 (10%), max 0.45, min 0.0
```

```
open var barSpace: CGFloat
```

```
{
```

```
    get
```

```
    {
```

```
        return _barSpace
```

```
    }
```

```
    set
```



```

    {

        _barSpace = newValue.clamped(to: 0...0.45)

    }

}

```

```

/// should the candle bars show?

```

```

/// when false, only "ticks" will show

```

```

///

```

```

/// **default**: true

```

```

open var showCandleBar: Bool = true

```

```

/// the width of the candle-shadow-line in pixels.

```

```

///

```

```

/// **default**: 1.5

```

```

open var shadowWidth = CGFloat(1.5)

```

```

/// the color of the shadow line

```

```
open var shadowColor: NSUIColor?
```

```
/// use candle color for the shadow
```

```
open var shadowColorSameAsCandle = false
```

```
/// Is the shadow color same as the candle color?
```

```
open var isShadowColorSameAsCandle: Bool { return shadowColorSameAsCandle }
```

```
/// color for open == close
```

```
open var neutralColor: NSUIColor?
```

```
/// color for open > close
```

```
open var increasingColor: NSUIColor?
```

```
/// color for open < close
```

```
open var decreasingColor: NSUIColor?
```

```

/// Are increasing values drawn as filled?

/// increasing candlesticks are traditionally hollow

open var increasingFilled = false


/// Are increasing values drawn as filled?

open var isIncreasingFilled: Bool { return increasingFilled }


/// Are decreasing values drawn as filled?

/// decreasing candlesticks are traditionally filled

open var decreasingFilled = true


/// Are decreasing values drawn as filled?

open var isDecreasingFilled: Bool { return decreasingFilled }

}

//

// CandleStickChartRenderer.swift

// Charts

```

```

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class CandleStickChartRenderer: LineScatterCandleRadarRenderer

{

    @objc open weak var dataProvider: CandleChartDataProvider?

    @objc public init(dataProvider: CandleChartDataProvider, animator: Animator,
viewPortHandler: ViewPortHandler)

    {

```

```

super.init(animator: animator, viewPortHandler: viewPortHandler)

self.dataProvider = dataProvider
}

open override func drawData(context: CGContext)

{

    guard let dataProvider = dataProvider, let candleData = dataProvider.candleData else
    { return }

    // If we redraw the data, remove and repopulate accessible elements to update label
    values and frames

    accessibleChartElements.removeAll()

    // Make the chart header the first element in the accessible elements array

    if let chart = dataProvider as? CandleStickChartView {

        let element = createAccessibleHeader(usingChart: chart,

                                            andData: candleData,

```

```
withDefaultDescription: "CandleStick Chart")
```

```
accessibleChartElements.append(element)
```

```
}
```

```
for set in candleData.dataSets as! [ICandleChartDataSet] where set.isVisible
```

```
{
```

```
drawDataSet(context: context, dataSet: set)
```

```
}
```

```
}
```

```
private var _shadowPoints = [CGPoint](repeating: CGPoint(), count: 4)
```

```
private var _rangePoints = [CGPoint](repeating: CGPoint(), count: 2)
```

```
private var _openPoints = [CGPoint](repeating: CGPoint(), count: 2)
```

```
private var _closePoints = [CGPoint](repeating: CGPoint(), count: 2)
```

```
private var _bodyRect = CGRect()
```

```
private var _lineSegments = [CGPoint](repeating: CGPoint(), count: 2)
```

```

@objc open func drawDataSet(context: CGContext, dataSet: ICandleChartDataSet)

{

    guard

        let dataProvider = dataProvider

    else { return }


    let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)


    let phaseY = animator.phaseY


    let barSpace = dataSet.barSpace


    let showCandleBar = dataSet.showCandleBar


    _xBounds.set(chart: dataProvider, dataSet: dataSet, animator: animator)


    context.saveGState()


    context.setLineWidth(dataSet.shadowWidth)

```

```

for j in _xBounds

{

    // get the entry

    guard let e = dataSet.entryForIndex(j) as? CandleChartDataEntry else { continue }

    let xPos = e.x

    let open = e.open

    let close = e.close

    let high = e.high

    let low = e.low

    let doesContainMultipleDataSets = (dataProvider.candleData?.dataSets.count ??
1) > 1

    var accessibilityMovementDescription = "neutral"

    var accessibilityRect = CGRect(x: CGFloat(xPos) + 0.5 - barSpace,

                                   y: CGFloat(low * phaseY),

```



```

        width: (2 * barSpace) - 1.0,

        height: (CGFloat(abs(high - low) * phaseY)))

trans.rectValueToPixel(&accessibilityRect)

if showCandleBar

{

    // calculate the shadow

    _shadowPoints[0].x = CGFloat(xPos)

    _shadowPoints[1].x = CGFloat(xPos)

    _shadowPoints[2].x = CGFloat(xPos)

    _shadowPoints[3].x = CGFloat(xPos)

    if open > close

    {

        _shadowPoints[0].y = CGFloat(high * phaseY)

        _shadowPoints[1].y = CGFloat(open * phaseY)

```

```

        _shadowPoints[2].y = CGFloat(low * phaseY)

        _shadowPoints[3].y = CGFloat(close * phaseY)

    }

    else if open < close

    {

        _shadowPoints[0].y = CGFloat(high * phaseY)

        _shadowPoints[1].y = CGFloat(close * phaseY)

        _shadowPoints[2].y = CGFloat(low * phaseY)

        _shadowPoints[3].y = CGFloat(open * phaseY)

    }

    else

    {

        _shadowPoints[0].y = CGFloat(high * phaseY)

        _shadowPoints[1].y = CGFloat(open * phaseY)

        _shadowPoints[2].y = CGFloat(low * phaseY)

        _shadowPoints[3].y = _shadowPoints[1].y

    }

```

```
trans.pointValuesToPixel(&_shadowPoints)
```

```
// draw the shadows
```

```
var shadowColor: NSUIColor! = nil
```

```
if dataSet.shadowColorSameAsCandle
```

```
{
```

```
    if open > close
```

```
    {
```

```
        shadowColor = dataSet.decreasingColor ?? dataSet.color(atIndex: j)
```

```
    }
```

```
    else if open < close
```

```
    {
```

```
        shadowColor = dataSet.increasingColor ?? dataSet.color(atIndex: j)
```

```
    }
```

```
else
```

```

    {

        shadowColor = dataSet.neutralColor ?? dataSet.color(atIndex: j)

    }

}

```

```

if shadowColor === nil

```

```

{

    shadowColor = dataSet.shadowColor ?? dataSet.color(atIndex: j)

}

```

```

context.setStrokeColor(shadowColor.cgColor)

```

```

context.strokeLineSegments(between: _shadowPoints)

```

```

// calculate the body

```

```

_bodyRect.origin.x = CGFloat(xPos) - 0.5 + barSpace

```

```

_bodyRect.origin.y = CGFloat(close * phaseY)

```

```
_bodyRect.size.width = (CGFloat(xPos) + 0.5 - barSpace) - _bodyRect.origin.x
```

```
_bodyRect.size.height = CGFloat(open * phaseY) - _bodyRect.origin.y
```

```
trans.rectValueToPixel(&_bodyRect)
```

```
// draw body differently for increasing and decreasing entry
```

```
if open > close
```

```
{
```

```
    accessibilityMovementDescription = "decreasing"
```

```
    let color = dataSet.decreasingColor ?? dataSet.color(atIndex: j)
```

```
    if dataSet.isDecreasingFilled
```

```
    {
```

```
        context.setFillColor(color.cgColor)
```

```
        context.fill(_bodyRect)
```

```

    }

    else

    {

        context.setStrokeColor(color.cgColor)

        context.stroke(_bodyRect)

    }

}

else if open < close

{

    accessibilityMovementDescription = "increasing"

    let color = dataSet.increasingColor ?? dataSet.color(atIndex: j)

    if dataSet.isIncreasingFilled

    {

        context.setFillColor(color.cgColor)

        context.fill(_bodyRect)

```

```

    }

    else

    {

        context.setStrokeColor(color.cgColor)

        context.stroke(_bodyRect)

    }

}

else

{

    let color = dataSet.neutralColor ?? dataSet.color(atIndex: j)

    context.setStrokeColor(color.cgColor)

    context.stroke(_bodyRect)

}

}

else

{

```

```

_rangePoints[0].x = CGFloat(xPos)

_rangePoints[0].y = CGFloat(high * phaseY)

_rangePoints[1].x = CGFloat(xPos)

_rangePoints[1].y = CGFloat(low * phaseY)


_openPoints[0].x = CGFloat(xPos) - 0.5 + barSpace

_openPoints[0].y = CGFloat(open * phaseY)

_openPoints[1].x = CGFloat(xPos)

_openPoints[1].y = CGFloat(open * phaseY)


_closePoints[0].x = CGFloat(xPos) + 0.5 - barSpace

_closePoints[0].y = CGFloat(close * phaseY)

_closePoints[1].x = CGFloat(xPos)

_closePoints[1].y = CGFloat(close * phaseY)


trans.pointValuesToPixel(&_rangePoints)

trans.pointValuesToPixel(&_openPoints)

```



```

trans.pointValuesToPixel(&_closePoints)

// draw the ranges

var barColor: NSUIColor! = nil

if open > close

{

    accessibilityMovementDescription = "decreasing"

    barColor = dataSet.decreasingColor ?? dataSet.color(atIndex: j)

}

else if open < close

{

    accessibilityMovementDescription = "increasing"

    barColor = dataSet.increasingColor ?? dataSet.color(atIndex: j)

}

else

{

```

```

        barColor = dataSet.neutralColor ?? dataSet.color(atIndex: j)

    }

    context.setStrokeColor(barColor.cgColor)

    context.strokeLineSegments(between: _rangePoints)

    context.strokeLineSegments(between: _openPoints)

    context.strokeLineSegments(between: _closePoints)

}

let axElement = createAccessibleElement(withIndex: j,

                                       container: dataProvider,

                                       dataSet: dataSet)

{ (element) in

    element.accessibilityLabel = "\(\(doesContainMultipleDataSets ?
"\(dataSet.label ?? "Dataset")" : "") " + "\(\(xPos) - \(\(accessibilityMovementDescription).
low: \(\(low), high: \(\(high), opening: \(\(open), closing: \(\(close))"

    element.accessibilityFrame = accessibilityRect

}

```

```
accessibleChartElements.append(axElement)
```

```
}
```

```
// Post this notification to let VoiceOver account for the redrawn frames
```

```
accessibilityPostLayoutChangedNotification()
```

```
context.restoreGState()
```

```
}
```

```
open override func drawValues(context: CGContext)
```

```
{
```

```
    guard
```

```
        let dataProvider = dataProvider,
```

```
        let candleData = dataProvider.candleData
```

```
    else { return }
```

```

// if values are drawn

if isDrawingValuesAllowed(dataProvider: dataProvider)

{

    let dataSets = candleData.dataSets


    let phaseY = animator.phaseY


    var pt = CGPoint()


    for i in 0 ..< dataSets.count

    {

        guard let

            dataSet = dataSets[i] as? IBarLineScatterCandleBubbleChartDataSet,

            shouldDrawValues(forDataSet: dataSet)

        else { continue }

```

```
let valueFont = dataSet.valueFont
```

```
guard let formatter = dataSet.valueFormatter else { continue }
```

```
let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)
```

```
let valueToPixelMatrix = trans.valueToPixelMatrix
```

```
let iconsOffset = dataSet.iconsOffset
```

```
_xBounds.set(chart: dataProvider, dataSet: dataSet, animator: animator)
```

```
let lineHeight = valueFont.lineHeight
```

```
let yOffset: CGFloat = lineHeight + 5.0
```

```
for j in _xBounds
```

```
{
```

```
    guard let e = dataSet.entryForIndex(j) as? CandleChartDataEntry else { break }
```

```

    pt.x = CGFloat(e.x)

    pt.y = CGFloat(e.high * phaseY)

    pt = pt.applying(valueToPixelMatrix)

    if (!viewPortHandler.isInBoundsRight(pt.x))

    {

        break

    }

    if (!viewPortHandler.isInBoundsLeft(pt.x)
|| !viewPortHandler.isInBoundsY(pt.y))

    {

        continue

    }

    if dataSet.isDrawValuesEnabled

    {

```

```

ChartUtils.drawText(

    context: context,

    text: formatter.stringForValue(

        e.high,

        entry: e,

        dataSetIndex: i,

        viewPortHandler: viewPortHandler),

    point: CGPoint(

        x: pt.x,

        y: pt.y - yOffset),

    align: .center,

    attributes: [NSAttributedString.Key.font: valueFont,
NSAttributedString.Key.foregroundColor: dataSet.valueTextColorAt(j)]

    }

    if let icon = e.icon, dataSet.isDrawIconsEnabled

    {

        ChartUtils.drawImage(context: context,

```

```

        image: icon,

        x: pt.x + iconsOffset.x,

        y: pt.y + iconsOffset.y,

        size: icon.size)

    }

}

}

}

}

```

```

open override func drawExtras(context: CGContext)

```

```

{

}

```

```

open override func drawHighlighted(context: CGContext, indices: [Highlight])

```

```

{

    guard

```



```

        let dataProvider = dataProvider,

        let candleData = dataProvider.candleData

        else { return }

    context.saveGState()

    for high in indices

    {

        guard

            let set = candleData.getDataSetByIndex(high.dataSetIndex) as?
ICandleChartDataSet,

            set.isHighlightEnabled

        else { continue }

        guard let e = set.entryForXValue(high.x, closestToY: high.y) as?
CandleChartDataEntry else { continue }

        if !isInBoundsX(entry: e, dataSet: set)

```

```
{  
  
    continue  
  
}
```

```
let trans = dataProvider.getTransformer(forAxis: set.axisDependency)
```

```
context.setStrokeColor(set.highlightColor.cgColor)
```

```
context.setLineWidth(set.highlightLineWidth)
```

```
if set.highlightLineDashLengths != nil
```

```
{  
  
    context.setLineDash(phase:      set.highlightLineDashPhase,      lengths:  
set.highlightLineDashLengths!)
```

```
}
```

```
else
```

```
{
```

```
    context.setLineDash(phase: 0.0, lengths: [])
```

```
}
```

```
let lowValue = e.low * Double(animotor.phaseY)
```

```
let highValue = e.high * Double(animotor.phaseY)
```

```
let y = (lowValue + highValue) / 2.0
```

```
let pt = trans.pixelForValues(x: e.x, y: y)
```

```
high.setDraw(pt: pt)
```

```
// draw the lines
```

```
drawHighlightLines(context: context, point: pt, set: set)
```

```
}
```

```
context.restoreGState()
```

```
}
```

```
private func createAccessibleElement(withIndex idx: Int,
```

```

        container: CandleChartDataProvider,

        dataSet: ICandleChartDataSet,

        modifier: (NSUIAccessibilityElement) -> () ->
NSUIAccessibilityElement {

    let element = NSUIAccessibilityElement(accessibilityContainer: container)


    // The modifier allows changing of traits and frame depending on highlight, rotation,
    etc

    modifier(element)


    return element

}

}

//

// CandleStickChartView.swift

// Charts

//

```

```
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```
// A port of MPAndroidChart for iOS
```

```
// Licensed under Apache License 2.0
```

```
//
```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
/// Financial chart type that draws candle-sticks.
```

```
open class CandleStickChartView: BarLineChartViewBase, CandleChartDataProvider
```

```
{
```

```
    internal override func initialize()
```

```
    {
```

```
        super.initialize()
```

```
        renderer = CandleStickChartRenderer(dataProvider: self, animator: _animator,  
viewPortHandler: _viewPortHandler)
```

```
        self.xAxis.spaceMin = 0.5
```

```
        self.xAxis.spaceMax = 0.5
```

```
    }
```

```
// MARK: - CandleChartDataProvider
```

```
open var candleData: CandleChartData?
```

```
{
```

```
    return _data as? CandleChartData
```

```
}
```

```
}
```

```
/*
```

See LICENSE folder for this sample's licensing information.

Abstract:

A table cell that summarizes one characteristic.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
class CharacterCell: UITableViewCell {
```

```
    @IBOutlet weak var slider: UISlider!
```

```
    @IBOutlet weak var toggle: UISwitch!
```

```
    @IBOutlet weak var nameLabel: UILabel!
```

```
    @IBOutlet weak var valueLabel: UILabel!
```

```
    @IBOutlet weak var segmentedControl: UISegmentedControl!
```

```
    /// Writes to the characteristic as the user moves the slider back and forth.
```

```
    @IBAction func sliderDidChange(_ sender: UISlider) {
```

```
        guard let characteristic = characteristic else { return }
```

```

// Convert to a value in range, and on the grid.

let value = characteristic.valueFor(fraction: sender.value)


// Only write when there's a change.

guard

    let oldValue = characteristic.decimalValue,

    value != oldValue else { return }


// Write the new value.

characteristic.writeValue(value) { error in

    if let error = error {

        print(error.localizedDescription)

    } else {

        // Update the label to match.

        self.redrawValueLabel()

    }

```



```
}
```

```
}
```

```
/// Ensures the final state of the UI matches reality after the user finishes sliding.
```

```
@IBAction func sliderTouchUp(_ sender: Any) {
```

```
    readAndRedrawValue(animated: false)
```

```
}
```

```
/// Updates the characteristic value when the toggle switch changes state.
```

```
@IBAction func toggleDidChange(_ sender: UISwitch) {
```

```
    guard let characteristic = characteristic else { return }
```

```
    characteristic.writeValue(sender.isOn) { error in
```

```
        if let error = error {
```

```
            print(error.localizedDescription)
```

```
        }
```

```
    }
```

```
}
```

```
/// Updates the characteristic value when the segmented control changes state.
```

```
@IBAction func segDidChange(_ sender: UISegmentedControl) {
```

```
    guard let characteristic = characteristic else { return }
```

```
    characteristic.writeValue(sender.selectedSegmentIndex) { error in
```

```
        if let error = error {
```

```
            print(error.localizedDescription)
```

```
        }
```

```
    }
```

```
}
```

```
/// The characteristic that this cell displays.
```

```
var characteristic: HMCharacteristic? {
```

```
    // Update the UI when a new characteristic is set.
```

```

didSet {

    // A cell with no characteristic shows as empty.

    guard

        let characteristic = characteristic,

        let accessory = characteristic.service?.accessory else {

            nameLabel.text = nil

            valueLabel.text = nil

            slider.isHidden = true

            toggle.isHidden = true

            return

        }

    // Set the characteristic's name

    nameLabel.text = characteristic.displayName

    // Show the toggle for reachable, writeable, Boolean characteristics.

    toggle.isHidden = !accessory.isReachable || !characteristic.isBool
|| !characteristic.isWriteable

```

```

toggle.isOn = false

// Show the segmented control for writeable values with declared state names.

segmentedControl.isHidden = !(accessory.isReachable &&
characteristic.isWriteable && !characteristic.stateNames.isEmpty)

if !segmentedControl.isHidden {

    segmentedControl.removeAllSegments()

    for index in 0...(characteristic.stateNames.count - 1) {

        let name = characteristic.stateNames[index]

        segmentedControl.insertSegment(withTitle: name, at: index, animated: false)

    }

}

// Show the slider for other numbers.

//     slider.isHidden = !(segmentedControl.isHidden && accessory.isReachable &&
characteristic.isNumeric && characteristic.stateNames.isEmpty)

//     slider.isEnabled = characteristic.isWriteable

//     slider.value = 0

```

```

// Value label shows when toggle and segment are hidden.

valueLabel.isHidden = !(toggle.isHidden && segmentedControl.isHidden)

valueLabel.text = "—"

// Execute a read, and then update the UI with it.

readAndRedrawValue(animated: false)

}

}

/// Reads the characteristic value from the HomeKit database, and updates the UI.

func readAndRedrawValue(animated: Bool) {

    guard

        let characteristic = characteristic,

        characteristic.isReadable,

        let accessory = characteristic.service?.accessory,

        accessory.isReachable else { return }

```

```

characteristic.readValue { error in

    if let error = error {

        print(error.localizedDescription)

    } else {

        self.redrawValueLabel()

        self.redrawControls(animated: animated)

    }

}

}

}

/// Sets the cell's secondary text according to the current characteristic value.

func redrawValueLabel() {

    valueLabel.text = characteristic?.formattedValueString

}

/// Sets any visible graphical control to the current characteristic value.

```

```

func redrawControls(animated: Bool) {

    guard

        let characteristic = characteristic,

        let value = characteristic.value else { return }

    if !slider.isHidden {

        if let decimalValue = characteristic.decimalValue {

            let fraction = (decimalValue / characteristic.span) as NSNumber

            slider.setValue(fraction.floatValue, animated: animated)

        }

    } else if !toggle.isHidden {

        if let boolValue = value as? Bool {

            toggle.setOn(boolValue, animated: animated)

        }

    } else if !segmentedControl.isHidden {

```

```

        if let intValue = value as? Int {

            segmentedControl.selectedSegmentIndex = intValue

        }

    }

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

A table cell that summarizes one characteristic.

```

*/

```

```

import UIKit

```

```

import HomeKit

```

```

class CharacteristicCell: UITableViewCell {

```



```
@IBOutlet weak var slider: UISlider!
```

```
@IBOutlet weak var toggle: UISwitch!
```

```
@IBOutlet weak var nameLabel: UILabel!
```

```
@IBOutlet weak var valueLabel: UILabel!
```

```
@IBOutlet weak var segmentedControl: UISegmentedControl!
```

```
/// Writes to the characteristic as the user moves the slider back and forth.
```

```
@IBAction func sliderDidChange(_ sender: UISlider) {
```

```
    guard let characteristic = characteristic else { return }
```

```
    // Convert to a value in range, and on the grid.
```

```
    let value = characteristic.valueFor(fraction: sender.value)
```

```
    // Only write when there's a change.
```

```
    guard
```

```
        let oldValue = characteristic.decimalValue,
```

```

        value != oldValue else { return }

    // Write the new value.

    characteristic.writeValue(value) { error in

        if let error = error {

            print(error.localizedDescription)

        } else {

            // Update the label to match.

            self.redrawValueLabel()

        }

    }

}

/// Ensures the final state of the UI matches reality after the user finishes sliding.

@IBAction func sliderTouchUp(_ sender: Any) {

    readAndRedrawValue(animated: false)

}

```

```
/// Updates the characteristic value when the toggle switch changes state.
```

```
@IBAction func toggleDidChange(_ sender: UISwitch) {
```

```
    guard let characteristic = characteristic else { return }
```

```
    characteristic.writeValue(sender.isOn) { error in
```

```
        if let error = error {
```

```
            print(error.localizedDescription)
```

```
        }
```

```
    }
```

```
}
```

```
/// Updates the characteristic value when the segmented control changes state.
```

```
@IBAction func segDidChange(_ sender: UISegmentedControl) {
```

```
    guard let characteristic = characteristic else { return }
```

```
    characteristic.writeValue(sender.selectedSegmentIndex) { error in
```

```

        if let error = error {

            print(error.localizedDescription)

        }

    }

}

```

/// The characteristic that this cell displays.

```

var characteristic: HMCharacteristic? {

    // Update the UI when a new characteristic is set.

    didSet {

        // A cell with no characteristic shows as empty.

        guard

            let characteristic = characteristic,

            let accessory = characteristic.service?.accessory else {

                nameLabel.text = nil

                valueLabel.text = nil
            }
        }
    }
}

```

```

        slider.isHidden = true

        toggle.isHidden = true

        return
    }

    // Set the characteristic's name

    nameLabel.text = characteristic.displayName

    // Show the toggle for reachable, writeable, Boolean characteristics.

    toggle.isHidden    =    !accessory.isReachable    ||    !characteristic.isBool
|| !characteristic.isWriteable

    toggle.isOn = false

    // Show the segmented control for writeable values with declared state names.

    segmentedControl.isHidden    =    !(accessory.isReachable    &&
characteristic.isWriteable && !characteristic.stateNames.isEmpty)

    if !segmentedControl.isHidden {

        segmentedControl.removeAllSegments()

```

```

    for index in 0...(characteristic.stateNames.count - 1) {

        let name = characteristic.stateNames[index]

        segmentedControl.insertSegment(withTitle: name, at: index, animated: false)

    }

}

// Show the slider for other numbers.

slider.isHidden = !(segmentedControl.isHidden && accessory.isReachable &&
characteristic.isNumeric && characteristic.stateNames.isEmpty)

slider.isEnabled = characteristic.isWriteable

slider.value = 0

// Value label shows when toggle and segment are hidden.

valueLabel.isHidden = !(toggle.isHidden && segmentedControl.isHidden)

valueLabel.text = "—"

// Execute a read, and then update the UI with it.

readAndRedrawValue(animated: false)

```

```

    }

}

/// Reads the characteristic value from the HomeKit database, and updates the UI.

func readAndRedrawValue(animated: Bool) {

    guard

        let characteristic = characteristic,

        characteristic.isReadable,

        let accessory = characteristic.service?.accessory,

        accessory.isReachable else { return }

    characteristic.readValue { error in

        if let error = error {

            print(error.localizedDescription)

        } else {

            self.redrawValueLabel()

            self.redrawControls(animated: animated)

```

```

    }

}

}

```

/// Sets the cell's secondary text according to the current characteristic value.

```

func redrawValueLabel() {

    valueLabel.text = characteristic?.formattedValueString

}

```

/// Sets any visible graphical control to the current characteristic value.

```

func redrawControls(animated: Bool) {

    guard

        let characteristic = characteristic,

        let value = characteristic.value else { return }

    if !slider.isHidden {

        if let decimalValue = characteristic.decimalValue {

```



```

        let fraction = (decimalValue / characteristic.span) as NSNumber

        slider.setValue(fraction.floatValue, animated: animated)

    }

} else if !toggle.isHidden {

    if let boolValue = value as? Bool {

        toggle.setOn(boolValue, animated: animated)

    }

} else if !segmentedControl.isHidden {

    if let intValue = value as? Int {

        segmentedControl.selectedSegmentIndex = intValue

    }

}

}

}

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

`CharacteristicCell` is a superclass which represents the state of a HomeKit characteristic.

*/

import UIKit

import HomeKit

typealias CellValueType = NSCopying

/// Methods for handling cell reads and updates.

protocol CharacteristicCellDelegate {

/**

Called whenever the control within the cell updates its value.

- parameter cell: The cell which has updated its value.
- parameter newValue: The new value represented by the cell's control.
- parameter characteristic: The characteristic the cell represents.
- parameter immediate: Whether or not to update external values immediately.

For example, Slider cells should not update immediately upon value change, so their values are cached and updates are coalesced. Subclasses can decide whether or not their values are meant to be updated immediately.

*/

```
func characteristicCell(_ cell: CharacteristicCell, didUpdateValue value:
CellValueType, forCharacteristic characteristic: HMCharacteristic, immediate: Bool)
```

/**

Called when the characteristic cell needs to reload its value from an external source.

Consider using this call to look up values in memory or query them from an accessory.

- parameter cell: The cell requesting a value update.

- parameter characteristic: The characteristic for whose value the cell is asking.

- parameter completion: The closure that the cell provides to be called when values have been read successfully.

*/

```
func characteristicCell(_ cell: CharacteristicCell, readInitialValueForCharacteristic
characteristic: HMCharacteristic, completion: @escaping (CellValueType?, Error?) ->
Void)
```

```
}
```

/**

A 'UITableViewCell' subclass that displays the current value of an 'HMCharacteristic' and

notifies its delegate of changes. Subclasses of this class will provide additional controls to display different kinds of data.

*/

```
class CharacteristicCell: UITableViewCell {
```

```
/// An alpha percentage used when disabling cells.
```

```
static let DisabledAlpha: CGFloat = 0.4
```

```
/// Required init.
```

```
required init?(coder aDecoder: NSCoder) {
```

```
    super.init(coder: aDecoder)
```

```
}
```

```
/// Subclasses can return false if they have many frequent updates that should be deferred.
```

```
class var updatesImmediately: Bool {
```

```
    return true
```

```
}
```

```
// MARK: Properties
```

```
@IBOutlet weak var typeLabel: UILabel!
```

```
@IBOutlet weak var valueLabel: UILabel!
```

```
@IBOutlet weak var favoriteButton: UIButton!
```

```
@IBOutlet weak var favoriteButtonWidthConstraint: NSLayoutConstraint!
```

```
@IBOutlet weak var favoriteButtonHeightConstraint: NSLayoutConstraint!
```

```
/**
```

```
    Show / hide the favoriteButton and adjust the constraints
```

```
    to ensure proper layout.
```

```
*/
```

```
var showsFavorites = false {
```

```
    didSet {
```

```
        if showsFavorites {
```

```
            favoriteButton.isHidden = false
```

```
            favoriteButtonWidthConstraint.constant =  
favoriteButtonHeightConstraint.constant
```

```
        }
```

```
    else {
```

```
        favoriteButton.isHidden = true
```

```
        favoriteButtonWidthConstraint.constant = 15.0
```

```
    }
```

```
}
```

```
}
```

```
/**
```

```
- returns: `true` if the represented characteristic is reachable;
```

```
  `false` otherwise.
```

```
*/
```

```
var enabled: Bool {
```

```
  return (characteristic.service?.accessory?.isReachable ?? false)
```

```
}
```

```
/**
```

```
The value currently represented by the cell.
```

```
This is not necessarily the value of this cell's characteristic,
```

```
because the cell's value changes independently of the characteristic.
```

```
*/
```

```
var value: AnyObject?
```

```
/// The delegate that will respond to cell value changes.
```

```
var delegate: CharacteristicCellDelegate?
```

```
/**
```

```
    The characteristic represented by this cell.
```

```
    When this is set, the cell populates based on
```

```
    the characteristic's value and requests an initial value
```

```
    from its delegate.
```

```
*/
```

```
var characteristic: HMCharacteristic! {
```

```
    didSet {
```

```
        typeLabel.text = characteristic.localizedCharacteristicType
```

```
        selectionStyle = characteristic.isIdentify ? .default : .none
```



```

setValue(characteristic.value as? CellValueType, notify: false)

if characteristic.isWriteOnly {

    // Don't read the value for write-only characteristics.

    return

}

// Set initial state of the favorite button

//     favoriteButton.isSelected = characteristic.isFavorite

// "Enable" the cell if the accessory is reachable or we are displaying the favorites.

// Configure the views.

typeLabel.alpha = enabled ? 1.0 : CharacteristicCell.DisabledAlpha

valueLabel?.alpha = enabled ? 1.0 : CharacteristicCell.DisabledAlpha

if enabled {

```

```

        delegate?.characteristicCell(self, readInitialValueForCharacteristic:
characteristic) { value, error in

            if let error = error {

                print("HomeKit: Error reading value for characteristic
\\(self.characteristic?.description ?? ""): \\(error.localizedDescription).")

            }

            else {

                self.setValue(value, notify: false)

            }

        }

    }

}

```

/// Resets the value label to the localized description from HMCharacteristic+Readability.

```

func resetValueLabel() {

    if let value = value {

```

```

        valueLabel?.text = characteristic.localizedDescriptionForValue(value)

    }

}

/**

Toggles the star button and saves the favorite status

of the characteristic in the FavoriteManager.

*/

// @IBAction func didTapFavoriteButton(_ sender: UIButton) {

//     sender.isSelected = !sender.isSelected

//     characteristic.isFavorite = sender.isSelected

// }

/**

```

Sets the cell's value and resets the label.

- parameter newValue: The new value.

- parameter notify: If true, the cell notifies its delegate of the change.

```
*/
```

```
func setValue(_ newValue: CellValueType?, notify: Bool) {
```

```
    value = newValue
```

```
    if let newValue = newValue {
```

```
        resetValueLabel()
```

```
        /*
```

```
            We do not allow the setting of nil values from the app,
```

```
            but we do have to deal with incoming nil values.
```

```
        */
```

```
        if notify {
```

```
            delegate?.characteristicCell(self, didUpdateValue: newValue, forCharacteristic:
characteristic, immediate: type(of: self).updatesImmediately)
```

```
        }
```

```
    }
```

```
}
```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `CharacteristicSelectionViewController` allows for the selection of characteristics.

This is mainly used for creating characteristic events and conditions

*/

import UIKit

import HomeKit

/**

Allows for the selection of characteristics.

This is mainly used for creating characteristic events and conditions

*/

class CharacteristicSelectionViewController: HMCatalogViewController {

// MARK: Types

```
struct Identifiers {  
  
    static let accessoryCell = "AccessoryCell"  
  
    static let unreachableAccessoryCell = "UnreachableAccessoryCell"  
  
    static let showServicesSegue = "Show Services"  
  
}
```

```
// MARK: Properties
```

```
var eventTrigger: HMEventTrigger?
```

```
var triggerCreator: EventTriggerCreator!
```

```
/// An internal copy of all controllable accessories in the home.
```

```
private var accessories = [HMAccessory]()
```

```
@IBOutlet weak var saveButton: UIBarButtonItem!
```

```
// MARK: View Methods
```

```
/// Resets the internal array of accessories from the home.
```

```
override func viewWillAppear(_ animated: Bool) {
```

```
    super.viewWillAppear(animated)
```

```
    // Only take accessories which have one control service.
```

```
    accessories = home.sortedControlAccessories
```

```
}
```

```
/// Configures the 'ServicesViewController' and passes it the correct accessory.
```

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
```

```
    if segue.identifier == Identifiers.showServicesSegue {
```

```
        let senderCell = sender as! UITableViewCell
```

```
        let servicesVC = segue.intendedDestinationViewController as!  
ServicesViewController
```

```
        let cellIndex = tableView.indexPath(for: senderCell)!.row
```

```
        servicesVC.allowsAllWrites = true
```

```
        servicesVC.onlyShowsControlServices = true
```

```

        servicesVC.accessory = accessories[cellIndex]

        servicesVC.cellDelegate = triggerCreator
    }
}

// MARK: IBAction Methods

/**
    Updates the predicates in the trigger creator and then
    dismisses the view controller.

    */

@IBAction func didTapSave(_ sender: UIBarButtonItem) {

    /**
        We should not save the trigger completely, the user still has a chance to bail out.

        Instead, we generate all of the predicates that were in the map.

        */

    triggerCreator.updatePredicates()

```



```
dismiss(animated: true, completion: nil)

}
```

```
// MARK: Table View Methods
```

```
/// Single section view controller.
```

```
override func numberOfSections(in tableView: UITableView) -> Int {

    return 1

}
```

```
/// - returns: The number of accessories. If there are none, will return 1 (for the 'none
row').
```

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {

    return max(accessories.count, 1)

}
```

```
/// - returns: An Accessory cell that contains an accessory's name.
```

```

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

        let accessory = accessories.sortByLocalizedNames()[indexPath.row]

        let cellIdentifier = accessory.isReachable ? Identifiers.accessoryCell :
Identifiers.unreachableAccessoryCell

        let cell = tableView.dequeueReusableCell(withIdentifier: cellIdentifier, for:
indexPath)

        cell.textLabel?.text = accessory.name

        return cell

    }

```

/// Shows the services in the selected accessory.

```

    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {

        tableView.deselectRow(at: indexPath, animated: true)

        let cell = tableView.cellForRow(at: indexPath)!

        if cell.selectionStyle == .none {

```

```

        return

    }

    performSegue(withIdentifier: Identifiers.showServicesSegue, sender: cell)

}

/// - returns: Localized "Accessories" string.

override func tableView(_ tableView: UITableView, titleForHeaderInSection section:
Int) -> String? {

    return NSLocalizedString("Accessories", comment: "Accessories")

}

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The 'CharacteristicTriggerCreator' creates characteristic triggers.

```

*/

```

```

import UIKit

import HomeKit

/// Represents modes for a `CharacteristicTriggerCreator`.

enum CharacteristicTriggerCreatorMode: Int {

    case event, condition

}

/**

    An `EventTriggerCreator` subclass which allows for the creation

    of characteristic triggers.

*/

class CharacteristicTriggerCreator: EventTriggerCreator {

    // MARK: Properties

    var eventTrigger: HMEventTrigger? {

```

```
        return self.trigger as? HMEventTrigger
    }
}
```

```
/**
```

This object will be a characteristic cell delegate and will therefore be receiving updates when UI elements change value. However, this object can construct both characteristic events and characteristic triggers. Setting the `mode` determines how this trigger creator will handle cell delegate callbacks.

```
*/
```

```
var mode: CharacteristicTriggerCreatorMode = .event
```

```
/**
```

Contains the new pending mapping of `HMCharacteristic`'s to their trigger (`NSCopying`) values.

When `saveTriggerWithName(name:completion:)` is called, all of these mappings will be converted

into `HMCharacteristicEvent`'s and added to the `HMEventTrigger`.

```

*/

private let targetValueMap = NSMapTable<HMCharacteristic,
CellValueType>.strongToStrongObjects()

/// `HMCharacteristicEvent`s that should be removed if
`saveTriggerWithName(name:completion:)` is called.

private var removalCharacteristicEvents = [HMCharacteristicEvent<NSCopying>]()

// MARK: Trigger Creator Methods

/// Syncs the stored event trigger using internal values.

override func updateTrigger() {

    guard let eventTrigger = eventTrigger else { return }

    matchEventsFromTriggerIfNecessary()

    removePendingEventsFromTrigger()

    for (characteristic, triggerValue) in pairsFromMapTable(targetValueMap) {

        let newEvent = HMCharacteristicEvent(characteristic: characteristic, triggerValue:
triggerValue)

        self.saveTriggerGroup.enter()

```

```

eventTrigger.addEvent(newEvent) { error in

    if let error = error {

        self.errors.append(error)

    }

    self.saveTriggerGroup.leave()

}

}

savePredicate()

}

/**

- returns: A new `HMEventTrigger` with the pending

        characteristic events and constructed predicate.

*/

override func newTrigger() -> HMTrigger? {

    return HMEventTrigger(name: name, events: pendingCharacteristicEvents, predicate:
newPredicate())

}

```

```
/**
```

Remove all objects from the map so they don't show up

in the `events` computed array.

```
*/
```

```
override func cleanUp() {
```

```
    targetValueMap.removeAllObjects()
```

```
}
```

```
/**
```

Removes an event from the map table if it's a new event and

queues it for removal if it already existed in the event trigger.

- parameter event: `HMCharacteristicEvent` to be removed.

```
*/
```

```
func removeEvent(_ event: HMCharacteristicEvent<CellValueType>) {
```

```
    if targetValueMap.object(forKey: event.characteristic) != nil {
```



```

        // Remove the characteristic from the target value map.

        targetValueMap.removeObject(forKey: event.characteristic)

    }

    if let characteristicEvents = eventTrigger?.characteristicEvents,
        characteristicEvents.contains(event) {

        // If the given event is in the event array, queue it for removal.

        removalCharacteristicEvents.append(event)

    }

}

// MARK: Helper Methods

/**

Any characteristic events in the map table that have not yet been

added to the trigger.

*/

var pendingCharacteristicEvents: [HMCharacteristicEvent<CellValueType>] {

```

```

let pairs = pairsFromMapTable(targetValueMap)

return pairs.map { (characteristic, triggerValue) ->
HMCharacteristicEvent<CellValueType> in

    return HMCharacteristicEvent(characteristic: characteristic, triggerValue:
triggerValue)

}

}

```

```
/**
```

Loops through the characteristic events in the trigger.

If any characteristics in our map table are also in the event,

replace the value with the one we have stored and remove that entry from

our map table.

```
*/
```

```
private func matchEventsFromTriggerIfNecessary() {
```

```
    guard let eventTrigger = eventTrigger else { return }
```

```
    for event in eventTrigger.characteristicEvents {
```

```
        // Find events who's characteristic is in our map table.
```

```

        if let triggerValue = targetValueMap.object(forKey: event.characteristic) {

            self.saveTriggerGroup.enter()

            event.updateTriggerValue(triggerValue) { error in

                if let error = error {

                    self.errors.append(error)

                }

                self.saveTriggerGroup.leave()

            }

        }

    }

}

```

```

/**

```

Removes all `HMCharacteristicEvent`s from the `removalCharacteristicEvents` array and stores any errors that accumulate.

```

*/

```

```

private func removePendingEventsFromTrigger() {

```

```

guard let eventTrigger = eventTrigger else { return }

for event in removalCharacteristicEvents {

    saveTriggerGroup.enter()

    eventTrigger.removeEvent(event) { error in

        if let error = error {

            self.errors.append(error)

        }

        self.saveTriggerGroup.leave()

    }

}

removalCharacteristicEvents.removeAll()

}

```

/// All `HMCharacteristic`s in the `targetValueMap`.

```
private var allCharacteristics: [HMCharacteristic] {
```

```

var characteristics = Set<HMCharacteristic>()

for characteristic in targetValueMap.keyEnumerator().allObjects as!
[HMCharacteristic] {

    characteristics.insert(characteristic)

}

return Array(characteristics)

}

```

/**

Saves a characteristic and value into the pending map

of characteristic events.

- parameter value: The value of the characteristic.

- parameter characteristic: The `HMCharacteristic` that has been updated.

*/

```

private func updateEventValue(_ value: NSCopying, forCharacteristic characteristic:
HMCharacteristic) {

```

```

    for (index, event) in removalCharacteristicEvents.enumerated() {

```

```

if event.characteristic == characteristic {

    /*

        We have this event pending for deletion,

        but we are going to want to update it.

        remove it from the removal array.

    */

    removalCharacteristicEvents.remove(at: index)

    break

}

}

targetValueMap.setObject(value, forKey: characteristic)

}

/**

```

The current, sorted collection of `HMCharacteristicEvent`'s accumulated by filtering out the events pending removal from the original trigger events and then adding new pending events.

```
*/
```

```
var events: [HMCharacteristicEvent<CellValueType>] {
```

```
    let characteristicEvents = eventTrigger?.characteristicEvents ?? []
```

```
    let originalEvents = characteristicEvents.filter {
```

```
        return !removalCharacteristicEvents.contains($0)
```

```
    }
```

```
    let allEvents = originalEvents + pendingCharacteristicEvents
```

```
    return allEvents.sorted { (event1: HMCharacteristicEvent, event2: HMCharacteristicEvent) in
```

```
        let type1 = event1.characteristic.localizedCharacteristicType
```

```
        let type2 = event2.characteristic.localizedCharacteristicType
```

```
        return type1.localizedCompare(type2) == .orderedAscending
```

```
    }
```

```
}
```

```

// MARK: CharacteristicCellDelegate Methods

/**

    If the mode is event, update the event value.

    Otherwise, default to super implementation

*/

override func characteristicCell(_ cell: CharacteristicCell, didUpdateValue value:
CellValueType, forCharacteristic characteristic: HMCharacteristic, immediate: Bool) {

    switch mode {

        case .event:

            updateEventValue(value, forCharacteristic: characteristic)

        default:

            super.characteristicCell(cell, didUpdateValue: value, forCharacteristic:
characteristic, immediate: immediate)

    }

}

```



```
/**
```

Tries to find the characteristic in either the event map or the

condition map (based on the current mode). Then calls read value.

When the value comes back, we check the selected map for the value

```
*/
```

```
override fun characteristicCell(_ cell: CharacteristicCell,
readInitialValueForCharacteristic characteristic: HMCharacteristic, completion:
@escaping (CellValueType?, Error?) -> Void) {
```

```
    if mode == .condition {
```

```
        // This is a condition, fall back to the `EventTriggerCreator` read.
```

```
        super.characteristicCell(cell, readInitialValueForCharacteristic: characteristic,
completion: completion)
```

```
        return
```

```
    }
```

```
    if let value = targetValueMap.object(forKey: characteristic) {
```

```
        completion(value, nil)
```

```
        return
```

```
    }
```

```

characteristic.readValue { error in

    /*

        The user may have updated the cell value while the

        read was happening. We check the map one more time.

    */

    if let value = self.targetValueMap.object(forKey: characteristic) {

        completion(value, nil)

    }

    else {

        completion(characteristic.value as? CellValueType, error)

    }

}

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `CharacteristicTriggerViewController` allows the user to create a characteristic trigger.

*/

import UIKit

import HomeKit

/// A view controller which facilitates the creation of characteristic triggers.

class CharacteristicTriggerViewController: EventTriggerViewController {

 // MARK: Types

 struct Identifiers {

 static let selectCharacteristicSegue = "Select Characteristic"

 static let addCell = "AddCell"

```

static let conditionCell = "ConditionCell"

static let showTimeConditionSegue = "Show Time Condition"

}

// MARK: Properties

private var characteristicTriggerCreator: CharacteristicTriggerCreator {

    return triggerCreator as! CharacteristicTriggerCreator

}

var eventTrigger: HMEventTrigger? {

    return trigger as? HMEventTrigger

}

/// An internal array of `HMCharacteristicEvent`s to save into the trigger.

```

```
private var events = [HMCharacteristicEvent<CellValueType>]()
```

```
// MARK: View Methods
```

```
/// Creates the trigger creator.
```

```
override func viewDidLoad() {
```

```
    super.viewDidLoad()
```

```
    triggerCreator = CharacteristicTriggerCreator(trigger: eventTrigger, home: home)
```

```
}
```

```
/// Reloads the internal data.
```

```
override func viewWillAppear(_ animated: Bool) {
```

```
    super.viewWillAppear(animated)
```

```
    reloadData()
```

```
}
```

```
/// Passes our event trigger and trigger creator to the  
`CharacteristicSelectionViewController`
```

```

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

    super.prepare(for: segue, sender: sender)

    if segue.identifier == Identifiers.selectCharacteristicSegue {

        if let destinationVC = segue.intendedDestinationViewController as?
CharacteristicSelectionViewController {

            destinationVC.eventTrigger = eventTrigger

            destinationVC.triggerCreator = characteristicTriggerCreator

        }

    }

}

```

// MARK: Table View Methods

/**

- returns: The characteristic events for the Characteristics section.

Defaults to super implementation.

*/

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
```

```
    switch sectionForIndex(section) {
```

```
        case .characteristics?:
```

```
            // Plus one for the add row.
```

```
            return events.count + 1
```

```
        case nil:
```

```
            fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
        default:
```

```
            return super.tableView(tableView, numberOfRowsInSection: section)
```

```
    }
```

```
}
```

```
/**
```

Switches based on cell type to generate the correct cell for the index path.

Defaults to super implementation.

```
*/
```

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
```

```
    if indexPathIsAdd(indexPath) {
```

```
        return self.tableView(tableView, addCellForRowAtIndexPath: indexPath)
```

```
    }
```

```
    switch sectionForIndex(indexPath.section) {
```

```
        case .characteristics?:
```

```
            return self.tableView(tableView, cellForRowAtIndexPath: indexPath)
```

```
        case nil:
```

```
            fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
        default:
```

```
            return super.tableView(tableView, cellForRowAt: indexPath)
```

```
    }
```

```
}
```


/// - returns: A 'condition cell' with the event at the specified index path.

```
private func tableView(_ tableView: UITableView, cellForRowAtIndexPath  
indexPath: IndexPath) -> UITableViewCell {
```

```
    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.conditionCell,  
for: indexPath) as! ConditionCell
```

```
    let event = events[indexPath.row]
```

```
    cell.setCharacteristic(event.characteristic, targetValue: event.triggerValue!)
```

```
    return cell
```

```
}
```

/**

- returns: An 'add cell' with localized text.

Defaults to super implementation.

*/

```
override func tableView(_ tableView: UITableView, addCellForRowAtIndexPath  
indexPath: IndexPath) -> UITableViewCell {
```

```
    switch sectionForIndex(indexPath.section) {
```

```
        case .characteristics?:
```

```
        let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.addCell, for:
indexPath)
```

```
        cell.textLabel?.text = NSLocalizedString("Add Characteristic...", comment:
"Add Characteristic")
```

```
        cell.textLabel?.textColor = .editableBlue
```

```
        return cell
```

```
    case nil:
```

```
        fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
    default:
```

```
        return super.tableView(tableView, addCellForRowAtIndexPath: indexPath)
```

```
    }
```

```
}
```

```
/**
```

Handles the selection of characteristic events.

Defaults to super implementation for other sections.

```
*/
```

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
```

```
    tableView.deselectRow(at: indexPath, animated: true)
```

```
    switch sectionForIndex(indexPath.section) {
```

```
        case .characteristics?:
```

```
            if indexPathIsAdd(indexPath) {
```

```
                addEvent()
```

```
                return
```

```
            }
```

```
            let cell = tableView.cellForRow(at: indexPath)
```

```
            performSegue(withIdentifier: Identifiers.selectCharacteristicSegue, sender: cell)
```

```
        case nil:
```

```
            fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
        default:
```

```
            super.tableView(tableView, didSelectRowAt: indexPath)
```

```

    }

}

/**

- returns: `true` for characteristic cells,

        otherwise defaults to super implementation.

*/

override func tableView(_ tableView: UITableView, canEditRowAt indexPath:
IndexPath) -> Bool {

    if indexPathIsAdd(indexPath) {

        return false

    }

    switch sectionForIndex(indexPath.section) {

        case .characteristics?:

            return true

        case nil:

            fatalError("Unexpected `TriggerTableViewSection` raw value.")

```

default:

```
return super.tableView(tableView, canEditRowAt: indexPath)
```

```
}
```

```
}
```

```
/**
```

Removes events from the trigger creator.

Defaults to super implementation for other sections.

```
*/
```

```
override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {
```

```
if editingStyle == .delete {
```

```
switch sectionForIndex(indexPath.section) {
```

```
case .characteristics?:
```

```
characteristicTriggerCreator.removeEvent(events[indexPath.row])
```

```
events = characteristicTriggerCreator.events
```

```
tableView.deleteRows(at: [indexPath], with: .automatic)
```

```
case nil:
```

```
    fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
default:
```

```
    super.tableView(tableView, commit: editingStyle, forRowAt: indexPath)
```

```
    }
```

```
    }
```

```
}
```

```
/**
```

```
- returns: A localized description of characteristic events
```

```
Defaults to super implementation for other sections.
```

```
*/
```

```
override func tableView(_ tableView: UITableView, titleForFooterInSection section:  
Int) -> String? {
```

```
    switch sectionForIndex(section) {
```

```
        case .characteristics?:
```

```
        return NSLocalizedString("This trigger will activate when any of these
characteristics change to their value. For example, 'run when the garage door is opened'.",
comment: "Characteristic Trigger Description")
```

```
    case nil:
```

```
        fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
    default:
```

```
        return super.tableView(tableView, titleForFooterInSection: section)
```

```
    }
```

```
}
```

```
// MARK: Helper Methods
```

```
/// Resets the internal events array from the trigger creator.
```

```
private func reloadData() {
```

```
    events = characteristicTriggerCreator.events
```

```
    tableView.reloadData()
```

```
}
```

```
/// Performs a segue to the `CharacteristicSelectionViewController`.
```

```
private func addEvent() {
```

```
    characteristicTriggerCreator.mode = .event
```

```
    self.performSegue(withIdentifier: Identifiers.selectCharacteristicSegue, sender: nil)
```

```
}
```

```
/// - returns: `true` if the section is the Characteristic 'add row'; otherwise defaults to  
super implementation.
```

```
override func indexPathIsAdd(_ indexPath: IndexPath) -> Bool {
```

```
    switch sectionForIndex(indexPath.section) {
```

```
        case .characteristics?:
```

```
            return indexPath.row == events.count
```

```
        case nil:
```

```
            fatalError("Unexpected `TriggerTableViewSection` raw value.")
```


default:

```
return super.indexPathIsAdd(indexPath)
```

```
}
```

```
}
```

```
// MARK: Trigger Controller Methods
```

```
/**
```

```
- parameter index: The section index.
```

```
- returns: The `TriggerTableViewSection` for the given index.
```

```
*/
```

```
override func sectionForIndex(_ index: Int) -> TriggerTableViewSection? {
```

```
switch index {
```

```
case 0:
```

```
return .name
```

case 1:

return .enabled

case 2:

return .characteristics

case 3:

return .conditions

case 4:

return .actionSets

default:

return nil

}

}

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The ``CharacteristicsTableViewDataSource`` provides the data for the ``CharacteristicsViewController``.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
/// Represents the sections in the `CharacteristicsViewController`.
```

```
enum CharacteristicTableViewSection: Int {
```

```
    case characteristics, associatedServiceType
```

```
}
```

```
/// A `UITableViewDataSource` that populates a `CharacteristicsViewController`.
```

```
class CharacteristicsTableViewDataSource: NSObject, UITableViewDelegate,  
UITableViewDataSource {
```

```
    // MARK: Types
```

```
    struct Identifiers {
```

```
        static let characteristicCell = "CharacteristicCell"
```

```
        static let sliderCharacteristicCell = "SliderCharacteristicCell"
```

```
        static let switchCharacteristicCell = "SwitchCharacteristicCell"
```

```
        static let segmentedControlCharacteristicCell =  
        "SegmentedControlCharacteristicCell"
```

```
        static let textCharacteristicCell = "TextCharacteristicCell"
```

```
        static let serviceTypeCell = "ServiceTypeCell"
```

```
    }
```

```
    // MARK: Properties
```

```
    var service: HMService
```

```
var tableView: UITableView
```

```
var delegate: CharacteristicCellDelegate
```

```
var showsFavorites: Bool
```

```
var allowsAllWrites: Bool
```

```
/// Sets up properties from specified values, configures the table view, and cell reuse  
identifiers.
```

```
required init(service: HMService, tableView: UITableView, delegate:  
CharacteristicCellDelegate, showsFavorites: Bool = false, allowsAllWrites: Bool = false)  
{
```

```
    self.service = service
```

```
    self.tableView = tableView
```

```
    self.delegate = delegate
```

```
    self.showsFavorites = showsFavorites
```

```
    self.allowsAllWrites = allowsAllWrites
```

```
    super.init()
```

```
    tableView.dataSource = self
```

```
    tableView.rowHeight = UITableView.automaticDimension
```

```
    tableView.estimatedRowHeight = 50.0
```

```

        registerReuseIdentifiers()

    }

    /// Registers all of the characteristic cell reuse identifiers with this table.

    func registerReuseIdentifiers() {

        let characteristicNib = UINib(nibName: Identifiers.characteristicCell, bundle: nil)

        tableView.register(characteristicNib,                                forCellReuseIdentifier:
Identifiers.characteristicCell)

        let sliderNib = UINib(nibName: Identifiers.sliderCharacteristicCell, bundle: nil)

        tableView.register(sliderNib,                                    forCellReuseIdentifier:
Identifiers.sliderCharacteristicCell)

        let switchNib = UINib(nibName: Identifiers.switchCharacteristicCell, bundle: nil)

        tableView.register(switchNib,                                    forCellReuseIdentifier:
Identifiers.switchCharacteristicCell)

        let segmentedNib = UINib(nibName: Identifiers.segmentedControlCharacteristicCell,
bundle: nil)

```

```
tableView.register(segmentedNib, forCellReuseIdentifier:
Identifiers.segmentedControlCharacteristicCell)
```

```
let textNib = UINib(nibName: Identifiers.textCharacteristicCell, bundle: nil)
```

```
tableView.register(textNib, forCellReuseIdentifier: Identifiers.textCharacteristicCell)
```

```
tableView.register(UITableViewCell.self, forCellReuseIdentifier:
Identifiers.serviceTypeCell)
```

```
}
```

```
/**
```

```
- returns: The number of sections, computed from whether or not
```

```
the services supports an associated service type.
```

```
*/
```

```
func numberOfSections(in tableView: UITableView) -> Int {
```

```
return service.supportsAssociatedServiceType ? 2 : 1
```

```
}
```

```
/**
```

The characteristics section uses the services count to generate the number of rows.

The associated service type uses the valid associated service types.

```
*/
```

```
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) ->
Int {

    switch CharacteristicTableViewSection(rawValue: section) {

        case .characteristics?:

            return service.characteristics.count


        case .associatedServiceType?:

            // For 'None'.

            return HMService.validAssociatedServiceTypes.count + 1


        case nil:

            fatalError("Unexpected `CharacteristicTableViewSection` raw value.")

    }

}
```



```
/**
```

Looks up the appropriate service type for the row in the list and returns a localized version,

or 'None' if the row doesn't correspond to any valid service type.

- parameter row: The row to look up.

- returns: The localized service type in that row, or 'None'.

```
*/
```

```
func displayedServiceTypeForRow(_ row: Int) -> String {
```

```
    let serviceTypes = HMService.validAssociatedServiceTypes
```

```
    if row < serviceTypes.count {
```

```
        return HMService.localizedDescriptionForServiceType(serviceTypes[row])
```

```
    }
```

```
    return NSLocalizedString("None", comment: "None")
```

```
}
```

```
/**
```

Evaluates whether or not a service type is selected for a given row.

- parameter row: The selected row.

- returns: `true` if the current row is a valid service type, `false` otherwise

```
*/
```

```
func serviceTypeIsSelectedForRow(_ row: Int) -> Bool {
```

```
    let serviceTypes = HMService.validAssociatedServiceTypes
```

```
    if row >= serviceTypes.count {
```

```
        return service.associatedServiceType == nil
```

```
    }
```

```
    if let associatedServiceType = service.associatedServiceType {
```

```
        return serviceTypes[row] == associatedServiceType
```

```
    }
```

```

        return false
    }

    /// Generates a cell for an associated service.

    private func tableView(_ tableView: UITableView,
associatedServiceTypeCellForRowAtIndexPath indexPath: IndexPath) ->
UITableViewCell {

        let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.serviceTypeCell,
for: indexPath)

        cell.textLabel?.text = displayedServiceTypeForRow(indexPath.row)

        cell.accessoryType =
serviceTypeIsSelectedForRow(indexPath.row) ? .checkmark : .none

        return cell
    }

    /**

```

Generates a characteristic cell based on the type of characteristic

located at the specified index path.

*/

```
private func tableView(_ tableView: UITableView,
characteristicCellForRowAtIndexPath indexPath: IndexPath) -> UITableViewCell {

    let characteristic = service.characteristics[indexPath.row]

    var reuseIdentifier = Identifiers.characteristicCell

    if (characteristic.isReadOnly || characteristic.isWriteOnly) && !allowsAllWrites {

        reuseIdentifier = Identifiers.characteristicCell

    }

    else if characteristic.isBoolean {

        reuseIdentifier = Identifiers.switchCharacteristicCell

    }

    else if characteristic.hasPredeterminedValueDescriptions {

        reuseIdentifier = Identifiers.segmentedControlCharacteristicCell

    }
```

```

//     else if characteristic.isNumeric {

//         reuseIdentifier = Identifiers.sliderCharacteristicCell

//     }

    else if characteristic.isTextWritable {

        reuseIdentifier = Identifiers.textCharacteristicCell

    }

    let cell = tableView.dequeueReusableCell(withIdentifier: reuseIdentifier, for:
indexPath) as! CharacteristicCell

    cell.showsFavorites = showsFavorites

    cell.delegate = delegate

    cell.characteristic = characteristic

    return cell

}

```

/// Uses convenience methods to generate a cell based on the index path's section.

```

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {

    switch CharacteristicTableViewSection(rawValue: indexPath.section) {

        case .characteristics?:

            return self.tableView(tableView, characteristicCellForRowAtIndexPath:
indexPath)

        case .associatedServiceType?:

            return self.tableView(tableView,
associatedServiceTypeCellForRowAtIndexPath: indexPath)

        case nil:

            fatalError("Unexpected `CharacteristicTableViewSection` raw value.")

    }

}

```

/// - returns: A localized string for the section.

```

func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) ->
String? {

```

```

switch CharacteristicTableViewSection(rawValue: section) {

    case .characteristics?:

        return NSLocalizedString("Characteristics", comment: "Characteristics")


    case .associatedServiceType?:

        return NSLocalizedString("Associated Service Type", comment: "Associated
Service Type")


    case nil:

        fatalError("Unexpected `CharacteristicTableViewSection` raw value.")

    }

}

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `CharacteristicsViewController` displays characteristics within a service.

*/

```
import UIKit
```

```
import HomeKit
```

```
/// A view controller that displays a list of characteristics within an `HMService`.
```

```
class CharacteristicsViewController: HMCatalogViewController, HMAccessoryDelegate  
{
```

```
    // MARK: Properties
```

```
    var service: HMService!
```

```
    var cellDelegate: CharacteristicCellDelegate!
```

```
    private var tableViewDataSource: CharacteristicsTableViewDataSource!
```

```
    var showsFavorites = false
```

```
    var allowsAllWrites = false
```



```
// MARK: View Methods
```

```
/// Initializes the data source.
```

```
override func viewDidLoad() {
```

```
    super.viewDidLoad()
```

```
    tableViewDataSource = CharacteristicsTableViewDataSource(service: service,  
    tableView: tableView, delegate: cellDelegate, showsFavorites: showsFavorites,  
    allowsAllWrites: allowsAllWrites)
```

```
}
```

```
/// Reloads the view and enabled notifications for all relevant characteristics.
```

```
override func viewWillAppear(_ animated: Bool) {
```

```
    super.viewWillAppear(animated)
```

```
    title = service.name
```

```
    setNotificationsEnabled(true)
```

```
    reloadTableView()
```

```
}
```

```

/// Disables notifications for characteristics.

override func viewWillDisappear(_ animated: Bool) {

    super.viewWillDisappear(animated)

    setNotificationsEnabled(false)

}

/**

Registers as the delegate for the current home and

the service's accessory.

*/

override func registerAsDelegate() {

    super.registerAsDelegate()

    service.accessory?.delegate = self

}

/**

```

Enables or disables notifications on all characteristics within this service.

- parameter notificationsEnabled: A `Bool`; whether to enable or disable.

```
*/
```

```
func setNotificationsEnabled(_ notificationsEnabled: Bool) {  
  
    for characteristic in service.characteristics {  
  
        if characteristic.supportsEventNotification {  
  
            characteristic.enableNotification(notificationsEnabled) { error in  
  
                if let error = error {  
  
                    print("HomeKit: Error enabling notification on characteristic  
\"(characteristic)\": \"(error.localizedDescription)\"")  
  
                }  
  
            }  
  
        }  
  
    }  
  
}
```

```
/// Reloads the table view and stops the refresh control.
```

```

func reloadData() {

    setNotificationsEnabled(true)

    tableViewDataSource.service = service

    refreshControl?.endRefreshing()

    tableView.reloadData()

}

```

// MARK: Table View Methods

```

override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {

    switch CharacteristicTableViewSection(rawValue: indexPath.section) {

        case .characteristics?:

            let characteristic = service.characteristics[indexPath.row]

            didSelectCharacteristic(characteristic, atIndexPath: indexPath)

        case .associatedServiceType?:

            didSelectAssociatedServiceTypeAtIndexPath(indexPath)

```

```

        case nil:

            fatalError("Unexpected `CharacteristicTableViewSection` raw value.")

        }

    }

/**

    If a characteristic is selected, and it is the 'Identify' characteristic,

    perform an identify on that accessory.

*/

    private func didSelectCharacteristic(_ characteristic: HMCharacteristic, atIndexPath
indexPath: IndexPath) {

        if characteristic.isIdentify {

            service.accessory?.identify { error in

                if let error = error {

                    self.displayError(error)

                    return

                }

            }

```

```

    }

}

}

/**

Handles selection of one of the associated service types in the list.

- parameter indexPath: The selected index path.

*/

private func didSelectAssociatedServiceTypeAtIndexPath(_ indexPath: IndexPath) {

    let serviceTypes = HMService.validAssociatedServiceTypes

    var newServiceType: String?

    if indexPath.row < serviceTypes.count {

        newServiceType = serviceTypes[indexPath.row]

    }

    service.updateAssociatedServiceType(newServiceType) { error in

        if let error = error {

```

```

        self.displayError(error)

        return

    }

    self.didUpdateAssociatedServiceType()

}

}

/// Reloads the associated service section in the table view.

private func didUpdateAssociatedServiceType() {

    let associatedServiceTypeIndexSet = IndexSet(integer:
CharacteristicTableViewSection.associatedServiceType.rawValue)

    tableView.reloadSections(associatedServiceTypeIndexSet, with: .automatic)

}

// MARK: HMHomeDelegate Methods

```

```

/// If our accessory was removed, pop to root view controller.

func home(_ home: HMHome, didRemoveAccessory accessory: HMAccessory) {

    if accessory == service.accessory {

        _ = navigationController?.popToRootViewController(animated: true)

    }

}

// MARK: HMAccessoryDelegate Methods

/// If our accessory becomes unreachable, pop to root view controller.

func accessoryDidUpdateReachability(_ accessory: HMAccessory) {

    if accessory == service.accessory && !accessory.isReachable {

        _ = navigationController?.popToRootViewController(animated: true)

    }

}

/**

```


Search for the cell corresponding to that characteristic and

update its value.

*/

```
func accessory(_ accessory: HMAccessory, service: HMService, didUpdateValueFor
characteristic: HMCharacteristic) {
```

```
    if let index = service.characteristics.firstIndex(of: characteristic) {
```

```
        let indexPath = IndexPath(row: index, section: 0)
```

```
        if let cell = tableView.cellForRow(at: indexPath) as? CharacteristicCell {
```

```
            cell.setValue(characteristic.value as? CellValueType, notify: false)
```

```
        }
```

```
    }
```

```
}
```

```
}
```

```
//
```

```
// ChartAnimationUtils.swift
```

```
// Charts
```

```
//
```

```
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```
// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//
```

```
import Foundation
```

```
import CoreGraphics
```

```
@objc
```

```
public enum ChartEasingOption: Int
```

```
{
```

```
    case linear
```

```
    case easeInQuad
```

```
    case easeOutQuad
```

```
    case easeInOutQuad
```

```
    case easeInCubic
```

case easeOutCubic

case easeInOutCubic

case easeInQuart

case easeOutQuart

case easeInOutQuart

case easeInQuint

case easeOutQuint

case easeInOutQuint

case easeInSine

case easeOutSine

case easeInOutSine

case easeInExpo

case easeOutExpo

case easeInOutExpo

case easeInCirc

case easeOutCirc

case easeInOutCirc

```

    case easeInElastic

    case easeOutElastic

    case easeInOutElastic

    case easeInBack

    case easeOutBack

    case easeInOutBack

    case easeInBounce

    case easeOutBounce

    case easeInOutBounce

}

public typealias ChartEasingFunctionBlock = ((_ elapsed: TimeInterval, _ duration:
TimeInterval) -> Double)

internal func easingFunctionFromOption(_ easing: ChartEasingOption) ->
ChartEasingFunctionBlock

{

    switch easing

```

```
{  
  
case .linear:  
  
    return EasingFunctions.Linear  
  
case .easeInQuad:  
  
    return EasingFunctions.EaseInQuad  
  
case .easeOutQuad:  
  
    return EasingFunctions.EaseOutQuad  
  
case .easeInOutQuad:  
  
    return EasingFunctions.EaseInOutQuad  
  
case .easeInCubic:  
  
    return EasingFunctions.EaseInCubic  
  
case .easeOutCubic:  
  
    return EasingFunctions.EaseOutCubic  
  
case .easeInOutCubic:  
  
    return EasingFunctions.EaseInOutCubic  
  
case .easeInQuart:  
  
    return EasingFunctions.EaseInQuart
```

```
case .easeOutQuart:

    return EasingFunctions.EaseOutQuart

case .easeInOutQuart:

    return EasingFunctions.EaseInOutQuart

case .easeInQuint:

    return EasingFunctions.EaseInQuint

case .easeOutQuint:

    return EasingFunctions.EaseOutQuint

case .easeInOutQuint:

    return EasingFunctions.EaseInOutQuint

case .easeInSine:

    return EasingFunctions.EaseInSine

case .easeOutSine:

    return EasingFunctions.EaseOutSine

case .easeInOutSine:

    return EasingFunctions.EaseInOutSine

case .easeInExpo:
```

```
        return EasingFunctions.EaseInExpo

    case .easeOutExpo:

        return EasingFunctions.EaseOutExpo

    case .easeInOutExpo:

        return EasingFunctions.EaseInOutExpo

    case .easeInCirc:

        return EasingFunctions.EaseInCirc

    case .easeOutCirc:

        return EasingFunctions.EaseOutCirc

    case .easeInOutCirc:

        return EasingFunctions.EaseInOutCirc

    case .easeInElastic:

        return EasingFunctions.EaseInElastic

    case .easeOutElastic:

        return EasingFunctions.EaseOutElastic

    case .easeInOutElastic:

        return EasingFunctions.EaseInOutElastic
```

```

    case .easeInBack:

        return EasingFunctions.EaseInBack

    case .easeOutBack:

        return EasingFunctions.EaseOutBack

    case .easeInOutBack:

        return EasingFunctions.EaseInOutBack

    case .easeInBounce:

        return EasingFunctions.EaseInBounce

    case .easeOutBounce:

        return EasingFunctions.EaseOutBounce

    case .easeInOutBounce:

        return EasingFunctions.EaseInOutBounce

    }

}

internal struct EasingFunctions

{

```



```
internal static let Linear = { (elapsed: TimeInterval, duration: TimeInterval) -> Double
in return Double(elapsed / duration) }
```

```
internal static let EaseInQuad = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in
```

```
    var position = Double(elapsed / duration)
```

```
    return position * position
```

```
}
```

```
internal static let EaseOutQuad = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in
```

```
    var position = Double(elapsed / duration)
```

```
    return -position * (position - 2.0)
```

```
}
```

```
internal static let EaseInOutQuad = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in
```

```
    var position = Double(elapsed / (duration / 2.0))
```

```
    if position < 1.0
```

```

{

    return 0.5 * position * position

}

return -0.5 * ((position - 1.0) * (position - 3.0) - 1.0)

}

```

```

internal static let EaseInCubic = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

```

```

    var position = Double(elapsed / duration)

    return position * position * position

}

```

```

internal static let EaseOutCubic = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

```

```

    var position = Double(elapsed / duration)

    position -= 1.0

    return (position * position * position + 1.0)

```

```
}
```

```
internal static let EaseInOutCubic = { (elapsed: TimeInterval, duration: TimeInterval) ->  
Double in
```

```
var position = Double(elapsed / (duration / 2.0))
```

```
if position < 1.0
```

```
{
```

```
    return 0.5 * position * position * position
```

```
}
```

```
position -= 2.0
```

```
return 0.5 * (position * position * position + 2.0)
```

```
}
```

```
internal static let EaseInQuart = { (elapsed: TimeInterval, duration: TimeInterval) ->  
Double in
```

```
var position = Double(elapsed / duration)
```

```
return position * position * position * position
```

```
}
```

```
internal static let EaseOutQuart = { (elapsed: TimeInterval, duration: TimeInterval) ->  
Double in
```

```
var position = Double(elapsed / duration)
```

```
position -= 1.0
```

```
return -(position * position * position * position - 1.0)
```

```
}
```

```
internal static let EaseInOutQuart = { (elapsed: TimeInterval, duration: TimeInterval) ->  
Double in
```

```
var position = Double(elapsed / (duration / 2.0))
```

```
if position < 1.0
```

```
{
```

```
return 0.5 * position * position * position * position
```

```
}
```

```
position -= 2.0
```

```
return -0.5 * (position * position * position * position - 2.0)
```

```
}
```

```

internal static let EaseInQuint = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

    var position = Double(elapsed / duration)

    return position * position * position * position * position

}

```

```

internal static let EaseOutQuint = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

    var position = Double(elapsed / duration)

    position -= 1.0

    return (position * position * position * position * position + 1.0)

}

```

```

internal static let EaseInOutQuint = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

    var position = Double(elapsed / (duration / 2.0))

    if position < 1.0

    {

```

```

        return 0.5 * position * position * position * position * position

    }

    else

    {

        position -= 2.0

        return 0.5 * (position * position * position * position * position + 2.0)

    }

}

```

```

internal static let EaseInSine = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

```

```

    var position: TimeInterval = elapsed / duration

    return Double( -cos(position * Double.pi / 2) + 1.0 )

}

```

```

internal static let EaseOutSine = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

```

```

    var position: TimeInterval = elapsed / duration

```

```

    return Double( sin(position * Double.pi / 2) )

}

```

```

internal static let EaseInOutSine = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

```

```

    var position: TimeInterval = elapsed / duration

```

```

    return Double( -0.5 * (cos(Double.pi * position) - 1.0) )

```

```

}

```

```

internal static let EaseInExpo = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

```

```

    return (elapsed == 0) ? 0.0 : Double(pow(2.0, 10.0 * (elapsed / duration - 1.0)))

```

```

}

```

```

internal static let EaseOutExpo = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

```

```

    return (elapsed == duration) ? 1.0 : (-Double(pow(2.0, -10.0 * elapsed / duration)) +
1.0)

```

```

}

```

```

internal static let EaseInOutExpo = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

    if elapsed == 0

    {

        return 0.0

    }

    if elapsed == duration

    {

        return 1.0

    }

    var position: TimeInterval = elapsed / (duration / 2.0)

    if position < 1.0

    {

        return Double( 0.5 * pow(2.0, 10.0 * (position - 1.0)) )

    }

```



```

    position = position - 1.0

    return Double( 0.5 * (-pow(2.0, -10.0 * position) + 2.0) )

}

```

```

internal static let EaseInCirc = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

```

```

    var position = Double(elapsed / duration)

    return -(Double(sqrt(1.0 - position * position)) - 1.0)

}

```

```

internal static let EaseOutCirc = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

```

```

    var position = Double(elapsed / duration)

    position -= 1.0

    return Double( sqrt(1 - position * position) )

}

```

```

internal static let EaseInOutCirc = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

```

```

var position: TimeInterval = elapsed / (duration / 2.0)

if position < 1.0

{

    return Double( -0.5 * (sqrt(1.0 - position * position) - 1.0) )

}

position -= 2.0

return Double( 0.5 * (sqrt(1.0 - position * position) + 1.0) )

}

```

```

internal static let EaseInElastic = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

```

```

if elapsed == 0.0

{

    return 0.0

}

```

```

var position: TimeInterval = elapsed / duration

if position == 1.0

```

```

    {

        return 1.0

    }

    var p = duration * 0.3

    var s = p / (2.0 * Double.pi) * asin(1.0)

    position -= 1.0

    return Double( -(pow(2.0, 10.0 * position) * sin((position * duration - s) * (2.0 *
Double.pi) / p)) )

}

```

```

internal static let EaseOutElastic = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

```

```

    if elapsed == 0.0

```

```

    {

        return 0.0

    }

```

```

var position: TimeInterval = elapsed / duration

if position == 1.0

{

    return 1.0

}


var p = duration * 0.3

var s = p / (2.0 * Double.pi) * asin(1.0)

return Double( pow(2.0, -10.0 * position) * sin((position * duration - s) * (2.0 *
Double.pi) / p) + 1.0 )

}


internal static let EaseInOutElastic = { (elapsed: TimeInterval, duration: TimeInterval)
-> Double in

    if elapsed == 0.0

    {

        return 0.0

    }

```

```

var position: TimeInterval = elapsed / (duration / 2.0)

if position == 2.0

{

    return 1.0

}


var p = duration * (0.3 * 1.5)

var s = p / (2.0 * Double.pi) * asin(1.0)

if position < 1.0

{

    position -= 1.0

    return Double( -0.5 * (pow(2.0, 10.0 * position) * sin((position * duration - s) *
(2.0 * Double.pi) / p)) )

}

position -= 1.0

return Double( pow(2.0, -10.0 * position) * sin((position * duration - s) * (2.0 *
Double.pi) / p) * 0.5 + 1.0 )

```

```
}
```

```
internal static let EaseInBack = { (elapsed: TimeInterval, duration: TimeInterval) ->  
Double in
```

```
let s: TimeInterval = 1.70158
```

```
var position: TimeInterval = elapsed / duration
```

```
return Double( position * position * ((s + 1.0) * position - s) )
```

```
}
```

```
internal static let EaseOutBack = { (elapsed: TimeInterval, duration: TimeInterval) ->  
Double in
```

```
let s: TimeInterval = 1.70158
```

```
var position: TimeInterval = elapsed / duration
```

```
position -= 1.0
```

```
return Double( position * position * ((s + 1.0) * position + s) + 1.0 )
```

```
}
```

```
internal static let EaseInOutBack = { (elapsed: TimeInterval, duration: TimeInterval) ->  
Double in
```

```

var s: TimeInterval = 1.70158

var position: TimeInterval = elapsed / (duration / 2.0)

if position < 1.0

{

    s *= 1.525

    return Double( 0.5 * (position * position * ((s + 1.0) * position - s)) )

}

s *= 1.525

position -= 2.0

return Double( 0.5 * (position * position * ((s + 1.0) * position + s) + 2.0) )

}

internal static let EaseInBounce = { (elapsed: TimeInterval, duration: TimeInterval) ->
Double in

    return 1.0 - EaseOutBounce(duration - elapsed, duration)

}

```

```
internal static let EaseOutBounce = { (elapsed: TimeInterval, duration: TimeInterval) ->  
Double in
```

```
var position: TimeInterval = elapsed / duration
```

```
if position < (1.0 / 2.75)
```

```
{
```

```
    return Double( 7.5625 * position * position )
```

```
}
```

```
else if position < (2.0 / 2.75)
```

```
{
```

```
    position -= (1.5 / 2.75)
```

```
    return Double( 7.5625 * position * position + 0.75 )
```

```
}
```

```
else if position < (2.5 / 2.75)
```

```
{
```

```
    position -= (2.25 / 2.75)
```

```
    return Double( 7.5625 * position * position + 0.9375 )
```

```
}
```

```
else
```



```

    {

        position -= (2.625 / 2.75)

        return Double( 7.5625 * position * position + 0.984375 )

    }

}

internal static let EaseInOutBounce = { (elapsed: TimeInterval, duration: TimeInterval)
-> Double in

    if elapsed < (duration / 2.0)

    {

        return EaseInBounce(elapsed * 2.0, duration) * 0.5

    }

    return EaseOutBounce(elapsed * 2.0 - duration, duration) * 0.5 + 0.5

}

}

//

// BaseDataSet.swift

// Charts

```

```
//  
  
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda  
  
// A port of MPAndroidChart for iOS  
  
// Licensed under Apache License 2.0  
  
//  
  
// https://github.com/danielgindi/Charts  
  
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
open class ChartBaseDataSet: NSObject, IChartDataSet, NSCopying
```

```
{
```

```
    public required override init()
```

```
    {
```

```
        super.init()
```

```

        // default color

        colors.append(NSUIColor(red: 140.0/255.0, green: 234.0/255.0, blue: 255.0/255.0,
alpha: 1.0))

        valueColors.append(.labelOrBlack)

    }

@objc public init(label: String?)

{

    super.init()

    // default color

    colors.append(NSUIColor(red: 140.0/255.0, green: 234.0/255.0, blue: 255.0/255.0,
alpha: 1.0))

    valueColors.append(.labelOrBlack)

    self.label = label

}

```

```
// MARK: - Data functions and accessors
```

```
/// Use this method to tell the data set that the underlying data has changed
```

```
open func notifyDataSetChanged()
```

```
{
```

```
    calcMinMax()
```

```
}
```

```
open func calcMinMax()
```

```
{
```

```
    fatalError("calcMinMax is not implemented in ChartBaseDataSet")
```

```
}
```

```
open func calcMinMaxY(fromX: Double, toX: Double)
```

```
{
```

```
    fatalError("calcMinMaxY(fromX:, toX:) is not implemented in ChartBaseDataSet")
```

```
}
```

```
open var yMin: Double
```

```
{
```

```
    fatalError("yMin is not implemented in ChartBaseDataSet")
```

```
}
```

```
open var yMax: Double
```

```
{
```

```
    fatalError("yMax is not implemented in ChartBaseDataSet")
```

```
}
```

```
open var xMin: Double
```

```
{
```

```
    fatalError("xMin is not implemented in ChartBaseDataSet")
```

```
}
```

```

open var xMax: Double

{

    fatalError("xMax is not implemented in ChartBaseDataSet")

}


open var entryCount: Int

{

    fatalError("entryCount is not implemented in ChartBaseDataSet")

}


open fun entryForIndex(_ i: Int) -> ChartDataEntry?

{

    fatalError("entryForIndex is not implemented in ChartBaseDataSet")

}


open fun entryForXValue(

    _ x: Double,

```

```

        closestToY y: Double,

        rounding: ChartDataSetRounding) -> ChartDataEntry?

    {

        fatalError("entryForXValue(x, closestToY, rounding) is not implemented in
ChartBaseDataSet")

    }

open func entryForXValue(

    _ x: Double,

    closestToY y: Double) -> ChartDataEntry?

    {

        fatalError("entryForXValue(x, closestToY) is not implemented in ChartBaseDataSet")

    }

open func entriesForXValue(_ x: Double) -> [ChartDataEntry]

    {

        fatalError("entriesForXValue is not implemented in ChartBaseDataSet")

    }

```

```

open func entryIndex(
    x xValue: Double,
    closestToY y: Double,
    rounding: ChartDataSetRounding) -> Int
{
    fatalError("entryIndex(x, closestToY, rounding) is not implemented in
ChartBaseDataSet")
}

```

```

open func entryIndex(entry e: ChartDataEntry) -> Int

```

```

{
    fatalError("entryIndex(entry) is not implemented in ChartBaseDataSet")
}

```

```

open func addEntry(_ e: ChartDataEntry) -> Bool

```

```

{
    fatalError("addEntry is not implemented in ChartBaseDataSet")
}

```



```
}
```

```
open func addEntryOrdered(_ e: ChartDataEntry) -> Bool
```

```
{
```

```
    fatalError("addEntryOrdered is not implemented in ChartBaseDataSet")
```

```
}
```

```
@discardableResult open func removeEntry(_ entry: ChartDataEntry) -> Bool
```

```
{
```

```
    fatalError("removeEntry is not implemented in ChartBaseDataSet")
```

```
}
```

```
@discardableResult open func removeEntry(index: Int) -> Bool
```

```
{
```

```
    if let entry = entryForIndex(index)
```

```
    {
```

```
        return removeEntry(entry)
```

```

    }

    return false
}

@discardableResult open fun removeEntry(x: Double) -> Bool
{
    if let entry = entryForXValue(x, closestToY: Double.nan)
    {
        return removeEntry(entry)
    }

    return false
}

```

```

@discardableResult open fun removeFirst() -> Bool
{
    if entryCount > 0
    {

```

```

        if let entry = entryForIndex(0)

        {

            return removeEntry(entry)

        }

    }

    return false

}

```

```

@discardableResult open func removeLast() -> Bool

{

    if entryCount > 0

    {

        if let entry = entryForIndex(entryCount - 1)

        {

            return removeEntry(entry)

        }

    }

}

```

```
return false
```

```
}
```

```
open func contains(_ e: ChartDataEntry) -> Bool
```

```
{
```

```
    fatalError("removeEntry is not implemented in ChartBaseDataSet")
```

```
}
```

```
open func clear()
```

```
{
```

```
    fatalError("clear is not implemented in ChartBaseDataSet")
```

```
}
```

```
// MARK: - Styling functions and accessors
```

```
/// All the colors that are used for this DataSet.
```

```
/// Colors are reused as soon as the number of Entries the DataSet represents is higher  
than the size of the colors array.
```

```

open var colors = [NSUIColor]()

/// List representing all colors that are used for drawing the actual values for this DataSet

open var valueColors = [NSUIColor]()

/// The label string that describes the DataSet.

open var label: String? = "DataSet"

/// The axis this DataSet should be plotted against.

open var axisDependency = YAxis.AxisDependency.left

/// - Returns: The color at the given index of the DataSet's color array.

/// This prevents out-of-bounds by performing a modulus on the color index, so colours
will repeat themselves.

open func color(atIndex index: Int) -> NSUIColor

{

    var index = index

    if index < 0

```

```

    {

        index = 0

    }

    return colors[index % colors.count]

}

/// Resets all colors of this DataSet and recreates the colors array.

open func resetColors()

{

    colors.removeAll(keepingCapacity: false)

}

/// Adds a new color to the colors array of the DataSet.

///

/// - Parameters:

/// - color: the color to add

open func addColor(_ color: NSUIColor)

```

```

{

    colors.append(color)

}

/// Sets the one and **only** color that should be used for this DataSet.

/// Internally, this recreates the colors array and adds the specified color.

///

/// - Parameters:

///   - color: the color to set

open func setColor(_ color: NSUIColor)

{

    colors.removeAll(keepingCapacity: false)

    colors.append(color)

}

/// Sets colors to a single color a specific alpha value.

///

```

```

/// - Parameters:

/// - color: the color to set

/// - alpha: alpha to apply to the set `color`

@objc open func setColor(_ color: NSUIColor, alpha: CGFloat)

{

    setColor(color.withAlphaComponent(alpha))

}

/// Sets colors with a specific alpha value.

///

/// - Parameters:

/// - colors: the colors to set

/// - alpha: alpha to apply to the set `colors`

@objc open func setColors(_ colors: [NSUIColor], alpha: CGFloat)

{

    self.colors = colors.map { $0.withAlphaComponent(alpha) }

}

```



```

/// Sets colors with a specific alpha value.

///

/// - Parameters:

/// - colors: the colors to set

/// - alpha: alpha to apply to the set `colors`

open func setColors(_ colors: NSUIColor...)

{

    self.colors = colors

}


/// if true, value highlighting is enabled

open var highlightEnabled = true


/// `true` if value highlighting is enabled for this dataset

open var isHighlightEnabled: Bool { return highlightEnabled }

```

```
/// Custom formatter that is used instead of the auto-formatter if set
```

```
internal var _valueFormatter: IValueFormatter?
```

```
/// Custom formatter that is used instead of the auto-formatter if set
```

```
open var valueFormatter: IValueFormatter?
```

```
{
```

```
    get
```

```
    {
```

```
        if needsFormatter
```

```
        {
```

```
            return ChartUtils.defaultValueFormatter()
```

```
        }
```

```
        return _valueFormatter
```

```
    }
```

```
    set
```

```
    {
```

```
if newValue == nil { return }
```

```
    _valueFormatter = newValue
```

```
}
```

```
}
```

```
open var needsFormatter: Bool
```

```
{
```

```
    return _valueFormatter == nil
```

```
}
```

```
/// Sets/get a single color for value text.
```

```
/// Setting the color clears the colors array and adds a single color.
```

```
/// Getting will return the first color in the array.
```

```
open var valueTextColor: UIColor
```

```
{
```

```
    get
```

```

{

    return valueColors[0]

}

set

{

    valueColors.removeAll(keepingCapacity: false)

    valueColors.append(newValue)

}

}

```

/// - Returns: The color at the specified index that is used for drawing the values inside the chart. Uses modulus internally.

```

open fun valueTextColorAt(_ index: Int) -> NSUIColor

{

    var index = index

    if index < 0

    {

        index = 0
    }
}

```

```

    }

    return valueColors[index % valueColors.count]

}

/// the font for the value-text labels

open var valueFont: NSUIFont = NSUIFont.systemFont(ofSize: 7.0)

/// The form to draw for this dataset in the legend.

open var form = Legend.Form.default

/// The form size to draw for this dataset in the legend.

///

/// Return `NaN` to use the default legend form size.

open var formSize: CGFloat = CGFloat.nan

/// The line width for drawing the form of this dataset in the legend

///

```

```
/// Return `NaN` to use the default legend form line width.
```

```
open var formLineWidth: CGFloat = CGFloat.nan
```

```
/// Line dash configuration for legend shapes that consist of lines.
```

```
///
```

```
/// This is how much (in pixels) into the dash pattern are we starting from.
```

```
open var formLineDashPhase: CGFloat = 0.0
```

```
/// Line dash configuration for legend shapes that consist of lines.
```

```
///
```

```
/// This is the actual dash pattern.
```

```
/// I.e. [2, 3] will paint [-- -- ]
```

```
/// [1, 3, 4, 2] will paint [- ---- - ---- ]
```

```
open var formLineDashLengths: [CGFloat]? = nil
```

```
/// Set this to true to draw y-values on the chart.
```

```
///
```

/// - Note: For bar and line charts: if `maxVisibleCount` is reached, no values will be drawn even if this is enabled.

```
open var drawValuesEnabled = true
```

```
/// `true` if y-value drawing is enabled, `false` if not
```

```
open var isDrawValuesEnabled: Bool
```

```
{
```

```
    return drawValuesEnabled
```

```
}
```

```
/// Set this to true to draw y-icons on the chart.
```

```
///
```

/// - Note: For bar and line charts: if `maxVisibleCount` is reached, no icons will be drawn even if this is enabled.

```
open var drawIconsEnabled = true
```

```
/// Returns true if y-icon drawing is enabled, false if not
```

```
open var isDrawIconsEnabled: Bool
```

```

{

    return drawIconsEnabled

}

/// Offset of icons drawn on the chart.

///

/// For all charts except Pie and Radar it will be ordinary (x offset, y offset).

///

/// For Pie and Radar chart it will be (y offset, distance from center offset); so if you want
icon to be rendered under value, you should increase X component of CGPoint, and if you
want icon to be rendered closet to center, you should decrease height component of
CGPoint.

open var iconsOffset = CGPoint(x: 0, y: 0)

/// Set the visibility of this DataSet. If not visible, the DataSet will not be drawn to the
chart upon refreshing it.

open var visible = true

/// `true` if this DataSet is visible inside the chart, or `false` if it is currently hidden.

```



```
open var isVisible: Bool
```

```
{  
  
    return visible  
  
}
```

```
// MARK: - NSObject
```

```
open override var description: String
```

```
{  
  
    return String(format: "%@, label: %@, %i entries", arguments:  
[NSStringFromClass(type(of: self)), self.label ?? "", self.entryCount])  
  
}
```

```
open override var debugDescription: String
```

```
{  
  
    return (0..  
entryCount).reduce(description + ":",) {  
  
        "($0)\n\n(self.entryForIndex($1)?.description ?? "")"  
  
    }  
  
}
```

```
}
```

```
// MARK: - NSCopying
```

```
open func copy(with zone: NSZone? = nil) -> Any
```

```
{
```

```
    let copy = type(of: self).init()
```

```
    copy.colors = colors
```

```
    copy.valueColors = valueColors
```

```
    copy.label = label
```

```
    copy.axisDependency = axisDependency
```

```
    copy.highlightEnabled = highlightEnabled
```

```
    copy._valueFormatter = _valueFormatter
```

```
    copy.valueFont = valueFont
```

```
    copy.form = form
```

```
    copy.formSize = formSize
```

```
        copy.formLineWidth = formLineWidth

        copy.formLineDashPhase = formLineDashPhase

        copy.formLineDashLengths = formLineDashLengths

        copy.drawValuesEnabled = drawValuesEnabled

        copy.drawValuesEnabled = drawValuesEnabled

        copy.iconsOffset = iconsOffset

        copy.visible = visible

    }

}

//

// ChartColorTemplates.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS
```

```
// Licensed under Apache License 2.0
```

```
//
```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
open class ChartColorTemplates: NSObject
```

```
{
```

```
    @objc open class func liberty () -> [NSUIColor]
```

```
{
```

```
    return [
```

```
        NSUIColor(red: 207/255.0, green: 248/255.0, blue: 246/255.0, alpha: 1.0),
```

```
        NSUIColor(red: 148/255.0, green: 212/255.0, blue: 212/255.0, alpha: 1.0),
```

```
        NSUIColor(red: 136/255.0, green: 180/255.0, blue: 187/255.0, alpha: 1.0),
```

```
        NSUIColor(red: 118/255.0, green: 174/255.0, blue: 175/255.0, alpha: 1.0),
```

```
NSUIColor(red: 42/255.0, green: 109/255.0, blue: 130/255.0, alpha: 1.0)
```

```
]
```

```
}
```

```
@objc open class func joyful () -> [UIColor]
```

```
{
```

```
return [
```

```
    UIColor(red: 217/255.0, green: 80/255.0, blue: 138/255.0, alpha: 1.0),
```

```
    UIColor(red: 254/255.0, green: 149/255.0, blue: 7/255.0, alpha: 1.0),
```

```
    UIColor(red: 254/255.0, green: 247/255.0, blue: 120/255.0, alpha: 1.0),
```

```
    UIColor(red: 106/255.0, green: 167/255.0, blue: 134/255.0, alpha: 1.0),
```

```
    UIColor(red: 53/255.0, green: 194/255.0, blue: 209/255.0, alpha: 1.0)
```

```
]
```

```
}
```

```
@objc open class func pastel () -> [UIColor]
```

```
{
```

```

return [

    NSUIColor(red: 64/255.0, green: 89/255.0, blue: 128/255.0, alpha: 1.0),

    NSUIColor(red: 149/255.0, green: 165/255.0, blue: 124/255.0, alpha: 1.0),

    NSUIColor(red: 217/255.0, green: 184/255.0, blue: 162/255.0, alpha: 1.0),

    NSUIColor(red: 191/255.0, green: 134/255.0, blue: 134/255.0, alpha: 1.0),

    NSUIColor(red: 179/255.0, green: 48/255.0, blue: 80/255.0, alpha: 1.0)

]

}

```

```

@objc open class func colorful () -> [NSUIColor]

```

```

{

    return [

        NSUIColor(red: 193/255.0, green: 37/255.0, blue: 82/255.0, alpha: 1.0),

        NSUIColor(red: 255/255.0, green: 102/255.0, blue: 0/255.0, alpha: 1.0),

        NSUIColor(red: 245/255.0, green: 199/255.0, blue: 0/255.0, alpha: 1.0),

        NSUIColor(red: 106/255.0, green: 150/255.0, blue: 31/255.0, alpha: 1.0),

        NSUIColor(red: 179/255.0, green: 100/255.0, blue: 53/255.0, alpha: 1.0)

    ]

}

```

```
]
}
```

```
@objc open class func vordiplom () -> [NSUIColor]
```

```
{
    return [
        NSUIColor(red: 192/255.0, green: 255/255.0, blue: 140/255.0, alpha: 1.0),
        NSUIColor(red: 255/255.0, green: 247/255.0, blue: 140/255.0, alpha: 1.0),
        NSUIColor(red: 255/255.0, green: 208/255.0, blue: 140/255.0, alpha: 1.0),
        NSUIColor(red: 140/255.0, green: 234/255.0, blue: 255/255.0, alpha: 1.0),
        NSUIColor(red: 255/255.0, green: 140/255.0, blue: 157/255.0, alpha: 1.0)
    ]
}
```

```
@objc open class func material () -> [NSUIColor]
```

```
{
    return [
```

NSUIColor(red: 46/255.0, green: 204/255.0, blue: 113/255.0, alpha: 1.0),

NSUIColor(red: 241/255.0, green: 196/255.0, blue: 15/255.0, alpha: 1.0),

NSUIColor(red: 231/255.0, green: 76/255.0, blue: 60/255.0, alpha: 1.0),

NSUIColor(red: 52/255.0, green: 152/255.0, blue: 219/255.0, alpha: 1.0)

]

}

@objc open class func colorFromString(_ colorString: String) -> NSUIColor

{

let leftParenCharset: CharacterSet = CharacterSet(charactersIn: "(")

let commaCharset: CharacterSet = CharacterSet(charactersIn: ", ")

let colorString = colorString.lowercased()

if colorString.hasPrefix("#")

{

var argb: [UInt] = [255, 0, 0, 0]


```

let colorString = colorString.unicodeScalars

var length = colorString.count

var index = colorString.startIndex

let endIndex = colorString.endIndex


index = colorString.index(after: index)

length = length - 1


if length == 3 || length == 6 || length == 8

{

    var i = length == 8 ? 0 : 1

    while index < endIndex

    {

        var c = colorString[index]

        index = colorString.index(after: index)


        var val = (c.value >= 0x61 && c.value <= 0x66) ? (c.value - 0x61 + 10) :
c.value - 0x30

```

```

    argb[i] = UInt(val) * 16

    if length == 3

    {

        argb[i] = argb[i] + UInt(val)

    }

    else

    {

        c = colorString[index]

        index = colorString.index(after: index)

        val = (c.value >= 0x61 && c.value <= 0x66) ? (c.value - 0x61 + 10) : c.value
- 0x30

        argb[i] = argb[i] + UInt(val)

    }

    i += 1

}

}

```

```

        return NSUIColor(red: CGFloat(Argb[1]) / 255.0, green: CGFloat(Argb[2]) / 255.0,
blue: CGFloat(Argb[3]) / 255.0, alpha: CGFloat(Argb[0]) / 255.0)

    }

    else if colorString.hasPrefix("rgba")

    {

        var a: Float = 1.0

        var r: Int32 = 0

        var g: Int32 = 0

        var b: Int32 = 0

        let scanner: Scanner = Scanner(string: colorString)

        scanner.scanString("rgba", into: nil)

        scanner.scanCharacters(from: leftParenCharset, into: nil)

        scanner.scanInt32(&r)

        scanner.scanCharacters(from: commaCharset, into: nil)

        scanner.scanInt32(&g)

        scanner.scanCharacters(from: commaCharset, into: nil)

        scanner.scanInt32(&b)

```

```

scanner.scanCharacters(from: commaCharset, into: nil)

scanner.scanFloat(&a)

return NSUIColor(

    red: CGFloat(r) / 255.0,

    green: CGFloat(g) / 255.0,

    blue: CGFloat(b) / 255.0,

    alpha: CGFloat(a)

)

}

else if colorString.hasPrefix("argb")

{

    var a: Float = 1.0

    var r: Int32 = 0

    var g: Int32 = 0

    var b: Int32 = 0

    let scanner: Scanner = Scanner(string: colorString)

    scanner.scanString("argb", into: nil)

```

```

scanner.scanCharacters(from: leftParenCharset, into: nil)

scanner.scanFloat(&a)

scanner.scanCharacters(from: commaCharset, into: nil)

scanner.scanInt32(&r)

scanner.scanCharacters(from: commaCharset, into: nil)

scanner.scanInt32(&g)

scanner.scanCharacters(from: commaCharset, into: nil)

scanner.scanInt32(&b)

return NSUIColor(

    red: CGFloat(r) / 255.0,

    green: CGFloat(g) / 255.0,

    blue: CGFloat(b) / 255.0,

    alpha: CGFloat(a)

)

}

else if colorString.hasPrefix("rgb")

{

```

```

var r: Int32 = 0

var g: Int32 = 0

var b: Int32 = 0

let scanner: Scanner = Scanner(string: colorString)

scanner.scanString("rgb", into: nil)

scanner.scanCharacters(from: leftParenCharset, into: nil)

scanner.scanInt32(&r)

scanner.scanCharacters(from: commaCharset, into: nil)

scanner.scanInt32(&g)

scanner.scanCharacters(from: commaCharset, into: nil)

scanner.scanInt32(&b)

return NSUIColor(

    red: CGFloat(r) / 255.0,

    green: CGFloat(g) / 255.0,

    blue: CGFloat(b) / 255.0,

    alpha: 1.0

)

```

```

    }

    return NSUIColor.clear

}

}

//

// ChartData.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

```

```

open class ChartData: NSObject

{

    internal var _yMax: Double = -Double.greatestFiniteMagnitude

    internal var _yMin: Double = Double.greatestFiniteMagnitude

    internal var _xMax: Double = -Double.greatestFiniteMagnitude

    internal var _xMin: Double = Double.greatestFiniteMagnitude

    internal var _leftAxisMax: Double = -Double.greatestFiniteMagnitude

    internal var _leftAxisMin: Double = Double.greatestFiniteMagnitude

    internal var _rightAxisMax: Double = -Double.greatestFiniteMagnitude

    internal var _rightAxisMin: Double = Double.greatestFiniteMagnitude


    internal var _dataSets = [IChartDataSet]()


    public override init()

    {

        super.init()
    }
}

```



```
    _dataSets = [IChartDataSet]()  
  
}
```

```
@objc public init(dataSets: [IChartDataSet]?)  
  
{  
  
    super.init()  
  

```

```
    _dataSets = dataSets ?? [IChartDataSet]()
```

```
    self.initialize(dataSets: _dataSets)  
  
}
```

```
@objc public convenience init(dataSet: IChartDataSet?)  
  
{  
  
    self.init(dataSets: dataSet === nil ? nil : [dataSet!])  
  
}
```

```

internal func initialize(dataSets: [IChartDataSet])

{

    notifyDataChanged()

}

/// Call this method to let the ChartData know that the underlying data has changed.

/// Calling this performs all necessary recalculations needed when the contained data has
changed.

@objc open func notifyDataChanged()

{

    calcMinMax()

}

@objc open func calcMinMaxY(fromX: Double, toX: Double)

{

    _dataSets.forEach { $0.calcMinMaxY(fromX: fromX, toX: toX) }

    // apply the new data

```

```

    calcMinMax()

}

/// calc minimum and maximum y value over all datasets

@objc open func calcMinMax()

{

    _yMax = -Double.greatestFiniteMagnitude

    _yMin = Double.greatestFiniteMagnitude

    _xMax = -Double.greatestFiniteMagnitude

    _xMin = Double.greatestFiniteMagnitude


    _dataSets.forEach { calcMinMax(dataSet: $0) }


    _leftAxisMax = -Double.greatestFiniteMagnitude

    _leftAxisMin = Double.greatestFiniteMagnitude

    _rightAxisMax = -Double.greatestFiniteMagnitude

    _rightAxisMin = Double.greatestFiniteMagnitude

```

```

// left axis

let firstLeft = getFirstLeft(dataSets: dataSets)

if firstLeft !== nil

{

    _leftAxisMax = firstLeft!.yMax

    _leftAxisMin = firstLeft!.yMin

    for dataSet in _dataSets

    {

        if dataSet.axisDependency == .left

        {

            if dataSet.yMin < _leftAxisMin

            {

                _leftAxisMin = dataSet.yMin

            }

        }

    }

}

```

```

        if dataSet.yMax > _leftAxisMax

        {

            _leftAxisMax = dataSet.yMax

        }

    }

}

// right axis

let firstRight = getFirstRight(dataSets: dataSets)

if firstRight !== nil

{

    _rightAxisMax = firstRight!.yMax

    _rightAxisMin = firstRight!.yMin

```

```

for dataSet in _dataSets

{

    if dataSet.axisDependency == .right

    {

        if dataSet.yMin < _rightAxisMin

        {

            _rightAxisMin = dataSet.yMin

        }

        if dataSet.yMax > _rightAxisMax

        {

            _rightAxisMax = dataSet.yMax

        }

    }

}

```

/// Adjusts the current minimum and maximum values based on the provided Entry object.

```
@objc open func calcMinMax(entry e: ChartDataEntry, axis: YAxis.AxisDependency)
```

```
{
```

```
    if _yMax < e.y
```

```
    {
```

```
        _yMax = e.y
```

```
    }
```

```
    if _yMin > e.y
```

```
    {
```

```
        _yMin = e.y
```

```
    }
```

```
    if _xMax < e.x
```

```
    {
```

```
        _xMax = e.x
```

```
}
```

```
if _xMin > e.x
```

```
{
```

```
    _xMin = e.x
```

```
}
```

```
if axis == .left
```

```
{
```

```
    if _leftAxisMax < e.y
```

```
    {
```

```
        _leftAxisMax = e.y
```

```
    }
```

```
    if _leftAxisMin > e.y
```

```
    {
```

```
        _leftAxisMin = e.y
```



```

    }

}

else

{

    if _rightAxisMax < e.y

    {

        _rightAxisMax = e.y

    }

    if _rightAxisMin > e.y

    {

        _rightAxisMin = e.y

    }

}

}

```

/// Adjusts the minimum and maximum values based on the given DataSet.

```
@objc open func calcMinMax(dataSet d: IChartDataSet)
```

```
{
```

```
    if _yMax < d.yMax
```

```
    {
```

```
        _yMax = d.yMax
```

```
    }
```

```
    if _yMin > d.yMin
```

```
    {
```

```
        _yMin = d.yMin
```

```
    }
```

```
    if _xMax < d.xMax
```

```
    {
```

```
        _xMax = d.xMax
```

```
    }
```

```

if _xMin > d.xMin

{

    _xMin = d.xMin

}


if d.axisDependency == .left

{

    if _leftAxisMax < d.yMax

    {

        _leftAxisMax = d.yMax

    }


    if _leftAxisMin > d.yMin

    {

        _leftAxisMin = d.yMin

    }

}

```

```

else

{

    if _rightAxisMax < d.yMax

    {

        _rightAxisMax = d.yMax

    }

    if _rightAxisMin > d.yMin

    {

        _rightAxisMin = d.yMin

    }

}

}

/// The number of LineDataSets this object contains

@objc open var dataSetCount: Int

{

```

```

        return _dataSets.count
    }

    /// The smallest y-value the data object contains.

    @objc open var yMin: Double

    {

        return _yMin
    }

    @nonobjc

    open func getYMin() -> Double

    {

        return _yMin
    }

    @objc open func getYMin(axis: YAxis.AxisDependency) -> Double

    {

```

```

if axis == .left

{

    if _leftAxisMin == Double.greatestFiniteMagnitude

    {

        return _rightAxisMin

    }

    else

    {

        return _leftAxisMin

    }

}

else

{

    if _rightAxisMin == Double.greatestFiniteMagnitude

    {

        return _leftAxisMin

    }

}

```

```

        else

        {

            return _rightAxisMin

        }

    }

}

```

/// The greatest y-value the data object contains.

```
@objc open var yMax: Double
```

```

{

    return _yMax

}

```

```
@nonobjc
```

```
open func getYMax() -> Double
```

```

{

    return _yMax

}

```

```
}
```

```
@objc open func getYMax(axis: YAxis.AxisDependency) -> Double
```

```
{
```

```
    if axis == .left
```

```
    {
```

```
        if _leftAxisMax == -Double.greatestFiniteMagnitude
```

```
        {
```

```
            return _rightAxisMax
```

```
        }
```

```
    else
```

```
    {
```

```
        return _leftAxisMax
```

```
    }
```

```
}
```

```
    else
```

```
    {
```



```

        if _rightAxisMax == -Double.greatestFiniteMagnitude

        {

            return _leftAxisMax

        }

        else

        {

            return _rightAxisMax

        }

    }

}

```

/// The minimum x-value the data object contains.

```
@objc open var xMin: Double
```

```

{

    return _xMin

}

```

/// The maximum x-value the data object contains.

```
@objc open var xMax: Double
```

```
{  
  
    return _xMax  
  
}
```

```
/// All DataSet objects this ChartData object holds.
```

```
@objc open var dataSets: [IChartDataSet]
```

```
{  
  
    get  
  
    {  
  
        return _dataSets  
  
    }  
  
    set  
  
    {  
  
        _dataSets = newValue  
  
        notifyDataChanged()  
  
    }  
  
}
```

```
}
```

```
/// Retrieve the index of a ChartDataSet with a specific label from the ChartData. Search  
can be case sensitive or not.
```

```
///
```

```
/// **IMPORTANT: This method does calculations at runtime, do not over-use in  
performance critical situations.**
```

```
///
```

```
/// - Parameters:
```

```
/// - dataSets: the DataSet array to search
```

```
/// - type:
```

```
/// - ignorecase: if true, the search is not case-sensitive
```

```
/// - Returns: The index of the DataSet Object with the given label. Sensitive or not.
```

```
internal func getDataSetIndexByLabel(_ label: String, ignorecase: Bool) -> Int
```

```
{
```

```
    // TODO: Return nil instead of -1
```

```
    if ignorecase
```

```
    {
```

```

        return dataSets.firstIndex { $0.label?.caseInsensitiveCompare(label)
== .orderedSame }

        ?? -1

    }

    else

    {

        return dataSets.firstIndex { $0.label == label }

        ?? -1

    }

}

```

/// Get the Entry for a corresponding highlight object

///

/// - Parameters:

/// - highlight:

/// - Returns: The entry that is highlighted

@objc open func entryForHighlight(_ highlight: Highlight) -> ChartDataEntry?

{

```

    if highlight.dataSetIndex >= dataSets.count

    {

        return nil

    }

    else

    {

        return dataSets[highlight.dataSetIndex].entryForXValue(highlight.x, closestToY:
highlight.y)

    }

}

```

/// ****IMPORTANT:** This method does calculations at runtime. Use with care in performance critical situations.**

///

/// - Parameters:

/// - label:

/// - ignorecase:

/// - Returns: The DataSet Object with the given label. Sensitive or not.

```
@objc open func getDataSetByLabel(_ label: String, ignorecase: Bool) ->
IChartDataSet?
```

```
{
```

```
    let index = getDataSetIndexByLabel(label, ignorecase: ignorecase)
```

```
    if index < 0 || index >= _dataSets.count
```

```
    {
```

```
        return nil
```

```
    }
```

```
    else
```

```
    {
```

```
        return _dataSets[index]
```

```
    }
```

```
}
```

```
@objc open func getDataSetByIndex(_ index: Int) -> IChartDataSet!
```

```
{
```

```
    if index < 0 || index >= _dataSets.count
```

```

    {

        return nil

    }

    return _dataSets[index]

}

@objc open func addDataSet(_ dataSet: IChartDataSet!)

{

    calcMinMax(dataSet: dataSet)

    _dataSets.append(dataSet)

}

/// Removes the given DataSet from this data object.

/// Also recalculates all minimum and maximum values.

///

```

/// - Returns: `true` if a DataSet was removed, `false` ifno DataSet could be removed.

```
@objc @discardableResult open func removeDataSet(_ dataSet: IChartDataSet) ->
Bool
```

```
{

    guard let i = _dataSets.firstIndex(where: { $0 === dataSet }) else { return false }

    return removeDataSetByIndex(i)

}
```

/// Removes the DataSet at the given index in the DataSet array from the data object.

/// Also recalculates all minimum and maximum values.

///

/// - Returns: `true` if a DataSet was removed, `false` ifno DataSet could be removed.

```
@objc @discardableResult open func removeDataSetByIndex(_ index: Int) -> Bool
```

```
{

    if index >= _dataSets.count || index < 0

    {

        return false

    }

}
```



```
_dataSets.remove(at: index)
```

```
calcMinMax()
```

```
return true
```

```
}
```

/// Adds an Entry to the DataSet at the specified index. Entries are added to the end of the list.

```
@objc open func addEntry(_ e: ChartDataEntry, dataSetIndex: Int)
```

```
{
```

```
    if _dataSets.count > dataSetIndex && dataSetIndex >= 0
```

```
    {
```

```
        let set = _dataSets[dataSetIndex]
```

```
        if !set.addEntry(e) { return }
```

```

        calcMinMax(entry: e, axis: set.axisDependency)

    }

    else

    {

        print("ChartData.addEntry() - Cannot add Entry because dataSetIndex too high or
too low.", terminator: "\n")

    }

}

```

/// Removes the given Entry object from the DataSet at the specified index.

```

@objc @discardableResult open func removeEntry(_ entry: ChartDataEntry,
dataSetIndex: Int) -> Bool

{

    // entry outofbounds

    if dataSetIndex >= _dataSets.count

    {

        return false

    }

}

```

```

// remove the entry from the dataset

let removed = _dataSets[dataSetIndex].removeEntry(entry)

if removed

{

    calcMinMax()

}

return removed

}

```

```

/// Removes the Entry object closest to the given xIndex from the ChartDataSet at the

/// specified index.

///

/// - Returns: `true` if an entry was removed, `false` if no Entry was found that meets the
specified requirements.

```

```
@objc @discardableResult open func removeEntry(xValue: Double, dataSetIndex: Int)
-> Bool
```

```
{

    if dataSetIndex >= _dataSets.count

    {

        return false

    }


    if let entry = _dataSets[dataSetIndex].entryForXValue(xValue, closestToY:
Double.nan)

    {

        return removeEntry(entry, dataSetIndex: dataSetIndex)

    }


    return false

}
```

/// - Returns: The DataSet that contains the provided Entry, or null, if no DataSet contains this entry.

```
@objc open func getDataSetForEntry(_ e: ChartDataEntry) -> IChartDataSet?
```

```
{
```

```
    return _dataSets.first { $0.entryForXValue(e.x, closestToY: e.y) === e }
```

```
}
```

/// - Returns: The index of the provided DataSet in the DataSet array of this data object, or -1 if it does not exist.

```
@objc open func indexOfDataSet(_ dataSet: IChartDataSet) -> Int
```

```
{
```

```
    // TODO: Return nil instead of -1
```

```
    return _dataSets.firstIndex { $0 === dataSet } ?? -1
```

```
}
```

/// - Returns: The first DataSet from the datasets-array that has it's dependency on the left axis. Returns null if no DataSet with left dependency could be found.

```
@objc open func getFirstLeft(dataSets: [IChartDataSet]) -> IChartDataSet?
```

```
{
```

```
    return dataSets.first { $0.axisDependency == .left }
```

```
}
```

/// - Returns: The first DataSet from the datasets-array that has it's dependency on the right axis. Returns null if no DataSet with right dependency could be found.

```
@objc open func getFirstRight(dataSets: [IChartDataSet]) -> IChartDataSet?
```

```
{
```

```
    return dataSets.first { $0.axisDependency == .right }
```

```
}
```

/// - Returns: All colors used across all DataSet objects this object represents.

```
@objc open func getColors() -> [NSUIColor]?
```

```
{
```

```
    // TODO: Don't return nil
```

```
    return _dataSets.flatMap { $0.colors }
```

```
}
```

/// Sets a custom IValueFormatter for all DataSets this data object contains.

```
@objc open func setValueFormatter(_ formatter: IValueFormatter)
```

```
{  
  
    dataSets.forEach { $0.valueFormatter = formatter }  
  
}
```

/// Sets the color of the value-text (color in which the value-labels are drawn) for all DataSets this data object contains.

```
@objc open func setValueTextColor(_ color: NSUIColor)  
  
{  
  
    dataSets.forEach { $0.valueTextColor = color }  
  
}
```

/// Sets the font for all value-labels for all DataSets this data object contains.

```
@objc open func setValueFont(_ font: NSUIFont)  
  
{  
  
    dataSets.forEach { $0.valueFont = font }  
  
}
```

/// Enables / disables drawing values (value-text) for all DataSets this data object contains.

```
@objc open func setDrawValues(_ enabled: Bool)

{

    dataSets.forEach { $0.drawValuesEnabled = enabled }

}
```

/// Enables / disables highlighting values for all DataSets this data object contains.

/// If set to true, this means that values can be highlighted programmatically or by touch gesture.

```
@objc open var highlightEnabled: Bool

{

    get { return dataSets.allSatisfy { $0.highlightEnabled } }

    set { dataSets.forEach { $0.highlightEnabled = newValue } }

}
```

/// if true, value highlightning is enabled

```
@objc open var isHighlightEnabled: Bool { return highlightEnabled }
```



```
/// Clears this data object from all DataSets and removes all Entries.
```

```
/// Don't forget to invalidate the chart after this.
```

```
@objc open func clearValues()
```

```
{  
  
    dataSets.removeAll(keepingCapacity: false)  
  
    notifyDataChanged()  
  
}
```

```
/// Checks if this data object contains the specified DataSet.
```

```
///
```

```
/// - Returns: `true` if so, `false` ifnot.
```

```
@objc open func contains(dataSet: IChartDataSet) -> Bool
```

```
{  
  
    return dataSets.contains { $0 === dataSet }  
  
}
```

/// The total entry count across all DataSet objects this data object contains.

@objc open var entryCount: Int

{

return _dataSets.reduce(0) { \$0 + \$1.entryCount }

}

/// The DataSet object with the maximum number of entries or null if there are no DataSets.

@objc open var maxEntryCountSet: IChartDataSet?

{

return dataSets.max { \$0.entryCount < \$1.entryCount }

}

// MARK: - Accessibility

/// When the data entry labels are generated identifiers, set this property to prepend a string before each identifier

///

/// For example, if a label is "#3", settings this property to "Item" allows it to be spoken as "Item #3"

@objc open var accessibilityEntryLabelPrefix: String?

/// When the data entry value requires a unit, use this property to append the string representation of the unit to the value

///

/// For example, if a value is "44.1", setting this property to "m" allows it to be spoken as "44.1 m"

@objc open var accessibilityEntryLabelSuffix: String?

/// If the data entry value is a count, set this to true to allow plurals and other grammatical changes

/// **default**: false

@objc open var accessibilityEntryLabelSuffixIsCount: Bool = false

}

//

// ChartDataEntry.swift

// Charts

```
//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

open class ChartDataEntry: ChartDataEntryBase, NSCopying

{

    /// the x value

    @objc open var x = 0.0

    public required init()

    {
```

```

        super.init()

    }

    /// An Entry represents one single entry in the chart.

    ///

    /// - Parameters:

    ///   - x: the x value

    ///   - y: the y value (the actual value of the entry)

    @objc public init(x: Double, y: Double)

    {

        super.init(y: y)

        self.x = x

    }

    /// An Entry represents one single entry in the chart.

    ///

    /// - Parameters:

```

```
/// - x: the x value
```

```
/// - y: the y value (the actual value of the entry)
```

```
/// - data: Space for additional data this Entry represents.
```

```
@objc public convenience init(x: Double, y: Double, data: Any?)
```

```
{
```

```
    self.init(x: x, y: y)
```

```
    self.data = data
```

```
}
```

```
/// An Entry represents one single entry in the chart.
```

```
///
```

```
/// - Parameters:
```

```
/// - x: the x value
```

```
/// - y: the y value (the actual value of the entry)
```

```
/// - icon: icon image
```

```
@objc public convenience init(x: Double, y: Double, icon: NSUIImage?)
```

```
{
```

```
    self.init(x: x, y: y)
```

```
    self.icon = icon
```

```
}
```

```
/// An Entry represents one single entry in the chart.
```

```
///
```

```
/// - Parameters:
```

```
/// - x: the x value
```

```
/// - y: the y value (the actual value of the entry)
```

```
/// - icon: icon image
```

```
/// - data: Space for additional data this Entry represents.
```

```
@objc public convenience init(x: Double, y: Double, icon: NSUIImage?, data: Any?)
```

```
{
```

```
    self.init(x: x, y: y)
```

```

        self.icon = icon

        self.data = data

    }

// MARK: NSObject

open override var description: String

{

    return "ChartDataEntry, x: \(x), y \(y)"

}

// MARK: NSCopying

open func copy(with zone: NSZone? = nil) -> Any

{

    let copy = type(of: self).init()

```



```

    copy.x = x

    copy.y = y

    copy.data = data

    return copy
}

}

// MARK: Equatable

extension ChartDataEntry/*: Equatable*/ {

    open override func isEqual(_ object: Any?) -> Bool {

        guard let object = object as? ChartDataEntry else { return false }

        if self === object

        {

            return true

        }

```

```
        return y == object.y

        && x == object.x

    }

}

//

// ChartDataEntryBase.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation
```

```

open class ChartDataEntryBase: NSObject

{

    /// the y value

    @objc open var y = 0.0


    /// optional spot for additional data this Entry represents

    @objc open var data: Any?


    /// optional icon image

    @objc open var icon: NSUIImage?


    public override required init()

    {

        super.init()

    }

```

```
/// An Entry represents one single entry in the chart.
```

```
///
```

```
/// - Parameters:
```

```
/// - y: the y value (the actual value of the entry)
```

```
@objc public init(y: Double)
```

```
{
```

```
    super.init()
```

```
    self.y = y
```

```
}
```

```
/// - Parameters:
```

```
/// - y: the y value (the actual value of the entry)
```

```
/// - data: Space for additional data this Entry represents.
```

```
@objc public convenience init(y: Double, data: Any?)
```

```
{
```

```
self.init(y: y)
```

```
self.data = data
```

```
}
```

```
/// - Parameters:
```

```
/// - y: the y value (the actual value of the entry)
```

```
/// - icon: icon image
```

```
@objc public convenience init(y: Double, icon: NSUIImage?)
```

```
{
```

```
self.init(y: y)
```

```
self.icon = icon
```

```
}
```

```
/// - Parameters:
```

```
/// - y: the y value (the actual value of the entry)
```

```
/// - icon: icon image
```

```
/// - data: Space for additional data this Entry represents.
```

```
@objc public convenience init(y: Double, icon: NSUIImage?, data: Any?)
```

```
{
```

```
    self.init(y: y)
```

```
    self.icon = icon
```

```
    self.data = data
```

```
}
```

```
// MARK: NSObject
```

```
open override var description: String
```

```
{
```

```
    return "ChartDataEntryBase, y \ \(y)"
```

```

    }

}

// MARK: Equatable

extension ChartDataEntryBase/*: Equatable*/ {

    open override func isEqual(_ object: Any?) -> Bool {

        guard let object = object as? ChartDataEntryBase else { return false }

        if self === object

        {

            return true

        }

        return y == object.y

    }

}

//

```

```

// ChartDataProvider.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc

public protocol ChartDataProvider

{

    /// The minimum x-value of the chart, regardless of zoom or translation.

```



```
var chartXMin: Double { get }
```

```
/// The maximum x-value of the chart, regardless of zoom or translation.
```

```
var chartXMax: Double { get }
```

```
/// The minimum y-value of the chart, regardless of zoom or translation.
```

```
var chartYMin: Double { get }
```

```
/// The maximum y-value of the chart, regardless of zoom or translation.
```

```
var chartYMax: Double { get }
```

```
var maxHighlightDistance: CGFloat { get }
```

```
var xRange: Double { get }
```

```
var centerOffsets: CGPoint { get }
```

```

var data: ChartData? { get }

var maxVisibleCount: Int { get }

}

//

// DataRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

```

```
#if canImport(UiKit)
```

```
import UIKit
```

```
#endif
```

```
#if canImport(Cocoa)
```

```
import Cocoa
```

```
#endif
```

```
@objc(ChartDataRendererBase)
```

```
open class DataRenderer: Renderer
```

```
{
```

```
    /// An array of accessibility elements that are presented to the ChartViewBase  
    accessibility methods.
```

```
    ///
```

```
    /// Note that the order of elements in this array determines the order in which they are  
    presented and navigated by
```

```
    /// Accessibility clients such as VoiceOver.
```

```
///
```

/// Renderers should ensure that the order of elements makes sense to a client presenting an audio-only interface to a user.

/// Subclasses should populate this array in drawData() or drawDataSet() to make the chart accessible.

```
@objc final var accessibleChartElements: [NSUIAccessibilityElement] = []
```

```
@objc public let animator: Animator
```

```
@objc public init(animator: Animator, viewPortHandler: ViewPortHandler)
```

```
{
```

```
    self.animator = animator
```

```
    super.init(viewPortHandler: viewPortHandler)
```

```
}
```

```
@objc open func drawData(context: CGContext)
```

```
{
```

```

        fatalError("drawData() cannot be called on DataRenderer")

    }

    @objc open func drawValues(context: CGContext)

    {

        fatalError("drawValues() cannot be called on DataRenderer")

    }

```

```

    @objc open func drawExtras(context: CGContext)

    {

        fatalError("drawExtras() cannot be called on DataRenderer")

    }

```

```

/// Draws all highlight indicators for the values that are currently highlighted.

```

```

///

```

```

/// - Parameters:

```

```

/// - indices: the highlighted values

```

```

@objc open func drawHighlighted(context: CGContext, indices: [Highlight])

{

    fatalError("drawHighlighted() cannot be called on DataRenderer")

}


/// An opportunity for initializing internal buffers used for rendering with a new size.

/// Since this might do memory allocations, it should only be called if necessary.

@objc open func initBuffers() { }


@objc open func isDrawingValuesAllowed(dataProvider: ChartDataProvider?) -> Bool

{

    guard let data = dataProvider?.data else { return false }

    return data.entryCount < Int(CGFloat(dataProvider?.maxVisibleCount ?? 0) *
viewPortHandler.scaleX)

}


/// Creates an ``NSUIAccessibilityElement`` that acts as the first and primary header
describing a chart view.

```

```

///

/// - Parameters:

/// - chart: The chartView object being described

/// - data: A non optional data source about the chart

/// - defaultDescription: A simple string describing the type/design of Chart.

/// - Returns: A header ``NSUIAccessibilityElement`` that can be added to
accessibleChartElements.

@objc internal func createAccessibleHeader(usingChart chart: ChartViewBase,

                                         andData data: ChartData,

                                         withDefaultDescription defaultDescription: String = "Chart") ->
NSUIAccessibilityElement

{

    let chartDescriptionText = chart.chartDescription?.text ?? defaultDescription

    let dataSetDescriptions = data.dataSets.map { $0.label ?? "" }

    let dataSetDescriptionText = dataSetDescriptions.joined(separator: ", ")

    let dataSetCount = data.dataSets.count

    let

```

```

        element = NSUIAccessibilityElement(accessibilityContainer: chart)

        element.accessibilityLabel = chartDescriptionText + ". \ \(dataSetCount)
dataset\ \(dataSetCount == 1 ? "" : "s"). \ \(dataSetDescriptionText)"

        element.accessibilityFrame = chart.bounds

        element.isHeader = true

        return element
    }

}

//

// ChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

```



```
//

import Foundation

/// Determines how to round DataSet index values for `ChartDataSet.entryIndex(x,
rounding)` when an exact x-value is not found.

@objc

public enum ChartDataSetRounding: Int

{

    case up = 0

    case down = 1

    case closest = 2

}

/// The DataSet class represents one group or type of entries (Entry) in the Chart that belong
together.

/// It is designed to logically separate different groups of values inside the Chart (e.g. the
values for a specific line in the LineChart, or the values of a specific group of bars in the
BarChart).
```

```

open class ChartDataSet: ChartBaseDataSet

{

    public required init()

    {

        entries = []

        super.init()

    }

    public override convenience init(label: String?)

    {

        self.init(entries: nil, label: label)

    }

    @objc public init(entries: [ChartDataEntry]?, label: String?)

    {

        self.entries = entries ?? []
    }

```

```
super.init(label: label)
```

```
self.calcMinMax()
```

```
}
```

```
@objc public convenience init(entries: [ChartDataEntry]?)
```

```
{
```

```
    self.init(entries: entries, label: "DataSet")
```

```
}
```

```
// MARK: - Data functions and accessors
```

```
/// - Note: Calls `notifyDataSetChanged()` after setting a new value.
```

```
/// - Returns: The array of y-values that this DataSet represents.
```

```
/// the entries that this dataset represents / holds together
```

```
@available(*, unavailable, renamed: "entries")
```

@objc

```
open var values: [ChartDataEntry] { return entries }
```

@objc

```
open private(set) var entries: [ChartDataEntry]
```

```
/// Used to replace all entries of a data set while retaining styling properties.
```

```
/// This is a separate method from a setter on `entries` to encourage usage
```

```
/// of `Collection` conformances.
```

```
///
```

```
/// - Parameter entries: new entries to replace existing entries in the dataset
```

@objc

```
public func replaceEntries(_ entries: [ChartDataEntry]) {
```

```
    self.entries = entries
```

```
    notifyDataSetChanged()
```

```
}
```

```

/// maximum y-value in the value array

internal var _yMax: Double = -Double.greatestFiniteMagnitude


/// minimum y-value in the value array

internal var _yMin: Double = Double.greatestFiniteMagnitude


/// maximum x-value in the value array

internal var _xMax: Double = -Double.greatestFiniteMagnitude


/// minimum x-value in the value array

internal var _xMin: Double = Double.greatestFiniteMagnitude


open override func calcMinMax()

{

    _yMax = -Double.greatestFiniteMagnitude

    _yMin = Double.greatestFiniteMagnitude

    _xMax = -Double.greatestFiniteMagnitude

```

```
_xMin = Double.greatestFiniteMagnitude
```

```
guard !isEmpty else { return }
```

```
forEach(calcMinMax)
```

```
}
```

```
open override func calcMinMaxY(fromX: Double, toX: Double)
```

```
{
```

```
_yMax = -Double.greatestFiniteMagnitude
```

```
_yMin = Double.greatestFiniteMagnitude
```

```
guard !isEmpty else { return }
```

```
let indexFrom = entryIndex(x: fromX, closestToY: Double.nan, rounding: .down)
```

```
let indexTo = entryIndex(x: toX, closestToY: Double.nan, rounding: .up)
```

```

guard !(indexTo < indexFrom) else { return }

// only recalculate y

self[indexFrom...indexTo].forEach(calcMinMaxY)

}

@objc open func calcMinMaxX(entry e: ChartDataEntry)

{

    if e.x < _xMin

    {

        _xMin = e.x

    }

    if e.x > _xMax

    {

        _xMax = e.x

    }

}

```

```
@objc open func calcMinMaxY(entry e: ChartDataEntry)
```

```
{
```

```
    if e.y < _yMin
```

```
    {
```

```
        _yMin = e.y
```

```
    }
```

```
    if e.y > _yMax
```

```
    {
```

```
        _yMax = e.y
```

```
    }
```

```
}
```

```
/// Updates the min and max x and y value of this DataSet based on the given Entry.
```

```
///
```

```
/// - Parameters:
```

```
///   - e:
```

```
internal func calcMinMax(entry e: ChartDataEntry)
```



```

{

    calcMinMaxX(entry: e)

    calcMinMaxY(entry: e)

}


/// The minimum y-value this DataSet holds

open override var yMin: Double { return _yMin }


/// The maximum y-value this DataSet holds

open override var yMax: Double { return _yMax }


/// The minimum x-value this DataSet holds

open override var xMin: Double { return _xMin }


/// The maximum x-value this DataSet holds

open override var xMax: Double { return _xMax }

```

```

/// The number of y-values this DataSet represents

@available(*, deprecated, message: "Use `count` instead")

open override var entryCount: Int { return count }


/// - Throws: out of bounds

/// if `i` is out of bounds, it may throw an out-of-bounds exception

/// - Returns: The entry object found at the given index (not x-value!)

@available(*, deprecated, message: "Use `subscript(index)` instead.")

open override func entryForIndex(_ i: Int) -> ChartDataEntry?

{

    guard i >= startIndex, i < endIndex else {

        return nil

    }

    return self[i]

}


/// - Parameters:

```

```

/// - xValue: the x-value

/// - closestToY: If there are multiple y-values for the specified x-value,

/// - rounding: determine whether to round up/down/closest if there is no Entry matching
the provided x-value

/// - Returns: The first Entry object found at the given x-value with binary search.

/// If the no Entry at the specified x-value is found, this method returns the Entry at the
closest x-value according to the rounding.

/// nil if no Entry object at that x-value.

open override func entryForXValue(

    _ xValue: Double,

    closestToY yValue: Double,

    rounding: ChartDataSetRounding) -> ChartDataEntry?

{

    let index = entryIndex(x: xValue, closestToY: yValue, rounding: rounding)

    if index > -1

    {

        return self[index]

    }

```

```

        return nil
    }

    /// - Parameters:

    /// - xValue: the x-value

    /// - closestToY: If there are multiple y-values for the specified x-value,

    /// - Returns: The first Entry object found at the given x-value with binary search.

    /// If the no Entry at the specified x-value is found, this method returns the Entry at the
    closest x-value.

    /// nil if no Entry object at that x-value.

    open override func entryForXValue(
        _ xValue: Double,
        closestToY yValue: Double) -> ChartDataEntry?
    {
        return entryForXValue(xValue, closestToY: yValue, rounding: .closest)
    }

    /// - Returns: All Entry objects found at the given xIndex with binary search.

```

```

/// An empty array if no Entry object at that index.

open override fun entriesForXValue(_ xValue: Double) -> [ChartDataEntry]

{

    var entries = [ChartDataEntry]()

    var low = startIndex

    var high = endIndex - 1

    while low <= high

    {

        var m = (high + low) / 2

        var entry = self[m]

        // if we have a match

        if xValue == entry.x

        {

            while m > 0 && self[m - 1].x == xValue

```

```

{

    m -= 1

}

high = endIndex

// loop over all "equal" entries

while m < high

{

    entry = self[m]

    if entry.x == xValue

    {

        entries.append(entry)

    }

    else

    {

        break
    }
}

```

```
}
```

```
    m += 1
```

```
}
```

```
break
```

```
}
```

```
else
```

```
{
```

```
    if xValue > entry.x
```

```
    {
```

```
        low = m + 1
```

```
    }
```

```
else
```

```
{
```

```
    high = m - 1
```

```
}
```

```

    }

}

return entries

}

/// - Parameters:

/// - xValue: x-value of the entry to search for

/// - closestToY: If there are multiple y-values for the specified x-value,

/// - rounding: Rounding method if exact value was not found

/// - Returns: The array-index of the specified entry.

/// If the no Entry at the specified x-value is found, this method returns the index of the
Entry at the closest x-value according to the rounding.

open override func entryIndex(

    x xValue: Double,

    closestToY yValue: Double,

    rounding: ChartDataSetRounding) -> Int

{

```



```

var low = startIndex

var high = endIndex - 1

var closest = high


while low < high

{

    let m = (low + high) / 2


    let d1 = self[m].x - xValue

    let d2 = self[m + 1].x - xValue

    let ad1 = abs(d1), ad2 = abs(d2)


    if ad2 < ad1

    {

        // [m + 1] is closer to xValue


        // Search in an higher place


        low = m + 1

```

```

}

else if ad1 < ad2

{

    // [m] is closer to xValue

    // Search in a lower place

    high = m

}

else

{

    // We have multiple sequential x-value with same distance

    if d1 >= 0.0

    {

        // Search in a lower place

        high = m

    }

    else if d1 < 0.0

```

```

    {

        // Search in an higher place

        low = m + 1

    }

}

closest = high

}

if closest != -1

{

    let closestXValue = self[closest].x

    if rounding == .up

    {

        // If rounding up, and found x-value is lower than specified x, and we can go
upper...

        if closestXValue < xValue && closest < endIndex - 1

```

```

        {

            closest += 1

        }

    }

    else if rounding == .down

    {

        // If rounding down, and found x-value is upper than specified x, and we can go
lower...

        if closestXValue > xValue && closest > 0

        {

            closest -= 1

        }

    }


    // Search by closest to y-value

    if !yValue.isNaN

    {

        while closest > 0 && self[closest - 1].x == closestXValue

```

```

{

    closest -= 1

}

var closestYValue = self[closest].y

var closestYIndex = closest

while true

{

    closest += 1

    if closest >= endIndex { break }

    let value = self[closest]

    if value.x != closestXValue { break }

    if abs(value.y - yValue) <= abs(closestYValue - yValue)

    {

```

```

        closestYValue = yValue

        closestYIndex = closest

    }

}

closest = closestYIndex

}

}

return closest

}

/// - Parameters:

/// - e: the entry to search for

/// - Returns: The array-index of the specified entry

@available(*, deprecated, message: "Use `firstIndex(of:)` or `lastIndex(of:)`")

open override func entryIndex(entry e: ChartDataEntry) -> Int

```

```

{

    return firstIndex(of: e) ?? -1

}

```

```

/// Adds an Entry to the DataSet dynamically.

```

```

/// Entries are added to the end of the list.

```

```

/// This will also recalculate the current minimum and maximum values of the DataSet
and the value-sum.

```

```

///

```

```

/// - Parameters:

```

```

/// - e: the entry to add

```

```

/// - Returns: True

```

```

@available(*, deprecated, message: "Use `append(_:)` instead")

```

```

open override func addEntry(_ e: ChartDataEntry) -> Bool

```

```

{

    append(e)

    return true

}

```

```
/// Adds an Entry to the DataSet dynamically.
```

```
/// Entries are added to their appropriate index respective to it's x-index.
```

```
/// This will also recalculate the current minimum and maximum values of the DataSet  
and the value-sum.
```

```
///
```

```
/// - Parameters:
```

```
/// - e: the entry to add
```

```
/// - Returns: True
```

```
open override func addEntryOrdered(_ e: ChartDataEntry) -> Bool
```

```
{
```

```
    calcMinMax(entry: e)
```

```
    if let last = last, last.x > e.x
```

```
    {
```

```
        var closestIndex = entryIndex(x: e.x, closestToY: e.y, rounding: .up)
```

```
        while self[closestIndex].x < e.x
```

```
        {
```



```

        closestIndex += 1

    }

    entries.insert(e, at: closestIndex)

}

else

{

    append(e)

}

return true

}

@available(*, renamed: "remove(_:)")

open override func removeEntry(_ entry: ChartDataEntry) -> Bool

{

    return remove(entry)

}

```

```
/// Removes an Entry from the DataSet dynamically.
```

```
/// This will also recalculate the current minimum and maximum values of the DataSet  
and the value-sum.
```

```
///
```

```
/// - Parameters:
```

```
/// - entry: the entry to remove
```

```
/// - Returns: `true` if the entry was removed successfully, else if the entry does not exist
```

```
open func remove(_ entry: ChartDataEntry) -> Bool
```

```
{
```

```
    guard let index = firstIndex(of: entry) else { return false }
```

```
    _ = remove(at: index)
```

```
    return true
```

```
}
```

```
/// Removes the first Entry (at index 0) of this DataSet from the entries array.
```

```
///
```

```
/// - Returns: `true` if successful, `false` if not.
```

```
@available(*, deprecated, message: "Use `func removeFirst() -> ChartDataEntry`  
instead.")
```

```
open override fun removeFirst() -> Bool
```

```
{
```

```
    let entry: ChartDataEntry? = isEmpty ? nil : removeFirst()
```

```
    return entry != nil
```

```
}
```

```
/// Removes the last Entry (at index size-1) of this DataSet from the entries array.
```

```
///
```

```
/// - Returns: `true` if successful, `false` if not.
```

```
@available(*, deprecated, message: "Use `func removeLast() -> ChartDataEntry`  
instead.")
```

```
open override fun removeLast() -> Bool
```

```
{
```

```
    let entry: ChartDataEntry? = isEmpty ? nil : removeLast()
```

```
    return entry != nil
```

```
}
```

```
/// Removes all values from this DataSet and recalculates min and max value.
```

```
@available(*, deprecated, message: "Use `removeAll(keepingCapacity:)` instead.")
```

```
open override func clear()
```

```
{  
  
    removeAll(keepingCapacity: true)  
  
}
```

```
// MARK: - Data functions and accessors
```

```
// MARK: - NSCopying
```

```
open override func copy(with zone: NSZone? = nil) -> Any
```

```
{  
  
    let copy = super.copy(with: zone) as! ChartDataSet
```

```
    copy.entries = entries
```

```

        copy._yMax = _yMax

        copy._yMin = _yMin

        copy._xMax = _xMax

        copy._xMin = _xMin

    }

    return copy
}

// MARK: MutableCollection

extension ChartDataSet: MutableCollection {

    public typealias Index = Int

    public typealias Element = ChartDataEntry

    public var startIndex: Index {

        return entries.startIndex

    }
}

```

```
public var endIndex: Index {  
  
    return entries.endIndex  
  
}
```

```
public func index(after: Index) -> Index {  
  
    return entries.index(after: after)  
  
}
```

```
@objc
```

```
public subscript(position: Index) -> Element {  
  
    get {  
  
        // This is intentionally not a safe subscript to mirror  
  
        // the behaviour of the built in Swift Collection Types  
  
        return entries[position]  
  
    }  
  
    set {
```

```

        calcMinMax(entry: newValue)

        entries[position] = newValue
    }

}

}

// MARK: RandomAccessCollection

extension ChartDataSet: RandomAccessCollection {

    public func index(before: Index) -> Index {

        return entries.index(before: before)

    }

}

```

```

// MARK: RangeReplaceableCollection

extension ChartDataSet: RangeReplaceableCollection {

    public func append(_ newElement: Element) {

        calcMinMax(entry: newElement)
    }
}

```

```

        entries.append(newElement)

    }

    public func remove(at position: Index) -> Element {

        let element = entries.remove(at: position)

        notifyDataSetChanged()

        return element

    }

    public func removeFirst() -> Element {

        let element = entries.removeFirst()

        notifyDataSetChanged()

        return element

    }

    public func removeFirst(_ n: Int) {

        entries.removeFirst(n)

```



```

        notifyDataSetChanged()
    }

```

```

public func removeLast() -> Element {

    let element = entries.removeLast()

    notifyDataSetChanged()

    return element

}

```

```

public func removeLast(_ n: Int) {

    entries.removeLast(n)

    notifyDataSetChanged()

}

```

```

public func removeSubrange<R>(_ bounds: R) where R : RangeExpression, Index ==
R.Bound {

    entries.removeSubrange(bounds)

    notifyDataSetChanged()
}

```

```

    }

    @objc

    public func removeAll(keepingCapacity keepCapacity: Bool) {

        entries.removeAll(keepingCapacity: keepCapacity)

        notifyDataSetChanged()

    }

}

//

// ChartHighlighter.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

```

```
//

import Foundation

import CoreGraphics

open class ChartHighlighter : NSObject, IHighlighter

{

    /// instance of the data-provider

    @objc open weak var chart: ChartDataProvider?

    @objc public init(chart: ChartDataProvider)

    {

        self.chart = chart

    }

    open func getHighlight(x: CGFloat, y: CGFloat) -> Highlight?

    {
```

```

        let xVal = Double(getValsForTouch(x: x, y: y).x)

        return getHighlight(xValue: xVal, x: x, y: y)

    }

    /// - Parameters:

    ///   - x:

    /// - Returns: The corresponding x-pos for a given touch-position in pixels.

    @objc open func getValsForTouch(x: CGFloat, y: CGFloat) -> CGPoint

    {

        guard let chart = self.chart as? BarLineScatterCandleBubbleChartDataProvider else
        { return .zero }

        // take any transformer to determine the values

        return chart.getTransformer(forAxis: .left).valueForTouchPoint(x: x, y: y)

    }

    /// - Parameters:

    ///   - xValue:

```

```
/// - x:
```

```
/// - y:
```

/// - Returns: The corresponding ChartHighlight for a given x-value and xy-touch position in pixels.

```
@objc open func getHighlight(xValue xVal: Double, x: CGFloat, y: CGFloat) -> Highlight?
```

```
{
```

```
    guard let chart = chart else { return nil }
```

```
    let closestValues = getHighlights(xValue: xVal, x: x, y: y)
```

```
    guard !closestValues.isEmpty else { return nil }
```

```
    let leftAxisMinDist = getMinimumDistance(closestValues: closestValues, y: y, axis: .left)
```

```
    let rightAxisMinDist = getMinimumDistance(closestValues: closestValues, y: y, axis: .right)
```

```
    let axis: YAxis.AxisDependency = leftAxisMinDist < rightAxisMinDist ? .left : .right
```

```
        let detail = closestSelectionDetailByPixel(closestValues: closestValues, x: x, y: y,
axis: axis, minSelectionDistance: chart.maxHighlightDistance)
```

```
        return detail
```

```
    }
```

```
/// - Parameters:
```

```
/// - xValue: the transformed x-value of the x-touch position
```

```
/// - x: touch position
```

```
/// - y: touch position
```

```
/// - Returns: A list of Highlight objects representing the entries closest to the given xVal.
```

```
/// The returned list contains two objects per DataSet (closest rounding up, closest
rounding down).
```

```
@objc open func getHighlights(xValue: Double, x: CGFloat, y: CGFloat) -> [Highlight]
```

```
{
```

```
    var vals = [Highlight]()
```

```
    guard let data = self.data else { return vals }
```

```

for i in 0 ..< data.dataSetCount

{

    guard

        let dataSet = data.getDataSetByIndex(i),

        dataSet.isHighlightEnabled    // don't include datasets that cannot be highlighted

    else { continue }


    // extract all y-values from all DataSets at the given x-value.


    // some datasets (i.e bubble charts) make sense to have multiple values for an x-
    value. We'll have to find a way to handle that later on. It's more complicated now when x-
    indices are floating point.


    vals.append(contentsOf: buildHighlights(dataSet: dataSet, dataSetIndex: i, xValue:
    xValue, rounding: .closest))

}


return vals

}

```

/// - Returns: An array of `Highlight` objects corresponding to the selected xValue and dataSetIndex.

```
internal fun buildHighlights(

    dataSet set: IChartDataSet,

    dataSetIndex: Int,

    xValue: Double,

    rounding: ChartDataSetRounding) -> [Highlight]

{

    guard let chart = self.chart as? BarLineScatterCandleBubbleChartDataProvider else
    { return [] }

    var entries = set.entriesForXValue(xValue)

    if entries.count == 0, let closest = set.entryForXValue(xValue, closestToY: .nan,
    rounding: rounding)

    {

        // Try to find closest x-value and take all entries for that x-value

        entries = set.entriesForXValue(closest.x)

    }

}
```



```

return entries.map { e in

    let px = chart.getTransformer(forAxis: set.axisDependency)

    .pixelForValues(x: e.x, y: e.y)

    return Highlight(x: e.x, y: e.y, xPx: px.x, yPx: px.y, dataSetIndex: dataSetIndex,
axis: set.axisDependency)

}

}

```

// - MARK: - Utilities

/// - Returns: The `ChartHighlight` of the closest value on the x-y cartesian axes

```
internal func closestSelectionDetailByPixel(
```

```
    closestValues: [Highlight],
```

```
    x: CGFloat,
```

```
    y: CGFloat,
```

```
    axis: YAxis.AxisDependency?,
```

```
minSelectionDistance: CGFloat) -> Highlight?
```

```
{
```

```
    var distance = minSelectionDistance
```

```
    var closest: Highlight?
```

```
    for high in closestValues
```

```
    {
```

```
        if axis == nil || high.axis == axis
```

```
        {
```

```
            let cDistance = getDistance(x1: x, y1: y, x2: high.xPx, y2: high.yPx)
```

```
            if cDistance < distance
```

```
            {
```

```
                closest = high
```

```
                distance = cDistance
```

```
            }
```

```
    }
```

```
}
```

```
return closest
```

```
}
```

/// - Returns: The minimum distance from a touch-y-value (in pixels) to the closest y-value (in pixels) that is displayed in the chart.

```
internal func getMinimumDistance(
```

```
    closestValues: [Highlight],
```

```
    y: CGFloat,
```

```
    axis: YAxis.AxisDependency
```

```
) -> CGFloat {
```

```
    var distance = CGFloat.greatestFiniteMagnitude
```

```
    for high in closestValues where high.axis == axis
```

```
    {
```

```
        let tempDistance = abs(getHighlightPos(high: high) - y)
```

```
        if tempDistance < distance
```

```

    {

        distance = tempDistance

    }

}

```

```

return distance

}

```

```

internal func getHighlightPos(high: Highlight) -> CGFloat

```

```

{

    return high.yPx

}

```

```

internal func getDistance(x1: CGFloat, y1: CGFloat, x2: CGFloat, y2: CGFloat) ->
CGFloat

```

```

{

    return hypot(x1 - x2, y1 - y2)

}

```

```
internal var data: ChartData?

{

    return chart?.data

}

}

//

// ChartLimitLine.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//
```

```
import Foundation
```

```
import CoreGraphics
```

```
/// The limit line is an additional feature for all Line, Bar and ScatterCharts.
```

```
/// It allows the displaying of an additional line in the chart that marks a certain maximum  
/ limit on the specified axis (x- or y-axis).
```

```
open class ChartLimitLine: ComponentBase
```

```
{
```

```
    @objc(ChartLimitLabelPosition)
```

```
    public enum LabelPosition: Int
```

```
{
```

```
    case topLeft
```

```
    case topRight
```

```
    case bottomLeft
```

```
    case bottomRight
```

```
}
```

```
/// limit / maximum (the y-value or xIndex)
```

```
@objc open var limit = Double(0.0)
```

```
private var _lineWidth = CGFloat(2.0)
```

```
@objc open var lineColor = NSUIColor(red: 237.0/255.0, green: 91.0/255.0, blue:  
91.0/255.0, alpha: 1.0)
```

```
@objc open var lineDashPhase = CGFloat(0.0)
```

```
@objc open var lineDashLengths: [CGFloat]?
```

```
@objc open var valueTextColor = NSUIColor.labelOrBlack
```

```
@objc open var valueFont = NSUIFont.systemFont(ofSize: 13.0)
```

```
@objc open var drawLabelEnabled = true
```

```
@objc open var label = ""
```

```
@objc open var labelPosition = LabelPosition.topRight
```

```
public override init()
```

```
{
```

```
    super.init()

}
```

```
@objc public init(limit: Double)

{

    super.init()

    self.limit = limit

}
```

```
@objc public init(limit: Double, label: String)

{

    super.init()

    self.limit = limit

    self.label = label

}
```

```
/// set the line width of the chart (min = 0.2, max = 12); default 2
```



```

@objc open var lineWidth: CGFloat

{

    get

    {

        return _lineWidth

    }

    set

    {

        _lineWidth = newValue.clamped(to: 0.2...12)

    }

}

//

// Utils.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

```

```
// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//
```

```
import Foundation
```

```
import CoreGraphics
```

```
#if canImport(Uikit)
```

```
    import UIKit
```

```
#endif
```

```
#if canImport(Cocoa)
```

```
    import Cocoa
```

```
#endif
```

```

extension Comparable

{

    func clamped(to range: ClosedRange<Self>) -> Self

    {

        if self > range.upperBound

        {

            return range.upperBound

        }

        else if self < range.lowerBound

        {

            return range.lowerBound

        }

        else

        {

            return self

        }

    }

}

```

```
}
```

```
extension FloatingPoint
```

```
{
```

```
  var DEG2RAD: Self
```

```
  {
```

```
    return self * .pi / 180
```

```
  }
```

```
  var RAD2DEG: Self
```

```
  {
```

```
    return self * 180 / .pi
```

```
  }
```

```
/// - Note: Value must be in degrees
```

```
/// - Returns: An angle between  $0.0 < 360.0$  (not less than zero, less than 360)
```

```
var normalizedAngle: Self
```

```

{

    let angle = truncatingRemainder(dividingBy: 360)

    return (sign == .minus) ? angle + 360 : angle

}

}

```

extension CGSize

```

{

    func rotatedBy(degrees: CGFloat) -> CGSize

    {

        let radians = degrees.DEG2RAD

        return rotatedBy(radians: radians)

    }

}

```

```

func rotatedBy(radians: CGFloat) -> CGSize

{

    return CGSize(

```

```

        width: abs(width * cos(radians)) + abs(height * sin(radians)),

        height: abs(width * sin(radians)) + abs(height * cos(radians))

    )

}

}

```

```

extension Double

```

```

{

```

```

    /// Rounds the number to the nearest multiple of it's order of magnitude, rounding away
    from zero if halfway.

```

```

    func roundedToNextSignificant() -> Double

```

```

    {

```

```

        guard

```

```

            !isInfinite,

```

```

            !isNaN,

```

```

            self != 0

```

```

        else { return self }
    }

```

```

let d = ceil(log10(self < 0 ? -self : self))

let pw = 1 - Int(d)

let magnitude = pow(10.0, Double(pw))

let shifted = (self * magnitude).rounded()

return shifted / magnitude

}

```

```

var decimalPlaces: Int

```

```

{

guard

    !isNaN,

    !isInfinite,

    self != 0.0

else { return 0 }


```

```

let i = self.roundedToNextSignificant()

```

```

guard

    !i.isInfinite,

    !i.isNaN

else { return 0 }

return Int(ceil(-log10(i))) + 2

}

}

extension CGPoint

{

    /// Calculates the position around a center point, depending on the distance from the
    center, and the angle of the position around the center.

    func moving(distance: CGFloat, atAngle angle: CGFloat) -> CGPoint

    {

        return CGPoint(x: x + distance * cos(angle.DEG2RAD),

            y: y + distance * sin(angle.DEG2RAD))

    }

```



```
}
```

```
open class ChartUtils
```

```
{
```

```
    private static var _defaultValueFormatter: IValueFormatter =  
    ChartUtils.generateDefaultValueFormatter()
```

```
open class func drawImage(  
    context: CGContext,  
    image: NSUIImage,  
    x: CGFloat,  
    y: CGFloat,  
    size: CGSize)  
  
{  
    var drawOffset = CGPoint()  
  
    drawOffset.x = x - (size.width / 2)  
  
    drawOffset.y = y - (size.height / 2)
```

```
NSUIGraphicsPushContext(context)
```

```
if image.size.width != size.width && image.size.height != size.height
```

```
{
```

```
    let key = "resized_\(size.width)_\(size.height)"
```

```
    // Try to take scaled image from cache of this image
```

```
    var scaledImage = objc_getAssociatedObject(image, key) as? NSUIImage
```

```
    if scaledImage == nil
```

```
{
```

```
    // Scale the image
```

```
    NSUIGraphicsBeginImageContextWithOptions(size, false, 0.0)
```

```
    image.draw(in: CGRect(origin: CGPoint(x: 0, y: 0), size: size))
```

```
    scaledImage = NSUIGraphicsGetImageFromCurrentImageContext()
```

```
    NSUIGraphicsEndImageContext()
```

```

        // Put the scaled image in a cache owned by the original image

        objc_setAssociatedObject(image, key,
scaledImage, .OBJC_ASSOCIATION_RETAIN_NONATOMIC)

    }

    scaledImage?.draw(in: CGRect(origin: drawOffset, size: size))

}

else

{

    image.draw(in: CGRect(origin: drawOffset, size: size))

}

NSUIGraphicsPopContext()

}

```

```

open class func drawText(context: CGContext, text: String, point: CGPoint, align:
NSTextAlignment, attributes: [NSAttributedString.Key : Any]?)

```

```

{

    var point = point

    if align == .center

    {

        point.x -= text.size(withAttributes: attributes).width / 2.0

    }

    else if align == .right

    {

        point.x -= text.size(withAttributes: attributes).width

    }

    NSUIGraphicsPushContext(context)

    (text as NSString).draw(at: point, withAttributes: attributes)

    NSUIGraphicsPopContext()

```

```
}
```

```
open class func drawText(context: CGContext, text: String, point: CGPoint, attributes:  
[NSAttributedString.Key : Any]?, anchor: CGPoint, angleRadians: CGFloat)
```

```
{
```

```
    var drawOffset = CGPoint()
```

```
    NSUIGraphicsPushContext(context)
```

```
    if angleRadians != 0.0
```

```
    {
```

```
        let size = text.size(withAttributes: attributes)
```

```
        // Move the text drawing rect in a way that it always rotates around its center
```

```
        drawOffset.x = -size.width * 0.5
```

```
        drawOffset.y = -size.height * 0.5
```

```
        var translate = point
```

```

// Move the "outer" rect relative to the anchor, assuming its centered

if anchor.x != 0.5 || anchor.y != 0.5

{

    let rotatedSize = size.rotatedBy(radians: angleRadians)

    translate.x -= rotatedSize.width * (anchor.x - 0.5)

    translate.y -= rotatedSize.height * (anchor.y - 0.5)

}


context.saveGState()

context.translateBy(x: translate.x, y: translate.y)

context.rotate(by: angleRadians)


(text as NSString).draw(at: drawOffset, withAttributes: attributes)


context.restoreGState()

```

```

    }

    else

    {

        if anchor.x != 0.0 || anchor.y != 0.0

        {

            let size = text.size(withAttributes: attributes)

            drawOffset.x = -size.width * anchor.x

            drawOffset.y = -size.height * anchor.y

        }

        drawOffset.x += point.x

        drawOffset.y += point.y

        (text as NSString).draw(at: drawOffset, withAttributes: attributes)

    }

```

```
    NSUIGraphicsPopContext()
```

```
}
```

```
    internal class func drawMultilineText(context: CGContext, text: String, knownTextSize:
CGSize, point: CGPoint, attributes: [NSAttributedString.Key : Any]?, constrainedToSize:
CGSize, anchor: CGPoint, angleRadians: CGFloat)
```

```
{
```

```
    var rect = CGRect(origin: CGPoint(), size: knownTextSize)
```

```
    NSUIGraphicsPushContext(context)
```

```
    if angleRadians != 0.0
```

```
{
```

```
    // Move the text drawing rect in a way that it always rotates around its center
```

```
    rect.origin.x = -knownTextSize.width * 0.5
```

```
    rect.origin.y = -knownTextSize.height * 0.5
```

```
    var translate = point
```



```

// Move the "outer" rect relative to the anchor, assuming its centered

if anchor.x != 0.5 || anchor.y != 0.5

{

    let rotatedSize = knownTextSize.rotatedBy(radians: angleRadians)

    translate.x -= rotatedSize.width * (anchor.x - 0.5)

    translate.y -= rotatedSize.height * (anchor.y - 0.5)

}


context.saveGState()

context.translateBy(x: translate.x, y: translate.y)

context.rotate(by: angleRadians)


(text as NSString).draw(with: rect, options: .usesLineFragmentOrigin, attributes:
attributes, context: nil)


context.restoreGState()

```

```

    }

    else

    {

        if anchor.x != 0.0 || anchor.y != 0.0

        {

            rect.origin.x = -knownTextSize.width * anchor.x

            rect.origin.y = -knownTextSize.height * anchor.y

        }

        rect.origin.x += point.x

        rect.origin.y += point.y

        (text as NSString).draw(with: rect, options: .usesLineFragmentOrigin, attributes:
attributes, context: nil)

    }

    NSUIGraphicsPopContext()

}

```

```

        internal class func drawMultilineText(context: CGContext, text: String, point: CGPoint,
attributes: [NSAttributedString.Key : Any]?, constrainedToSize: CGSize, anchor:
CGPoint, angleRadians: CGFloat)

    {

        let rect = text.boundingRect(with: constrainedToSize,
options: .usesLineFragmentOrigin, attributes: attributes, context: nil)

        drawMultilineText(context: context, text: text, knownTextSize: rect.size, point: point,
attributes: attributes, constrainedToSize: constrainedToSize, anchor: anchor, angleRadians:
angleRadians)

    }

```

```

private class func generateDefaultValueFormatter() -> IValueFormatter

{

    let formatter = DefaultValueFormatter(decimals: 1)

    return formatter

}

```

/// - Returns: The default value formatter used for all chart components that needs a default

```
open class func defaultValueFormatter() -> IValueFormatter

{

    return _defaultValueFormatter

}

}

//

// ChartViewBase.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

// Based on https://github.com/PhilJay/MPAndroidChart/commit/c42b880
```

```
import Foundation
```

```
import CoreGraphics
```

```
#if canImport(Uikit)
```

```
    import UIKit
```

```
#endif
```

```
#if canImport(Cocoa)
```

```
import Cocoa
```

```
#endif
```

```
@objc
```

```
public protocol ChartViewDelegate
```

```
{
```

```
    /// Called when a value has been selected inside the chart.
```

```
    ///
```

```
    /// - Parameters:
```

/// - entry: The selected Entry.

/// - highlight: The corresponding highlight object that contains information about the highlighted position such as dataSetIndex etc.

@objc optional func chartValueSelected(_ chartView: ChartViewBase, entry: ChartDataEntry, highlight: Highlight)

/// Called when a user stops panning between values on the chart

@objc optional func chartViewDidEndPanning(_ chartView: ChartViewBase)

// Called when nothing has been selected or an "un-select" has been made.

@objc optional func chartValueNothingSelected(_ chartView: ChartViewBase)

// Callbacks when the chart is scaled / zoomed via pinch zoom gesture.

@objc optional func chartScaled(_ chartView: ChartViewBase, scaleX: CGFloat, scaleY: CGFloat)

// Callbacks when the chart is moved / translated via drag gesture.

@objc optional func chartTranslated(_ chartView: ChartViewBase, dX: CGFloat, dY: CGFloat)

```

// Callbacks when Animator stops animating

@objc optional func chartView(_ chartView: ChartViewBase, animatorDidStop
animator: Animator)

}

open class ChartViewBase: NSUIView, ChartDataProvider, AnimatorDelegate

{

// MARK: - Properties

/// - Returns: The object representing all x-labels, this method can be used to
/// acquire the XAxis object and modify it (e.g. change the position of the
/// labels)

@objc open var xAxis: XAxis

{

return _xAxis

}

```

/// The default IValueFormatter that has been determined by the chart considering the provided minimum and maximum values.

```
internal var _defaultValueFormatter: IValueFormatter? =
DefaultValueFormatter(decimals: 0)
```

/// object that holds all data that was originally set for the chart, before it was modified or any filtering algorithms had been applied

```
internal var _data: ChartData?
```

/// Flag that indicates if highlighting per tap (touch) is enabled

```
private var _highlightPerTapEnabled = true
```

/// If set to true, chart continues to scroll after touch up

```
@objc open var dragDecelerationEnabled = true
```

/// Deceleration friction coefficient in [0 ; 1] interval, higher values indicate that speed will decrease slowly, for example if it set to 0, it will stop immediately.

/// 1 is an invalid value, and will be converted to 0.999 automatically.

```
private var _dragDecelerationFrictionCoef: CGFloat = 0.9
```



```
/// if true, units are drawn next to the values in the chart
```

```
internal var _drawUnitInChart = false
```

```
/// The object representing the labels on the x-axis
```

```
internal var _xAxis: XAxis!
```

```
/// The `Description` object of the chart.
```

```
/// This should have been called just "description", but
```

```
@objc open var chartDescription: Description?
```

```
/// The legend object containing all data associated with the legend
```

```
internal var _legend: Legend!
```

```
/// delegate to receive chart events
```

```
@objc open weak var delegate: ChartViewDelegate?
```

```

/// text that is displayed when the chart is empty

@objc open var noDataText = "No chart data available."


/// Font to be used for the no data text.

@objc open var noDataFont = NSUIFont.systemFont(ofSize: 12)


/// color of the no data text

@objc open var noDataTextColor: NSUIColor = .labelOrBlack


/// alignment of the no data text

@objc open var noDataTextAlignment: NSTextAlignment = .left


internal var _legendRenderer: LegendRenderer!


/// object responsible for rendering the data

@objc open var renderer: DataRenderer?

```

```
@objc open var highlighter: IHighlighter?
```

```
/// object that manages the bounds and drawing constraints of the chart
```

```
internal var _viewPortHandler: ViewPortHandler!
```

```
/// object responsible for animations
```

```
internal var _animator: Animator!
```

```
/// flag that indicates if offsets calculation has already been done or not
```

```
private var _offsetsCalculated = false
```

```
/// array of Highlight objects that reference the highlighted slices in the chart
```

```
internal var _indicesToHighlight = [Highlight]()
```

```
/// `true` if drawing the marker is enabled when tapping on values
```

```
/// (use the `marker` property to specify a marker)
```

```
@objc open var drawMarkers = true
```

/// - Returns: `true` if drawing the marker is enabled when tapping on values

/// (use the `marker` property to specify a marker)

@objc open var isDrawMarkersEnabled: Bool { return drawMarkers }

/// The marker that is displayed when a value is clicked on the chart

@objc open var marker: IMarker?

private var _interceptTouchEvents = false

/// An extra offset to be appended to the viewport's top

@objc open var extraTopOffset: CGFloat = 0.0

/// An extra offset to be appended to the viewport's right

@objc open var extraRightOffset: CGFloat = 0.0

/// An extra offset to be appended to the viewport's bottom

```
@objc open var extraBottomOffset: CGFloat = 0.0
```

```
/// An extra offset to be appended to the viewport's left
```

```
@objc open var extraLeftOffset: CGFloat = 0.0
```

```
@objc open func setExtraOffsets(left: CGFloat, top: CGFloat, right: CGFloat, bottom:
CGFloat)
```

```
{
```

```
    extraLeftOffset = left
```

```
    extraTopOffset = top
```

```
    extraRightOffset = right
```

```
    extraBottomOffset = bottom
```

```
}
```

```
// MARK: - Initializers
```

```
public override init(frame: CGRect)
```

```
{
```

```
super.init(frame: frame)
```

```
initialize()
```

```
}
```

```
public required init?(coder aDecoder: NSCoder)
```

```
{
```

```
super.init(coder: aDecoder)
```

```
initialize()
```

```
}
```

```
deinit
```

```
{
```

```
self.removeObserver(self, forKeyPath: "bounds")
```

```
self.removeObserver(self, forKeyPath: "frame")
```

```
}
```

```
internal func initialize()
```

```

{

    #if os(iOS)

    self.backgroundColor = NSUIColor.clear

    #endif


    _animator = Animator()

    _animator.delegate = self


    _viewPortHandler = ViewPortHandler(width: bounds.size.width, height:
bounds.size.height)


    chartDescription = Description()


    _legend = Legend()

    _legendRenderer = LegendRenderer(viewPortHandler: _viewPortHandler, legend:
_legend)


    _xAxis = XAxis()

```

```
self.addObserver(self, forKeyPath: "bounds", options: .new, context: nil)

self.addObserver(self, forKeyPath: "frame", options: .new, context: nil)

}
```

```
// MARK: - ChartViewBase
```

```
/// The data for the chart
```

```
open var data: ChartData?
```

```
{

    get

    {

        return _data

    }

    set

    {

        _data = newValue

    }

}
```



```
_offsetsCalculated = false
```

```
guard let _data = _data else
```

```
{
```

```
    setNeedsDisplay()
```

```
    return
```

```
}
```

```
// calculate how many digits are needed
```

```
setupDefaultFormatter(min: _data.getYMin(), max: _data.getYMax())
```

```
for set in _data.dataSets
```

```
{
```

```
    if set.needsFormatter || set.valueFormatter === _defaultValueFormatter
```

```
    {
```

```
        set.valueFormatter = _defaultValueFormatter
```

```
    }
```

```
}
```

```
// let the chart know there is new data
```

```
notifyDataSetChanged()
```

```
}
```

```
}
```

```
/// Clears the chart from all data (sets it to null) and refreshes it (by calling  
setNeedsDisplay()).
```

```
@objc open func clear()
```

```
{
```

```
    _data = nil
```

```
    _offsetsCalculated = false
```

```
    _indicesToHighlight.removeAll()
```

```
    lastHighlighted = nil
```

```
    setNeedsDisplay()
```

```
}
```

/// Removes all DataSets (and thereby Entries) from the chart. Does not set the data object to nil. Also refreshes the chart by calling setNeedsDisplay().

```
@objc open func clearValues()
```

```
{
```

```
    _data?.clearValues()
```

```
    setNeedsDisplay()
```

```
}
```

/// - Returns: `true` if the chart is empty (meaning it's data object is either null or contains no entries).

```
@objc open func isEmpty() -> Bool
```

```
{
```

```
    guard let data = _data else { return true }
```

```
    if data.entryCount <= 0
```

```
    {
```

```
        return true
```

```

    }

    else

    {

        return false

    }

}

```

/// Lets the chart know its underlying data has changed and should perform all necessary recalculations.

/// It is crucial that this method is called everytime data is changed dynamically. Not calling this method can lead to crashes or unexpected behaviour.

```

@objc open func notifyDataSetChanged()

{

    fatalError("notifyDataSetChanged() cannot be called on ChartViewBase")

}

```

/// Calculates the offsets of the chart to the border depending on the position of an eventual legend or depending on the length of the y-axis and x-axis labels and their position

```

internal func calculateOffsets()

```

```

{

    fatalError("calculateOffsets() cannot be called on ChartViewBase")

}

```

/// calculates the y-min and y-max value and the y-delta and x-delta value

```
internal func calcMinMax()
```

```

{

    fatalError("calcMinMax() cannot be called on ChartViewBase")

}

```

/// calculates the required number of digits for the values that might be drawn in the chart (if enabled), and creates the default value formatter

```
internal func setupDefaultFormatter(min: Double, max: Double)
```

```

{

    // check if a custom formatter is set or not

    var reference = Double(0.0)


```

```
    if let data = _data , data.entryCount >= 2
```

```

{

    reference = fabs(max - min)

}

else

{

    let absMin = fabs(min)

    let absMax = fabs(max)

    reference = absMin > absMax ? absMin : absMax

}

```

if `_defaultValueFormatter` is `DefaultValueFormatter`

```

{

    // setup the formatter with a new number of digits

    let digits = reference.decimalPlaces

    (_defaultValueFormatter as? DefaultValueFormatter)?.decimals

```

```

        = digits

    }

}

open override func draw(_ rect: CGRect)

{

    let optionalContext = NSUIGraphicsGetCurrentContext()

    guard let context = optionalContext else { return }

    let frame = self.bounds

    if _data === nil && noDataText.count > 0

    {

        context.saveGState()

        defer { context.restoreGState() }

        let paragraphStyle = NSMutableParagraphStyle.default.mutableCopy() as!
NSMutableParagraphStyle

```

```
paragraphStyle.minimumLineHeight = noDataFont.lineHeight
```

```
paragraphStyle.lineBreakMode = .byWordWrapping
```

```
paragraphStyle.alignment = noDataTextAlignment
```

```
ChartUtils.drawMultilineText(
```

```
    context: context,
```

```
    text: noDataText,
```

```
    point: CGPoint(x: frame.width / 2.0, y: frame.height / 2.0),
```

```
    attributes:
```

```
        [.font: noDataFont,
```

```
        .foregroundColor: noDataTextColor,
```

```
        .paragraphStyle: paragraphStyle],
```

```
    constrainedToSize: self.bounds.size,
```

```
    anchor: CGPoint(x: 0.5, y: 0.5),
```

```
    angleRadians: 0.0)
```

```
return
```



```
}
```

```
if !_offsetsCalculated
```

```
{
```

```
    calculateOffsets()
```

```
    _offsetsCalculated = true
```

```
}
```

```
}
```

```
/// Draws the description text in the bottom right corner of the chart (per default)
```

```
internal fun drawDescription(context: CGContext)
```

```
{
```

```
    // check if description should be drawn
```

```
    guard
```

```
        let description = chartDescription,
```

```
        description.isEnabled,
```

```
        let descriptionText = description.text,
```

```
descriptionText.count > 0
```

```
else { return }
```

```
let position = description.position ?? CGPoint(x: bounds.width -  
_viewPortHandler.offsetRight - description.xOffset,
```

```
y: bounds.height - _viewPortHandler.offsetBottom -  
description.yOffset - description.font.lineHeight)
```

```
var attrs = [NSAttributedString.Key : Any]()
```

```
attrs[NSAttributedString.Key.font] = description.font
```

```
attrs[NSAttributedString.Key.foregroundColor] = description.textColor
```

```
ChartUtils.drawText(  

```

```
context: context,
```

```
text: descriptionText,
```

```
point: position,
```

```
align: description.textAlign,
```

```
        attributes: attrs)
    }
```

```
// MARK: - Accessibility
```

```
open override func accessibilityChildren() -> [Any]? {

    return renderer?.accessibleChartElements

}
```

```
// MARK: - Highlighting
```

/// The array of currently highlighted values. This might be an empty array if nothing is highlighted.

```
@objc open var highlighted: [Highlight]

{

    return _indicesToHighlight

}
```

```
/// Set this to false to prevent values from being highlighted by tap gesture.
```

```
/// Values can still be highlighted via drag or programmatically.
```

```
/// default: true
```

```
@objc open var highlightPerTapEnabled: Bool
```

```
{  
  
    get { return _highlightPerTapEnabled }  
  
    set { _highlightPerTapEnabled = newValue }  
  
}
```

```
/// `true` if values can be highlighted via tap gesture, `false` if not.
```

```
@objc open var isHighLightPerTapEnabled: Bool
```

```
{  
  
    return highlightPerTapEnabled  
  
}
```

```
/// Checks if the highlight array is null, has a length of zero or if the first object is null.
```

```
///
```

/// - Returns: `true` if there are values to highlight, `false` if there are no values to highlight.

```
@objc open func valuesToHighlight() -> Bool
```

```
{
```

```
    return !_indicesToHighlight.isEmpty
```

```
}
```

/// Highlights the values at the given indices in the given DataSets. Provide

/// null or an empty array to undo all highlighting.

/// This should be used to programmatically highlight values.

/// This method *will not* call the delegate.

```
@objc open func highlightValues(_ highs: [Highlight]?)
```

```
{
```

```
    // set the indices to highlight
```

```
    _indicesToHighlight = highs ?? [Highlight]()
```

```
    if !_indicesToHighlight.isEmpty
```

```
    {
```

```

        self.lastHighlighted = nil

    }

    else

    {

        self.lastHighlighted = _indicesToHighlight[0]

    }


    // redraw the chart

    setNeedsDisplay()

}


/// Highlights any y-value at the given x-value in the given DataSet.

/// Provide -1 as the dataSetIndex to undo all highlighting.

/// This method will call the delegate.

///

/// - Parameters:

/// - x: The x-value to highlight

```

```

/// - dataSetIndex: The dataset index to search in

/// - dataIndex: The data index to search in (only used in CombinedChartView currently)

@objc open func highlightValue(x: Double, dataSetIndex: Int, dataIndex: Int = -1)

{

    highlightValue(x: x, dataSetIndex: dataSetIndex, dataIndex: dataIndex, callDelegate:
true)

}

/// Highlights the value at the given x-value and y-value in the given DataSet.

/// Provide -1 as the dataSetIndex to undo all highlighting.

/// This method will call the delegate.

///

/// - Parameters:

/// - x: The x-value to highlight

/// - y: The y-value to highlight. Supply `NaN` for "any"

/// - dataSetIndex: The dataset index to search in

/// - dataIndex: The data index to search in (only used in CombinedChartView currently)

```

```

@objc open func highlightValue(x: Double, y: Double, dataSetIndex: Int, dataIndex: Int
= -1)

{

    highlightValue(x: x, y: y, dataSetIndex: dataSetIndex, dataIndex: dataIndex,
callDelegate: true)

}

/// Highlights any y-value at the given x-value in the given DataSet.

/// Provide -1 as the dataSetIndex to undo all highlighting.

///

/// - Parameters:

/// - x: The x-value to highlight

/// - dataSetIndex: The dataset index to search in

/// - dataIndex: The data index to search in (only used in CombinedChartView currently)

/// - callDelegate: Should the delegate be called for this change

@objc open func highlightValue(x: Double, dataSetIndex: Int, dataIndex: Int = -1,
callDelegate: Bool)

{

```



```
        highlightValue(x: x, y: .nan, dataSetIndex: dataSetIndex, dataIndex: dataIndex,  
callDelegate: callDelegate)
```

```
    }
```

```
/// Highlights the value at the given x-value and y-value in the given DataSet.
```

```
/// Provide -1 as the dataSetIndex to undo all highlighting.
```

```
///
```

```
/// - Parameters:
```

```
/// - x: The x-value to highlight
```

```
/// - y: The y-value to highlight. Supply `NaN` for "any"
```

```
/// - dataSetIndex: The dataset index to search in
```

```
/// - dataIndex: The data index to search in (only used in CombinedChartView currently)
```

```
/// - callDelegate: Should the delegate be called for this change
```

```
@objc open func highlightValue(x: Double, y: Double, dataSetIndex: Int, dataIndex: Int  
= -1, callDelegate: Bool)
```

```
{
```

```
    guard let data = _data else
```

```
    {
```

```

        Swift.print("Value not highlighted because data is nil")

        return

    }

    if dataSetIndex < 0 || dataSetIndex >= data.dataSetCount

    {

        highlightValue(nil, callDelegate: callDelegate)

    }

    else

    {

        highlightValue(Highlight(x: x, y: y, dataSetIndex: dataSetIndex, dataIndex:
dataIndex), callDelegate: callDelegate)

    }

}

/// Highlights the values represented by the provided Highlight object

/// This method *will not* call the delegate.

///

```

```
/// - Parameters:
```

```
/// - highlight: contains information about which entry should be highlighted
```

```
@objc open func highlightValue(_ highlight: Highlight?)
```

```
{
```

```
    highlightValue(highlight, callDelegate: false)
```

```
}
```

```
/// Highlights the value selected by touch gesture.
```

```
@objc open func highlightValue(_ highlight: Highlight?, callDelegate: Bool)
```

```
{
```

```
    var entry: ChartDataEntry?
```

```
    var h = highlight
```

```
    if h == nil
```

```
    {
```

```
        self.lastHighlighted = nil
```

```
        _indicesToHighlight.removeAll(keepingCapacity: false)
```

```

    }

    else

    {

        // set the indices to highlight

        entry = _data?.entryForHighlight(h!)

        if entry == nil

        {

            h = nil

            _indicesToHighlight.removeAll(keepingCapacity: false)

        }

        else

        {

            _indicesToHighlight = [h!]

        }

    }

```

if callDelegate, let delegate = delegate

```

{

    if let h = h

    {

        // notify the listener

        delegate.chartValueSelected?(self, entry: entry!, highlight: h)

    }

    else

    {

        delegate.chartValueNothingSelected?(self)

    }

}

// redraw the chart

setNeedsDisplay()

}

```

/// - Returns: The Highlight object (contains x-index and DataSet index) of the

```

/// selected value at the given touch point inside the Line-, Scatter-, or

/// CandleStick-Chart.

@objc open func getHighlightByTouchPoint(_ pt: CGPoint) -> Highlight?

{

    if _data === nil

    {

        Swift.print("Can't select by touch. No data set.")

        return nil

    }

    return self.highlighter?.getHighlight(x: pt.x, y: pt.y)

}

/// The last value that was highlighted via touch.

@objc open var lastHighlighted: Highlight?

// MARK: - Markers

```

```

/// draws all MarkerViews on the highlighted positions

internal func drawMarkers(context: CGContext)

{

    // if there is no marker view or drawing marker is disabled

    guard

        let marker = marker

        , isDrawMarkersEnabled &&

        valuesToHighlight()

    else { return }

    for i in 0 ..< _indicesToHighlight.count

    {

        let highlight = _indicesToHighlight[i]

        guard let

            set = data?.getDataSetByIndex(highlight.dataSetIndex),

```

```

    let e = _data?.entryForHighlight(highlight)

    else { continue }

let entryIndex = set.entryIndex(entry: e)

if entryIndex > Int(Double(set.entryCount) * _animator.phaseX)

{

    continue

}

let pos = getMarkerPosition(highlight: highlight)

// check bounds

if !_viewPortHandler.isInBounds(x: pos.x, y: pos.y)

{

    continue

}

```



```
// callbacks to update the content
```

```
marker.refreshContent(entry: e, highlight: highlight)
```

```
// draw the marker
```

```
marker.draw(context: context, point: pos)
```

```
}
```

```
}
```

/// - Returns: The actual position in pixels of the MarkerView for the given Entry in the given DataSet.

```
@objc open func getMarkerPosition(highlight: Highlight) -> CGPoint
```

```
{
```

```
    return CGPoint(x: highlight.drawX, y: highlight.drawY)
```

```
}
```

```
// MARK: - Animation
```

```
/// The animator responsible for animating chart values.
```

```

@objc open var chartAnimator: Animator!

{

    return _animator

}

/// Animates the drawing / rendering of the chart on both x- and y-axis with the specified
animation time.

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the
chart.

///

/// - Parameters:

/// - xAxisDuration: duration for animating the x axis

/// - yAxisDuration: duration for animating the y axis

/// - easingX: an easing function for the animation on the x axis

/// - easingY: an easing function for the animation on the y axis

@objc open func animate(xAxisDuration: TimeInterval, yAxisDuration: TimeInterval,
easingX: ChartEasingFunctionBlock?, easingY: ChartEasingFunctionBlock?)

{

```

```
    _animator.animate(xAxisDuration: xAxisDuration, yAxisDuration: yAxisDuration,  
easingX: easingX, easingY: easingY)
```

```
}
```

```
/// Animates the drawing / rendering of the chart on both x- and y-axis with the specified  
animation time.
```

```
/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the  
chart.
```

```
///
```

```
/// - Parameters:
```

```
/// - xAxisDuration: duration for animating the x axis
```

```
/// - yAxisDuration: duration for animating the y axis
```

```
/// - easingOptionX: the easing function for the animation on the x axis
```

```
/// - easingOptionY: the easing function for the animation on the y axis
```

```
@objc open func animate(xAxisDuration: TimeInterval, yAxisDuration: TimeInterval,  
easingOptionX: ChartEasingOption, easingOptionY: ChartEasingOption)
```

```
{
```

```
    _animator.animate(xAxisDuration: xAxisDuration, yAxisDuration: yAxisDuration,  
easingOptionX: easingOptionX, easingOptionY: easingOptionY)
```

```
}
```

/// Animates the drawing / rendering of the chart on both x- and y-axis with the specified animation time.

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the chart.

///

/// - Parameters:

/// - xAxisDuration: duration for animating the x axis

/// - yAxisDuration: duration for animating the y axis

/// - easing: an easing function for the animation

@objc open func animate(xAxisDuration: TimeInterval, yAxisDuration: TimeInterval, easing: ChartEasingFunctionBlock?)

{

 _animator.animate(xAxisDuration: xAxisDuration, yAxisDuration: yAxisDuration, easing: easing)

}

/// Animates the drawing / rendering of the chart on both x- and y-axis with the specified animation time.

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the chart.

///

/// - Parameters:

/// - xAxisDuration: duration for animating the x axis

/// - yAxisDuration: duration for animating the y axis

/// - easingOption: the easing function for the animation

@objc open func animate(xAxisDuration: TimeInterval, yAxisDuration: TimeInterval, easingOption: ChartEasingOption)

{

 _animator.animate(xAxisDuration: xAxisDuration, yAxisDuration: yAxisDuration, easingOption: easingOption)

}

/// Animates the drawing / rendering of the chart on both x- and y-axis with the specified animation time.

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the chart.

///

/// - Parameters:

```

/// - xAxisDuration: duration for animating the x axis

/// - yAxisDuration: duration for animating the y axis

@objc open func animate(xAxisDuration: TimeInterval, yAxisDuration: TimeInterval)

{

    _animator.animate(xAxisDuration: xAxisDuration, yAxisDuration: yAxisDuration)

}

/// Animates the drawing / rendering of the chart the x-axis with the specified animation
time.

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the
chart.

///

/// - Parameters:

/// - xAxisDuration: duration for animating the x axis

/// - easing: an easing function for the animation

@objc open func animate(xAxisDuration: TimeInterval, easing:
ChartEasingFunctionBlock?)

{

    _animator.animate(xAxisDuration: xAxisDuration, easing: easing)

```

```
}
```

```
/// Animates the drawing / rendering of the chart the x-axis with the specified animation  
time.
```

```
/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the  
chart.
```

```
///
```

```
/// - Parameters:
```

```
/// - xAxisDuration: duration for animating the x axis
```

```
/// - easingOption: the easing function for the animation
```

```
@objc open func animate(xAxisDuration: TimeInterval, easingOption:  
ChartEasingOption)
```

```
{
```

```
    _animator.animate(xAxisDuration: xAxisDuration, easingOption: easingOption)
```

```
}
```

```
/// Animates the drawing / rendering of the chart the x-axis with the specified animation  
time.
```

```
/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the  
chart.
```

```
///
```

```
/// - Parameters:
```

```
/// - xAxisDuration: duration for animating the x axis
```

```
@objc open func animate(xAxisDuration: TimeInterval)
```

```
{
```

```
    _animator.animate(xAxisDuration: xAxisDuration)
```

```
}
```

/// Animates the drawing / rendering of the chart the y-axis with the specified animation time.

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the chart.

```
///
```

```
/// - Parameters:
```

```
/// - yAxisDuration: duration for animating the y axis
```

```
/// - easing: an easing function for the animation
```

```
@objc open func animate(yAxisDuration: TimeInterval, easing:
ChartEasingFunctionBlock?)
```

```
{
```



```

    _animator.animate(yAxisDuration: yAxisDuration, easing: easing)

}

/// Animates the drawing / rendering of the chart the y-axis with the specified animation
time.

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the
chart.

///

/// - Parameters:

/// - yAxisDuration: duration for animating the y axis

/// - easingOption: the easing function for the animation

@objc open func animate(yAxisDuration: TimeInterval, easingOption:
ChartEasingOption)

{

    _animator.animate(yAxisDuration: yAxisDuration, easingOption: easingOption)

}

/// Animates the drawing / rendering of the chart the y-axis with the specified animation
time.

```

/// If `animate(...)` is called, no further calling of `invalidate()` is necessary to refresh the chart.

///

/// - Parameters:

/// - yAxisDuration: duration for animating the y axis

@objc open func animate(yAxisDuration: TimeInterval)

{

 animator.animate(yAxisDuration: yAxisDuration)

}

// MARK: - Accessors

/// The current y-max value across all DataSets

open var chartYMax: Double

{

 return _data?.yMax ?? 0.0

}

```
/// The current y-min value across all DataSets
```

```
open var chartYMin: Double
```

```
{  
  
    return _data?.yMin ?? 0.0  
  
}
```

```
open var chartXMax: Double
```

```
{  
  
    return _xAxis._axisMaximum  
  
}
```

```
open var chartXMin: Double
```

```
{  
  
    return _xAxis._axisMinimum  
  
}
```

```
open var xRange: Double
```

```

{

    return _xAxis.axisRange

}

```

/// - Note: (Equivalent of `getCenter()` in `MPAndroidChart`, as `center` is already a standard in iOS that returns the center point relative to superview, and `MPAndroidChart` returns relative to `self`)*

/// The center point of the chart (the whole View) in pixels.

```
@objc open var midPoint: CGPoint
```

```

{

    let bounds = self.bounds

    return CGPoint(x: bounds.origin.x + bounds.size.width / 2.0, y: bounds.origin.y +
bounds.size.height / 2.0)

}

```

/// The center of the chart taking offsets under consideration. (returns the center of the content rectangle)

```
open var centerOffsets: CGPoint
```

```
{
```

```

        return _viewPortHandler.contentCenter
    }

```

/// The Legend object of the chart. This method can be used to get an instance of the legend in order to customize the automatically generated Legend.

```

@objc open var legend: Legend

{

    return _legend
}

```

/// The renderer object responsible for rendering / drawing the Legend.

```

@objc open var legendRenderer: LegendRenderer!

{

    return _legendRenderer
}

```

/// The rectangle that defines the borders of the chart-value surface (into which the actual values are drawn).

```
@objc open var contentRect: CGRect
```

```
{
```

```
    return _viewPortHandler.contentRect
```

```
}
```

```
/// - Returns: The ViewPortHandler of the chart that is responsible for the
```

```
/// content area of the chart and its offsets and dimensions.
```

```
@objc open var viewPortHandler: ViewPortHandler!
```

```
{
```

```
    return _viewPortHandler
```

```
}
```

```
/// - Returns: The bitmap that represents the chart.
```

```
@objc open func getChartImage(transparent: Bool) -> NSUIImage?
```

```
{
```

```
    NSUIGraphicsBeginImageContextWithOptions(bounds.size, isOpaque || !transparent,  
NSUIScreen.nsuiMain?.nsuiScale ?? 1.0)
```

```

guard let context = NSUIGraphicsGetCurrentContext()

    else { return nil }

let rect = CGRect(origin: CGPoint(x: 0, y: 0), size: bounds.size)

if isOpaque || !transparent

{

    // Background color may be partially transparent, we must fill with white if we want
    to output an opaque image

    context.setFillColor(NSUIColor.white.cgColor)

    context.fill(rect)

    if let backgroundColor = self.backgroundColor

    {

        context.setFillColor(backgroundColor.cgColor)

        context.fill(rect)

    }

}

```

```
nsuiLayer?.render(in: context)
```

```
let image = NSUIGraphicsGetImageFromCurrentImageContext()
```

```
NSUIGraphicsEndImageContext()
```

```
return image
```

```
}
```

```
public enum ImageFormat
```

```
{
```

```
    case jpeg
```

```
    case png
```

```
}
```

```
/// Saves the current chart state with the given name to the given path on
```



```

/// the sdcard leaving the path empty "" will put the saved file directly on

/// the SD card chart is saved as a PNG image, example:

/// saveToPath("myfilename", "foldername1/foldername2")

///

/// - Parameters:

/// - to: path to the image to save

/// - format: the format to save

/// - compressionQuality: compression quality for lossless formats (JPEG)

/// - Returns: `true` if the image was saved successfully

open func save(to path: String, format: ImageFormat, compressionQuality: Double) ->
Bool

{

    guard let image = getChartImage(transparent: format != .jpeg) else { return false }

    let imageData: Data?

    switch (format)

    {

    case .png: imageData = NSUIImagePNGRepresentation(image)

```

```

        case .jpeg: imageData = NSUIImageJPEGRepresentation(image,
CGFloat(compressionQuality))

    }

    guard let data = imageData else { return false }

    do

    {

        try data.write(to: URL(fileURLWithPath: path), options: .atomic)

    }

    catch

    {

        return false

    }

    return true

}

```

```

internal var _viewportJobs = [ViewPortJob]()

open override fun observeValue(forKeyPath keyPath: String?, of object: Any?, change:
[NSKeyValueChangeKey : Any]?, context: UnsafeMutableRawPointer?)

{

    if keyPath == "bounds" || keyPath == "frame"

    {

        let bounds = self.bounds

        if (_viewportHandler !== nil &&

            (bounds.size.width != _viewportHandler.chartWidth ||

            bounds.size.height != _viewportHandler.chartHeight))

        {

            _viewportHandler.setChartDimens(width:      bounds.size.width,      height:
bounds.size.height)

            // This may cause the chart view to mutate properties affecting the view port --
            // lets do this

            // before we try to run any pending jobs on the view port itself

```

```
notifyDataSetChanged()
```

```
// Finish any pending viewport changes
```

```
while (!_viewportJobs.isEmpty)
```

```
{
```

```
    let job = _viewportJobs.remove(at: 0)
```

```
    job.doJob()
```

```
}
```

```
}
```

```
}
```

```
}
```

```
@objc open func removeViewportJob(_ job: ViewPortJob)
```

```
{
```

```
    if let index = _viewportJobs.firstIndex(where: { $0 === job })
```

```
    {
```

```
        _viewportJobs.remove(at: index)
```

```
}
```

```
}
```

```
@objc open func clearAllViewportJobs()
```

```
{
```

```
    _viewportJobs.removeAll(keepingCapacity: false)
```

```
}
```

```
@objc open func addViewportJob(_ job: ViewPortJob)
```

```
{
```

```
    if _viewPortHandler.hasChartDimens
```

```
    {
```

```
        job.doJob()
```

```
    }
```

```
    else
```

```
    {
```

```
        _viewportJobs.append(job)
```

```

    }

}

/// default: true

/// `true` if chart continues to scroll after touch up, `false` if not.

@objc open var isDragDecelerationEnabled: Bool

{

    return dragDecelerationEnabled

}

```

/// Deceleration friction coefficient in [0 ; 1] interval, higher values indicate that speed will decrease slowly, for example if it set to 0, it will stop immediately.

/// 1 is an invalid value, and will be converted to 0.999 automatically.

///

/// **default**: true

@objc open var dragDecelerationFrictionCoef: CGFloat

{

get

```

{

    return _dragDecelerationFrictionCoef

}

set

{

    var val = newValue

    if val < 0.0

    {

        val = 0.0

    }

    if val >= 1.0

    {

        val = 0.999

    }

    _dragDecelerationFrictionCoef = val

}

```

```
}
```

```
/// The maximum distance in screen pixels away from an entry causing it to highlight.
```

```
/// default: 500.0
```

```
open var maxHighlightDistance: CGFloat = 500.0
```

```
/// the number of maximum visible drawn values on the chart only active when  
`drawValuesEnabled` is enabled
```

```
open var maxVisibleCount: Int
```

```
{
```

```
    return Int(INT_MAX)
```

```
}
```

```
// MARK: - AnimatorDelegate
```

```
open func animatorUpdated(_ chartAnimator: Animator)
```

```
{
```

```
    setNeedsDisplay()
```



```
}
```

```
open func animatorStopped(_ chartAnimator: Animator)
```

```
{
```

```
    delegate?.chartView?(self, animatorDidStop: chartAnimator)
```

```
}
```

```
// MARK: - Touches
```

```
open override func nsuiTouchesBegan(_ touches: Set<NSUITouch>, withEvent event:  
NSUIEvent?)
```

```
{
```

```
    if !_interceptTouchEvent
```

```
    {
```

```
        super.nsuiTouchesBegan(touches, withEvent: event)
```

```
    }
```

```
}
```

```
open override func nsuiTouchesMoved(_ touches: Set<NSUITouch>, withEvent event:
NSUIEvent?)
```

```
{

    if !_interceptTouchEvents

    {

        super.nsuiTouchesMoved(touches, withEvent: event)

    }

}
```

```
open override func nsuiTouchesEnded(_ touches: Set<NSUITouch>, withEvent event:
NSUIEvent?)
```

```
{

    if !_interceptTouchEvents

    {

        super.nsuiTouchesEnded(touches, withEvent: event)

    }

}
```

```

    open override func nsuiTouchesCancelled(_ touches: Set<NSUITouch>?, withEvent
event: NSUIEvent?)

    {

        if !_interceptTouchEvent

        {

            super.nsuiTouchesCancelled(touches, withEvent: event)

        }

    }

}

//

// ChevronDownShapeRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

```

```
//

import Foundation

import CoreGraphics

open class ChevronDownShapeRenderer : NSObject, IShapeRenderer

{

    open func renderShape(

        context: CGContext,

        dataSet: IScatterChartDataSet,

        viewPortHandler: ViewPortHandler,

        point: CGPoint,

        color: NSUIColor)

    {

        let shapeSize = dataSet.scatterShapeSize

        let shapeHalf = shapeSize / 2.0

        context.setLineWidth(1.0)
```

```

context.setStrokeColor(color.cgColor)

context.beginPath()

context.move(to: CGPoint(x: point.x, y: point.y + 2 * shapeHalf))

context.addLine(to: CGPoint(x: point.x + 2 * shapeHalf, y: point.y))

context.move(to: CGPoint(x: point.x, y: point.y + 2 * shapeHalf))

context.addLine(to: CGPoint(x: point.x - 2 * shapeHalf, y: point.y))

context.strokePath()

}

}

//

// ChevronUpShapeRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

```

```

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class ChevronUpShapeRenderer : NSObject, IShapeRendererer

{

    open func renderShape(

        context: CGContext,

        dataSet: IScatterChartDataSet,

        viewPortHandler: ViewPortHandler,

        point: CGPoint,

        color: NSUIColor)

    {

        let shapeSize = dataSet.scatterShapeSize

        let shapeHalf = shapeSize / 2.0

```

```

context.setLineWidth(1.0)

context.setStrokeColor(color.cgColor)


context.beginPath()

context.move(to: CGPoint(x: point.x, y: point.y - 2 * shapeHalf))

context.addLine(to: CGPoint(x: point.x + 2 * shapeHalf, y: point.y))

context.move(to: CGPoint(x: point.x, y: point.y - 2 * shapeHalf))

context.addLine(to: CGPoint(x: point.x - 2 * shapeHalf, y: point.y))

context.strokePath()

}

}

//

// CircleShapeRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

```

```

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class CircleShapeRenderer : NSObject, IShapeRenderer

{

    open func renderShape(

        context: CGContext,

        dataSet: IScatterChartDataSet,

        viewPortHandler: ViewPortHandler,

        point: CGPoint,

        color: NSUIColor)

    {

```



```

let shapeSize = dataSet.scatterShapeSize

let shapeHalf = shapeSize / 2.0

let shapeHoleSizeHalf = dataSet.scatterShapeHoleRadius

let shapeHoleSize = shapeHoleSizeHalf * 2.0

let shapeHoleColor = dataSet.scatterShapeHoleColor

let shapeStrokeSize = (shapeSize - shapeHoleSize) / 2.0

let shapeStrokeSizeHalf = shapeStrokeSize / 2.0


if shapeHoleSize > 0.0
{

    context.setStrokeColor(color.cgColor)

    context.setLineWidth(shapeStrokeSize)

    var rect = CGRect()

    rect.origin.x = point.x - shapeHoleSizeHalf - shapeStrokeSizeHalf

    rect.origin.y = point.y - shapeHoleSizeHalf - shapeStrokeSizeHalf

    rect.size.width = shapeHoleSize + shapeStrokeSize

    rect.size.height = shapeHoleSize + shapeStrokeSize

```

```

context.strokeEllipse(in: rect)

if let shapeHoleColor = shapeHoleColor
{
    context.setFillColor(shapeHoleColor.cgColor)

    rect.origin.x = point.x - shapeHoleSizeHalf

    rect.origin.y = point.y - shapeHoleSizeHalf

    rect.size.width = shapeHoleSize

    rect.size.height = shapeHoleSize

    context.fillEllipse(in: rect)
}

}

else

{

    context.setFillColor(color.cgColor)

    var rect = CGRect()

    rect.origin.x = point.x - shapeHalf

```

```

        rect.origin.y = point.y - shapeHalf

        rect.size.width = shapeSize

        rect.size.height = shapeSize

        context.fillEllipse(in: rect)

    }

}

}

//

// CombinedChartData.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

```

```
import Foundation
```

```
open class CombinedChartData: BarLineScatterCandleBubbleChartData
```

```
{
```

```
    private var _lineData: LineChartData!
```

```
    private var _barData: BarChartData!
```

```
    private var _scatterData: ScatterChartData!
```

```
    private var _candleData: CandleChartData!
```

```
    private var _bubbleData: BubbleChartData!
```

```
    public override init()
```

```
{
```

```
        super.init()
```

```
}
```

```
    public override init(dataSets: [IChartDataSet]?)
```

```

{

    super.init(dataSets: dataSets)

}

@objc open var lineData: LineChartData!

{

    get

    {

        return _lineData

    }

    set

    {

        _lineData = newValue

        notifyDataChanged()

    }

}

```

```
@objc open var barData: BarChartData!
```

```
{  
  
    get  
  
    {  
  
        return _barData  
  
    }  
  
    set  
  
    {  
  
        _barData = newValue  
  
        notifyDataSetChanged()  
  
    }  
  
}
```

```
@objc open var scatterData: ScatterChartData!
```

```
{  
  
    get  
  
    {
```

```

        return _scatterData
    }

    set

    {

        _scatterData = newValue

        notifyDataChanged()

    }

}

```

@objc open var candleData: CandleChartData!

```

{

    get

    {

        return _candleData

    }

    set

    {

```

```

        _candleData = newValue

        notifyDataChanged()

    }

}

@objc open var bubbleData: BubbleChartData!

{

    get

    {

        return _bubbleData

    }

    set

    {

        _bubbleData = newValue

        notifyDataChanged()

    }

}

```



```

open override func calcMinMax()

{

    _dataSets.removeAll()

    _yMax = -Double.greatestFiniteMagnitude

    _yMin = Double.greatestFiniteMagnitude

    _xMax = -Double.greatestFiniteMagnitude

    _xMin = Double.greatestFiniteMagnitude

    _leftAxisMax = -Double.greatestFiniteMagnitude

    _leftAxisMin = Double.greatestFiniteMagnitude

    _rightAxisMax = -Double.greatestFiniteMagnitude

    _rightAxisMin = Double.greatestFiniteMagnitude

    let allData = self.allData

```

```
for data in allData

{

    data.calcMinMax()

    let sets = data.dataSets

    _dataSets.append(contentsOf: sets)

    if data.yMax > _yMax

    {

        _yMax = data.yMax

    }

    if data.yMin < _yMin

    {

        _yMin = data.yMin

    }

}
```

```
if data.xMax > _xMax
```

```
{
```

```
    _xMax = data.xMax
```

```
}
```

```
if data.xMin < _xMin
```

```
{
```

```
    _xMin = data.xMin
```

```
}
```

```
for dataset in sets
```

```
{
```

```
    if dataset.axisDependency == .left
```

```
    {
```

```
        if dataset.yMax > _leftAxisMax
```

```
        {
```

```
            _leftAxisMax = dataset.yMax
```

```

    }

    if dataset.yMin < _leftAxisMin

    {

        _leftAxisMin = dataset.yMin

    }

}

else

{

    if dataset.yMax > _rightAxisMax

    {

        _rightAxisMax = dataset.yMax

    }

    if dataset.yMin < _rightAxisMin

    {

        _rightAxisMin = dataset.yMin

    }

}

```

```
    }  
  }  
}
```

/// All data objects in row: line-bar-scatter-candle-bubble if not null.

```
@objc open var allData: [ChartData]
```

```
{
```

```
    var data = [ChartData]()
```

```
    if lineData != nil
```

```
    {
```

```
        data.append(lineData)
```

```
    }
```

```
    if barData != nil
```

```
    {
```

```
        data.append(barData)
```

```
    }
```

```

    if scatterData !== nil

    {

        data.append(scatterData)

    }

    if candleData !== nil

    {

        data.append(candleData)

    }

    if bubbleData !== nil

    {

        data.append(bubbleData)

    }

    return data

}

```

```

@objc open func dataByIndex(_ index: Int) -> ChartData

```

```
{  
  
    return allData[index]  
  
}
```

```
open func dataIndex(_ data: ChartData) -> Int?
```

```
{  
  
    return allData.firstIndex(of: data)  
  
}
```

```
open override func removeDataSet(_ dataSet: IChartDataSet) -> Bool
```

```
{  
  
    return allData.contains { $0.removeDataSet(dataSet) }  
  
}
```

```
open override func removeDataSetByIndex(_ index: Int) -> Bool
```

```
{  
  
    print("removeDataSet(index) not supported for CombinedData", terminator: "\n")  
  
}
```

```

        return false
    }

    open override fun removeEntry(_ entry: ChartDataEntry, dataSetIndex: Int) -> Bool

    {

        print("removeEntry(entry, dataSetIndex) not supported for CombinedData",
terminator: "\n")

        return false
    }

    open override fun removeEntry(xValue: Double, dataSetIndex: Int) -> Bool

    {

        print("removeEntry(xValue, dataSetIndex) not supported for CombinedData",
terminator: "\n")

        return false
    }

    open override fun notifyDataChanged()

```



```
{  
  
    if _lineData !== nil  
  
    {  
  
        _lineData.notifyDataChanged()  
  
    }  
  
    if _barData !== nil  
  
    {  
  
        _barData.notifyDataChanged()  
  
    }  
  
    if _scatterData !== nil  
  
    {  
  
        _scatterData.notifyDataChanged()  
  
    }  
  
    if _candleData !== nil  
  
    {  
  
        _candleData.notifyDataChanged()  
  
    }  
  
}
```

```

    if _bubbleData !== nil

    {

        _bubbleData.notifyDataChanged()

    }

    super.notifyDataChanged() // recalculate everything

}

/// Get the Entry for a corresponding highlight object

///

/// - Parameters:

/// - highlight:

/// - Returns: The entry that is highlighted

open override func entryForHighlight(_ highlight: Highlight) -> ChartDataEntry?

{

    if highlight.dataIndex >= allData.count

    {

```

```

        return nil
    }

    let data = dataByIndex(highlight.dataIndex)

    if highlight.dataSetIndex >= data.dataSetCount
    {
        return nil
    }

    // The value of the highlighted entry could be NaN - if we are not interested in
    highlighting a specific value.

    let entries =
data.getDataSetByIndex(highlight.dataSetIndex).entriesForXValue(highlight.x)

    return entries.first { $0.y == highlight.y || highlight.y.isNaN }
}

/// Get dataset for highlight

```

```

///

/// - Parameters:

/// - highlight: current highlight

/// - Returns: dataset related to highlight

@objc open func getDataSetByHighlight(_ highlight: Highlight) -> IChartDataSet!

{

    if highlight.dataIndex >= allData.count

    {

        return nil

    }

    let data = dataByIndex(highlight.dataIndex)

    if highlight.dataSetIndex >= data.dataSetCount

    {

        return nil

    }

```

```

        return data.dataSets[highlight.dataSetIndex]

    }

}

//

// CombinedChartDataProvider.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

```

@objc

```
public protocol CombinedChartDataProvider: LineChartDataProvider,  
BarChartDataProvider, BubbleChartDataProvider, CandleChartDataProvider,  
ScatterChartDataProvider
```

```
{
```

```
    var combinedData: CombinedChartData? { get }
```

```
}//
```

```
// CombinedChartRenderer.swift
```

```
// Charts
```

```
//
```

```
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```
// A port of MPAndroidChart for iOS
```

```
// Licensed under Apache License 2.0
```

```
//
```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
open class CombinedChartRenderer: DataRenderer
```

```
{
```

```
    @objc open weak var chart: CombinedChartView?
```

```
    /// if set to true, all values are drawn above their bars, instead of below their top
```

```
    @objc open var drawValueAboveBarEnabled = true
```

```
    /// if set to true, a grey area is drawn behind each bar that indicates the maximum value
```

```
    @objc open var drawBarShadowEnabled = false
```

```
    internal var _renderers = [DataRenderer]()
```

```
    internal var _drawOrder: [CombinedChartView.DrawOrder] =  
    [.bar, .bubble, .line, .candle, .scatter]
```

```
@objc public init(chart: CombinedChartView, animator: Animator, viewPortHandler:
ViewPortHandler)
```

```
{
```

```
    super.init(animator: animator, viewPortHandler: viewPortHandler)
```

```
    self.chart = chart
```

```
    createRenderers()
```

```
}
```

/// Creates the renderers needed for this combined-renderer in the required order. Also takes the DrawOrder into consideration.

```
internal func createRenderers()
```

```
{
```

```
    _renderers = [DataRenderer]()
```

```
    guard let chart = chart else { return }
```



```

for order in drawOrder

{

    switch (order)

    {

        case .bar:

            if chart.barData != nil

            {

                _renderers.append(BarChartRenderer(dataProvider: chart, animator: animator,
viewPortHandler: viewPortHandler))

            }

            break

        case .line:

            if chart.lineData != nil

            {

                _renderers.append(LineChartRenderer(dataProvider: chart, animator:
animator, viewPortHandler: viewPortHandler))

            }

```

```
break
```

```
case .candle:
```

```
    if chart.candleData !== nil
```

```
    {
```

```
        _renderers.append(CandleStickChartRenderer(dataProvider: chart, animator:  
animator, viewPortHandler: viewPortHandler))
```

```
    }
```

```
break
```

```
case .scatter:
```

```
    if chart.scatterData !== nil
```

```
    {
```

```
        _renderers.append(ScatterChartRenderer(dataProvider: chart, animator:  
animator, viewPortHandler: viewPortHandler))
```

```
    }
```

```
break
```

```

        case .bubble:

            if chart.bubbleData != nil

            {

                _renderers.append(BubbleChartRenderer(dataProvider: chart, animator:
animator, viewPortHandler: viewPortHandler))

            }

            break

        }

    }

}

```

```

open override func initBuffers()

```

```

{

    _renderers.forEach { $0.initBuffers() }

}

```

```

open override func drawData(context: CGContext)

```

```

{

    // If we redraw the data, remove and repopulate accessible elements to update label
    values and frames

    accessibleChartElements.removeAll()

    if

        let combinedChart = chart,

        let data = combinedChart.data {

            // Make the chart header the first element in the accessible elements array

            let element = createAccessibleHeader(usingChart: combinedChart,

                                                andData: data,

                                                withDefaultDescription: "Combined Chart")

            accessibleChartElements.append(element)

        }

    // TODO: Due to the potential complexity of data presented in Combined charts, a
    more usable way

    // for VO accessibility would be to use axis based traversal rather than by dataset.

```

// Hence, accessibleChartElements is not populated below. (Individual renderers guard against dataSource being their respective views)

```
_renderers.forEach { $0.drawData(context: context) }  
  
}
```

open override func drawValues(context: CGContext)

```
{  
  
    _renderers.forEach { $0.drawValues(context: context) }  
  
}
```

open override func drawExtras(context: CGContext)

```
{  
  
    _renderers.forEach { $0.drawExtras(context: context) }  
  
}
```

open override func drawHighlighted(context: CGContext, indices: [Highlight])

```
{  
  
    for renderer in _renderers
```

```

{

    var data: ChartData?

    if renderer is BarChartRenderer

    {

        data = (renderer as! BarChartRenderer).dataProvider?.barData

    }

    else if renderer is LineChartRenderer

    {

        data = (renderer as! LineChartRenderer).dataProvider?.lineData

    }

    else if renderer is CandleStickChartRenderer

    {

        data = (renderer as! CandleStickChartRenderer).dataProvider?.candleData

    }

    else if renderer is ScatterChartRenderer

    {

```

```

        data = (renderer as! ScatterChartRenderer).dataProvider?.scatterData
    }

    else if renderer is BubbleChartRenderer
    {

        data = (renderer as! BubbleChartRenderer).dataProvider?.bubbleData
    }

    let dataIndex: Int? = {

        guard let data = data else { return nil }

        return (chart?.data as? CombinedChartData)?

            .allData

            .firstIndex(of: data)

    }()

    let dataIndices = indices.filter{ $0.dataIndex == dataIndex || $0.dataIndex == -1 }

    renderer.drawHighlighted(context: context, indices: dataIndices)

```

```
}  
  
}
```

/// - Returns: The sub-renderer object at the specified index.

```
@objc open func getSubRenderer(index: Int) -> DataRenderer?
```

```
{  
  
    if index >= _renderers.count || index < 0  
  
    {  
  
        return nil  
  
    }  
  
    else  
  
    {  
  
        return _renderers[index]  
  
    }  
  
}
```

/// All sub-renderers.


```

@objc open var subRenderers: [DataRenderer]

{

    get { return _renderers }

    set { _renderers = newValue }

}


// MARK: Accessors


/// `true` if drawing values above bars is enabled, `false` ifnot


@objc open var isDrawValueAboveBarEnabled: Bool { return
drawValueAboveBarEnabled }


/// `true` if drawing shadows (maxvalue) for each bar is enabled, `false` ifnot


@objc open var isDrawBarShadowEnabled: Bool { return drawBarShadowEnabled }


/// the order in which the provided data objects should be drawn.


/// The earlier you place them in the provided array, the further they will be in the
background.

```

/// e.g. if you provide [DrawOrder.Bar, DrawOrder.Line], the bars will be drawn behind the lines.

```
open var drawOrder: [CombinedChartView.DrawOrder]

{

    get

    {

        return _drawOrder

    }

    set

    {

        if newValue.count > 0

        {

            _drawOrder = newValue

        }

    }

}

//
```

```

// CombinedChartView.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

/// This chart class allows the combination of lines, bars, scatter and candle data all
displayed in one chart area.

open class CombinedChartView: BarLineChartViewBase, CombinedChartDataProvider

{

    /// the fill-formatter used for determining the position of the fill-line

```

```
internal var _fillFormatter: IFillFormatter!
```

```
/// enum that allows to specify the order in which the different data objects for the  
combined-chart are drawn
```

```
@objc(CombinedChartDrawOrder)
```

```
public enum DrawOrder: Int
```

```
{
```

```
    case bar
```

```
    case bubble
```

```
    case line
```

```
    case candle
```

```
    case scatter
```

```
}
```

```
open override func initialize()
```

```
{
```

```
    super.initialize()
```

```
self.highlighter = CombinedHighlighter(chart: self, barDataProvider: self)
```

```
// Old default behaviour
```

```
self.highlightFullBarEnabled = true
```

```
_fillFormatter = DefaultFillFormatter()
```

```
    renderer = CombinedChartRenderer(chart: self, animator: _animator,  
viewPortHandler: _viewPortHandler)  
  
}
```

```
open override var data: ChartData?
```

```
{  
  
    get  
  
    {  
  
        return super.data  
  
    }  
  
    set
```

```

{

    super.data = newValue

    self.highlighter = CombinedHighlighter(chart: self, barDataProvider: self)

    (renderer as? CombinedChartRenderer)?.createRenderers()

    renderer?.initBuffers()

}

}

```

```

@objc open var fillFormatter: IFillFormatter

```

```

{

    get

    {

        return _fillFormatter

    }

    set

```

```

{

    _fillFormatter = newValue

    if _fillFormatter == nil

    {

        _fillFormatter = DefaultFillFormatter()

    }

}

}

```

/// - Returns: The Highlight object (contains x-index and DataSet index) of the selected value at the given touch point inside the CombinedChart.

open override func getHighlightByTouchPoint(_ pt: CGPoint) -> Highlight?

```

{

    if _data === nil

    {

        Swift.print("Can't select by touch. No data set.")

        return nil

    }

}

```

```

guard let h = self.highlighter?.getHighlight(x: pt.x, y: pt.y)

    else { return nil }

if !isHighlightFullBarEnabled { return h }

// For isHighlightFullBarEnabled, remove stackIndex

return Highlight(

    x: h.x, y: h.y,

    xPx: h.xPx, yPx: h.yPx,

    dataIndex: h.dataIndex,

    dataSetIndex: h.dataSetIndex,

    stackIndex: -1,

    axis: h.axis)

}

```

```

// MARK: - CombinedChartDataProvider

```



```
open var combinedData: CombinedChartData?
```

```
{
```

```
    get
```

```
    {
```

```
        return _data as? CombinedChartData
```

```
    }
```

```
}
```

```
// MARK: - LineChartDataProvider
```

```
open var lineData: LineChartData?
```

```
{
```

```
    get
```

```
    {
```

```
        return combinedData?.lineData
```

```
    }
```

```
}
```

```
// MARK: - BarChartDataProvider
```

```
open var barData: BarChartData?
```

```
{
```

```
    get
```

```
    {
```

```
        return combinedData?.barData
```

```
    }
```

```
}
```

```
// MARK: - ScatterChartDataProvider
```

```
open var scatterData: ScatterChartData?
```

```
{
```

```
    get
```

```
{  
  
    return combinedData?.scatterData  
  
}  
  
}
```

// MARK: - CandleChartDataProvider

open var candleData: CandleChartData?

```
{  
  
    get  
  
    {  
  
        return combinedData?.candleData  
  
    }  
  
}
```

// MARK: - BubbleChartDataProvider

```
open var bubbleData: BubbleChartData?
```

```
{
```

```
    get
```

```
    {
```

```
        return combinedData?.bubbleData
```

```
    }
```

```
}
```

```
// MARK: - Accessors
```

```
/// if set to true, all values are drawn above their bars, instead of below their top
```

```
@objc open var drawValueAboveBarEnabled: Bool
```

```
{
```

```
    get { return (renderer as! CombinedChartRenderer).drawValueAboveBarEnabled }
```

```
    set { (renderer as! CombinedChartRenderer).drawValueAboveBarEnabled =  
newValue }
```

```
}
```

/// if set to true, a grey area is drawn behind each bar that indicates the maximum value

@objc open var drawBarShadowEnabled: Bool

{

get { return (renderer as! CombinedChartRenderer).drawBarShadowEnabled }

set { (renderer as! CombinedChartRenderer).drawBarShadowEnabled = newValue }

}

/// `true` if drawing values above bars is enabled, `false` if not

open var isDrawValueAboveBarEnabled: Bool { return (renderer as!
CombinedChartRenderer).drawValueAboveBarEnabled }

/// `true` if drawing shadows (maxvalue) for each bar is enabled, `false` if not

open var isDrawBarShadowEnabled: Bool { return (renderer as!
CombinedChartRenderer).drawBarShadowEnabled }

/// the order in which the provided data objects should be drawn.

/// The earlier you place them in the provided array, the further they will be in the background.

/// e.g. if you provide [DrawOrder.Bar, DrawOrder.Line], the bars will be drawn behind the lines.

```
@objc open var drawOrder: [Int]

{

    get

    {

        return (renderer as! CombinedChartRenderer).drawOrder.map { $0.rawValue }

    }

    set

    {

        (renderer as! CombinedChartRenderer).drawOrder = newValue.map
        { DrawOrder(rawValue: $0)! }

    }

}
```

/// Set this to `true` to make the highlight operation full-bar oriented, `false` to make it highlight single values

```
@objc open var highlightFullBarEnabled: Bool = false
```

```

/// `true` the highlight is be full-bar oriented, `false` if single-value

open var isHighlightFullBarEnabled: Bool { return highlightFullBarEnabled }


// MARK: - ChartViewBase


/// draws all MarkerViews on the highlighted positions

override func drawMarkers(context: CGContext)

{

    guard

        let marker = marker,

        isDrawMarkersEnabled && valuesToHighlight()

    else { return }


    for i in 0 ..< _indicesToHighlight.count

    {

        let highlight = _indicesToHighlight[i]

```

guard

let set = combinedData?.getDataSetByHighlight(highlight),

let e = _data?.entryForHighlight(highlight)

else { continue }

let entryIndex = set.entryIndex(entry: e)

if entryIndex > Int(Double(set.entryCount) * _animator.phaseX)

{

continue

}

let pos = getMarkerPosition(highlight: highlight)

// check bounds

if !_viewPortHandler.isInBounds(x: pos.x, y: pos.y)

{

continue


```

    }

    // callbacks to update the content

    marker.refreshContent(entry: e, highlight: highlight)

    // draw the marker

    marker.draw(context: context, point: pos)

    }

    }

    }

    //

    // CombinedHighlighter.swift

    // Charts

    //

    // Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

    // A port of MPAndroidChart for iOS

    // Licensed under Apache License 2.0

```

```

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc(CombinedChartHighlighter)

open class CombinedHighlighter: ChartHighlighter

{

    /// bar highlighter for supporting stacked highlighting

    private var barHighlighter: BarHighlighter?

    @objc public init(chart: CombinedChartDataProvider, barDataProvider:
BarChartDataProvider)

    {

        super.init(chart: chart)

```

```

        // if there is BarData, create a BarHighlighter

        self.barHighlighter = barDataProvider.barData == nil ? nil : BarHighlighter(chart:
barDataProvider)

    }

    open override func getHighlights(xValue: Double, x: CGFloat, y: CGFloat) ->
[Highlight]

    {

        var vals = [Highlight]()

        guard

            let chart = self.chart as? CombinedChartDataProvider,

            let dataObjects = chart.combinedData?.allData

        else { return vals }

        for i in 0..

```

```

// in case of BarData, let the BarHighlighter take over

if barHighlighter != nil && dataObject is BarChartData,

    let high = barHighlighter?.getHighlight(x: x, y: y)

    {

        high.dataIndex = i

        vals.append(high)

    }

else

    {

        for j in 0..

```

```
        let highs = buildHighlights(dataSet: dataSet, dataSetIndex: j, xValue: xValue,  
rounding: .closest)
```

```
        for high in highs
```

```
        {
```

```
            high.dataIndex = i
```

```
            vals.append(high)
```

```
        }
```

```
    }
```

```
}
```

```
}
```

```
    return vals
```

```
}
```

```
}
```

```
//
```

```
// ComponentBase.swift
```

```
// Charts
```

```

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

/// This class encapsulates everything both Axis, Legend and LimitLines have in common

@objc(ChartComponentBase)

open class ComponentBase: NSObject

{

    /// flag that indicates if this component is enabled or not

```

```
@objc open var enabled = true
```

```
/// The offset this component has on the x-axis
```

```
/// default: 5.0
```

```
@objc open var xOffset = CGFloat(5.0)
```

```
/// The offset this component has on the x-axis
```

```
/// default: 5.0 (or 0.0 on ChartYAxis)
```

```
@objc open var yOffset = CGFloat(5.0)
```

```
public override init()
```

```
{
```

```
    super.init()
```

```
}
```

```
@objc open var isEnabled: Bool { return enabled }
```

```
}
```

/*

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `ConditionCell` displays characteristic and location conditions.

*/

```
import UIKit
```

```
import HomeKit
```

```
/// A `UITableViewCell` subclass that displays a trigger condition.
```

```
class ConditionCell: UITableViewCell {
```

```
    /// A static, short date formatter.
```

```
    static let dateFormatter: DateFormatter = {
```

```
        let dateFormatter = DateFormatter()
```

```
        dateFormatter.dateStyle = .none
```



```
dateFormatter.timeStyle = .short
```

```
dateFormatter.locale = Locale.current
```

```
return dateFormatter
```

```
}()
```

```
/// Ignores the passed-in style and overrides it with .Subtitle.
```

```
override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
```

```
    super.init(style: .subtitle, reuseIdentifier: reuseIdentifier)
```

```
    selectionStyle = .none
```

```
    detailTextLabel?.textColor = UIColor.lightGray
```

```
    accessoryType = .none
```

```
}
```

```
/// Required because we overwrote a designated initializer.
```

```
required init?(coder aDecoder: NSCoder) {
```

```
    super.init(coder: aDecoder)
```

```
}
```

```
/**
```

Sets the cell's text to represent a characteristic and target value.

For example, "Brightness → 60%"

Sets the subtitle to the service and accessory that this characteristic represents.

- parameter characteristic: The characteristic this cell represents.

- parameter targetValue: The target value from this action.

```
*/
```

```
func setCharacteristic(_ characteristic: HMCharacteristic, targetValue: AnyObject) {
```

```
    let    targetDescription    =    "\(characteristic.localizedDescription)    →  
    \(characteristic.localizedDescriptionForValue(targetValue))"
```

```
    textLabel?.text = targetDescription
```

```
    let contextDescription = NSLocalizedString("%@ in %@", comment: "Service in  
Accessory")
```

```
    if let service = characteristic.service, let accessory = service.accessory {
```

```
        detailTextLabel?.text    =    String(format:    contextDescription,    service.name,  
        accessory.name)
```

```

    }

    else {

        detailTextLabel?.text = NSLocalizedString("Unknown Characteristic", comment:
"Unknown Characteristic")

    }

}

```

```
/**
```

Sets the cell's text to represent an ordered time with a set context string.

- parameter order: A `TimeConditionOrder` which will map to a localized string.

- parameter timeString: The localized time string.

- parameter contextString: A localized string describing the time type.

```
*/
```

```

private func setOrder(_ order: TimeConditionOrder, timeString: String, contextString:
String) {

    let formatString: String

    switch order {

```

case .before:

```
formatString = NSLocalizedString("Before %@", comment: "Before Time")
```

case .after:

```
formatString = NSLocalizedString("After %@", comment: "After Time")
```

case .at:

```
formatString = NSLocalizedString("At %@", comment: "At Time")
```

```
}
```

```
textLabel?.text = String(format: formatString, timeString)
```

```
detailTextLabel?.text = contextString
```

```
}
```

```
/**
```

Sets the cell's text to represent an exact time condition.

- parameter order: A `TimeConditionOrder` which will map to a localized string.

- parameter dateComponents: The date components of the exact time.

*/

```
func setOrder(_ order: TimeConditionOrder, dateComponents: DateComponents) {
```

```
    let date = Calendar.current.date(from: dateComponents)
```

```
    let timeString = ConditionCell.dateFormatter.string(from: date!)
```

```
    setOrder(order, timeString: timeString, contextString: NSLocalizedString("Relative  
to Time", comment: "Relative to Time"))
```

```
}
```

/**

Sets the cell's text to represent a solar event time condition.

- parameter order: A `TimeConditionOrder` which will map to a localized string.

- parameter sunState: A `TimeConditionSunState` which will map to localized string.

*/

```
func setOrder(_ order: TimeConditionOrder, sunState: TimeConditionSunState) {
```

```
    let timeString: String
```

```
    switch sunState {
```

```

        case .sunrise:

            timeString = NSLocalizedString("Sunrise", comment: "Sunrise")

        case .sunset:

            timeString = NSLocalizedString("Sunset", comment: "Sunset")

        }

        setOrder(order, timeString: timeString , contextString: NSLocalizedString("Relative
to sun", comment: "Relative to Sun"))

    }

    /// Sets the cell's text to indicate the given condition is not handled by the app.

    func setUnknown() {

        let unknownString = NSLocalizedString("Unknown Condition", comment:
"Unknown Condition")

        detailTextLabel?.text = unknownString

       .textLabel?.text = unknownString

    }

}

```

```
//  
  
// MIT License  
  
//  
  
// Copyright (c) 2017 Robert-Hein Hooijmans <rh.hooijmans@gmail.com>  
  
//  
  
// Permission is hereby granted, free of charge, to any person obtaining a copy  
  
// of this software and associated documentation files (the "Software"), to deal  
  
// in the Software without restriction, including without limitation the rights  
  
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell  
  
// copies of the Software, and to permit persons to whom the Software is  
  
// furnished to do so, subject to the following conditions:  
  
//  
  
// The above copyright notice and this permission notice shall be included in  
  
// all copies or substantial portions of the Software.  
  
//  
  
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY  
KIND, EXPRESS OR
```

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

#if os(OSX)

import AppKit

#else

import UIKit

#endif

extension View: Constraintable {


```

@discardableResult

public func prepareForLayout() -> Self {

    translatesAutoresizingMaskIntoConstraints = false

    return self

}

}

```

```

extension LayoutGuide: Constraintable {

    @discardableResult

    public func prepareForLayout() -> Self { return self }

}

```

```

public protocol Constraintable {

    var topAnchor: NSLayoutYAxisAnchor { get }

    var bottomAnchor: NSLayoutYAxisAnchor { get }

    var leftAnchor: NSLayoutXAxisAnchor { get }

    var rightAnchor: NSLayoutXAxisAnchor { get }

```

```

var leadingAnchor: NSLayoutXAxisAnchor { get }

var trailingAnchor: NSLayoutXAxisAnchor { get }


var centerXAnchor: NSLayoutXAxisAnchor { get }

var centerYAnchor: NSLayoutYAxisAnchor { get }


var widthAnchor: NSLayoutDimension { get }

var heightAnchor: NSLayoutDimension { get }


@discardableResult

func prepareForLayout() -> Self

}

//

// MIT License

//

// Copyright (c) 2017 Robert-Hein Hooijmans <rh.hooijmans@gmail.com>

//

```

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
KIND, EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM,

```
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER  
DEALINGS IN
```

```
// THE SOFTWARE.
```

```
//
```

```
#if os(OSX)
```

```
    import AppKit
```

```
#else
```

```
    import UIKit
```

```
#endif
```

```
public typealias Constraint = NSLayoutConstraint
```

```
public typealias Constraints = [Constraint]
```

```
public enum ConstraintRelation: Int {
```

```
    case equal = 0
```

```
    case equalOrLess = -1
```

```
    case equalOrGreater = 1
```

```
}
```

```
public extension Collection where Iterator.Element == Constraint {
```

```
  func activate() {
```

```
    if let constraints = self as? Constraints {
```

```
      Constraint.activate(constraints)
```

```
    }
```

```
  }
```

```
  func deActivate() {
```

```
    if let constraints = self as? Constraints {
```

```
      Constraint.deactivate(constraints)
```

```
    }
```

```
  }
```

```
}
```

```
#if os(OSX)
```

```
public extension Constraint {
```

```
  @objc
```

```
  func with(_ p: Constraint.Priority) -> Self {
```

```
    priority = p
```

```
    return self
```

```
  }
```

```
  func set(_ active: Bool) -> Self {
```

```
    isActive = active
```

```
    return self
```

```
  }
```

```
}
```

```
#else
```

```
public extension Constraint {
```

@objc

```
func with(_ p: LayoutPriority) -> Self {
```

```
    priority = p
```

```
    return self
```

```
}
```

```
func set(_ active: Bool) -> Self {
```

```
    isActive = active
```

```
    return self
```

```
}
```

```
}
```

```
#endif
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `ControlsTableViewDataSource` provides data for the `ControlsViewController`.

*/

```
import UIKit
```

```
import HomeKit
```

```
/// A `UITableViewDataSource` that populates the table in `ControlsViewController`.
```

```
class ControlsTableViewDataSource: NSObject, UITableViewDataSource {
```

```
    // MARK: Types
```

```
    struct Identifiers {
```

```
        static let serviceCell = "ServiceCell"
```

```
        static let unreachableServiceCell = "UnreachableServiceCell"
```

```
    }
```

```
    // MARK: Properties
```



```
var serviceTable: [String: [HMService]]?
```

```
var sortedKeys: [String]?
```

```
let tableView: UITableView
```

```
var home: HMHome? {
```

```
    return HomeStore.shared.home
```

```
}
```

```
/// Initializes the table view and data source.
```

```
required init(tableView: UITableView) {
```

```
    self.tableView = tableView
```

```
    super.init()
```

```
    self.tableView.dataSource = self
```

```
}
```

```
/**
```

Reloads the table, sets the table's dataSource to self,
regenerated the service table, creates a sorted list of keys,
sets the home's delegate, and reloads the table.

*/

```
func reloadData() {  
  
    if let home = home {  
  
        serviceTable = home.serviceTable  
  
        sortedKeys = serviceTable!.keys.sorted()  
  
        tableView.reloadData()  
  
    }  
  
    else {  
  
        serviceTable = nil  
  
        sortedKeys = nil  
  
    }  
  
    tableView.reloadData()  
  
}
```

/// - returns: The localized description of the service type for that section.

```
func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) ->
String? {

    return sortedKeys?[section]

}
```

```
func numberOfSections(in tableView: UITableView) -> Int {

    return sortedKeys?.count ?? 0

}
```

/**

- returns: A message that corresponds to the current most important reason

that there are no services in the table. Either "No Accessories"

or "No Services".

*/

```
func emptyMessage() -> String {

    if home?.accessories.count == 0 {
```

```

        return NSLocalizedString("No Accessories", comment: "No Accessories")

    }

    else {

        return NSLocalizedString("No Services", comment: "No Services")

    }

}

```

/// - returns: The number of services matching the service type in that section.

```

func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) ->
Int {

    return serviceTable![sortedKeys![section]]!.count

}

```

/// - returns: A `ServiceCell` set for the service at the provided index path.

```

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {

    let service = serviceForIndexPath(indexPath)!

```

```
        let reuseIdentifier = service.accessory!.isReachable ? Identifiers.serviceCell :
Identifiers.unreachableServiceCell
```

```
        let cell = tableView.dequeueReusableCell(withIdentifier: reuseIdentifier, for:
indexPath) as! ServiceCell
```

```
        cell.service = service
```

```
        return cell
```

```
    }
```

```
/// - returns: The service represented at the index path in the table.
```

```
func serviceForIndexPath(_ indexPath: IndexPath) -> HMService? {
```

```
    if let sortedKeys = sortedKeys,
```

```
        let serviceTable = serviceTable,
```

```
        let services = serviceTable[sortedKeys[indexPath.section]] {
```

```
            return services[indexPath.row]
```

```
    }
```

```
        return nil
    }
}
```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `ControlsViewController` lists services in the selected home.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
/// A view controller which displays a list of `HMServices`, separated by Service Type.
```

```

class ControlsViewController: HMCatalogViewController, HMAccessoryDelegate {

    // MARK: Types

    struct Identifiers {

        static let showServiceSegue = "Show Service"

    }

    // MARK: Properties

    var tableViewDataSource: ControlsTableViewDataSource!

    var cellController = AccessoryUpdateController()

    @IBOutlet weak var addButton: UIBarButtonItem!

    // MARK: View Methods

    /// Sends the selected service into the destination view controller.

```

```

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

    super.prepare(for: segue, sender: sender)

    if segue.identifier == Identifiers.showServiceSegue {

        if let indexPath = tableView.indexPath(for: sender as! UITableViewCell) {

            let characteristicsViewController = segue.intendedDestinationViewController as!
CharacteristicsViewController

            if let selectedService = tableViewDataSource.serviceForIndexPath(indexPath) {

                characteristicsViewController.service = selectedService

            }

            characteristicsViewController.cellDelegate = cellController

        }

    }

}

/// Initializes the table view data source.

override func viewDidLoad() {

```



```

super.viewDidLoad()

tableViewDataSource = ControlsTableViewDataSource(tableView: tableView)

}

/// Reloads the view.

override func viewWillAppear(_ animated: Bool) {

    super.viewWillAppear(animated)

    navigationItem.title = home.name

    reloadData()

}

// MARK: Helper Methods

private func reloadData() {

    tableViewDataSource.reloadTable()

    let sections = tableViewDataSource.numberOfSections(in: tableView)

```

```

if sections == 0 {

    setBackgroundImage(tableViewDataSource.emptyMessage())

}

else {

    setBackgroundImage(nil)

}

}

// MARK: Delegate Registration

/// Registers as the delegate for the current home and all accessories in the home.

override func registerAsDelegate() {

    super.registerAsDelegate()

    for accessory in home.accessories {

        accessory.delegate = self

    }

}

```

```
/*
```

Any delegate methods which could change data will reload the
table view data source.

```
*/
```

```
// MARK: HMHomeDelegate Methods
```

```
func home(_ home: HMHome, didAddAccessory accessory: HMAccessory) {
```

```
    accessory.delegate = self
```

```
    reloadData()
```

```
}
```

```
func home(_ home: HMHome, didRemoveAccessory accessory: HMAccessory) {
```

```
    reloadData()
```

```
}
```

```
// MARK: HMAccessoryDelegate Methods
```

```
func accessoryDidUpdateReachability(_ accessory: HMAccessory) {  
  
    reloadData()  
  
}
```

```
func accessory(_ accessory: HMAccessory, didUpdateNameFor service: HMService) {  
  
    reloadData()  
  
}
```

```
func accessory(_ accessory: HMAccessory, didUpdateAssociatedServiceTypeFor  
service: HMService) {  
  
    reloadData()  
  
}
```

```
func accessoryDidUpdateServices(_ accessory: HMAccessory) {  
  
    reloadData()  
  
}
```

```
func accessoryDidUpdateName(_ accessory: HMAccessory) {  
  
    reloadData()  
  
}  
  
}  
  
//  
  
// CrossShapeRenderer.swift  
  
// Charts  
  
//  
  
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda  
  
// A port of MPAndroidChart for iOS  
  
// Licensed under Apache License 2.0  
  
//  
  
// https://github.com/danielgindi/Charts  
  
//  
  
import Foundation  
  
import CoreGraphics
```

```

open class CrossShapeRenderer : NSObject, IShapeRenderer

{

    open func renderShape(

        context: CGContext,

        dataSet: IScatterChartDataSet,

        viewPortHandler: ViewPortHandler,

        point: CGPoint,

        color: NSUIColor)

    {

        let shapeSize = dataSet.scatterShapeSize

        let shapeHalf = shapeSize / 2.0


        context.setLineWidth(1.0)

        context.setStrokeColor(color.cgColor)


        context.beginPath()

```

```
        context.move(to: CGPoint(x: point.x - shapeHalf, y: point.y))

        context.addLine(to: CGPoint(x: point.x + shapeHalf, y: point.y))

        context.move(to: CGPoint(x: point.x, y: point.y - shapeHalf))

        context.addLine(to: CGPoint(x: point.x, y: point.y + shapeHalf))

        context.strokePath()

    }

}

//

// DataApproximator+N.swift

// Charts

//

// Created by M Ivaniushchenko on 9/6/17.

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//
```

```
import Foundation
```

```
import CoreGraphics
```

```
extension CGPoint {
```

```
    fileprivate func distanceToLine(from linePoint1: CGPoint, to linePoint2: CGPoint) ->  
    CGFloat {
```

```
        let dx = linePoint2.x - linePoint1.x
```

```
        let dy = linePoint2.y - linePoint1.y
```

```
        let dividend = abs(dy * self.x - dx * self.y - linePoint1.x * linePoint2.y + linePoint2.x  
        * linePoint1.y)
```

```
        let divisor = sqrt(dx * dx + dy * dy)
```

```
        return dividend / divisor
```

```
    }
```

```
}
```

```
private struct LineAlt {
```



```
let start: Int
```

```
let end: Int
```

```
var distance: CGFloat = 0
```

```
var index: Int = 0
```

```
init(start: Int, end: Int, points: [CGPoint]) {
```

```
    self.start = start
```

```
    self.end = end
```

```
    let startPoint = points[start]
```

```
    let endPoint = points[end]
```

```
    guard (end > start + 1) else {
```

```
        return
```

```
    }
```

```

for i in start + 1 ..< end {

    let currentPoint = points[i]

    let distance = currentPoint.distanceToLine(from: startPoint, to: endPoint)

    if distance > self.distance {

        self.index = i

        self.distance = distance

    }

}

}

}

}

extension LineAlt: Comparable {

    static func ==(lhs: LineAlt, rhs: LineAlt) -> Bool {

        return (lhs.start == rhs.start) && (lhs.end == rhs.end) && (lhs.index == rhs.index)

    }

}

```

```

static func <(lhs: LineAlt, rhs: LineAlt) -> Bool {

    return lhs.distance < rhs.distance

}

}

extension DataApproximator {

    /// uses the douglas peuker algorithm to reduce the given arraylist of entries to given
    number of points

    /// More algorithm details here - http://psimpl.sourceforge.net/douglas-peucker.html

    @objc open class func reduceWithDouglasPeukerN(_ points: [CGPoint], resultCount:
    Int) -> [CGPoint]

    {

        // if a shape has 2 or less points it cannot be reduced

        if resultCount <= 2 || resultCount >= points.count

        {

            return points

```

```
}
```

```
var keep = [Bool](repeating: false, count: points.count)
```

```
// first and last always stay
```

```
keep[0] = true
```

```
keep[points.count - 1] = true
```

```
var currentStoredPoints = 2
```

```
var queue = [LineAlt]()
```

```
let line = LineAlt(start: 0, end: points.count - 1, points: points)
```

```
queue.append(line)
```

```
repeat {
```

```
    let line = queue.popLast()!
```

```
    // store the key
```

```
    keep[line.index] = true
```

```

// check point count tolerance

currentStoredPoints += 1

if (currentStoredPoints == resultCount) {

    break;

}

// split the polyline at the key and recurse

let left = LineAlt(start: line.start, end: line.index, points: points)

if (left.index > 0) {

    self.insertLine(left, into: &queue)

}

let right = LineAlt(start: line.index, end: line.end, points: points)

if (right.index > 0) {

    self.insertLine(right, into: &queue)

```

```
}
```

```
} while !queue.isEmpty
```

```
// create a new array with series, only take the kept ones
```

```
let reducedEntries = points.enumerated().compactMap { (index: Int, point: CGPoint)  
-> CGPoint? in
```

```
    return keep[index] ? point : nil
```

```
}
```

```
return reducedEntries
```

```
}
```

```
// Keeps array sorted
```

```
private static func insertLine(_ line: LineAlt, into array: inout [LineAlt]) {
```

```
    let insertionIndex = self.insertionIndex(for: line, into: &array)
```

```
    array.insert(line, at: insertionIndex)
```

```
}
```

```

private static func insertionIndex(for line: LineAlt, into array: inout [LineAlt]) -> Int {

    var indices = array.indices

    while !indices.isEmpty {

        let midIndex = indices.lowerBound.advanced(by: indices.count / 2)

        let midLine = array[midIndex]

        if midLine == line {

            return midIndex

        }

        else if (line < midLine) {

            // perform search in left half

            indices = indices.lowerBound..

```

```

        indices = (midIndex + 1)..<indices.upperBound

    }

}

return indices.lowerBound

}

}

//

// DataApproximator.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

```



```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
@objc(ChartDataApproximator)
```

```
open class DataApproximator: NSObject
```

```
{
```

```
    /// uses the douglas peuker algorithm to reduce the given arraylist of entries
```

```
    @objc open class func reduceWithDouglasPeuker(_ points: [CGPoint], tolerance: CGFloat) -> [CGPoint]
```

```
{
```

```
    // if a shape has 2 or less points it cannot be reduced
```

```
    if tolerance <= 0 || points.count < 3
```

```
{
```

```
        return points
```

```
}
```

```

var keep = [Bool](repeating: false, count: points.count)

// first and last always stay

keep[0] = true

keep[points.count - 1] = true


// first and last entry are entry point to recursion

reduceWithDouglasPeuker(points: points,

    tolerance: tolerance,

    start: 0,

    end: points.count - 1,

    keep: &keep)

// create a new array with series, only take the kept ones

return zip(keep, points).compactMap { $0 ? nil : $1 }

}

```

```
/// apply the Douglas-Peucker-Reduction to an array of `CGPoint`s with a given
tolerance
```

```
///
```

```
/// - Parameters:
```

```
/// - points:
```

```
/// - tolerance:
```

```
/// - start:
```

```
/// - end:
```

```
open class func reduceWithDouglasPeuker(
```

```
    points: [CGPoint],
```

```
    tolerance: CGFloat,
```

```
    start: Int,
```

```
    end: Int,
```

```
    keep: inout [Bool])
```

```
{
```

```
    if end <= start + 1
```

```
    {
```

```

    // recursion finished

    return

}

var greatestIndex = Int(0)

var greatestDistance = CGFloat(0.0)

let line = Line(pt1: points[start], pt2: points[end])

for i in start + 1 ..< end

{

    let distance = line.distance(toPoint: points[i])

    if distance > greatestDistance

    {

        greatestDistance = distance

        greatestIndex = i
    }
}

```

```

    }

}

if greatestDistance > tolerance

{

    // keep max dist point

    keep[greatestIndex] = true


    // recursive call

    reduceWithDouglasPeuker(points: points, tolerance: tolerance, start: start, end:
greatestIndex, keep: &keep)

    reduceWithDouglasPeuker(points: points, tolerance: tolerance, start: greatestIndex,
end: end, keep: &keep)

} // else don't keep the point...

}

private class Line

{

```

```
var sxy: CGFloat
```

```
var exsy: CGFloat
```

```
var dx: CGFloat
```

```
var dy: CGFloat
```

```
var length: CGFloat
```

```
init(pt1: CGPoint, pt2: CGPoint)
```

```
{
```

```
    dx = pt1.x - pt2.x
```

```
    dy = pt1.y - pt2.y
```

```
    sxy = pt1.x * pt2.y
```

```
    exsy = pt2.x * pt1.y
```

```
    length = sqrt(dx * dx + dy * dy)
```

```
}
```

```

func distance(toPoint pt: CGPoint) -> CGFloat

{

    return abs(dy * pt.x - dx * pt.y + skey - exsy) / length

}

}

}

//

// DateValueFormatter.swift

// ChartTime

//

// Created by Anantaya on 3/5/2563 BE.

// Copyright © 2563 Anantaya. All rights reserved.

//

```

```

import Foundation

```

```

import Charts

```

```

public class DateValueFormatter: NSObject, IAxisValueFormatter {

    private let dateFormatter = DateFormatter()

    override init() {

        super.init()

        dateFormatter.dateFormat = "dd MMM HH:mm"

    }

    public func stringForValue(_ value: Double, axis: AxisBase?) -> String {

        return dateFormatter.string(from: Date(timeIntervalSince1970: value))

    }

}

//

// DefaultAxisValueFormatter.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

```



```

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

@objc(ChartDefaultAxisValueFormatter)

open class DefaultAxisValueFormatter: NSObject, IAxisValueFormatter

{

    public typealias Block = (

        _ value: Double,

        _ axis: AxisBase?) -> String

    )

    @objc open var block: Block?

```

```
@objc open var hasAutoDecimals: Bool = false
```

```
private var _formatter: NumberFormatter?
```

```
@objc open var formatter: NumberFormatter?
```

```
{  
  
    get { return _formatter }  
  
    set  
  
    {  
  
        hasAutoDecimals = false  
  
        _formatter = newValue  
  
    }  
  
}
```

```
// TODO: Documentation. Especially the nil case
```

```
private var _decimals: Int?
```

```
open var decimals: Int?
```

```
{
```

```

    get { return _decimals }

    set

    {

        _decimals = newValue

        if let digits = newValue

        {

            self.formatter?.minimumFractionDigits = digits

            self.formatter?.maximumFractionDigits = digits

            self.formatter?.usesGroupingSeparator = true

        }

    }

}

public override init()

{

    super.init()

```

```

        self.formatter = NumberFormatter()

        hasAutoDecimals = true
    }

    @objc public init(formatter: NumberFormatter)
    {
        super.init()

        self.formatter = formatter
    }

    @objc public init(decimals: Int)
    {
        super.init()

        self.formatter = NumberFormatter()
    }

```

```
self.formatter?.usesGroupingSeparator = true
```

```
self.decimals = decimals
```

```
hasAutoDecimals = true
```

```
}
```

```
@objc public init(block: @escaping Block)
```

```
{
```

```
    super.init()
```

```
    self.block = block
```

```
}
```

```
@objc public static func with(block: @escaping Block) -> DefaultAxisValueFormatter?
```

```
{
```

```
    return DefaultAxisValueFormatter(block: block)
```

```
}
```

```

open func stringForValue(_ value: Double,

                           axis: AxisBase?) -> String

{

    if let block = block {

        return block(value, axis)

    } else {

        return formatter?.string(from: NSNumber(floatLiteral: value)) ?? ""

    }

}

//

// DefaultFillFormatter.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

```

```

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

/// Default formatter that calculates the position of the filled line.

@objc(ChartDefaultFillFormatter)

open class DefaultFillFormatter: NSObject, IFillFormatter

{

    public typealias Block = (

        _ dataSet: ILineChartDataSet,

        _ dataProvider: LineChartDataProvider) -> CGFloat

    @objc open var block: Block?

```

```
public override init() { }
```

```
@objc public init(block: @escaping Block)
```

```
{
```

```
    self.block = block
```

```
}
```

```
@objc public static func with(block: @escaping Block) -> DefaultFillFormatter?
```

```
{
```

```
    return DefaultFillFormatter(block: block)
```

```
}
```

```
open func getFillLinePosition(
```

```
    dataSet: ILineChartDataSet,
```

```
    dataProvider: LineChartDataProvider) -> CGFloat
```

```
{
```

```
    guard block == nil else { return block!(dataSet, dataProvider) }
```



```

var fillMin: CGFloat = 0.0

if dataSet.yMax > 0.0 && dataSet.yMin < 0.0

{

    fillMin = 0.0

}

else if let data = dataProvider.data

{

    let max = data.yMax > 0.0 ? 0.0 : dataProvider.chartYMax

    let min = data.yMin < 0.0 ? 0.0 : dataProvider.chartYMin

    fillMin = CGFloat(dataSet.yMin >= 0.0 ? min : max)

}

return fillMin

}

}

```

```

//

// DefaultValueFormatter.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

@objc(ChartDefaultValueFormatter)

open class DefaultValueFormatter: NSObject, IValueFormatter

{

    public typealias Block = (

```

```
_ value: Double,  
  
_ entry: ChartDataEntry,  
  
_ dataSetIndex: Int,  
  
_ viewPortHandler: ViewPortHandler?) -> String
```

```
@objc open var block: Block?
```

```
@objc open var hasAutoDecimals: Bool = false
```

```
private var _formatter: NumberFormatter?
```

```
@objc open var formatter: NumberFormatter?
```

```
{
```

```
  get { return _formatter }
```

```
  set
```

```
  {
```

```
    hasAutoDecimals = false
```

```
    _formatter = newValue
```

```

    }

}

private var _decimals: Int?

open var decimals: Int?

{

    get { return _decimals }

    set

    {

        _decimals = newValue

        if let digits = newValue

        {

            self.formatter?.minimumFractionDigits = digits

            self.formatter?.maximumFractionDigits = digits

            self.formatter?.usesGroupingSeparator = true

        }

    }

}

```

```
}
```

```
}
```

```
public override init()
```

```
{
```

```
    super.init()
```

```
    self.formatter = NumberFormatter()
```

```
    hasAutoDecimals = true
```

```
}
```

```
@objc public init(formatter: NumberFormatter)
```

```
{
```

```
    super.init()
```

```
    self.formatter = formatter
```

```
}
```

```
@objc public init(decimals: Int)

{

    super.init()


    self.formatter = NumberFormatter()

    self.formatter?.usesGroupingSeparator = true

    self.decimals = decimals

    hasAutoDecimals = true

}
```

```
@objc public init(block: @escaping Block)

{

    super.init()


    self.block = block

}
```

```
@objc public static func with(block: @escaping Block) -> DefaultValueFormatter?
```

```
{
```

```
    return DefaultValueFormatter(block: block)
```

```
}
```

```
open func stringForValue(_ value: Double,
```

```
    entry: ChartDataEntry,
```

```
    dataSetIndex: Int,
```

```
    viewPortHandler: ViewPortHandler?) -> String
```

```
{
```

```
    if let block = block {
```

```
        return block(value, entry, dataSetIndex, viewPortHandler)
```

```
    } else {
```

```
        return formatter?.string(from: NSNumber(floatLiteral: value)) ?? ""
```

```
    }
```

```
}
```

```
}

//

// Description.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

#if canImport(UiKit)

import UIKit
```



```
#endif
```

```
#if canImport(Cocoa)
```

```
import Cocoa
```

```
#endif
```

```
@objc(ChartDescription)
```

```
open class Description: ComponentBase
```

```
{
```

```
    public override init()
```

```
    {
```

```
        #if os(tvOS)
```

```
            // 23 is the smallest recommended font size on the TV
```

```
            font = .systemFont(ofSize: 23)
```

```
        #elseif os(OSX)
```

```
            font = .systemFont(ofSize: NSUIFont.systemFontSize)
```

```
        #else
```

```

        font = .systemFont(ofSize: 8.0)

    #endif

    super.init()

}

/// The text to be shown as the description.

@objc open var text: String?

/// Custom position for the description text in pixels on the screen.

open var position: CGPoint? = nil

/// The text alignment of the description text. Default RIGHT.

@objc open var textAlign: NSTextAlignment = NSTextAlignment.right

/// Font object used for drawing the description text.

@objc open var font: UIFont

```

```
/// Text color used for drawing the description text

@objc open var textColor = NSUIColor.labelOrBlack

}

/*

See LICENSE folder for this sample's licensing information.

Abstract:

A single cell of the collection view that displays a single accessory.

*/
```

```
import UIKit
```

```
import HomeKit
```

```
class DeviceCell: UICollectionViewCell {
```

```
    @IBOutlet weak var imageView: UIImageView!
```

```

@IBOutlet weak var roomLabel: UILabel!

@IBOutlet weak var nameLabel: UILabel!

//@IBOutlet weak var stateLabel: UILabel!

@IBOutlet weak var disclosureButton: UIButton!


/// The service that this cell represents.

var service: HMService? {

    didSet {

        imageView.image = service?.icon

        roomLabel.text = service?.accessory?.room?.name

        nameLabel.text = service?.name

        //stateLabel.text = "Updating..."

        readAndRedraw(characteristic: service?.primaryDisplayCharacteristic, animated:
true)

    }

}

```

```
/// Reads the characteristic value from the HomeKit database, and updates the UI.
```

```
func readAndRedraw(characteristic: HMCharacteristic?, animated: Bool) {  
  
    guard  
  
        let characteristic = characteristic,  
  
        characteristic.properties.contains(HMCharacteristicPropertyReadable),  
  
        let accessory = characteristic.service?.accessory,  
  
        accessory.isReachable else {  
  
            //stateLabel.text = "Unreachable"  
  
            return  
  
        }  
  
    characteristic.readValue { error in  
  
        self.redrawState(error: error)  
  
    }  
  
}
```

```
/// Updates the UI to reflect the given state.
```

```
func redrawState(error: Error? = nil) {
```

```
    imageView.image = service?.icon
```

```
    if let error = error {
```

```
        print(error.localizedDescription)
```

```
        //stateLabel.text = "Update error!"
```

```
    } else {
```

```
        //stateLabel.text = service?.state
```

```
    }
```

```
}
```

```
/// Informs the cell that it's been tapped.
```

```
func tap() {
```

```
    if let characteristic = service?.primaryControlCharacteristic,
```

```
        let value = characteristic.value as? Bool {
```

```

        // Provide visual feedback that the item was tapped.

        bounce()

        // Write the new value to HomeKit.

        characteristic.writeValue(!value) { error in

            self.redrawState(error: error)

        }

    }

}

/// Animates the cell size in a way that looks like a little bounce.

private func bounce() {

    UIView.animate(withDuration: 0.05, animations: {

        self.transform = CGAffineTransform(scaleX: 0.95, y: 0.95)

    }) {_ in

        UIView.animate(withDuration: 0.10, animations: {

            self.transform = CGAffineTransform(scaleX: 1.05, y: 1.05)

```

```

    }) {_ in

        UIView.animate(withDuration: 0.15, animations: {

            self.transform = .identity

        })

    }

}

}

}

}

// Copyright (c) 2016-present, Facebook, Inc. All rights reserved.

//

// You are hereby granted a non-exclusive, worldwide, royalty-free license to use,

// copy, modify, and distribute this software in source code or binary form for use

// in connection with the web services and APIs provided by Facebook.

//

// As with any software that integrates with the Facebook platform, your use of

// this software is subject to the Facebook Developer Principles and Policies

// [http://developers.facebook.com/policy/]. This copyright notice shall be

```


// included in all copies or substantial portions of the software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS

// FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR

// COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER

// IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN

// CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

internal extension Dictionary {

func mapKeyValues<K, V>(_ transform: (Element) throws -> (K, V)) rethrows -> [K: V]
{

var dictionary: [K: V] = [:]

try forEach {

let transformed = try transform(\$0)

```

        dictionary[transformed.0] = transformed.1

    }

    return dictionary

}

}

//

// DispatchQueue+Alamofire.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

```

```
// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//
```

```
import Dispatch
```

```
import Foundation
```

```

extension DispatchQueue {

    /// Execute the provided closure after a `TimeInterval`.

    ///

    /// - Parameters:

    ///   - delay: `TimeInterval` to delay execution.

    ///   - closure: Closure to execute.

    func after(_ delay: TimeInterval, execute closure: @escaping () -> Void) {

        asyncAfter(deadline: .now() + delay, execute: closure)

    }

}

// Copyright (c) 2016-present, Facebook, Inc. All rights reserved.

//

// You are hereby granted a non-exclusive, worldwide, royalty-free license to use,

// copy, modify, and distribute this software in source code or binary form for use

// in connection with the web services and APIs provided by Facebook.

//

```

```
// As with any software that integrates with the Facebook platform, your use of

// this software is subject to the Facebook Developer Principles and Policies

// [http://developers.facebook.com/policy/]. This copyright notice shall be

// included in all copies or substantial portions of the software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS

// FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR

// COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER

// IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN

// CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

import FBSDKShareKit
```

```

/**

ShareDialog.Mode CustomStringConvertible

*/

extension ShareDialog.Mode: CustomStringConvertible {

    /// The string description

    public var description: String {

        return __NSStringFromFBSDKShareDialogMode(self)

    }

}

```

```

/**

AppGroupPrivacy CustomStringConvertible

*/

extension AppGroupPrivacy: CustomStringConvertible {

    /// The string description

    public var description: String {

        return __NSStringFromFBSDKAppGroupPrivacy(self)

    }

}

```

```

    }

}

/**

LikeObjectType CustomStringConvertible

*/

extension LikeObjectType: CustomStringConvertible {

    /// The string description

    public var description: String {

        return __NSStringFromFBSDKLikeObjectType(self)

    }

}

//

// EventMonitor.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

```

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,


```
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER  
DEALINGS IN
```

```
// THE SOFTWARE.
```

```
//
```

```
import Foundation
```

```
/// Protocol outlining the lifetime events inside Alamofire. It includes both events received  
from the various
```

```
/// `URLSession` delegate protocols as well as various events from the lifetime of `Request`  
and its subclasses.
```

```
public protocol EventMonitor {
```

```
    /// The `DispatchQueue` onto which Alamofire's root `CompositeEventMonitor` will  
dispatch events. `.main` by default.
```

```
    var queue: DispatchQueue { get }
```

```
    // MARK: - URLSession Events
```

```
    // MARK: URLSessionDelegate Events
```

/// Event called during `URLSessionDelegate`'s
`urlSession(\_:didBecomeInvalidWithError:)` method.

```
func urlSession(_ session: URLSession, didBecomeInvalidWithError error: Error?)
```

```
// MARK: URLSessionTaskDelegate Events
```

/// Event called during `URLSessionTaskDelegate`'s
`urlSession(\_:task:didReceive:completionHandler:)` method.

```
func urlSession(_ session: URLSession, task: URLSessionTask, didReceive challenge:  
URLAuthenticationChallenge)
```

/// Event called during `URLSessionTaskDelegate`'s
`urlSession(\_:task:didSendBodyData:totalBytesSent:totalBytesExpectedToSend:)`
method.

```
func urlSession(_ session: URLSession,  
  
                task: URLSessionTask,  
  
                didSendBodyData bytesSent: Int64,  
  
                totalBytesSent: Int64,  
  
                totalBytesExpectedToSend: Int64)
```

/// Event called during `URLSessionTaskDelegate`'s
`urlSession(\_:task:needNewBodyStream:)` method.

```
func urlSession(_ session: URLSession, taskNeedsNewBodyStream task:  
URLSessionTask)
```

/// Event called during `URLSessionTaskDelegate`'s
`urlSession(\_:task:willPerformHTTPRedirection:newRequest:completionHandler:)`
method.

```
func urlSession(_ session: URLSession,  
  
                task: URLSessionTask,  
  
                willPerformHTTPRedirection response: HTTPURLResponse,  
  
                newRequest request: URLRequest)
```

/// Event called during `URLSessionTaskDelegate`'s
`urlSession(\_:task:didFinishCollecting:)` method.

```
func urlSession(_ session: URLSession, task: URLSessionTask, didFinishCollecting  
metrics: URLSessionTaskMetrics)
```

/// Event called during `URLSessionTaskDelegate`'s
`urlSession(\_:task:didCompleteWithError:)` method.

```
func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError
error: Error?)
```

```
/// Event called during `URLSessionTaskDelegate`'s
`urlSession(_:taskIsWaitingForConnectivity:)` method.
```

```
@available(macOS 10.13, iOS 11.0, tvOS 11.0, watchOS 4.0, *)
```

```
func urlSession(_ session: URLSession, taskIsWaitingForConnectivity task:
URLSessionTask)
```

```
// MARK: URLSessionDataDelegate Events
```

```
/// Event called during `URLSessionDataDelegate`'s
`urlSession(_:dataTask:didReceive:)` method.
```

```
func urlSession(_ session: URLSession, dataTask: URLSessionDataTask, didReceive
data: Data)
```

```
/// Event called during `URLSessionDataDelegate`'s
`urlSession(_:dataTask:willCacheResponse:completionHandler:)` method.
```

```
func urlSession(_ session: URLSession, dataTask: URLSessionDataTask,
willCacheResponse proposedResponse: CachedURLResponse)
```

// MARK: URLSessionDownloadDelegate Events

/// Event called during `URLSessionDownloadDelegate`'s
`urlSession(\_:downloadTask:didResumeAtOffset:expectedTotalBytes:)` method.

```
func urlSession(_ session: URLSession,  
  
                downloadTask: URLSessionDownloadTask,  
  
                didResumeAtOffset fileOffset: Int64,  
  
                expectedTotalBytes: Int64)
```

/// Event called during `URLSessionDownloadDelegate`'s
`urlSession(\_:downloadTask:didWriteData:totalBytesWritten:totalBytesExpectedToWrite:)` method.

```
func urlSession(_ session: URLSession,  
  
                downloadTask: URLSessionDownloadTask,  
  
                didWriteData bytesWritten: Int64,  
  
                totalBytesWritten: Int64,  
  
                totalBytesExpectedToWrite: Int64)
```

/// Event called during `URLSessionDownloadDelegate`'s
`urlSession(\_:downloadTask:didFinishDownloadingTo:)` method.

```
func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask,  
didFinishDownloadingTo location: URL)
```

// MARK: - Request Events

/// Event called when a `URLRequest` is first created for a `Request`. If a
`RequestAdapter` is active, the

/// `URLRequest` will be adapted before being issued.

```
func request(_ request: Request, didCreateInitialURLRequest urlRequest: URLRequest)
```

/// Event called when the attempt to create a `URLRequest` from a `Request`'s original
`URLRequestConvertible` value fails.

```
func request(_ request: Request, didFailToCreateURLRequestWithError error: NSError)
```

/// Event called when a `RequestAdapter` adapts the `Request`'s initial `URLRequest`.

```
func request(_ request: Request, didAdaptInitialRequest initialRequest: URLRequest, to  
adaptedRequest: URLRequest)
```

/// Event called when a `RequestAdapter` fails to adapt the `Request`'s initial `URLRequest`.

```
func request(_ request: Request, didFailToAdaptURLRequest initialRequest: URLRequest, withError error: NSError)
```

/// Event called when a final `URLRequest` is created for a `Request`.

```
func request(_ request: Request, didCreateURLRequest urlRequest: URLRequest)
```

/// Event called when a `URLSessionTask` subclass instance is created for a `Request`.

```
func request(_ request: Request, didCreateTask task: URLSessionTask)
```

/// Event called when a `Request` receives a `URLSessionTaskMetrics` value.

```
func request(_ request: Request, didGatherMetrics metrics: URLSessionTaskMetrics)
```

/// Event called when a `Request` fails due to an error created by Alamofire. e.g. When certificate pinning fails.

```
func request(_ request: Request, didFailTask task: URLSessionTask, earlyWithError error: NSError)
```

/// Event called when a `Request`'s task completes, possibly with an error. A `Request` may receive this event

/// multiple times if it is retried.

func request(_ request: Request, didCompleteTask task: URLSessionTask, with error: NSError?)

/// Event called when a `Request` is about to be retried.

func requestIsRetrying(_ request: Request)

/// Event called when a `Request` finishes and response serializers are being called.

func requestDidFinish(_ request: Request)

/// Event called when a `Request` receives a `resume` call.

func requestDidResume(_ request: Request)

/// Event called when a `Request`'s associated `URLSessionTask` is resumed.

func request(_ request: Request, didResumeTask task: URLSessionTask)

/// Event called when a `Request` receives a `suspend` call.

```
func requestDidSuspend(_ request: Request)
```

/// Event called when a `Request`'s associated `URLSessionTask` is suspended.

```
func request(_ request: Request, didSuspendTask task: URLSessionTask)
```

/// Event called when a `Request` receives a `cancel` call.

```
func requestDidCancel(_ request: Request)
```

/// Event called when a `Request`'s associated `URLSessionTask` is cancelled.

```
func request(_ request: Request, didCancelTask task: URLSessionTask)
```

// MARK: DataRequest Events

/// Event called when a `DataRequest` calls a `Validation`.

```
func request(_ request: DataRequest,
```

```
    didValidateRequest urlRequest: URLRequest?,
```

```
response: HTTPURLResponse,  
  
data: Data?,  
  
withResult result: Request.ValidationResult)
```

/// Event called when a `DataRequest` creates a `DataResponse<Data?>` value without calling a `ResponseSerializer`.

```
func request(_ request: DataRequest, didParseResponse response: DataResponse<Data?,  
AFError>)
```

/// Event called when a `DataRequest` calls a `ResponseSerializer` and creates a generic `DataResponse<Value, AFError>`.

```
func request<Value>(_ request: DataRequest, didParseResponse response:  
DataResponse<Value, AFError>)
```

// MARK: UploadRequest Events

/// Event called when an `UploadRequest` creates its `Uploadable` value, indicating the type of upload it represents.

```
func request(_ request: UploadRequest, didCreateUploadable uploadable:  
UploadRequest.Uploadable)
```

/// Event called when an `UploadRequest` failed to create its `Uploadable` value due to an error.

```
func request(_ request: UploadRequest, didFailToCreateUploadableWithError error: AFError)
```

/// Event called when an `UploadRequest` provides the `InputStream` from its `Uploadable` value. This only occurs if

/// the `InputStream` does not wrap a `Data` value or file `URL`.

```
func request(_ request: UploadRequest, didProvideInputStream stream: InputStream)
```

// MARK: DownloadRequest Events

/// Event called when a `DownloadRequest`'s `URLSessionDownloadTask` finishes and the temporary file has been moved.

```
func request(_ request: DownloadRequest, didFinishDownloadingUsing task: URLSessionTask, with result: Result<URL, AFError>)
```

/// Event called when a `DownloadRequest`'s `Destination` closure is called and creates the destination URL the

/// downloaded file will be moved to.

```
func request(_ request: DownloadRequest, didCreateDestinationURL url: URL)
```

/// Event called when a `DownloadRequest` calls a `Validation`.

```
func request(_ request: DownloadRequest,  
  
             didValidateRequest urlRequest: URLRequest?,  
  
             response: HTTPURLResponse,  
  
             fileURL: URL?,  
  
             withResult result: Request.ValidationResult)
```

/// Event called when a `DownloadRequest` creates a `DownloadResponse<URL?, AFError>` without calling a `ResponseSerializer`.

```
func request(_ request: DownloadRequest, didParseResponse response:  
DownloadResponse<URL?, AFError>)
```

/// Event called when a `DownloadRequest` calls a `DownloadResponseSerializer` and creates a generic `DownloadResponse<Value, AFError>`

```
func request<Value>(_ request: DownloadRequest, didParseResponse response:  
DownloadResponse<Value, AFError>)
```

```
}
```

```
extension EventMonitor {
```

```
    /// The default queue on which `CompositeEventMonitor`'s will call the `EventMonitor`  
    methods. `.main` by default.
```

```
    public var queue: DispatchQueue { return .main }
```

```
    // MARK: Default Implementations
```

```
    public func urlSession(_ session: URLSession, didBecomeInvalidWithError error:  
Error?) {}
```

```
    public func urlSession(_ session: URLSession,
```

```
        task: URLSessionTask,
```

```
        didReceive challenge: URLAuthenticationChallenge) {}
```

```
    public func urlSession(_ session: URLSession,
```

```
        task: URLSessionTask,
```

```
        didSendBodyData bytesSent: Int64,
```

```
        totalBytesSent: Int64,
```

```

        totalBytesExpectedToSend: Int64) {}

    public func urlSession(_ session: URLSession, taskNeedsNewBodyStream task:
URLSessionTask) {}

    public func urlSession(_ session: URLSession,

        task: URLSessionTask,

        willPerformHTTPRedirection response: HTTPURLResponse,

        newRequest request: URLRequest) {}

    public func urlSession(_ session: URLSession,

        task: URLSessionTask,

        didFinishCollecting metrics: URLSessionTaskMetrics) {}

    public func urlSession(_ session: URLSession, task: URLSessionTask,
didCompleteWithError error: Error?) {}

    public func urlSession(_ session: URLSession, taskIsWaitingForConnectivity task:
URLSessionTask) {}

    public func urlSession(_ session: URLSession, dataTask: URLSessionDataTask,
didReceive data: Data) {}

    public func urlSession(_ session: URLSession,

        dataTask: URLSessionDataTask,

        willCacheResponse proposedResponse: CachedURLResponse) {}

```

```
public func urlSession(_ session: URLSession,  
  
    downloadTask: URLSessionDownloadTask,  
  
    didResumeAtOffset fileOffset: Int64,  
  
    expectedTotalBytes: Int64) {}
```

```
public func urlSession(_ session: URLSession,  
  
    downloadTask: URLSessionDownloadTask,  
  
    didWriteData bytesWritten: Int64,  
  
    totalBytesWritten: Int64,  
  
    totalBytesExpectedToWrite: Int64) {}
```

```
public func urlSession(_ session: URLSession,  
  
    downloadTask: URLSessionDownloadTask,  
  
    didFinishDownloadingTo location: URL) {}
```

```
public func request(_ request: Request, didCreateInitialURLRequest urlRequest:  
URLRequest) {}
```

```
public func request(_ request: Request, didFailToCreateURLRequestWithError error:  
NSError) {}
```

```
public func request(_ request: Request,  
  
    didAdaptInitialRequest initialRequest: URLRequest,
```

```

        to adaptedRequest: URLRequest) {}

public func request(_ request: Request,

                    didFailToAdaptURLRequest initialRequest: URLRequest,

                    withError error: AFError) {}

public func request(_ request: Request, didCreateURLRequest urlRequest: URLRequest)
{}

public func request(_ request: Request, didCreateTask task: URLSessionTask) {}

public func request(_ request: Request, didGatherMetrics metrics:
URLSessionTaskMetrics) {}

public func request(_ request: Request, didFailTask task: URLSessionTask,
earlyWithError error: AFError) {}

public func request(_ request: Request, didCompleteTask task: URLSessionTask, with
error: AFError?) {}

public func requestIsRetrying(_ request: Request) {}

public func requestDidFinish(_ request: Request) {}

public func requestDidResume(_ request: Request) {}

public func request(_ request: Request, didResumeTask task: URLSessionTask) {}

public func requestDidSuspend(_ request: Request) {}

public func request(_ request: Request, didSuspendTask task: URLSessionTask) {}

```



```

public func requestDidCancel(_ request: Request) {}

public func request(_ request: Request, didCancelTask task: URLSessionTask) {}

public func request(_ request: DataRequest,

                    didValidateRequest urlRequest: URLRequest?,

                    response: HTTPURLResponse,

                    data: Data?,

                    withResult result: Request.ValidationResult) {}

public func request(_ request: DataRequest, didParseResponse response:
DataResponse<Data?, AFError>) {}

public func request<Value>(_ request: DataRequest, didParseResponse response:
DataResponse<Value, AFError>) {}

public func request(_ request: UploadRequest, didCreateUploadable uploadable:
UploadRequest.Uploadable) {}

public func request(_ request: UploadRequest, didFailToCreateUploadableWithError
error: AFError) {}

public func request(_ request: UploadRequest, didProvideInputStream stream:
InputStream) {}

public func request(_ request: DownloadRequest, didFinishDownloadingUsing task:
URLSessionTask, with result: Result<URL, AFError>) {}

public func request(_ request: DownloadRequest, didCreateDestinationURL url: URL)
{}

```

```

public func request(_ request: DownloadRequest,

                    didValidateRequest urlRequest: URLRequest?,

                    response: HTTPURLResponse,

                    fileURL: URL?,

                    withResult result: Request.ValidationResult) {}

public func request(_ request: DownloadRequest, didParseResponse response:
DownloadResponse<URL?, AFError>) {}

public func request<Value>(_ request: DownloadRequest, didParseResponse response:
DownloadResponse<Value, AFError>) {}

}

```

/// An `EventMonitor` which can contain multiple `EventMonitor`s and calls their methods on their queues.

```

public final class CompositeEventMonitor: EventMonitor {

    public let queue = DispatchQueue(label: "org.alamofire.compositeEventMonitor",
qos: .background)

    let monitors: [EventMonitor]

```

```

init(monitors: [EventMonitor]) {

    self.monitors = monitors

}

```

```

func performEvent(_ event: @escaping (EventMonitor) -> Void) {

    queue.async {

        for monitor in self.monitors {

            monitor.queue.async { event(monitor) }

        }

    }

}

```

```

public func urlSession(_ session: URLSession, didBecomeInvalidWithError error:
Error?) {

    performEvent { $0.urlSession(session, didBecomeInvalidWithError: error) }

}

```

```

public func urlSession(_ session: URLSession,

```

```

        task: URLSessionTask,

        didReceive challenge: URLAuthenticationChallenge) {

performEvent { $0.urlSession(session, task: task, didReceive: challenge) }

}

```

```

public func urlSession(_ session: URLSession,

        task: URLSessionTask,

        didSendBodyData bytesSent: Int64,

        totalBytesSent: Int64,

        totalBytesExpectedToSend: Int64) {

performEvent {

    $0.urlSession(session,

        task: task,

        didSendBodyData: bytesSent,

        totalBytesSent: totalBytesSent,

        totalBytesExpectedToSend: totalBytesExpectedToSend)

}

```

```
}
```

```
public func urlSession(_ session: URLSession, taskNeedsNewBodyStream task:  
URLSessionTask) {
```

```
    performEvent {
```

```
        $0.urlSession(session, taskNeedsNewBodyStream: task)
```

```
    }
```

```
}
```

```
public func urlSession(_ session: URLSession,
```

```
    task: URLSessionTask,
```

```
    willPerformHTTPRedirection response: HTTPURLResponse,
```

```
    newRequest request: URLRequest) {
```

```
    performEvent {
```

```
        $0.urlSession(session,
```

```
            task: task,
```

```
            willPerformHTTPRedirection: response,
```

```
            newRequest: request)
```

```
}
```

```
}
```

```
public func urlSession(_ session: URLSession, task: URLSessionTask,  
didFinishCollecting metrics: URLSessionTaskMetrics) {
```

```
    performEvent { $0.urlSession(session, task: task, didFinishCollecting: metrics) }
```

```
}
```

```
public func urlSession(_ session: URLSession, task: URLSessionTask,  
didCompleteWithError error: Error?) {
```

```
    performEvent { $0.urlSession(session, task: task, didCompleteWithError: error) }
```

```
}
```

```
@available(macOS 10.13, iOS 11.0, tvOS 11.0, watchOS 4.0, *)
```

```
public func urlSession(_ session: URLSession, taskIsWaitingForConnectivity task:  
URLSessionTask) {
```

```
    performEvent { $0.urlSession(session, taskIsWaitingForConnectivity: task) }
```

```
}
```

```
public func urlSession(_ session: URLSession, dataTask: URLSessionDataTask,
didReceive data: Data) {
```

```
    performEvent { $0.urlSession(session, dataTask: dataTask, didReceive: data) }

}
```

```
public func urlSession(_ session: URLSession,

                        dataTask: URLSessionDataTask,

                        willCacheResponse proposedResponse: CachedURLResponse) {

    performEvent { $0.urlSession(session, dataTask: dataTask, willCacheResponse:
proposedResponse) }

}
```

```
public func urlSession(_ session: URLSession,

                        downloadTask: URLSessionDownloadTask,

                        didResumeAtOffset fileOffset: Int64,

                        expectedTotalBytes: Int64) {

    performEvent {

        $0.urlSession(session,
```

```

        downloadTask: downloadTask,

        didResumeAtOffset: fileOffset,

        expectedTotalBytes: expectedTotalBytes)

    }

}

```

```

public func urlSession(_ session: URLSession,

        downloadTask: URLSessionDownloadTask,

        didWriteData bytesWritten: Int64,

        totalBytesWritten: Int64,

        totalBytesExpectedToWrite: Int64) {

performEvent {

    $0.urlSession(session,

        downloadTask: downloadTask,

        didWriteData: bytesWritten,

        totalBytesWritten: totalBytesWritten,

        totalBytesExpectedToWrite: totalBytesExpectedToWrite)

```



```
}
```

```
}
```

```
public func urlSession(_ session: URLSession,
```

```
    downloadTask: URLSessionDownloadTask,
```

```
    didFinishDownloadingTo location: URL) {
```

```
    performEvent { $0.urlSession(session,    downloadTask:    downloadTask,  
didFinishDownloadingTo: location) }
```

```
}
```

```
public func request(_ request: Request, didCreateInitialURLRequest urlRequest:  
URLRequest) {
```

```
    performEvent { $0.request(request, didCreateInitialURLRequest: urlRequest) }
```

```
}
```

```
public func request(_ request: Request, didFailToCreateURLRequestWithError error:  
NSError) {
```

```
    performEvent { $0.request(request, didFailToCreateURLRequestWithError: error) }
```

```
}
```

```
public func request(_ request: Request, didAdaptInitialRequest initialRequest:
URLRequest, to adaptedRequest: URLRequest) {
```

```
    performEvent { $0.request(request, didAdaptInitialRequest: initialRequest, to:
adaptedRequest) }
```

```
}
```

```
public func request(_ request: Request, didFailToAdaptURLRequest initialRequest:
URLRequest, withError error: NSError) {
```

```
    performEvent { $0.request(request, didFailToAdaptURLRequest: initialRequest,
withError: error) }
```

```
}
```

```
public func request(_ request: Request, didCreateURLRequest urlRequest: URLRequest)
{
```

```
    performEvent { $0.request(request, didCreateURLRequest: urlRequest) }
```

```
}
```

```
public func request(_ request: Request, didCreateTask task: URLSessionTask) {
```

```
    performEvent { $0.request(request, didCreateTask: task) }
```

```
}
```

```
public func request(_ request: Request, didGatherMetrics metrics:
NSURLSessionTaskMetrics) {
```

```
    performEvent { $0.request(request, didGatherMetrics: metrics) }
```

```
}
```

```
public func request(_ request: Request, didFailTask task: URLSessionTask,
earlyWithError error: NSError) {
```

```
    performEvent { $0.request(request, didFailTask: task, earlyWithError: error) }
```

```
}
```

```
public func request(_ request: Request, didCompleteTask task: URLSessionTask, with
error: NSError?) {
```

```
    performEvent { $0.request(request, didCompleteTask: task, with: error) }
```

```
}
```

```
public func requestIsRetrying(_ request: Request) {
```

```
    performEvent { $0.requestIsRetrying(request) }
```

```
}
```

```
public func requestDidFinish(_ request: Request) {
```

```
    performEvent { $0.requestDidFinish(request) }
```

```
}
```

```
public func requestDidResume(_ request: Request) {
```

```
    performEvent { $0.requestDidResume(request) }
```

```
}
```

```
public func request(_ request: Request, didResumeTask task: URLSessionTask) {
```

```
    performEvent { $0.request(request, didResumeTask: task) }
```

```
}
```

```
public func requestDidSuspend(_ request: Request) {
```

```
    performEvent { $0.requestDidSuspend(request) }
```

```
}
```

```

public func request(_ request: Request, didSuspendTask task: URLSessionTask) {

    performEvent { $0.request(request, didSuspendTask: task) }

}

```

```

public func requestDidCancel(_ request: Request) {

    performEvent { $0.requestDidCancel(request) }

}

```

```

public func request(_ request: Request, didCancelTask task: URLSessionTask) {

    performEvent { $0.request(request, didCancelTask: task) }

}

```

```

public func request(_ request: DataRequest,

    didValidateRequest urlRequest: URLRequest?,

    response: HTTPURLResponse,

    data: Data?,

```

```

        withResult result: Request.ValidationResult) {

performEvent { $0.request(request,

        didValidateRequest: urlRequest,

        response: response,

        data: data,

        withResult: result)

    }

}

```

```

public func request(_ request: DataRequest, didParseResponse response:
DataResponse<Data?, AFError>) {

    performEvent { $0.request(request, didParseResponse: response) }

}

```

```

public func request<Value>(_ request: DataRequest, didParseResponse response:
DataResponse<Value, AFError>) {

    performEvent { $0.request(request, didParseResponse: response) }

}

```

```

    public func request(_ request: UploadRequest, didCreateUploadable uploadable:
UploadRequest.Uploadable) {

        performEvent { $0.request(request, didCreateUploadable: uploadable) }

    }

```

```

    public func request(_ request: UploadRequest, didFailToCreateUploadableWithError
error: AFError) {

        performEvent { $0.request(request, didFailToCreateUploadableWithError: error) }

    }

```

```

    public func request(_ request: UploadRequest, didProvideInputStream stream:
InputStream) {

        performEvent { $0.request(request, didProvideInputStream: stream) }

    }

```

```

    public func request(_ request: DownloadRequest, didFinishDownloadingUsing task:
URLSessionTask, with result: Result<URL, AFError>) {

        performEvent { $0.request(request, didFinishDownloadingUsing: task, with: result) }

```

```
}
```

```
public func request(_ request: DownloadRequest, didCreateDestinationURL url: URL)
{
    performEvent { $0.request(request, didCreateDestinationURL: url) }
}
```

```
public func request(_ request: DownloadRequest,
                    didValidateRequest urlRequest: URLRequest?,
                    response: HTTPURLResponse,
                    fileURL: URL?,
                    withResult result: Request.ValidationResult) {
    performEvent { $0.request(request,
                              didValidateRequest: urlRequest,
                              response: response,
                              fileURL: fileURL,
                              withResult: result) }
}
```



```

    public func request(_ request: DownloadRequest, didParseResponse response:
DownloadResponse<URL?, AFError>) {

        performEvent { $0.request(request, didParseResponse: response) }

    }

```

```

    public func request<Value>(_ request: DownloadRequest, didParseResponse response:
DownloadResponse<Value, AFError>) {

        performEvent { $0.request(request, didParseResponse: response) }

    }

}

```

/// `EventManager` that allows optional closures to be set to receive events.

```

open class ClosureEventManager: EventMonitor {

```

```

    /// Closure called on the `urlSession(_:didBecomeInvalidWithError)` event.

```

```

    open var sessionDidBecomeInvalidWithError: ((URLSession, Error?) -> Void)?

```

```

    /// Closure called on the `urlSession(_:task:didReceive:completionHandler)` .

```

```
open var taskDidReceiveChallenge: ((URLSession, URLSessionTask,
URLAuthenticationChallenge) -> Void)?
```

```
/// Closure that receives
`urlSession(_:task:didSendBodyData:totalBytesSent:totalBytesExpectedToSend:)` event.
```

```
open var taskDidSendBodyData: ((URLSession, URLSessionTask, Int64, Int64, Int64)
-> Void)?
```

```
/// Closure called on the `urlSession(_:task:needNewBodyStream:)` event.
```

```
open var taskNeedNewBodyStream: ((URLSession, URLSessionTask) -> Void)?
```

```
/// Closure called on the
`urlSession(_:task:willPerformHTTPRedirection:newRequest:completionHandler:)` event.
```

```
open var taskWillPerformHTTPRedirection: ((URLSession, URLSessionTask,
HTTPURLResponse, URLRequest) -> Void)?
```

```
/// Closure called on the `urlSession(_:task:didFinishCollecting:)` event.
```

```
open var taskDidFinishCollectingMetrics: ((URLSession, URLSessionTask,
URLSessionTaskMetrics) -> Void)?
```

/// Closure called on the `urlSession(\_:task:didCompleteWithError:)` event.

open var taskDidComplete: ((URLSession, URLSessionTask, Error?) -> Void)?

/// Closure called on the `urlSession(\_:taskIsWaitingForConnectivity:)` event.

open var taskIsWaitingForConnectivity: ((URLSession, URLSessionTask) -> Void)?

/// Closure that receives the `urlSession(\_:dataTask:didReceive:)` event.

open var dataTaskDidReceiveData: ((URLSession, URLSessionDataTask, Data) -> Void)?

/// Closure called on the
`urlSession(\_:dataTask:willCacheResponse:completionHandler:)` event.

open var dataTaskWillCacheResponse: ((URLSession, URLSessionDataTask, CachedURLResponse) -> Void)?

/// Closure called on the `urlSession(\_:downloadTask:didFinishDownloadingTo:)` event.

open var downloadTaskDidFinishDownloadingToURL: ((URLSession, URLSessionDownloadTask, URL) -> Void)?

```
/// Closure called on the  
`urlSession(_:downloadTask:didWriteData:totalBytesWritten:totalBytesExpectedToWrite)`
```

```
/// event.
```

```
open var downloadTaskDidWriteData: ((URLSession, URLSessionDownloadTask,  
Int64, Int64, Int64) -> Void)?
```

```
/// Closure called on the  
`urlSession(_:downloadTask:didResumeAtOffset:expectedTotalBytes)` event.
```

```
open var downloadTaskDidResumeAtOffset: ((URLSession,  
URLSessionDownloadTask, Int64, Int64) -> Void)?
```

```
// MARK: - Request Events
```

```
/// Closure called on the `request(_:didCreateInitialURLRequest)` event.
```

```
open var requestDidCreateInitialURLRequest: ((Request, URLRequest) -> Void)?
```

```
/// Closure called on the `request(_:didFailToCreateURLRequestWithError)` event.
```

```
open var requestDidFailToCreateURLRequestWithError: ((Request, NSError) -> Void)?
```

/// Closure called on the `request(\_:didAdaptInitialRequest:to:)` event.

open var requestDidAdaptInitialRequestToAdaptedRequest: ((Request, URLRequest, URLRequest) -> Void)?

/// Closure called on the `request(\_:didFailToAdaptURLRequest:withError:)` event.

open var requestDidFailToAdaptURLRequestWithError: ((Request, URLRequest, NSError) -> Void)?

/// Closure called on the `request(\_:didCreateURLRequest:)` event.

open var requestDidCreateURLRequest: ((Request, URLRequest) -> Void)?

/// Closure called on the `request(\_:didCreateTask:)` event.

open var requestDidCreateTask: ((Request, URLSessionTask) -> Void)?

/// Closure called on the `request(\_:didGatherMetrics:)` event.

open var requestDidGatherMetrics: ((Request, URLSessionTaskMetrics) -> Void)?

/// Closure called on the `request(\_:didFailTask:earlyWithError:)` event.

```
open var requestDidFailTaskEarlyWithError: ((Request, URLSessionTask, AFError) -> Void)?
```

```
/// Closure called on the `request(_:didCompleteTask:with:)` event.
```

```
open var requestDidCompleteTaskWithError: ((Request, URLSessionTask, AFError?) -> Void)?
```

```
/// Closure called on the `requestIsRetrying(_:)` event.
```

```
open var requestIsRetrying: ((Request) -> Void)?
```

```
/// Closure called on the `requestDidFinish(_:)` event.
```

```
open var requestDidFinish: ((Request) -> Void)?
```

```
/// Closure called on the `requestDidResume(_:)` event.
```

```
open var requestDidResume: ((Request) -> Void)?
```

```
/// Closure called on the `request(_:didResumeTask:)` event.
```

```
open var requestDidResumeTask: ((Request, URLSessionTask) -> Void)?
```

/// Closure called on the `requestDidSuspend(\_:)` event.

open var requestDidSuspend: ((Request) -> Void)?

/// Closure called on the `request(\_:didSuspendTask:)` event.

open var requestDidSuspendTask: ((Request, URLSessionTask) -> Void)?

/// Closure called on the `requestDidCancel(\_:)` event.

open var requestDidCancel: ((Request) -> Void)?

/// Closure called on the `request(\_:didCancelTask:)` event.

open var requestDidCancelTask: ((Request, URLSessionTask) -> Void)?

/// Closure called on the `request(\_:didValidateRequest:response:data:withResult:)` event.

open var requestDidValidateRequestResponseDataWithResult: ((DataRequest, URLRequest?, HTTPURLResponse, Data?, Request.ValidationResult) -> Void)?

/// Closure called on the `request(\_:didParseResponse:)` event.

open var requestDidParseResponse: ((DataRequest, DataResponse<Data?, AFError>) -> Void)?

/// Closure called on the `request(\_:didCreateUploadable:)` event.

open var requestDidCreateUploadable: ((UploadRequest, UploadRequest.Uploadable) -> Void)?

/// Closure called on the `request(\_:didFailToCreateUploadableWithError:)` event.

open var requestDidFailToCreateUploadableWithError: ((UploadRequest, AFError) -> Void)?

/// Closure called on the `request(\_:didProvideInputStream:)` event.

open var requestDidProvideInputStream: ((UploadRequest, InputStream) -> Void)?

/// Closure called on the `request(\_:didFinishDownloadingUsing:with:)` event.

open var requestDidFinishDownloadingUsingTaskWithResult: ((DownloadRequest, URLSessionTask, Result<URL, AFError>) -> Void)?


```
/// Closure called on the `request(_:didCreateDestinationURL:)` event.
```

```
open var requestDidCreateDestinationURL: ((DownloadRequest, URL) -> Void)?
```

```
/// Closure called on the  
`request(_:didValidateRequest:response:temporaryURL:destinationURL:withResult:)`  
event.
```

```
open var requestDidValidateRequestResponseFileURLWithResult:  
((DownloadRequest, URLRequest?, HTTPURLResponse, URL?,  
Request.ValidationResult) -> Void)?
```

```
/// Closure called on the `request(_:didParseResponse:)` event.
```

```
open var requestDidParseDownloadResponse: ((DownloadRequest,  
DownloadResponse<URL?, NSError>) -> Void)?
```

```
public let queue: DispatchQueue
```

```
/// Creates an instance using the provided queue.
```

```
///
```

```
/// - Parameter queue: `DispatchQueue` on which events will fired. `.main` by default.
```

```
public init(queue: DispatchQueue = .main) {
```

```

        self.queue = queue

    }

    open func urlSession(_ session: URLSession, didBecomeInvalidWithError error: Error?)
    {

        sessionDidBecomeInvalidWithError?(session, error)

    }

    open func urlSession(_ session: URLSession, task: URLSessionTask, didReceive
challenge: URLAuthenticationChallenge) {

        taskDidReceiveChallenge?(session, task, challenge)

    }

    open func urlSession(_ session: URLSession,

        task: URLSessionTask,

        didSendBodyData bytesSent: Int64,

        totalBytesSent: Int64,

        totalBytesExpectedToSend: Int64) {

```

```

        taskDidSendBodyData?(session, task, bytesSent, totalBytesSent,
totalBytesExpectedToSend)

    }

```

```

    open func urlSession(_ session: URLSession, taskNeedsNewBodyStream task:
URLSessionTask) {

        taskNeedNewBodyStream?(session, task)

    }

```

```

    open func urlSession(_ session: URLSession,

        task: URLSessionTask,

        willPerformHTTPRedirection response: HTTPURLResponse,

        newRequest request: URLRequest) {

        taskWillPerformHTTPRedirection?(session, task, response, request)

    }

```

```

    open func urlSession(_ session: URLSession, task: URLSessionTask,
didFinishCollecting metrics: URLSessionTaskMetrics) {

        taskDidFinishCollectingMetrics?(session, task, metrics)

```

```
}
```

```
open func urlSession(_ session: URLSession, task: URLSessionTask,  
didCompleteWithError error: Error?) {
```

```
    taskDidComplete?(session, task, error)
```

```
}
```

```
open func urlSession(_ session: URLSession, taskIsWaitingForConnectivity task:  
URLSessionTask) {
```

```
    taskIsWaitingForConnectivity?(session, task)
```

```
}
```

```
open func urlSession(_ session: URLSession, dataTask: URLSessionDataTask,  
didReceive data: Data) {
```

```
    dataTaskDidReceiveData?(session, dataTask, data)
```

```
}
```

```
open func urlSession(_ session: URLSession, dataTask: URLSessionDataTask,  
willCacheResponse proposedResponse: CachedURLResponse) {
```

```

dataTaskWillCacheResponse?(session, dataTask, proposedResponse)

}

```

```

open func urlSession(_ session: URLSession,

                      downloadTask: URLSessionDownloadTask,

                      didResumeAtOffset fileOffset: Int64,

                      expectedTotalBytes: Int64) {

    downloadTaskDidResumeAtOffset?(session, downloadTask, fileOffset,
expectedTotalBytes)

}

```

```

open func urlSession(_ session: URLSession,

                      downloadTask: URLSessionDownloadTask,

                      didWriteData bytesWritten: Int64,

                      totalBytesWritten: Int64,

                      totalBytesExpectedToWrite: Int64) {

    downloadTaskDidWriteData?(session, downloadTask, bytesWritten,
totalBytesWritten, totalBytesExpectedToWrite)

}

```

```
}
```

```
open func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didFinishDownloadingTo location: URL) {
```

```
    downloadTaskDidFinishDownloadingToURL?(session, downloadTask, location)
```

```
}
```

```
// MARK: Request Events
```

```
open func request(_ request: Request, didCreateInitialURLRequest urlRequest: URLRequest) {
```

```
    requestDidCreateInitialURLRequest?(request, urlRequest)
```

```
}
```

```
open func request(_ request: Request, didFailToCreateURLRequestWithError error: NSError) {
```

```
    requestDidFailToCreateURLRequestWithError?(request, error)
```

```
}
```

```

    open func request(_ request: Request, didAdaptInitialRequest initialRequest:
URLRequest, to adaptedRequest: URLRequest) {

        requestDidAdaptInitialRequestToAdaptedRequest?(request, initialRequest,
adaptedRequest)

    }

```

```

    open func request(_ request: Request, didFailToAdaptURLRequest initialRequest:
URLRequest, withError error: NSError) {

        requestDidFailToAdaptURLRequestWithError?(request, initialRequest, error)

    }

```

```

    open func request(_ request: Request, didCreateURLRequest urlRequest: URLRequest)
{

        requestDidCreateURLRequest?(request, urlRequest)

    }

```

```

    open func request(_ request: Request, didCreateTask task: URLSessionTask) {

        requestDidCreateTask?(request, task)

    }

```

```

    open func request(_ request: Request, didGatherMetrics metrics:
URLSessionTaskMetrics) {

        requestDidGatherMetrics?(request, metrics)

    }

```

```

    open func request(_ request: Request, didFailTask task: URLSessionTask,
earlyWithError error: AFError) {

        requestDidFailTaskEarlyWithError?(request, task, error)

    }

```

```

    open func request(_ request: Request, didCompleteTask task: URLSessionTask, with
error: AFError?) {

        requestDidCompleteTaskWithError?(request, task, error)

    }

```

```

    open func requestIsRetrying(_ request: Request) {

        requestIsRetrying?(request)

    }

```



```
open func requestDidFinish(_ request: Request) {  
  
    requestDidFinish?(request)  
  
}
```

```
open func requestDidResume(_ request: Request) {  
  
    requestDidResume?(request)  
  
}
```

```
public func request(_ request: Request, didResumeTask task: URLSessionTask) {  
  
    requestDidResumeTask?(request, task)  
  
}
```

```
open func requestDidSuspend(_ request: Request) {  
  
    requestDidSuspend?(request)  
  
}
```

```
public func request(_ request: Request, didSuspendTask task: URLSessionTask) {  
  
    requestDidSuspendTask?(request, task)  
  
}
```

```
open func requestDidCancel(_ request: Request) {  
  
    requestDidCancel?(request)  
  
}
```

```
public func request(_ request: Request, didCancelTask task: URLSessionTask) {  
  
    requestDidCancelTask?(request, task)  
  
}
```

```
open func request(_ request: DataRequest,  
  
    didValidateRequest urlRequest: URLRequest?,  
  
    response: HTTPURLResponse,  
  
    data: Data?,  
  
    withResult result: Request.ValidationResult) {
```

```
requestDidValidateRequestResponseDataWithResult?(request, urlRequest, response,  
data, result)
```

```
}
```

```
open func request(_ request: DataRequest, didParseResponse response:  
DataResponse<Data?, AFError>) {
```

```
requestDidParseResponse?(request, response)
```

```
}
```

```
open func request(_ request: UploadRequest, didCreateUploadable uploadable:  
UploadRequest.Uploadable) {
```

```
requestDidCreateUploadable?(request, uploadable)
```

```
}
```

```
open func request(_ request: UploadRequest, didFailToCreateUploadableWithError  
error: AFError) {
```

```
requestDidFailToCreateUploadableWithError?(request, error)
```

```
}
```

```

    open func request(_ request: UploadRequest, didProvideInputStream stream:
InputStream) {

        requestDidProvideInputStream?(request, stream)

    }

```

```

    open func request(_ request: DownloadRequest, didFinishDownloadingUsing task:
URLSessionTask, with result: Result<URL, AFError>) {

        requestDidFinishDownloadingUsingTaskWithResult?(request, task, result)

    }

```

```

    open func request(_ request: DownloadRequest, didCreateDestinationURL url: URL) {

        requestDidCreateDestinationURL?(request, url)

    }

```

```

    open func request(_ request: DownloadRequest,

        didValidateRequest urlRequest: URLRequest?,

        response: HTTPURLResponse,

        fileURL: URL?,

```

```

        withResult result: Request.ValidationResult) {

    requestDidValidateRequestResponseFileURLWithResult?(request,

                                urlRequest,

                                response,

                                fileURL,

                                result)

}

```

```

    open func request(_ request: DownloadRequest, didParseResponse response:
DownloadResponse<URL?, AFError>) {

        requestDidParseDownloadResponse?(request, response)

    }

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `EventTriggerCreator` is a superclass that creates Characteristic and Location triggers.

*/

import HomeKit

/**

A superclass for event trigger creators.

These classes manage the state for characteristic trigger conditions.

*/

class EventTriggerCreator: TriggerCreator, CharacteristicCellDelegate {

// MARK: Properties

/// A mapping of `HMCharacteristic`s to their values.

private let conditionValueMap = NSMapTable<HMCharacteristic,
CellValueType>.strongToStrongObjects()

```

private var eventTrigger: HMEventTrigger? {

    return trigger as? HMEventTrigger

}

/**

    An array of top-level `NSPredicate` objects.

    Currently, HMCatalog only supports top-level `NSPredicate`s

    which have type `AndPredicateType`.

*/

var originalConditions: [NSPredicate] {

    if let compoundPredicate = eventTrigger?.predicate as? NSCompoundPredicate,

        let subpredicates = compoundPredicate.subpredicates as? [NSPredicate] {

        return subpredicates

    }

    return []

```

```
}
```

```
/// An array of new conditions which will be written when the trigger is saved.
```

```
lazy var conditions: [NSPredicate] = self.originalConditions
```

```
/**
```

```
    Adds a predicate to the pending conditions.
```

```
    - parameter predicate: The new `NSPredicate` to add.
```

```
*/
```

```
func addCondition(_ predicate: NSPredicate) {
```

```
    conditions.append(predicate)
```

```
}
```

```
/**
```

```
    Removes a predicate from the pending conditions.
```


- parameter predicate: The `NSPredicate` to remove.

*/

```
func removeCondition(_ predicate: NSPredicate) {
```

```
    if let index = conditions.firstIndex(of: predicate) {
```

```
        conditions.remove(at: index)
```

```
    }
```

```
}
```

/**

- returns: The new `NSCompoundPredicate`, generated from

the pending conditions.

*/

```
func newPredicate() -> NSPredicate {
```

```
    return NSCompoundPredicate(type: .and, subpredicates: conditions)
```

```
}
```

/// Handles the value update and stores the value in the condition map.

```

func characteristicCell(_ cell: CharacteristicCell, didUpdateValue value:
CellValueType, forCharacteristic characteristic: HMCharacteristic, immediate: Bool) {

    conditionValueMap.setObject(value, forKey: characteristic)

}

```

```

/**

```

Tries to use the value from the condition-value map, but falls back

to reading the characteristic's value from HomeKit.

```

*/

```

```

func characteristicCell(_ cell: CharacteristicCell, readInitialValueForCharacteristic
characteristic: HMCharacteristic, completion: @escaping (CellValueType?, Error?) ->
Void) {

```

```

    if let value = conditionValueMap.object(forKey: characteristic) {

```

```

        completion(value, nil)

```

```

        return

```

```

    }

```

```

    characteristic.readValue { error in

```

```

        /*

```

The user may have updated the cell value while the

read was happening. We check the map one more time.

```
*/
```

```
if let value = self.conditionValueMap.object(forKey: characteristic) {
```

```
    completion(value, nil)
```

```
}
```

```
else {
```

```
    completion(characteristic.value as? CellValueType, error)
```

```
}
```

```
}
```

```
}
```

```
// MARK: Helper Methods
```

```
/**
```

Updates the predicates and saves the new, generated

predicate to the event trigger.

```
*/
```

```
func savePredicate() {  
  
    updatePredicates()  
  
    saveTriggerGroup.enter()  
  
    eventTrigger?.updatePredicate(newPredicate()) { error in  
  
        if let error = error {  
  
            self.errors.append(error)  
  
        }  
  
        self.saveTriggerGroup.leave()  
  
    }  
  
}
```

/// Generates predicates from the characteristic-value map and adds them to the pending conditions.

```
func updatePredicates() {  
  
    for (characteristic, value) in pairsFromMapTable(conditionValueMap) {  
  
        let predicate = HMEventTrigger.predicateForEvaluatingTrigger(characteristic,  
relatedBy: .equalTo, toValue: value)
```

```
addCondition(predicate)
```

```
}
```

```
conditionValueMap.removeAllObjects()
```

```
}
```

```
/**
```

- parameter table: The `NSMapTable` from which to generate the pairs.

- returns: Tuples representing `HMCharacteristic`s and their associated return trigger values.

```
*/
```

```
func pairsFromMapTable<Key, ValueType>(_ table: NSMapTable<Key, ValueType>)  
-> [(Key, ValueType)] {
```

```
    return table.keyEnumerator().allObjects.map { object in
```

```
        let characteristic = object as! Key
```

```
        let triggerValue = table.object(forKey: characteristic)
```

```
        return (characteristic, triggerValue!)
```

```
}  
  
}  
  
}  
  
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `EventTriggerViewController` is a superclass that helps users create Characteristic and Location triggers.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
/**
```

A superclass for all event-based view controllers.

It handles the process of creating and managing trigger conditions.

```
*/
```

```
class EventTriggerViewController: TriggerViewController {
```

```
    // MARK: Types
```

```
    struct Identifiers {
```

```
        static let addCell = "AddCell"
```

```
        static let conditionCell = "ConditionCell"
```

```
        static let showTimeConditionSegue = "Show Time Condition"
```

```
    }
```

```
    // MARK: Properties
```

```
    private var eventTriggerCreator: EventTriggerCreator {
```

```
        return triggerCreator as! EventTriggerCreator
```

```
    }
```

```

// MARK: View Methods

/// Registers table view for cells.

override func viewDidLoad() {

    super.viewDidLoad()

    tableView.register(UITableViewCell.self, forCellReuseIdentifier:Identifiers.addCell)

    tableView.register(ConditionCell.self,
forCellReuseIdentifier:Identifiers.conditionCell)

}

/// Hands off the trigger creator to the condition view controllers.

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

    switch segue.intendedDestinationViewController {

        case let timeVC as TimeConditionViewController:

            timeVC.triggerCreator = eventTriggerCreator

        case let characteristicEventVC as CharacteristicSelectionViewController:

            let characteristicTriggerCreator = triggerCreator as! EventTriggerCreator

```



```
characteristicEventVC.triggerCreator = characteristicTriggerCreator
```

```
default:
```

```
    break
```

```
}
```

```
}
```

```
// MARK: Table View Methods
```

```
/**
```

```
- returns: In the conditions section: the number of conditions, plus one
```

```
for the add row. Defaults to the super implementation.
```

```
*/
```

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section:  
Int) -> Int {
```

```
    switch sectionForIndex(section) {
```

```
        case .conditions?:
```

```
            // Add row.
```

```
return eventTriggerCreator.conditions.count + 1
```

```
case nil:
```

```
fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
default:
```

```
return super.tableView(tableView, numberOfRowsInSection: section)
```

```
}
```

```
}
```

```
/**
```

```
    Launches "Add Condition" if the 'add index path' is selected.
```

```
    Defaults to the super implementation.
```

```
*/
```

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
```

```
    tableView.deselectRow(at: indexPath, animated: true)
```

```
    switch sectionForIndex(indexPath.section) {
```

```
case .conditions?:
```

```
    if indexPathIsAdd(indexPath) {
```

```
        addCondition()
```

```
    }
```

```
case nil:
```

```
    fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
default:
```

```
    super.tableView(tableView, didSelectRowAt: indexPath)
```

```
}
```

```
}
```

```
/**
```

Switches to select the correct type of cell for the section.

Defaults to the super implementation.

```
*/
```

```

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

        if indexPathIsAdd(indexPath) {

            return self.tableView(tableView, addCellForRowAtIndexPath: indexPath)

        }

        switch sectionForIndex(indexPath.section) {

            case .conditions?:

                return self.tableView(tableView, conditionCellForRowAtIndexPath: indexPath)

            case nil:

                fatalError("Unexpected `TriggerTableViewSection` raw value.")

            default:

                return super.tableView(tableView, cellForRowAt: indexPath)

        }

    }

```

```
/**
```

The conditions can be removed, the 'add index path' cannot.

For all others, default to super implementation.

```
*/
```

```
override func tableView(_ tableView: UITableView, canEditRowAt indexPath:
IndexPath) -> Bool {
```

```
    if indexPathIsAdd(indexPath) {
```

```
        return false
```

```
    }
```

```
    switch sectionForIndex(indexPath.section) {
```

```
        case .conditions?:
```

```
            return true
```

```
        case nil:
```

```
            fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
        default:
```

```

        return false
    }
}

/// Remove the selected condition from the trigger creator.

override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {

    if editingStyle == .delete {

        let predicate = eventTriggerCreator.conditions[indexPath.row]

        eventTriggerCreator.removeCondition(predicate)

        tableView.deleteRows(at: [indexPath], with: .automatic)

    }

}

/// - returns: An 'add cell' with 'Add Condition' text.

func tableView(_ tableView: UITableView, addCellForRowAtIndexPath indexPath:
IndexPath) -> UITableViewCell {

    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.addCell, for:
indexPath)

```

```

let cellText: String

switch sectionForIndex(indexPath.section) {

    case .conditions?:

        cellText = NSLocalizedString("Add Condition...", comment: "Add Condition")


    case nil:

        fatalError("Unexpected `TriggerTableViewSection` raw value.")


    default:

        cellText = NSLocalizedString("Add...", comment: "Add")

}

cell.textLabel?.text = cellText

cell.textLabel?.textColor = .editableBlue


return cell

}

```

```
/// - returns: A localized description of a trigger. Falls back to super implementation.
```

```
override func tableView(_ tableView: UITableView, titleForFooterInSection section:  
Int) -> String? {
```

```
    switch sectionForIndex(section) {
```

```
        case .conditions?:
```

```
            return NSLocalizedString("When a trigger is activated by an event, it checks  
these conditions. If all of them are true, it will set its scenes.", comment: "Trigger  
Conditions Description")
```

```
        case nil:
```

```
            fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
        default:
```

```
            return super.tableView(tableView, titleForFooterInSection: section)
```

```
    }
```

```
}
```

```
// MARK: Helper Methods
```


/// - returns: A 'condition cell', which displays information about the condition.

```
private func tableView(_ tableView: UITableView, conditionCellForRowAtIndexPath
indexPath: IndexPath) -> UITableViewCell {
```

```
    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.conditionCell)
    as! ConditionCell
```

```
    let condition = eventTriggerCreator.conditions[indexPath.row]
```

```
    switch condition.homeKitConditionType {
```

```
        case .characteristic(let characteristic, let value):
```

```
            cell.setCharacteristic(characteristic, targetValue: value)
```

```
        case .exactTime(let order, let dateComponents):
```

```
            cell.setOrder(order, dateComponents: dateComponents)
```

```
        case .sunTime(let order, let sunState):
```

```
            cell.setOrder(order, sunState: sunState)
```

```

        case .unknown:

            cell.setUnknown()

        }

        return cell

    }

    /// Presents an alert controller to choose the type of trigger.

    private func addCondition() {

        let title = NSLocalizedString("Add Condition", comment: "Add Condition")

        let alertController = UIAlertController(title: title, message: nil,
        preferredStyle: .actionSheet)

        // Time Condition.

        let timeAction = UIAlertAction(title: NSLocalizedString("Time", comment: "Time"),
        style: .default) { _ in

            self.performSegue(withIdentifier: Identifiers.showTimeConditionSegue, sender:
            self)

        }

```

```
alertController.addAction(timeAction)
```

```
// Characteristic trigger.
```

```
let eventActionTitle = NSLocalizedString("Characteristic", comment:  
"Characteristic")
```

```
let eventAction = UIAlertAction(title: eventActionTitle, style: .default, handler: { _  
in
```

```
    if let triggerCreator = self.triggerCreator as? CharacteristicTriggerCreator {
```

```
        triggerCreator.mode = .condition
```

```
    }
```

```
    self.performSegue(withIdentifier: "Select Characteristic", sender: self)
```

```
})
```

```
alertController.addAction(eventAction)
```

```
// Cancel.
```

```
let cancelAction = UIAlertAction(title: NSLocalizedString("Cancel", comment:  
"Cancel"), style: .cancel, handler: nil)
```

```
alertController.addAction(cancelAction)
```

```
// Present alert.
```

```
if let presenter = alertController.popoverPresentationController {
```

```
    presenter.sourceView = self.view
```

```
    presenter.sourceRect = self.view.bounds
```

```
}
```

```
present(alertController, animated: true, completion: nil)
```

```
}
```

```
/// - returns: `true` if the index path is the 'add row'; `false` otherwise.
```

```
func indexPathIsAdd(_ indexPath: IndexPath) -> Bool {
```

```
    switch sectionForIndex(indexPath.section) {
```

```
        case .conditions?:
```

```
            return indexPath.row == eventTriggerCreator.conditions.count
```

```
        case nil:
```

```
fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
default:
```

```
    return false
```

```
    }
```

```
  }
```

```
}
```

```
// Copyright (c) 2016-present, Facebook, Inc. All rights reserved.
```

```
//
```

```
// You are hereby granted a non-exclusive, worldwide, royalty-free license to use,
```

```
// copy, modify, and distribute this software in source code or binary form for use
```

```
// in connection with the web services and APIs provided by Facebook.
```

```
//
```

```
// As with any software that integrates with the Facebook platform, your use of
```

```
// this software is subject to the Facebook Developer Principles and Policies
```

```
// [http://developers.facebook.com/policy/]. This copyright notice shall be
```

```
// included in all copies or substantial portions of the software.
```

```
//
```

```
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,  
EXPRESS OR
```

```
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
MERCHANTABILITY, FITNESS
```

```
// FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT  
SHALL THE AUTHORS OR
```

```
// COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER  
LIABILITY, WHETHER
```

```
// IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT  
OF OR IN
```

```
// CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN  
THE SOFTWARE.
```

```
import FacebookCore
```

```
import UIKit
```

```
/**
```

A button that initiates a log in or log out flow upon tapping.

`LoginButton` works with `AccessToken.current` to determine what to display,

and automatically starts authentication when tapped (i.e., you do not need to manually subscribe action targets).

Like `LoginManager`, you should make sure your app delegate is connected to `AppDelegate`

in order for the button's delegate to receive messages.

`LoginButton` has a fixed height of @c 30 pixels, but you may change the width.

Initializing the button with `nil` frame will size the button to its minimum frame.

*/

```
public extension FBLoginButton {
```

```
/**
```

Create a new `LoginButton` with a given optional frame and read permissions.

- Parameter frame: Optional frame to initialize with. Default: `nil`, which uses a default size for the button.

- Parameter permissions: Array of read permissions to request when logging in.

```
*/
```

```

convenience init(frame: CGRect = .zero, permissions: [Permission] = [.publicProfile]) {

    self.init(frame: frame)

    self.permissions = permissions.map { $0.name }

}

}

// Copyright (c) 2016-present, Facebook, Inc. All rights reserved.

//

// You are hereby granted a non-exclusive, worldwide, royalty-free license to use,

// copy, modify, and distribute this software in source code or binary form for use

// in connection with the web services and APIs provided by Facebook.

//

// As with any software that integrates with the Facebook platform, your use of

// this software is subject to the Facebook Developer Principles and Policies

// [http://developers.facebook.com/policy/]. This copyright notice shall be

// included in all copies or substantial portions of the software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

```


// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS

// FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR

// COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER

// IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN

// CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

import Foundation

import UIKit

/**

FBProfilePictureView Extension

*/

public extension FBProfilePictureView {

/**

Create a new instance of `UserProfilePictureView`.

- Parameter frame: Optional frame rectangle for the view, measured in points.
- Parameter profile: Optional profile to display a picture for. Default: `UserProfile.current`.

```
*/
```

```
convenience init(frame: CGRect = .zero, profile: Profile? = nil) {
```

```
    self.init(frame: frame)
```

```
    if let profile = profile {
```

```
        profileID = profile.userID
```

```
    }
```

```
    autoresizingMask = [.flexibleWidth, .flexibleHeight]
```

```
    setNeedsImageUpdate()
```

```
}
```

```
}
```

```
// Copyright (c) 2016-present, Facebook, Inc. All rights reserved.
```

```
//
```

// You are hereby granted a non-exclusive, worldwide, royalty-free license to use,

// copy, modify, and distribute this software in source code or binary form for use

// in connection with the web services and APIs provided by Facebook.

//

// As with any software that integrates with the Facebook platform, your use of

// this software is subject to the Facebook Developer Principles and Policies

// [<http://developers.facebook.com/policy/>]. This copyright notice shall be

// included in all copies or substantial portions of the software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS

// FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR

// COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER

// IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN

// CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

import FBSDKCoreKit

import Foundation

// Typealiases for FBSDKCoreKit types to avoid having to import

// dependent libraries. At somepoint these will be likely

// become "wrapper" types that extend and enhance functionality

// in addition to exposing it. For now it suffices to simply expose

// them to the correct library aka. FacebookCore

/**

Wrapper for `FBSDKCoreKit.AccessToken`

- SeeAlso: `FBSDKCoreKit.AccessToken`

*/

public typealias AccessToken = FBSDKCoreKit.AccessToken

/**

Wrapper for `FBSDKCoreKit.AppEvents`

- SeeAlso: `FBSDKCoreKit.AppEvents`

*/

public typealias AppEvents = FBSDKCoreKit.AppEvents

/**

Wrapper for `FBSDKCoreKit.AppLink`

- SeeAlso: `FBSDKCoreKit.AppLink`

*/

public typealias AppLink = FBSDKCoreKit.AppLink

/**

Wrapper for `FBSDKCoreKit.AppLinkNavigation`

- SeeAlso: `FBSDKCoreKit.AppLinkNavigation`

*/

public typealias AppLinkNavigation = FBSDKCoreKit.AppLinkNavigation

/**

Wrapper for `FBSDKCoreKit.AppLinkResolver`

- SeeAlso: `FBSDKCoreKit.AppLinkResolver`

*/

public typealias AppLinkResolver = FBSDKCoreKit.AppLinkResolver

/**

Wrapper for `FBSDKCoreKit.AppLinkReturnToRefererController`

- SeeAlso: `FBSDKCoreKit.AppLinkReturnToRefererController`

*/

```
public typealias AppLinkReturnToRefererController =  
FBSDKCoreKit.AppLinkReturnToRefererController
```

```
/**
```

```
Wrapper for `FBSDKCoreKit.AppLinkTarget`
```

```
- SeeAlso: `FBSDKCoreKit.AppLinkTarget`
```

```
*/
```

```
public typealias AppLinkTarget = FBSDKCoreKit.AppLinkTarget
```

```
/**
```

```
Wrapper for `FBSDKCoreKit.AppLinkURL`
```

```
- SeeAlso: `FBSDKCoreKit.AppLinkURL`
```

```
*/
```

```
public typealias AppLinkURL = FBSDKCoreKit.AppLinkURL
```

```
/**
```

Wrapper for `FBSDKCoreKit.AppLinkUtility`

- SeeAlso: `FBSDKCoreKit.AppLinkUtility`

*/

```
public typealias AppLinkUtility = FBSDKCoreKit.AppLinkUtility
```

/**

Wrapper for `FBSDKCoreKit.ApplicationDelegate`

- SeeAlso: `FBSDKCoreKit.ApplicationDelegate`

*/

```
public typealias AppDelegate = FBSDKCoreKit.ApplicationDelegate
```

/**

Wrapper for `FBSDKCoreKit.CoreError`

- SeeAlso: `FBSDKCoreKit.CoreError`


```
*/
```

```
public typealias CoreError = FBSDKCoreKit.CoreError
```

```
/**
```

```
Wrapper for `FBSDKCoreKit.FBAppLinkReturnToRefererView`
```

```
- SeeAlso: `FBSDKCoreKit.FBAppLinkReturnToRefererView`
```

```
*/
```

```
public          typealias          FBAppLinkReturnToRefererView          =  
FBSDKCoreKit.FBAppLinkReturnToRefererView
```

```
/**
```

```
Wrapper for `FBSDKCoreKit.FBButton`
```

```
- SeeAlso: `FBSDKCoreKit.FBButton`
```

```
*/
```

```
public typealias FBButton = FBSDKCoreKit.FBButton
```

/**

Wrapper for `FBSDKCoreKit.FBProfilePictureView`

- SeeAlso: `FBSDKCoreKit.FBProfilePictureView`

*/

public typealias FBProfilePictureView = FBSDKCoreKit.FBProfilePictureView

/**

Wrapper for `FBSDKCoreKit.GraphRequest`

- SeeAlso: `FBSDKCoreKit.GraphRequest`

*/

public typealias GraphRequest = FBSDKCoreKit.GraphRequest

/**

Wrapper for `FBSDKCoreKit.GraphRequestConnection`

- SeeAlso: `FBSDKCoreKit.GraphRequestConnection`

*/

public typealias GraphRequestConnection = FBSDKCoreKit.GraphRequestConnection

/**

Wrapper for `FBSDKCoreKit.GraphRequestDataAttachment`

- SeeAlso: `FBSDKCoreKit.GraphRequestDataAttachment`

*/

public typealias GraphRequestDataAttachment =
FBSDKCoreKit.GraphRequestDataAttachment

/**

Wrapper for `FBSDKCoreKit.GraphRequestError`

- SeeAlso: `FBSDKCoreKit.GraphRequestError`

*/

public typealias GraphRequestError = FBSDKCoreKit.GraphRequestError

/**

Wrapper for `FBSDKCoreKit.HTTPMethod`

- SeeAlso: `FBSDKCoreKit.HTTPMethod`

*/

public typealias HTTPMethod = FBSDKCoreKit.HTTPMethod

/**

Wrapper for `FBSDKCoreKit.LoggingBehavior`

- SeeAlso: `FBSDKCoreKit.LoggingBehavior`

*/

public typealias LoggingBehavior = FBSDKCoreKit.LoggingBehavior

/**

Wrapper for `FBSDKCoreKit.MeasurementEvent`

- SeeAlso: `FBSDKCoreKit.MeasurementEvent`

*/

public typealias MeasurementEvent = FBSDKCoreKit.MeasurementEvent

/**

Wrapper for `FBSDKCoreKit.Profile`

- SeeAlso: `FBSDKCoreKit.Profile`

*/

public typealias Profile = FBSDKCoreKit.Profile

/**

Wrapper for `FBSDKCoreKit.Settings`

- SeeAlso: `FBSDKCoreKit.Settings`

*/

```
public typealias Settings = FBSDKCoreKit.Settings
```

```
/**
```

```
Wrapper for `FBSDKCoreKit.TestUsersManager`
```

```
- SeeAlso: `FBSDKCoreKit.TestUsersManager`
```

```
*/
```

```
public typealias TestUsersManager = FBSDKCoreKit.TestUsersManager
```

```
/**
```

```
Wrapper for `FBSDKCoreKit.Utility`
```

```
- SeeAlso: `FBSDKCoreKit.Utility`
```

```
*/
```

```
public typealias Utility = FBSDKCoreKit.Utility
```

```
/**
```

Wrapper for `FBSDKCoreKit.WebViewAppLinkResolver`

- SeeAlso: `FBSDKCoreKit.WebViewAppLinkResolver`

*/

```
public          typealias          WebViewAppLinkResolver          =  
FBSDKCoreKit.WebViewAppLinkResolver
```

// MARK: - Protocols

/**

Wrapper for `FBSDKCoreKit.AppLinkResolving`

- SeeAlso: `FBSDKCoreKit.AppLinkResolving`

*/

```
public typealias AppLinkResolving = FBSDKCoreKit.AppLinkResolving
```

/**

Wrapper for `FBSDKCoreKit.AppLinkReturnToRefererControllerDelegate`

- SeeAlso: `FBSDKCoreKit.AppLinkReturnToRefererControllerDelegate`

*/

```
public          typealias          AppLinkReturnToRefererControllerDelegate          =  
FBSDKCoreKit.AppLinkReturnToRefererControllerDelegate
```

/**

Wrapper for `FBSDKCoreKit.AppLinkReturnToRefererViewDelegate`

- SeeAlso: `FBSDKCoreKit.AppLinkReturnToRefererViewDelegate`

*/

```
public          typealias          AppLinkReturnToRefererViewDelegate          =  
FBSDKCoreKit.AppLinkReturnToRefererViewDelegate
```

/**

Wrapper for `FBSDKCoreKit.Copying`

- SeeAlso: `FBSDKCoreKit.Copying`

*/

public typealias Copying = FBSDKCoreKit.Copying

/**

Wrapper for `FBSDKCoreKit.GraphErrorRecoveryProcessorDelegate`

- SeeAlso: `FBSDKCoreKit.GraphErrorRecoveryProcessorDelegate`

*/

public typealias GraphErrorRecoveryProcessorDelegate =
FBSDKCoreKit.GraphErrorRecoveryProcessorDelegate

/**

Wrapper for `FBSDKCoreKit.GraphRequestConnectionDelegate`

- SeeAlso: `FBSDKCoreKit.GraphRequestConnectionDelegate`

*/

public typealias GraphRequestConnectionDelegate =
FBSDKCoreKit.GraphRequestConnectionDelegate

```
/**
```

```
Wrapper for `FBSDKCoreKit.MutableCopying`
```

```
- SeeAlso: `FBSDKCoreKit.MutableCopying`
```

```
*/
```

```
public typealias MutableCopying = FBSDKCoreKit.MutableCopying
```

```
// Copyright (c) 2016-present, Facebook, Inc. All rights reserved.
```

```
//
```

```
// You are hereby granted a non-exclusive, worldwide, royalty-free license to use,
```

```
// copy, modify, and distribute this software in source code or binary form for use
```

```
// in connection with the web services and APIs provided by Facebook.
```

```
//
```

```
// As with any software that integrates with the Facebook platform, your use of
```

```
// this software is subject to the Facebook Developer Principles and Policies
```

```
// [http://developers.facebook.com/policy/]. This copyright notice shall be
```

```
// included in all copies or substantial portions of the software.
```

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS

// FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR

// COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER

// IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN

// CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

import FBSDKLoginKit

import Foundation

// Typealiases for FBSDKLoginKit types to avoid having to import

// dependent libraries. At somepoint these will be likely

// become "wrapper" types that extend and enhance functionality

```
// in addition to exposing it. For now it suffices to simply expose
```

```
// them to the correct library aka. FacebookLogin
```

```
// MARK: - Classes
```

```
/**
```

```
Wrapper for `FBSDKLoginKit.AccessToken`
```

```
- SeeAlso: `FBSDKLoginKit.AccessToken`
```

```
*/
```

```
public typealias AccessToken = FBSDKLoginKit.AccessToken
```

```
/**
```

```
Wrapper for `FBSDKLoginKit.DefaultAudience`
```

```
- SeeAlso: `FBSDKLoginKit.DefaultAudience`
```

```
*/
```

```
public typealias DefaultAudience = FBSDKLoginKit.DefaultAudience
```

```
/**
```

```
Wrapper for `FBSDKLoginKit.DeviceLoginCodeInfo`
```

```
- SeeAlso: `FBSDKLoginKit.DeviceLoginCodeInfo`
```

```
*/
```

```
public typealias DeviceLoginCodeInfo = FBSDKLoginKit.DeviceLoginCodeInfo
```

```
/**
```

```
Wrapper for `FBSDKLoginKit.DeviceLoginError`
```

```
- SeeAlso: `FBSDKLoginKit.DeviceLoginError`
```

```
*/
```

```
public typealias DeviceLoginError = FBSDKLoginKit.DeviceLoginError
```

```
/**
```

Wrapper for `FBSDKLoginKit.DeviceLoginManager`

- SeeAlso: `FBSDKLoginKit.DeviceLoginManager`

*/

public typealias DeviceLoginManager = FBSDKLoginKit.DeviceLoginManager

/**

Wrapper for `FBSDKLoginKit.DeviceLoginManagerResult`

- SeeAlso: `FBSDKLoginKit.DeviceLoginManagerResult`

*/

public typealias DeviceLoginManagerResult =
FBSDKLoginKit.DeviceLoginManagerResult

/**

Wrapper for `FBSDKLoginKit.FBButton`

- SeeAlso: `FBSDKLoginKit.FBButton`

*/

```
public typealias FBButton = FBSDKLoginKit.FBButton
```

/**

Wrapper for `FBSDKLoginKit.FBLoginButton`

- SeeAlso: `FBSDKLoginKit.FBLoginButton`

*/

```
public typealias FBLoginButton = FBSDKLoginKit.FBLoginButton
```

/**

Wrapper for `FBSDKLoginKit.FBLoginTooltipView`

- SeeAlso: `FBSDKLoginKit.FBLoginTooltipView`

*/

```
public typealias FBLoginTooltipView = FBSDKLoginKit.FBLoginTooltipView
```

/**

Wrapper for `FBSDKLoginKit.FBTooltipView`

- SeeAlso: `FBSDKLoginKit.FBTooltipView`

*/

public typealias FBTooltipView = FBSDKLoginKit.FBTooltipView

/**

Wrapper for `FBSDKLoginKit.GraphRequestConnection`

- SeeAlso: `FBSDKLoginKit.GraphRequestConnection`

*/

public typealias GraphRequestConnection = FBSDKLoginKit.GraphRequestConnection

/**

Wrapper for `FBSDKLoginKit.LoginAuthType`

- SeeAlso: `FBSDKLoginKit.LoginAuthType`

*/

public typealias LoginAuthType = FBSDKLoginKit.LoginAuthType

/**

Wrapper for `FBSDKLoginKit.LoginBehavior`

- SeeAlso: `FBSDKLoginKit.LoginBehavior`

*/

public typealias LoginBehavior = FBSDKLoginKit.LoginBehavior

/**

Wrapper for `FBSDKLoginKit.LoginError`

- SeeAlso: `FBSDKLoginKit.LoginError`

*/

public typealias LoginError = FBSDKLoginKit.LoginError

/**

Wrapper for `FBSDKLoginKit.LoginManager`

- SeeAlso: `FBSDKLoginKit.LoginManager`

*/

public typealias LoginManager = FBSDKLoginKit.LoginManager

/**

Wrapper for `FBSDKLoginKit.LoginManagerLoginResult`

- SeeAlso: `FBSDKLoginKit.LoginManagerLoginResult`

*/

public typealias LoginManagerLoginResult =
FBSDKLoginKit.LoginManagerLoginResult

/**

Wrapper for `FBSDKLoginKit.LoginManagerLoginResultBlock`

- SeeAlso: `FBSDKLoginKit.LoginManagerLoginResultBlock`

*/

```
public          typealias          LoginManagerLoginResultBlock          =  
FBSDKLoginKit.LoginManagerLoginResultBlock
```

// MARK: - Protocols

/**

Wrapper for `FBSDKLoginKit.Copying`

- SeeAlso: `FBSDKLoginKit.Copying`

*/

```
public typealias Copying = FBSDKLoginKit.Copying
```

/**

Wrapper for `FBSDKLoginKit.DeviceLoginManagerDelegate`

- SeeAlso: `FBSDKLoginKit.DeviceLoginManagerDelegate`

*/

```
public          typealias          DeviceLoginManagerDelegate          =  
FBSDKLoginKit.DeviceLoginManagerDelegate
```

/**

Wrapper for `FBSDKLoginKit.GraphRequestConnectionDelegate`

- SeeAlso: `FBSDKLoginKit.GraphRequestConnectionDelegate`

*/

```
public          typealias          GraphRequestConnectionDelegate          =  
FBSDKLoginKit.GraphRequestConnectionDelegate
```

/**

Wrapper for `FBSDKLoginKit.LoginButtonDelegate`

- SeeAlso: `FBSDKLoginKit.LoginButtonDelegate`

*/

```
public typealias LoginButtonDelegate = FBSDKLoginKit.LoginButtonDelegate
```

```
/**
```

```
Wrapper for `FBSDKLoginKit.LoginTooltipViewDelegate`
```

```
- SeeAlso: `FBSDKLoginKit.LoginTooltipViewDelegate`
```

```
*/
```

```
public typealias LoginTooltipViewDelegate =  
FBSDKLoginKit.LoginTooltipViewDelegate
```

```
// Copyright (c) 2014-present, Facebook, Inc. All rights reserved.
```

```
//
```

```
// You are hereby granted a non-exclusive, worldwide, royalty-free license to use,
```

```
// copy, modify, and distribute this software in source code or binary form for use
```

```
// in connection with the web services and APIs provided by Facebook.
```

```
//
```

```
// As with any software that integrates with the Facebook platform, your use of
```

```
// this software is subject to the Facebook Developer Principles and Policies
```

```
// [http://developers.facebook.com/policy/]. This copyright notice shall be
```

// included in all copies or substantial portions of the software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS

// FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR

// COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER

// IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN

// CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

#import <UIKit/UIKit.h>

NS_ASSUME_NONNULL_BEGIN

#if TARGET_OS_TV

```

// This is an unfortunate hack for Swift Package Manager support.

// SPM does not allow us to conditionally exclude Swift files for compilation by platform.

//

// So to support tvOS with SPM we need to use runtime availability checks in the Swift
files.

// This means that even though the code in `LoginManager.swift` will never be run for
tvOS

// targets, it still needs to be able to compile. Hence we need to declare it here.

//

// The way to fix this is to remove extensions of ObjC types in Swift.

// This will be done in the next major release (6.0)

@class LoginManagerLoginResult;

typedef NS_ENUM(NSUInteger, LoginBehavior) { LoginBehaviorBrowser };

typedef NS_ENUM(NSUInteger, DefaultAudience) { DefaultAudienceFriends };

typedef void (^LoginManagerLoginResultBlock)(LoginManagerLoginResult * _Nullable
result,

```

```
NSError *_Nullable error);
```

```
@interface LoginManager : NSObject
```

```
@property (assign, nonatomic) LoginBehavior loginBehavior;
```

```
@property (assign, nonatomic) DefaultAudience defaultAudience;
```

```
- (void)logInWithPermissions:(NSArray<NSString *> *)permissions
```

```
fromViewController:(nullable UIViewController *)fromViewController
```

```
handler:(nullable LoginManagerLoginResultBlock)handler
```

```
NS_SWIFT_NAME(logIn(permissions:from:handler:));
```

```
@end
```

```
#else
```

```
@class FBSDKLoginManagerLoginResult;
```



```
/// typedef for FBSDKLoginAuthType
```

```
typedef NSString *const FBSDKLoginAuthType NS_TYPED_EXTENSIBLE_ENUM  
NS_SWIFT_NAME(LoginAuthType);
```

```
/// Rerequest
```

```
FOUNDATION_EXPORT FBSDKLoginAuthType FBSDKLoginAuthTypeRerequest;
```

```
/// Reauthorize
```

```
FOUNDATION_EXPORT FBSDKLoginAuthType FBSDKLoginAuthTypeReauthorize;
```

```
/**
```

Describes the call back to the FBSDKLoginManager

@param result the result of the authorization

@param error the authorization error, if any.

```
*/
```

```
typedef void  
(^FBSDKLoginManagerLoginResultBlock)(FBSDKLoginManagerLoginResult  
*_Nullable result,
```

NSError *_Nullable error)

NS_SWIFT_NAME(LoginManagerLoginResultBlock);

/**

FBSDKDefaultAudience enum

Passed to open to indicate which default audience to use for sessions that post data to Facebook.

Certain operations such as publishing a status or publishing a photo require an audience. When the user

grants an application permission to perform a publish operation, a default audience is selected as the

publication ceiling for the application. This enumerated value allows the application to select which

audience to ask the user to grant publish permission for.

```
*/
```

```
typedef NS_ENUM(NSUInteger, FBSDKDefaultAudience)
```

```
{
```

```
/** Indicates that the user's friends are able to see posts made by the application */
```

```
FBSDKDefaultAudienceFriends = 0,
```

```
/** Indicates that only the user is able to see posts made by the application */
```

```
FBSDKDefaultAudienceOnlyMe,
```

```
/** Indicates that all Facebook users are able to see posts made by the application */
```

```
FBSDKDefaultAudienceEveryone,
```

```
} NS_SWIFT_NAME(DefaultAudience);
```

```
/**
```

```
FBSDKLoginBehavior enum
```

Passed to the \c FBSDKLoginManager to indicate how Facebook Login should be attempted.

Facebook Login authorizes the application to act on behalf of the user, using the user's Facebook account. Usually a Facebook Login will rely on an account maintained outside of the application, by the native Facebook application, the browser, or perhaps the device itself. This avoids the need for a user to enter their username and password directly, and provides the most secure and lowest friction way for a user to authorize the application to interact with Facebook.

The `\c FBSDKLoginBehavior` enum specifies which log-in methods may be used. The SDK

will determine the best behavior based on the current device (such as iOS version).

`*/`

```
typedef NS_ENUM(NSUInteger, FBSDKLoginBehavior)
```

```
{
```

```
/**
```

This is the default behavior, and indicates logging in via `ASWebAuthenticationSession` (iOS 12+) or `SFAuthenticationSession` (iOS 11),

which present specialized SafariViewControllers. Falls back to plain SFSafariViewController (iOS 9 and 10) or Safari (iOS 8).

```
*/
```

```
FBSDKLoginBehaviorBrowser = 0,
```

```
} NS_SWIFT_NAME(LoginBehavior)
```

```
DEPRECATED_MSG_ATTRIBUTE("All login flows utilize the browser. This will be  
removed in the next major release");
```

```
/**
```

`FBSDKLoginManager` provides methods for logging the user in and out.

`FBSDKLoginManager` works directly with `[FBSDKAccessToken
currentAccessToken]` and

sets the "currentAccessToken" upon successful authorizations (or sets `nil` in case of
`logout`).

You should check `[FBSDKAccessToken currentAccessToken]` before calling `login*` to
see if there is

a cached token available (typically in your `viewDidLoad`).

If you are managing your own token instances outside of "currentAccessToken", you will need to set

"currentAccessToken" before calling login* to authorize further permissions on your tokens.

```
*/
```

```
NS_SWIFT_NAME(LoginManager)
```

```
@interface FBSDKLoginManager : NSObject
```

```
/**
```

Auth type

```
*/
```

```
@property (strong, nonatomic) FBSDKLoginAuthType authType;
```

```
/**
```

the default audience.

you should set this if you intend to ask for publish permissions.

```
*/
```

```
@property (assign, nonatomic) FBSDKDefaultAudience defaultAudience;
```

/**

the login behavior

*/

@property (assign, nonatomic) FBSDKLoginBehavior loginBehavior

DEPRECATED_MSG_ATTRIBUTE("All login flows utilize the browser. This will be removed in the next major release");

/**

Logs the user in or authorizes additional permissions.

@param permissions the optional array of permissions. Note this is converted to NSSet and is only

an NSArray for the convenience of literal syntax.

@param fromViewController the view controller to present from. If nil, the topmost view controller will be

automatically determined as best as possible.

@param handler the callback.

Use this method when asking for read permissions. You should only ask for permissions when they

are needed and explain the value to the user. You can inspect the `result.declinedPermissions` to also

provide more information to the user if they decline permissions.

This method will present UI to the user. You typically should check if `[FBSDKAccessToken currentAccessToken]`

already contains the permissions you need before asking to reduce unnecessary app switching. For example,

you could make that check at `viewDidLoad`.

You can only do one login call at a time. Calling a login method before the completion handler is called

on a previous login will return an error.

*/

```
- (void)logInWithPermissions:(NSArray<NSString*>*)permissions
```

```
    fromViewController:(nullable UIViewController*)fromViewController
```

```
    handler:(nullable FBSDKLoginManagerLoginResultBlock)handler
```

```
NS_SWIFT_NAME(logIn(permissions:from:handler:));
```


/**

Requests user's permission to reauthorize application's data access, after it has expired due to inactivity.

@param fromViewController the view controller to present from. If nil, the topmost view controller will be

automatically determined as best as possible.

@param handler the callback.

Use this method when you need to reauthorize your app's access to user data via Graph API, after such an access has expired.

You should provide as much context to the user as possible as to why you need to reauthorize the access, the scope of

access being reauthorized, and what added value your app provides when the access is reauthorized.

You can inspect the result.declinedPermissions to also provide more information to the user if they decline permissions.

This method will present UI to the user. You typically should call this if `[FBSDKAccessToken isDataAccessExpired]` returns true.

*/

- (void)reauthorizeDataAccess:(UIViewController *)fromViewController

handler:(FBSDKLoginManagerLoginResultBlock)handler

NS_SWIFT_NAME(reauthorizeDataAccess(from:handler:));

```
/**
```

Logs the user out

This calls [FBSDKAccessToken setCurrentAccessToken:nil] and [FBSDKProfile setCurrentProfile:nil].

```
*/
```

```
- (void)logOut;
```

```
@end
```

```
#endif
```

```
NS_ASSUME_NONNULL_END
```

```
// Copyright (c) 2014-present, Facebook, Inc. All rights reserved.
```

```
//
```

```
// You are hereby granted a non-exclusive, worldwide, royalty-free license to use,
```

```
// copy, modify, and distribute this software in source code or binary form for use
```

// in connection with the web services and APIs provided by Facebook.

//

// As with any software that integrates with the Facebook platform, your use of

// this software is subject to the Facebook Developer Principles and Policies

// [<http://developers.facebook.com/policy/>]. This copyright notice shall be

// included in all copies or substantial portions of the software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS

// FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR

// COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER

// IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN

// CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

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

```
NS_ASSUME_NONNULL_BEGIN
```

```
#if TARGET_OS_TV
```

```
// This is an unfortunate hack for Swift Package Manager support.
```

```
// SPM does not allow us to conditionally exclude Swift files for compilation by platform.
```

```
//
```

```
// So to support tvOS with SPM we need to use runtime availability checks in the Swift files.
```

```
// This means that even though the code in `LoginManager.swift` will never be run for tvOS
```

```
// targets, it still needs to be able to compile. Hence we need to declare it here.
```

```
//
```

```
// The way to fix this is to remove extensions of ObjC types in Swift.
```

```
// This will be done in the next major release (6.0)
```

```
@interface LoginManagerLoginResult : NSObject
```

```
@property (copy, nonatomic, nullable) FBSDKAccessToken *token;
```

```
@property (readonly, nonatomic) BOOL isCancelled;
```

```
@property (copy, nonatomic) NSSet<NSString *> *grantedPermissions;
```

```
@property (copy, nonatomic) NSSet<NSString *> *declinedPermissions;
```

```
@end
```

```
#else
```

```
@class FBSDKAccessToken;
```

```
/**
```

```
Describes the result of a login attempt.
```

```
*/
```

```
NS_SWIFT_NAME(LoginManagerLoginResult)
```

```
@interface FBSDKLoginManagerLoginResult : NSObject
```

```
- (instancetype)init NS_UNAVAILABLE;
```

```
+ (instancetype)new NS_UNAVAILABLE;
```

```
/**
```

```
the access token.
```

```
*/
```

```
@property (copy, nonatomic, nullable) FBSDKAccessToken *token;
```

```
/**
```

```
whether the login was cancelled by the user.
```

```
*/
```

```
@property (readonly, nonatomic) BOOL isCancelled;
```

```
/**
```

```
the set of permissions granted by the user in the associated request.
```

inspect the token's permissions set for a complete list.

*/

@property (copy, nonatomic) NSSet<NSString *> *grantedPermissions;

/**

the set of permissions declined by the user in the associated request.

inspect the token's permissions set for a complete list.

*/

@property (copy, nonatomic) NSSet<NSString *> *declinedPermissions;

/**

Initializes a new instance.

@param token the access token

@param isCancelled whether the login was cancelled by the user

@param grantedPermissions the set of granted permissions

@param declinedPermissions the set of declined permissions

*/

- (instancetype)initWithToken:(nullable FBSDKAccessToken *)token

isCancelled:(BOOL)isCancelled

grantedPermissions:(NSSet<NSString *> *)grantedPermissions

declinedPermissions:(NSSet<NSString *> *)declinedPermissions

NS_DESIGNATED_INITIALIZER;

@end

#endif

NS_ASSUME_NONNULL_END

// Copyright (c) 2016-present, Facebook, Inc. All rights reserved.

//

// You are hereby granted a non-exclusive, worldwide, royalty-free license to use,

// copy, modify, and distribute this software in source code or binary form for use

// in connection with the web services and APIs provided by Facebook.

//

// As with any software that integrates with the Facebook platform, your use of

// this software is subject to the Facebook Developer Principles and Policies

// [<http://developers.facebook.com/policy/>]. This copyright notice shall be

// included in all copies or substantial portions of the software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS

// FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR

// COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER

// IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN

// CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

// swiftlint:disable type_name file_length

```

import FBSDKShareKit

import Foundation

// Typealiases for FBSDKShareKit types to avoid having to import
// dependent libraries. At somepoint these will be likely
// become "wrapper" types that extend and enhance functionality
// in addition to exposing it. For now it suffices to simply expose
// them to the correct library aka. FacebookShare

// MARK: - Classes

/**

Wrapper for `FBSDKShareKit.AppGroupContent`

- SeeAlso: `FBSDKShareKit.AppGroupContent`

*/

```

```
public typealias AppGroupContent = FBSDKShareKit.AppGroupContent
```

```
/**
```

```
Wrapper for `FBSDKShareKit.AppGroupPrivacy`
```

```
- SeeAlso: `FBSDKShareKit.AppGroupPrivacy`
```

```
*/
```

```
public typealias AppGroupPrivacy = FBSDKShareKit.AppGroupPrivacy
```

```
/**
```

```
Wrapper for `FBSDKShareKit.AppInviteContent`
```

```
- SeeAlso: `FBSDKShareKit.AppInviteContent`
```

```
*/
```

```
public typealias AppInviteContent = FBSDKShareKit.AppInviteContent
```

```
/**
```

Wrapper for `FBSDKShareKit.AppInviteDestination`

- SeeAlso: `FBSDKShareKit.AppInviteDestination`

*/

public typealias AppInviteDestination = FBSDKShareKit.AppInviteDestination

/**

Wrapper for `FBSDKShareKit.CameraEffectArguments`

- SeeAlso: `FBSDKShareKit.CameraEffectArguments`

*/

public typealias CameraEffectArguments = FBSDKShareKit.CameraEffectArguments

/**

Wrapper for `FBSDKShareKit.CameraEffectTextures`

- SeeAlso: `FBSDKShareKit.CameraEffectTextures`

*/

```
public typealias CameraEffectTextures = FBSDKShareKit.CameraEffectTextures
```

/**

Wrapper for `FBSDKShareKit.FBButton`

- SeeAlso: `FBSDKShareKit.FBButton`

*/

```
public typealias FBButton = FBSDKShareKit.FBButton
```

/**

Wrapper for `FBSDKShareKit.FBSendButton`

- SeeAlso: `FBSDKShareKit.FBSendButton`

*/

```
public typealias FBSendButton = FBSDKShareKit.FBSendButton
```

/**

Wrapper for `FBSDKShareKit.FBShareButton`

- SeeAlso: `FBSDKShareKit.FBShareButton`

*/

public typealias FBShareButton = FBSDKShareKit.FBShareButton

/**

Wrapper for `FBSDKShareKit.GameRequestActionType`

- SeeAlso: `FBSDKShareKit.GameRequestActionType`

*/

public typealias GameRequestActionType = FBSDKShareKit.GameRequestActionType

/**

Wrapper for `FBSDKShareKit.GameRequestContent`

- SeeAlso: `FBSDKShareKit.GameRequestContent`

*/

public typealias GameRequestContent = FBSDKShareKit.GameRequestContent

/**

Wrapper for `FBSDKShareKit.GameRequestDialog`

- SeeAlso: `FBSDKShareKit.GameRequestDialog`

*/

public typealias GameRequestDialog = FBSDKShareKit.GameRequestDialog

/**

Wrapper for `FBSDKShareKit.GameRequestFilter`

- SeeAlso: `FBSDKShareKit.GameRequestFilter`

*/

public typealias GameRequestFilter = FBSDKShareKit.GameRequestFilter

/**

Wrapper for `FBSDKShareKit.GraphRequestConnection`

- SeeAlso: `FBSDKShareKit.GraphRequestConnection`

*/

public typealias GraphRequestConnection = FBSDKShareKit.GraphRequestConnection

/**

Wrapper for `FBSDKShareKit.Hashtag`

- SeeAlso: `FBSDKShareKit.Hashtag`

*/

public typealias Hashtag = FBSDKShareKit.Hashtag

/**

Wrapper for `FBSDKShareKit.LikeObjectType`

- SeeAlso: `FBSDKShareKit.LikeObjectType`

*/

public typealias LikeObjectType = FBSDKShareKit.LikeObjectType

/**

Wrapper for `FBSDKShareKit.MessageDialog`

- SeeAlso: `FBSDKShareKit.MessageDialog`

*/

public typealias MessageDialog = FBSDKShareKit.MessageDialog

/**

Wrapper for `FBSDKShareKit.PHAsset`

- SeeAlso: `FBSDKShareKit.PHAsset`

*/

```
public typealias PHAsset = FBSDKShareKit.PHAsset
```

```
/**
```

```
Wrapper for `FBSDKShareKit.ShareAPI`
```

```
- SeeAlso: `FBSDKShareKit.ShareAPI`
```

```
*/
```

```
public typealias ShareAPI = FBSDKShareKit.ShareAPI
```

```
/**
```

```
Wrapper for `FBSDKShareKit.ShareBridgeOptions`
```

```
- SeeAlso: `FBSDKShareKit.ShareBridgeOptions`
```

```
*/
```

```
public typealias ShareBridgeOptions = FBSDKShareKit.ShareBridgeOptions
```

```
/**
```

Wrapper for `FBSDKShareKit.ShareCameraEffectContent`

- SeeAlso: `FBSDKShareKit.ShareCameraEffectContent`

*/

```
public          typealias          ShareCameraEffectContent          =  
FBSDKShareKit.ShareCameraEffectContent
```

/**

Wrapper for `FBSDKShareKit.ShareDialog`

- SeeAlso: `FBSDKShareKit.ShareDialog`

*/

```
public typealias ShareDialog = FBSDKShareKit.ShareDialog
```

/**

Wrapper for `FBSDKShareKit.ShareError`

- SeeAlso: `FBSDKShareKit.ShareError`

*/

public typealias ShareError = FBSDKShareKit.ShareError

/**

Wrapper for `FBSDKShareKit.ShareLinkContent`

- SeeAlso: `FBSDKShareKit.ShareLinkContent`

*/

public typealias ShareLinkContent = FBSDKShareKit.ShareLinkContent

/**

Wrapper for `FBSDKShareKit.ShareMediaContent`

- SeeAlso: `FBSDKShareKit.ShareMediaContent`

*/

public typealias ShareMediaContent = FBSDKShareKit.ShareMediaContent

/**

Wrapper for `FBSDKShareKit.ShareMessengerGenericTemplateContent`

- SeeAlso: `FBSDKShareKit.ShareMessengerGenericTemplateContent`

*/

```
public          typealias          ShareMessengerGenericTemplateContent          =  
FBSDKShareKit.ShareMessengerGenericTemplateContent
```

/**

Wrapper for `FBSDKShareKit.ShareMessengerGenericTemplateElement`

- SeeAlso: `FBSDKShareKit.ShareMessengerGenericTemplateElement`

*/

```
public          typealias          ShareMessengerGenericTemplateElement          =  
FBSDKShareKit.ShareMessengerGenericTemplateElement
```

/**

Wrapper for `FBSDKShareKit.ShareMessengerGenericTemplateImageAspectRatio`

- SeeAlso: `FBSDKShareKit.ShareMessengerGenericTemplateImageAspectRatio`

*/

public typealias ShareMessengerGenericTemplateImageAspectRatio =

FBSDKShareKit.ShareMessengerGenericTemplateImageAspectRatio

/**

Wrapper for `FBSDKShareKit.ShareMessengerMediaTemplateContent`

- SeeAlso: `FBSDKShareKit.ShareMessengerMediaTemplateContent`

*/

public typealias ShareMessengerMediaTemplateContent =
FBSDKShareKit.ShareMessengerMediaTemplateContent

/**

Wrapper for `FBSDKShareKit.ShareMessengerMediaTemplateMediaType`

- SeeAlso: `FBSDKShareKit.ShareMessengerMediaTemplateMediaType`

*/

```
public          typealias      ShareMessengerMediaTemplateMediaType      =  
FBSDKShareKit.ShareMessengerMediaTemplateMediaType
```

/**

Wrapper for `FBSDKShareKit.ShareMessengerOpenGraphMusicTemplateContent`

- SeeAlso: `FBSDKShareKit.ShareMessengerOpenGraphMusicTemplateContent`

*/

```
public          typealias      ShareMessengerOpenGraphMusicTemplateContent  =  
FBSDKShareKit.ShareMessengerOpenGraphMusicTemplateContent
```

/**

Wrapper for `FBSDKShareKit.ShareMessengerURLActionButton`

- SeeAlso: `FBSDKShareKit.ShareMessengerURLActionButton`

*/

```
public          typealias      ShareMessengerURLActionButton                =  
FBSDKShareKit.ShareMessengerURLActionButton
```

/**

Wrapper for `FBSDKShareKit.ShareOpenGraphAction`

- SeeAlso: `FBSDKShareKit.ShareOpenGraphAction`

*/

public typealias ShareOpenGraphAction = FBSDKShareKit.ShareOpenGraphAction

/**

Wrapper for `FBSDKShareKit.ShareOpenGraphContent`

- SeeAlso: `FBSDKShareKit.ShareOpenGraphContent`

*/

public typealias ShareOpenGraphContent = FBSDKShareKit.ShareOpenGraphContent

/**

Wrapper for `FBSDKShareKit.ShareOpenGraphObject`

- SeeAlso: `FBSDKShareKit.ShareOpenGraphObject`

*/

public typealias ShareOpenGraphObject = FBSDKShareKit.ShareOpenGraphObject

/**

Wrapper for `FBSDKShareKit.ShareOpenGraphValueContainer`

- SeeAlso: `FBSDKShareKit.ShareOpenGraphValueContainer`

*/

public typealias ShareOpenGraphValueContainer =
FBSDKShareKit.ShareOpenGraphValueContainer

/**

Wrapper for `FBSDKShareKit.SharePhoto`

- SeeAlso: `FBSDKShareKit.SharePhoto`

*/

```
public typealias SharePhoto = FBSDKShareKit.SharePhoto
```

```
/**
```

```
Wrapper for `FBSDKShareKit.SharePhotoContent`
```

```
- SeeAlso: `FBSDKShareKit.SharePhotoContent`
```

```
*/
```

```
public typealias SharePhotoContent = FBSDKShareKit.SharePhotoContent
```

```
/**
```

```
Wrapper for `FBSDKShareKit.ShareVideo`
```

```
- SeeAlso: `FBSDKShareKit.ShareVideo`
```

```
*/
```

```
public typealias ShareVideo = FBSDKShareKit.ShareVideo
```

```
/**
```

Wrapper for `FBSDKShareKit.ShareVideoContent`

- SeeAlso: `FBSDKShareKit.ShareVideoContent`

*/

```
public typealias ShareVideoContent = FBSDKShareKit.ShareVideoContent
```

// MARK: - Protocols

/**

Wrapper for `FBSDKShareKit.Copying`

- SeeAlso: `FBSDKShareKit.Copying`

*/

```
public typealias Copying = FBSDKShareKit.Copying
```

/**

Wrapper for `FBSDKShareKit.GameRequestDialogDelegate`

- SeeAlso: `FBSDKShareKit.GameRequestDialogDelegate`

*/

```
public          typealias          GameRequestDialogDelegate          =  
FBSDKShareKit.GameRequestDialogDelegate
```

/**

Wrapper for `FBSDKShareKit.GraphRequestConnectionDelegate`

- SeeAlso: `FBSDKShareKit.GraphRequestConnectionDelegate`

*/

```
public          typealias          GraphRequestConnectionDelegate          =  
FBSDKShareKit.GraphRequestConnectionDelegate
```

/**

Wrapper for `FBSDKShareKit.Liking`

- SeeAlso: `FBSDKShareKit.Liking`

*/

public typealias Liking = FBSDKShareKit.Liking

/**

Wrapper for `FBSDKShareKit.ShareMedia`

- SeeAlso: `FBSDKShareKit.ShareMedia`

*/

public typealias ShareMedia = FBSDKShareKit.ShareMedia

/**

Wrapper for `FBSDKShareKit.ShareMessengerActionButton`

- SeeAlso: `FBSDKShareKit.ShareMessengerActionButton`

*/

public typealias ShareMessengerActionButton =
FBSDKShareKit.ShareMessengerActionButton

/**

Wrapper for `FBSDKShareKit.ShareOpenGraphValueContaining`

- SeeAlso: `FBSDKShareKit.ShareOpenGraphValueContaining`

*/

```
public          typealias          ShareOpenGraphValueContaining          =  
FBSDKShareKit.ShareOpenGraphValueContaining
```

/**

Wrapper for `FBSDKShareKit.Sharing`

- SeeAlso: `FBSDKShareKit.Sharing`

*/

```
public typealias Sharing = FBSDKShareKit.Sharing
```

/**

Wrapper for `FBSDKShareKit.SharingButton`

- SeeAlso: `FBSDKShareKit.SharingButton`

*/

public typealias SharingButton = FBSDKShareKit.SharingButton

/**

Wrapper for `FBSDKShareKit.SharingContent`

- SeeAlso: `FBSDKShareKit.SharingContent`

*/

public typealias SharingContent = FBSDKShareKit.SharingContent

/**

Wrapper for `FBSDKShareKit.SharingDelegate`

- SeeAlso: `FBSDKShareKit.SharingDelegate`

*/

public typealias SharingDelegate = FBSDKShareKit.SharingDelegate

/**

Wrapper for `FBSDKShareKit.SharingDialog`

- SeeAlso: `FBSDKShareKit.SharingDialog`

*/

public typealias SharingDialog = FBSDKShareKit.SharingDialog

/**

Wrapper for `FBSDKShareKit.SharingScheme`

- SeeAlso: `FBSDKShareKit.SharingScheme`

*/

public typealias SharingScheme = FBSDKShareKit.SharingScheme

/**

Wrapper for `FBSDKShareKit.SharingValidation`


```

- SeeAlso: `FBSDKShareKit.SharingValidation`

*/

public typealias SharingValidation = FBSDKShareKit.SharingValidation

//

// Fill.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

```

```
@objc(ChartFillType)
```

```
public enum FillType: Int
```

```
{
```

```
    case empty
```

```
    case color
```

```
    case linearGradient
```

```
    case radialGradient
```

```
    case image
```

```
    case tiledImage
```

```
    case layer
```

```
}
```

```
@objc(ChartFill)
```

```
open class Fill: NSObject
```

```
{
```

```
    private var _type: FillType = FillType.empty
```

```

private var _color: CGColor?

private var _gradient: CGGradient?

private var _gradientAngle: CGFloat = 0.0

private var _gradientStartOffsetPercent: CGPoint = CGPoint()

private var _gradientStartRadiusPercent: CGFloat = 0.0

private var _gradientEndOffsetPercent: CGPoint = CGPoint()

private var _gradientEndRadiusPercent: CGFloat = 0.0

private var _image: CGImage?

private var _layer: CGLayer?


// MARK: Properties


@objc open var type: FillType

{

    return _type

}

```

@objc open var color: CGColor?

```
{  
  
    return _color  
  
}
```

@objc open var gradient: CGGradient?

```
{  
  
    return _gradient  
  
}
```

@objc open var gradientAngle: CGFloat

```
{  
  
    return _gradientAngle  
  
}
```

@objc open var gradientStartOffsetPercent: CGPoint

```
{
```

```

        return _gradientStartOffsetPercent
    }

    @objc open var gradientStartRadiusPercent: CGFloat

    {

        return _gradientStartRadiusPercent
    }

    @objc open var gradientEndOffsetPercent: CGPoint

    {

        return _gradientEndOffsetPercent
    }

    @objc open var gradientEndRadiusPercent: CGFloat

    {

        return _gradientEndRadiusPercent
    }

```

```
@objc open var image: CGImage?
```

```
{
```

```
    return _image
```

```
}
```

```
@objc open var layer: CGLayer?
```

```
{
```

```
    return _layer
```

```
}
```

```
// MARK: Constructors
```

```
public override init()
```

```
{
```

```
}
```

```
@objc public init(CGColor: CGColor)
```

```
{
```

```
    _type = .color
```

```
    _color = CGColor
```

```
}
```

```
@objc public convenience init(color: NSUIColor)
```

```
{
```

```
    self.init(CGColor: color.cgColor)
```

```
}
```

```
@objc public init(linearGradient: CGGradient, angle: CGFloat)
```

```
{
```

```
    _type = .linearGradient
```

```
    _gradient = linearGradient
```

```
    _gradientAngle = angle
```

```
}
```

```

@objc public init(

    radialGradient: CGGradient,

    startOffsetPercent: CGPoint,

    startRadiusPercent: CGFloat,

    endOffsetPercent: CGPoint,

    endRadiusPercent: CGFloat

)

{

    _type = .radialGradient

    _gradient = radialGradient

    _gradientStartOffsetPercent = startOffsetPercent

    _gradientStartRadiusPercent = startRadiusPercent

    _gradientEndOffsetPercent = endOffsetPercent

    _gradientEndRadiusPercent = endRadiusPercent

}

```



```
@objc public convenience init(radialGradient: CGGradient)
```

```
{  
  
    self.init(  
  
        radialGradient: radialGradient,  
  
        startOffsetPercent: CGPoint(x: 0.0, y: 0.0),  
  
        startRadiusPercent: 0.0,  
  
        endOffsetPercent: CGPoint(x: 0.0, y: 0.0),  
  
        endRadiusPercent: 1.0  
    )  
}
```

```
@objc public init(CGImage: CGImage, tiled: Bool)
```

```
{  
  
    _type = tiled ? .tiledImage : .image  
  
    _image = CGImage  
}
```

```
@objc public convenience init(image: NSUIImage, tiled: Bool)
```

```
{
```

```
    self.init(CGImage: image.cgImage!, tiled: tiled)
```

```
}
```

```
@objc public convenience init(CGImage: CGImage)
```

```
{
```

```
    self.init(CGImage: CGImage, tiled: false)
```

```
}
```

```
@objc public convenience init(image: NSUIImage)
```

```
{
```

```
    self.init(image: image, tiled: false)
```

```
}
```

```
@objc public init(CGLayer: CGLayer)
```

```
{
```

```
_type = .layer  
  
_layer = CGLayer  
  
}
```

// MARK: Constructors

```
@objc open class func fillWithCGColor(_ CGColor: CGColor) -> Fill  
  
{  
  
    return Fill(CGColor: CGColor)  
  
}
```

```
@objc open class func fillWithColor(_ color: NSUIColor) -> Fill  
  
{  
  
    return Fill(color: color)  
  
}
```

```
@objc open class func fillWithLinearGradient(  

```

```

    _linearGradient: CGGradient,

    angle: CGFloat) -> Fill

{

    return Fill(linearGradient: linearGradient, angle: angle)

}

```

```

@objc open class func fillWithRadialGradient(

```

```

    _radialGradient: CGGradient,

    startOffsetPercent: CGPoint,

    startRadiusPercent: CGFloat,

    endOffsetPercent: CGPoint,

    endRadiusPercent: CGFloat

) -> Fill

{

    return Fill(

        radialGradient: radialGradient,

        startOffsetPercent: startOffsetPercent,

```

```

        startRadiusPercent: startRadiusPercent,

        endOffsetPercent: endOffsetPercent,

        endRadiusPercent: endRadiusPercent

    )

}

```

```

@objc open class func fillWithRadialGradient(_ radialGradient: CGGradient) -> Fill

{

    return Fill(radialGradient: radialGradient)

}

```

```

@objc open class func fillWithCGImage(_ CGImage: CGImage, tiled: Bool) -> Fill

{

    return Fill(CGImage: CGImage, tiled: tiled)

}

```

```

@objc open class func fillWithImage(_ image: NSUIImage, tiled: Bool) -> Fill

```

```
{  
  
    return Fill(image: image, tiled: tiled)  
  
}
```

```
@objc open class func fillWithCGImage(_ CGImage: CGImage) -> Fill
```

```
{  
  
    return Fill(CGImage: CGImage)  
  
}
```

```
@objc open class func fillWithImage(_ image: NSUIImage) -> Fill
```

```
{  
  
    return Fill(image: image)  
  
}
```

```
@objc open class func fillWithCGLayer(_ CGLayer: CGLayer) -> Fill
```

```
{  
  
    return Fill(CGLayer: CGLayer)  
  
}
```

```
}
```

```
// MARK: Drawing code
```

```
/// Draws the provided path in filled mode with the provided area
```

```
@objc open func fillPath(  
  
    context: CGContext,  
  
    rect: CGRect)
```

```
{
```

```
    let fillType = _type
```

```
    if fillType == .empty
```

```
    {
```

```
        return
```

```
    }
```

```
    context.saveGState()
```

```
switch fillType
```

```
{
```

```
case .color:
```

```
    context.setFillColors(_color!)
```

```
    context.fillPath()
```

```
case .image:
```

```
    context.clip()
```

```
    context.draw(_image!, in: rect)
```

```
case .tiledImage:
```

```
    context.clip()
```

```
    context.draw(_image!, in: rect, byTiling: true)
```



```
case .layer:
```

```
    context.clip()
```

```
    context.draw(_layer!, in: rect)
```

```
case .linearGradient:
```

```
    let radians = (360.0 - _gradientAngle).DEG2RAD
```

```
    let centerPoint = CGPoint(x: rect.midX, y: rect.midY)
```

```
    let xAngleDelta = cos(radians) * rect.width / 2.0
```

```
    let yAngleDelta = sin(radians) * rect.height / 2.0
```

```
    let startPoint = CGPoint(
```

```
        x: centerPoint.x - xAngleDelta,
```

```
        y: centerPoint.y - yAngleDelta
```

```
    )
```

```
    let endPoint = CGPoint(
```

```
        x: centerPoint.x + xAngleDelta,
```

```

        y: centerPoint.y + yAngleDelta
    )

    context.clip()

    context.drawLinearGradient(_gradient!,

        start: startPoint,

        end: endPoint,

        options: [.drawsAfterEndLocation, .drawsBeforeStartLocation]
    )

```

```

case .radialGradient:

```

```

    let centerPoint = CGPoint(x: rect.midX, y: rect.midY)

```

```

    let radius = max(rect.width, rect.height) / 2.0

```

```

    context.clip()

```

```

    context.drawRadialGradient(_gradient!,

```

```

startCenter: CGPoint(

    x: centerPoint.x + rect.width * _gradientStartOffsetPercent.x,

    y: centerPoint.y + rect.height * _gradientStartOffsetPercent.y

),

startRadius: radius * _gradientStartRadiusPercent,

endCenter: CGPoint(

    x: centerPoint.x + rect.width * _gradientEndOffsetPercent.x,

    y: centerPoint.y + rect.height * _gradientEndOffsetPercent.y

),

endRadius: radius * _gradientEndRadiusPercent,

options: [.drawsAfterEndLocation, .drawsBeforeStartLocation]

)

```

```

case .empty:

    break

}

```

```

        context.restoreGState()

    }

}

//

// FirebaseAuthManager.swift

// FirebaseAuthApp

//

// Created by Florian Marcu on 2/23/19.

// Copyright © 2019 Instamobile. All rights reserved.

//

import FirebaseAuth

import UIKit

class FirebaseAuthManager {

```

```

func login(credential: AuthCredential, completionBlock: @escaping (_ success: Bool)
-> Void) {

    Auth.auth().signIn(with: credential, completion: { (firebaseUser, error) in

        print(firebaseUser)

        completionBlock(error == nil)

    })

}

```

```

func createUser(email: String, password: String, completionBlock: @escaping (_
success: Bool) -> Void) {

    Auth.auth().createUser(withEmail: email, password: password) {(authResult, error)
in

        if let user = authResult?.user {

            print(user)

            completionBlock(true)

        } else {

            completionBlock(true)

        }

    }

}

```

```

    }

    func signIn(email: String, pass: String, completionBlock: @escaping (_ success: Bool)
-> Void) {

        Auth.auth().signIn(withEmail: email, password: pass) { (result, error) in

            if let error = error, let _ = AuthErrorCode(rawValue: error._code) {

                completionBlock(false)

            } else {

                print(result)

                completionBlock(true)

            }

        }

    }

}

//

// FoldingCell.swift

//

// Copyright (c) 21/12/15. Ramotion Inc. (http://ramotion.com)

```

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

import UIKit

/// UITableViewCell with folding animation

open class FoldingCell: UITableViewCell {

 @objc open var isUnfolded = false

 /// UIView is displayed when cell open

 @IBOutlet open var containerView: UIView!

 @IBOutlet open var containerViewTop: NSLayoutConstraint!

 /// UIView which display when cell close


```
@IBOutlet open var foregroundView: RotatedView!
```

```
@IBOutlet open var foregroundViewTop: NSLayoutConstraint!
```

```
var animationView: UIView?
```

```
/// the number of folding elements. Default 2
```

```
@IBInspectable open var itemCount: NSInteger = 2
```

```
/// The color of the back cell
```

```
@IBInspectable open var backViewColor: UIColor = UIColor.brown
```

```
var animationItemViews: [RotatedView]?
```

```
/**
```

Folding animation types

- Open: Open direction

- Close: Close direction

```
*/
```

```
@objc public enum AnimationType : Int {
```

```
    case open
```

```
    case close
```

```
}
```

```
// MARK: Life Cycle
```

```
open override func awakeFromNib() {
```

```
    super.awakeFromNib()
```

```
    commonInit()
```

```
}
```

```
/**
```

Call this method in methods `init(style: UITableViewCellStyle, reuseIdentifier: String?)`
after creating Views

```
*/
```

```

@objc open func commonInit() {

    configureDefaultState()

    selectionStyle = .none

    containerView.layer.cornerRadius = foregroundView.layer.cornerRadius

    containerView.layer.masksToBounds = true

}

// MARK: configure

private func configureDefaultState() {

    guard let foregroundViewTop = self.foregroundViewTop,

        let containerViewTop = self.containerViewTop else {

        fatalError("set foregroundViewTop or containerViewTop outlets in storyboard")

    }

```

```
containerViewTop.constant = foregroundViewTop.constant
```

```
containerView.alpha = 0
```

```
if let height = (foregroundView.constraints.filter { $0.firstAttribute == .height &&  
$0.secondItem == nil }).first?.constant {
```

```
    foregroundView.layer.anchorPoint = CGPoint(x: 0.5, y: 1)
```

```
    foregroundViewTop.constant += height / 2
```

```
}
```

```
foregroundView.layer.transform = foregroundView.transform3d()
```

```
createAnimationView()
```

```
contentView.bringSubviewToFront(foregroundView)
```

```
}
```

```
private func createAnimationItemView() -> [RotatedView] {
```

```
    var items = [RotatedView]()
```

```

items.append(foregroundView)

var rotatedViews = [RotatedView]()

animationView?.subviews

    .lazy

    .compactMap({ $0 as? RotatedView })

    .sorted(by: { $0.tag < $1.tag })

    .forEach { itemView in

        rotatedViews.append(itemView)

        if let backView = itemView.backView {

            rotatedViews.append(backView)

        }

    }

items.append(contentsOf: rotatedViews)

return items

}

```

```

func configureAnimationItems(_ animationType: AnimationType) {

    if animationType == .open {

        animationView?.subviews

            .lazy

            .compactMap { $0 as? RotatedView }

            .forEach { $0.alpha = 0 }

    } else {

        animationView?.subviews

            .lazy

            .compactMap { $0 as? RotatedView }

            .forEach {

                $0.alpha = animationType == .open ? 0 : 1

                if animationType != .open { $0.backView?.alpha = 0 }

            }

    }

}

```

```
}
```

```
func createAnimationView() {  
  
    animationView = UIView(frame: containerView.frame)  
  
    animationView?.layer.cornerRadius = foregroundView.layer.cornerRadius  
  
    animationView?.backgroundColor = .clear  
  
    animationView?.translatesAutoresizingMaskIntoConstraints = false  
  
    animationView?.alpha = 0  
  
  
    guard let animationView = self.animationView else { return }  
  
  
    self.contentView.addSubview(animationView)  
  
  
    // copy constraints from containerView  
  
    var newConstraints = [NSLayoutConstraint]()  
  
    for constraint in self.contentView.constraints {  
  
        if let item = constraint.firstItem as? UIView, item == containerView {
```

```

        let newConstraint = NSLayoutConstraint(item: animationView, attribute:
constraint.firstAttribute,

                                relatedBy:      constraint.relation,      toItem:
constraint.secondItem, attribute: constraint.secondAttribute,

                                multiplier:     constraint.multiplier,     constant:
constraint.constant)

```

```

        newConstraints.append(newConstraint)

```

```

    } else if let firstItem = constraint.firstItem as? UIView, let secondItem: UIView =
constraint.secondItem as? UIView, secondItem == containerView {

```

```

        let newConstraint = NSLayoutConstraint(item: firstItem, attribute:
constraint.firstAttribute,

                                relatedBy: constraint.relation, toItem: animationView,
attribute: constraint.secondAttribute,

                                multiplier:     constraint.multiplier,     constant:
constraint.constant)

```

```

        newConstraints.append(newConstraint)

```

```

    }

```

```

}

```

```

self.contentView.addConstraints(newConstraints)

```



```

    for constraint in containerView.constraints { // added height constraint

        if constraint.firstAttribute == .height, let item: UIView = constraint.firstItem as?
UIView, item == containerView {

            let newConstraint = NSLayoutConstraint(item: animationView, attribute:
constraint.firstAttribute,

                                                relatedBy: constraint.relation, toItem: nil, attribute:
constraint.secondAttribute,

                                                multiplier: constraint.multiplier, constant:
constraint.constant)

            animationView.addConstraint(newConstraint)

        }

    }

}

```

```

func addImageItemsToAnimationView() {

    containerView.alpha = 1

    let containerViewSize = containerView.bounds.size

```

```

let foregroundViewSize = foregroundView.bounds.size

// added first item

var image = containerView.takeSnapshot(CGRect(x: 0, y: 0, width:
containerViewSize.width, height: foregroundViewSize.height))

var imageView = UIImageView(image: image)

imageView.tag = 0

imageView.layer.cornerRadius = foregroundView.layer.cornerRadius

animationView?.addSubview(imageView)

// added second item

image = containerView.takeSnapshot(CGRect(x: 0, y: foregroundViewSize.height,
width: containerViewSize.width, height: foregroundViewSize.height))

imageView = UIImageView(image: image)

let rotatedView = RotatedView(frame: imageView.frame)

rotatedView.tag = 1

rotatedView.layer.anchorPoint = CGPoint(x: 0.5, y: 0)

```

```

rotatedView.layer.transform = rotatedView.transform3d()

rotatedView.addSubview(imageView)

animationView?.addSubview(rotatedView)

rotatedView.frame = CGRect(x: imageView.frame.origin.x, y:
foregroundViewSize.height, width: containerViewSize.width, height:
foregroundViewSize.height)

// added other views

let itemHeight = (containerViewSize.height - 2 * foregroundViewSize.height) /
CGFloat(itemCount - 2)

if itemCount == 2 {

    // decrease containerView height or increase itemCount

    assert(containerViewSize.height - 2 * foregroundViewSize.height == 0,
"containerView.height too high")

} else {

    // decrease containerView height or increase itemCount

    assert(containerViewSize.height - 2 * foregroundViewSize.height >= itemHeight,
"containerView.height too high")

```

```
}
```

```
var yPosition = 2 * foregroundViewSize.height
```

```
var tag = 2
```

```
for _ in 2 ..< itemCount {
```

```
    image = containerView.takeSnapshot(CGRect(x: 0, y: yPosition, width:
containerViewSize.width, height: itemHeight))
```

```
    imageView = UIImageView(image: image)
```

```
    let rotatedView = RotatedView(frame: imageView.frame)
```

```
    rotatedView.addSubview(imageView)
```

```
    rotatedView.layer.anchorPoint = CGPoint(x: 0.5, y: 0)
```

```
    rotatedView.layer.transform = rotatedView.transform3d()
```

```
    animationView?.addSubview(rotatedView)
```

```
    rotatedView.frame = CGRect(x: 0, y: yPosition, width:
rotatedView.bounds.size.width, height: itemHeight)
```

```
    rotatedView.tag = tag
```

```

        yPosition += itemHeight

        tag += 1
    }

    containerView.alpha = 0

    if let animationView = self.animationView {

        // added back view

        var previuosView: RotatedView?

        for case let container as RotatedView in animationView.subviews.sorted(by:
{ $0.tag < $1.tag })

            where container.tag > 0 && container.tag < animationView.subviews.count {

                previuosView?.addBackView(container.bounds.size.height,          color:
backViewColor)

                previuosView = container

            }

        }
    }

```

```

        animationItemViews = createAnimationItemView()

    }

fileprivate func removeImageItemsFromAnimationView() {

    guard let animationView = self.animationView else {

        return

    }

    animationView.subviews.forEach({ $0.removeFromSuperview() })

}

// MARK: public

/// Unfold cell.

///

/// - Parameters:

```

/// - value: unfold = true; collapse = false.

/// - animated: animate changes.

/// - completion: A block object to be executed when the animation sequence ends.

```
@objc open func unfold(_ value: Bool, animated: Bool = true, completion: (() -> Void)?  
= nil) {
```

```
    if animated {
```

```
        value ? openAnimation(completion) : closeAnimation(completion)
```

```
    } else {
```

```
        foregroundView.alpha = value ? 0 : 1
```

```
        containerView.alpha = value ? 1 : 0
```

```
    }
```

```
}
```

```
@objc open func isAnimating() -> Bool {
```

```
    return animationView?.alpha == 1 ? true : false
```

```
}
```

```
// MARK: Animations
```

```

@objc open dynamic func animationDuration(_ itemIndex: NSInteger, type:
AnimationType) -> TimeInterval {

    return type == .close ? durationsForCollapsedState[itemIndex] :
durationsForExpandedState[itemIndex]

}

```

```

@objc open var durationsForExpandedState: [TimeInterval] = []

```

```

@objc open var durationsForCollapsedState: [TimeInterval] = []

```

```

func durationSequence(_ type: AnimationType) -> [TimeInterval] {

    var durations = [TimeInterval]()

    for i in 0 ..< itemCount - 1 {

        let duration = animationDuration(i, type: type)

        durations.append(TimeInterval(duration / 2.0))

        durations.append(TimeInterval(duration / 2.0))

    }

    return durations
}

```



```
}
```

```
func openAnimation(_ completion: (() -> Void)?) {
```

```
    isUnfolded = true
```

```
    removeImageItemsFromAnimationView()
```

```
    addImageItemsToAnimationView()
```

```
    animationView?.alpha = 1
```

```
    containerView.alpha = 0
```

```
    let durations = durationSequence(.open)
```

```
    var delay: TimeInterval = 0
```

```
    var timing =  
    convertFromCAMediaTimingFunctionName(CAMediaTimingFunctionName.easeIn)
```

```
    var from: CGFloat = 0.0
```

```
    var to: CGFloat = -CGFloat.pi / 2
```

```
    var hidden = true
```

```
configureAnimationItems(.open)
```

```
guard let animationItemViews = self.animationItemViews else {
```

```
    return
```

```
}
```

```
for index in 0 ..< animationItemViews.count {
```

```
    let animatedView = animationItemViews[index]
```

```
        animatedView.foldingAnimation(timing: from: from, to: to, duration:
durations[index], delay: delay, hidden: hidden)
```

```
        from = from == 0.0 ? CGFloat.pi / 2 : 0.0
```

```
        to = to == 0.0 ? -CGFloat.pi / 2 : 0.0
```

```
        timing = timing ==
convertFromCAMediaTimingFunctionName(CAMediaTimingFunctionName.easeIn) ?
convertFromCAMediaTimingFunctionName(CAMediaTimingFunctionName.easeOut) :
convertFromCAMediaTimingFunctionName(CAMediaTimingFunctionName.easeIn)
```

```
        hidden = !hidden
```

```
        delay += durations[index]
    }
}
```

```
let firstItemView = animationView?.subviews.filter { $0.tag == 0 }.first
```

```
firstItemView?.layer.masksToBounds = true
```

```
DispatchQueue.main.asyncAfter(deadline: .now() + durations[0], execute: {
```

```
    firstItemView?.layer.cornerRadius = 0
```

```
})
```

```
DispatchQueue.main.asyncAfter(deadline: .now() + delay) {
```

```
    self.animationView?.alpha = 0
```

```
    self.containerView.alpha = 1
```

```
    completion?()
```

```
}
```

```
}
```

```

func closeAnimation(_ completion: (() -> Void)?) {

    isUnfolded = false

    removeImageItemsFromAnimationView()

    addImageItemsToAnimationView()


    guard let animationItemViews = self.animationItemViews else {

        fatalError()

    }


    animationView?.alpha = 1

    containerView.alpha = 0


    let durations: [TimeInterval] = durationSequence(.close).reversed()


    var delay: TimeInterval = 0


    var timing =
    convertFromCAMediaTimingFunctionName(CAMediaTimingFunctionName.easeIn)


    var from: CGFloat = 0.0

```

```

var to: CGFloat = CGFloat.pi / 2

var hidden = true

configureAnimationItems(.close)

if durations.count < animationItemViews.count {

    fatalError("wrong override func animationDuration(itemIndex:NSInteger,
type:AnimationType)-> NSTimeInterval")

}

for index in 0 ..< animationItemViews.count {

    let animatedView = animationItemViews.reversed()[index]

    animatedView.foldingAnimation(timing, from: from, to: to, duration:
durations[index], delay: delay, hidden: hidden)

    to = to == 0.0 ? CGFloat.pi / 2 : 0.0

    from = from == 0.0 ? -CGFloat.pi / 2 : 0.0

    timing = timing ==
convertFromCAMediaTimingFunctionName(CAMediaTimingFunctionName.easeIn) ?
convertFromCAMediaTimingFunctionName(CAMediaTimingFunctionName.easeOut) :
convertFromCAMediaTimingFunctionName(CAMediaTimingFunctionName.easeIn)

```

```

        hidden = !hidden

        delay += durations[index]

    }

    DispatchQueue.main.asyncAfter(deadline: .now() + delay, execute: {

        self.animationView?.alpha = 0

        completion?()

    })

    let firstItemView = animationView?.subviews.filter { $0.tag == 0 }.first

    firstItemView?.layer.cornerRadius = 0

    firstItemView?.layer.masksToBounds = true

    if let durationFirst = durations.first {

        DispatchQueue.main.asyncAfter(deadline: .now() + delay - durationFirst * 2,
        execute: {

            firstItemView?.layer.cornerRadius = self.foregroundView.layer.cornerRadius

            firstItemView?.setNeedsDisplay()

            firstItemView?.setNeedsLayout()

        })
    }

```

```
    }  
  }  
}
```

```
// MARK: RotatedView
```

```
open class RotatedView: UIView {
```

```
  private enum Const {
```

```
    static let rotationX = "rotation.x"
```

```
    static let transformRotationX = "transform.rotation.x"
```

```
  }
```

```
  var hiddenAfterAnimation = false
```

```
  var backView: RotatedView?
```

```
  func addBackView(_ height: CGFloat, color: UIColor) {
```

```

let view = RotatedView(frame: CGRect.zero)

view.backgroundColor = color

view.layer.anchorPoint = CGPoint(x: 0.5, y: 1)

view.layer.transform = view.transform3d()

view.translatesAutoresizingMaskIntoConstraints = false

self.addSubview(view)

backView = view


view.addConstraint(NSLayoutConstraint(item: view, attribute: .height,
relatedBy: .equal, toItem: nil, attribute: .height,
multiplier: 1, constant: height))


self.addConstraints([

    NSLayoutConstraint(item: view, attribute: .top, relatedBy: .equal, toItem: self,
attribute: .top, multiplier: 1,
constant: self.bounds.size.height - height + height / 2),

    NSLayoutConstraint(item: view, attribute: .leading, relatedBy: .equal, toItem: self,
attribute: .leading,
multiplier: 1, constant: 0),

```



```
        NSLayoutConstraint(item: view, attribute: .trailing, relatedBy: .equal, toItem: self,  
attribute: .trailing,
```

```
        multiplier: 1, constant: 0),
```

```
    ])
```

```
    }
```

```
}
```

```
extension RotatedView: CAAAnimationDelegate {
```

```
    func rotatedX(_ angle: CGFloat) {
```

```
        var allTransoform = CATransform3DIdentity
```

```
        let rotateTransform = CATransform3DMakeRotation(angle, 1, 0, 0)
```

```
        allTransoform = CATransform3DConcat(allTransoform, rotateTransform)
```

```
        allTransoform = CATransform3DConcat(allTransoform, transform3d())
```

```
        self.layer.transform = allTransoform
```

```
    }
```

```
    func transform3d() -> CATransform3D {
```

```

var transform = CATransform3DIdentity

transform.m34 = 2.5 / -2000

return transform

}

// MARK: animations

func foldingAnimation(_ timing: String, from: CGFloat, to: CGFloat, duration:
TimeInterval, delay: TimeInterval, hidden: Bool) {

    let rotateAnimation = CABasicAnimation(keyPath: Const.transformRotationX)

    rotateAnimation.timingFunction = CAMediaTimingFunction(name:
convertToCAMediaTimingFunctionName(timing))

    rotateAnimation.fromValue = from

    rotateAnimation.toValue = to

    rotateAnimation.duration = duration

    rotateAnimation.delegate = self

    rotateAnimation.fillMode = CAMediaTimingFillMode.forwards

```

```
rotateAnimation.isRemovedOnCompletion = false
```

```
rotateAnimation.beginTime = CACurrentMediaTime() + delay
```

```
self.hiddenAfterAnimation = hidden
```

```
self.layer.add(rotateAnimation, forKey: Const.rotationX)
```

```
}
```

```
public func animationDidStart(_ : CAAnimation) {
```

```
    self.layer.shouldRasterize = true
```

```
    self.alpha = 1
```

```
}
```

```
public func animationDidStop(_ : CAAnimation, finished _ : Bool) {
```

```
    if hiddenAfterAnimation {
```

```
        self.alpha = 0
```

```
    }
```

```

        self.layer.removeAllAnimations()

        self.layer.shouldRasterize = false

        self.rotatedX(CGFloat(0))

    }

}

// MARK: UIView + extension

private extension UIView {

    func takeSnapshot(_ frame: CGRect) -> UIImage? {

        UIGraphicsBeginImageContextWithOptions(frame.size, false, 0)

        guard let context = UIGraphicsGetCurrentContext() else { return nil }

        context.translateBy(x: frame.origin.x * -1, y: frame.origin.y * -1)

        layer.render(in: context)

        let image = UIGraphicsGetImageFromCurrentImageContext()

```

```
UIGraphicsEndImageContext()
```

```
    return image
```

```
    }
```

```
}
```

```
// Helper function inserted by Swift 4.2 migrator.
```

```
fileprivate func convertFromCAMediaTimingFunctionName(_ input: CAMediaTimingFunctionName) -> String {
```

```
    return input.rawValue
```

```
}
```

```
// Helper function inserted by Swift 4.2 migrator.
```

```
fileprivate func convertToCAMediaTimingFunctionName(_ input: String) -> CAMediaTimingFunctionName {
```

```
    return CAMediaTimingFunctionName(rawValue: input)
```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `HMActionSet+BuiltIn` extension provides a method for determining whether or not an action set is built-in.

*/

```
import HomeKit
```

```
extension HMActionSet {
```

```
    struct Constants {
```

```
        static let builtInActionTypes = [HMActionSetTypeWakeUp,
HMActionSetTypeHomeDeparture,          HMActionSetTypeHomeArrival,
HMActionSetTypeSleep]
```

```
    }
```

```
    /// - returns: `true` if the action set is built-in; `false` otherwise.
```

```
    var isBuiltIn: Bool {
```

```
        return Constants.builtInActionTypes.contains(self.actionSetType)

    }

}

/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `HMCatalogViewController` is a super class which mainly provides easy-access methods for shared HomeKit objects.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
/**
```

The super class for most table view controllers in this app. It manages home delegate registration and facilitates 'popping back' when it's discovered that

a home has been deleted.

*/

```
class HMCatalogViewController: UITableViewController, HMHomeDelegate {
```

```
    // MARK: Properties
```

```
    var homeStore: HomeStore {
```

```
        return HomeStore.shared
```

```
    }
```

```
    var home: HMHome! {
```

```
        return homeStore.home
```

```
    }
```

```
    // MARK: View Methods
```

```
    /**
```

```
        Evaluates whether or not the view controller should pop to
```


the list of homes.

- returns: `true` if this instance is not the root view controller

and the `home` is nil; `false` otherwise.

*/

```
private func shouldPopViewController() -> Bool {
```

```
    if let rootViewController = navigationController?.viewControllers.first
```

```
        , rootViewController == self {
```

```
        return false
```

```
    }
```

```
    return home == nil
```

```
}
```

```
/// Pops the view controller, if required. Invokes the delegate registration method.
```

```
override func viewWillAppear(_ animated: Bool) {
```

```
    super.viewWillAppear(animated)
```

```

if shouldPopViewController() {

    // Pop to root view controller if our home was destroyed while we were away.

    _ = navigationController?.popToRootViewController(animated: true)

    return

}

```

```

registerAsDelegate()

}

```

// MARK: Delegate Registration

/**

A hierarchical method, to be overridden by superclasses.

The base implementation registers as the delegate for the `HomeStore`'s home.

Thus, any subclasses may override this, register as the delegate for any

objects they please, and then call `super.registerAsDelegate()` to register

as the home delegate as well.

This method will be called when the view appears.

```
*/
```

```
func registerAsDelegate() {
```

```
    homeStore.home?.delegate = self
```

```
}
```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `HMCharacteristic+Properties` methods are used to generate localized strings related to a characteristic or to evaluate a characteristic's type.

```
*/
```

```
import HomeKit
```

```
extension HMCharacteristic {
```

```
    private struct Constants {
```

```
        static let valueFormatter = NumberFormatter()
```

```
        static let numericFormats = [
```

```
            HMCharacteristicMetadataFormatInt,
```

```
            HMCharacteristicMetadataFormatFloat,
```

```
            HMCharacteristicMetadataFormatUInt8,
```

```
            HMCharacteristicMetadataFormatUInt16,
```

```
            HMCharacteristicMetadataFormatUInt32,
```

```
            HMCharacteristicMetadataFormatUInt64
```

```
        ]
```

```
    }
```

```
    /**
```

```
        Returns the localized description for a provided value, taking the characteristic's
        metadata and possible
```

values into account.

- parameter value: The value to look up.

- returns: A string representing the value in a localized way, e.g. `"24%"` or `"354°"

*/

```
func localizedDescriptionForValue(_ value: AnyObject) -> String {

    if self.isWriteOnly {

        return NSLocalizedString("Write-Only Characteristic", comment: "Write-Only
Characteristic")

    }

    else if self.isBoolean {

        if let boolValue = value.boolValue {

            return boolValue ? NSLocalizedString("On", comment: "On") :
NSLocalizedString("Off", comment: "Off")

        }

    }

    if let number = value as? Int {
```

```

        if let predeterminedValueString =
self.predeterminedValueDescriptionForNumber(number) {

            return predeterminedValueString

        }

        if let stepValue = self.metadata?.stepValue {

            Constants.valueFormatter.minimumFractionDigits = Int(log10(1.0 /
stepValue.doubleValue))

            if let string = Constants.valueFormatter.string(from: NSNumber(value: number))
{

                return string + self.localizedUnitDecoration

            }

        }

        return "\(value)"

    }
}

/**

```

- parameter number: The value of this characteristic.

- returns: An optional, localized string for the value.

*/

```
func predeterminedValueDescriptionForNumber(_ number: Int) -> String? {

    switch self.characteristicType {

        case HMCharacteristicTypePowerState, HMCharacteristicTypeInputEvent,
HMCharacteristicTypeOutputState:

            if number != 0 {

                return NSLocalizedString("On", comment: "On")

            }

            else {

                return NSLocalizedString("Off", comment: "Off")

            }

        case HMCharacteristicTypeOutletInUse, HMCharacteristicTypeMotionDetected,
HMCharacteristicTypeAdminOnlyAccess, HMCharacteristicTypeAudioFeedback,
HMCharacteristicTypeObstructionDetected:

            if number != 0 {

                return NSLocalizedString("Yes", comment: "Yes")
```

```

    }

    else {

        return NSLocalizedString("No", comment: "No")

    }

case                                     HMCharacteristicTypeTargetDoorState,
HMCharacteristicTypeCurrentDoorState:

    if let doorState = HMCharacteristicValueDoorState(rawValue: number) {

        switch doorState {

            case .open:

                return NSLocalizedString("Open", comment: "Open")

            case .opening:

                return NSLocalizedString("Opening", comment: "Opening")

            case .closed:

                return NSLocalizedString("Closed", comment: "Closed")

```



```
case .closing:
```

```
    return NSLocalizedString("Closing", comment: "Closing")
```

```
case .stopped:
```

```
    return NSLocalizedString("Stopped", comment: "Stopped")
```

```
@unknown default:
```

```
    return ""
```

```
}
```

```
}
```

```
case HMCharacteristicTypeTargetHeatingCooling:
```

```
    if let mode = HMCharacteristicValueHeatingCooling(rawValue: number) {
```

```
        switch mode {
```

```
            case .off:
```

```
                return NSLocalizedString("Off", comment: "Off")
```

```
case .heat:
```

```
    return NSLocalizedString("Heat", comment: "Heat")
```

```
case .cool:
```

```
    return NSLocalizedString("Cool", comment: "Cool")
```

```
case .auto:
```

```
    return NSLocalizedString("Auto", comment: "Auto")
```

```
@unknown default:
```

```
    return ""
```

```
}
```

```
}
```

```
case HMCharacteristicTypeCurrentHeatingCooling:
```

```
    if let mode = HMCharacteristicValueHeatingCooling(rawValue: number) {
```

```
        switch mode {
```

```
            case .off:
```

```
return NSLocalizedString("Off", comment: "Off")
```

```
case .heat:
```

```
return NSLocalizedString("Heating", comment: "Heating")
```

```
case .cool:
```

```
return NSLocalizedString("Cooling", comment: "Cooling")
```

```
case .auto:
```

```
return NSLocalizedString("Auto", comment: "Auto")
```

```
@unknown default:
```

```
return ""
```

```
}
```

```
}
```

```
case HMCharacteristicTypeTargetLockMechanismState,  
HMCharacteristicTypeCurrentLockMechanismState:
```

```

        if let lockState = HMCharacteristicValueLockMechanismState(rawValue:
number) {

            switch lockState {

                case .unsecured:

                    return NSLocalizedString("Unsecured", comment: "Unsecured")

                case .secured:

                    return NSLocalizedString("Secured", comment: "Secured")

                case .unknown:

                    return NSLocalizedString("Unknown", comment: "Unknown")

                case .jammed:

                    return NSLocalizedString("Jammed", comment: "Jammed")

                @unknown default:

                    return ""

            }

```

```
}
```

```
case HMCharacteristicTypeTemperatureUnits:
```

```
if let unit = HMCharacteristicValueTemperatureUnit(rawValue: number) {
```

```
    switch unit {
```

```
        case .celsius:
```

```
            return NSLocalizedString("Celsius", comment: "Celsius")
```

```
        case .fahrenheit:
```

```
            return NSLocalizedString("Fahrenheit", comment: "Fahrenheit")
```

```
        @unknown default:
```

```
            return ""
```

```
    }
```

```
}
```

```
case HMCharacteristicTypeLockMechanismLastKnownAction:
```

```

        if let lastKnownAction =
HMCharacteristicValueLockMechanismLastKnownAction(rawValue: number) {

        switch lastKnownAction {

        case .securedUsingPhysicalMovementInterior:

            return NSLocalizedString("Interior Secured", comment: "Interior
Secured")

        case .unsecuredUsingPhysicalMovementInterior:

            return NSLocalizedString("Exterior Unsecured", comment: "Exterior
Unsecured")

        case .securedUsingPhysicalMovementExterior:

            return NSLocalizedString("Exterior Secured", comment: "Exterior
Secured")

        case .unsecuredUsingPhysicalMovementExterior:

            return NSLocalizedString("Exterior Unsecured", comment: "Exterior
Unsecured")

        case .securedWithKeypad:

```

```
        return NSLocalizedString("Keypad Secured", comment: "Keypad  
Secured")
```

```
    case .unsecuredWithKeypad:
```

```
        return NSLocalizedString("Keypad Unsecured", comment: "Keypad  
Unsecured")
```

```
    case .securedRemotely:
```

```
        return NSLocalizedString("Secured Remotely", comment: "Secured  
Remotely")
```

```
    case .unsecuredRemotely:
```

```
        return NSLocalizedString("Unsecured Remotely", comment:  
"Unsecured Remotely")
```

```
    case .securedWithAutomaticSecureTimeout:
```

```
        return NSLocalizedString("Secured Automatically", comment: "Secured  
Automatically")
```

```
    case .securedUsingPhysicalMovement:
```

```
        return NSLocalizedString("Secured Using Physical Movement",
comment: "Secured Using Physical Movement")
```

```
    case .unsecuredUsingPhysicalMovement:
```

```
        return NSLocalizedString("Unsecured Using Physical Movement",
comment: "Unsecured Using Physical Movement")
```

```
    @unknown default:
```

```
        return ""
```

```
    }
```

```
}
```

```
case HMCharacteristicTypeRotationDirection:
```

```
    if let rotationDirection = HMCharacteristicValueRotationDirection(rawValue:
number) {
```

```
        switch rotationDirection {
```

```
            case .clockwise:
```

```
                return NSLocalizedString("Clockwise", comment: "Clockwise")
```



```
case .counterClockwise:

    return NSLocalizedString("Counter Clockwise", comment: "Counter
Clockwise")
```

```
@unknown default:

    return ""

}

}
```

```
case HMCharacteristicTypeAirParticulateSize:

    if let size = HMCharacteristicValueAirParticulateSize(rawValue: number) {

        switch size {

            case .size10:

                return NSLocalizedString("Size 10", comment: "Size 10")

            case .size2_5:

                return NSLocalizedString("Size 2.5", comment: "Size 2.5")
```

```
@unknown default:
```

```
    return ""
```

```
}
```

```
}
```

```
case HMCharacteristicTypePositionState:
```

```
    if let state = HMCharacteristicValuePositionState(rawValue: number) {
```

```
        switch state {
```

```
            case .opening:
```

```
                return NSLocalizedString("Opening", comment: "Opening")
```

```
            case .closing:
```

```
                return NSLocalizedString("Closing", comment: "Closing")
```

```
            case .stopped:
```

```
                return NSLocalizedString("Stopped", comment: "Stopped")
```

```

        @unknown default:

            return ""

        }

    }

case HMCharacteristicTypeCurrentSecuritySystemState:

    if let state = HMCharacteristicValueCurrentSecuritySystemState(rawValue:
number) {

        switch state {

            case .awayArm:

                return NSLocalizedString("Away", comment: "Away")

            case .stayArm:

                return NSLocalizedString("Home", comment: "Home")

            case .nightArm:

                return NSLocalizedString("Night", comment: "Night")

```

```
case .disarmed:
```

```
    return NSLocalizedString("Disarm", comment: "Disarm")
```

```
case .triggered:
```

```
    return NSLocalizedString("Triggered", comment: "Triggered")
```

```
@unknown default:
```

```
    return ""
```

```
}
```

```
}
```

```
case HMCharacteristicTypeTargetSecuritySystemState:
```

```
    if let state = HMCharacteristicValueTargetSecuritySystemState(rawValue:  
number) {
```

```
        switch state {
```

```
            case .awayArm:
```

```
                return NSLocalizedString("Away", comment: "Away")
```

```
case .stayArm:
```

```
    return NSLocalizedString("Home", comment: "Home")
```

```
case .nightArm:
```

```
    return NSLocalizedString("Night", comment: "Night")
```

```
case .disarm:
```

```
    return NSLocalizedString("Disarm", comment: "Disarm")
```

```
@unknown default:
```

```
    return ""
```

```
}
```

```
}
```

```
default:
```

```
    break
```

```
}
```

```
return nil
```

```
}
```

```
var supportsEventNotification: Bool {
```

```
    return
```

```
    self.properties.contains(NSNotification.Name.HMCharacteristicPropertySupportsEvent.r  
awValue)
```

```
}
```

```
/// - returns: A string representing the value in a localized way, e.g. `"24%"` or `"354°"
```

```
var localizedValueDescription: String {
```

```
    if let value = value {
```

```
        return self.localizedDescriptionForValue(value as AnyObject)
```

```
    }
```

```
    return ""
```

```
}
```

```
/// - returns: The decoration for the characteristic's units, localized, e.g. `""%"` or `""°"
```

```

var localizedUnitDecoration: String {

    if let units = self.metadata?.units {

        switch units {

            case HMCharacteristicMetadataUnitsCelsius:

                return NSLocalizedString("°C", comment: "Degrees Celsius")

            case HMCharacteristicMetadataUnitsArcDegree:

                return NSLocalizedString("°", comment: "Arc Degrees")

            case HMCharacteristicMetadataUnitsFahrenheit:

                return NSLocalizedString("°F", comment: "Degrees Fahrenheit")

            case HMCharacteristicMetadataUnitsPercentage:

                return NSLocalizedString("%", comment: "Percentage")

            default: break

        }
    }
}

```

```

    }

    return ""

}

/// - returns: The type of the characteristic, e.g. `Current Lock Mechanism State`

var localizedCharacteristicType: String {

    var type = self.localizedDescription

    var localizedDescription: String? = nil

    if isReadOnly {

        localizedDescription = NSLocalizedString("Read Only", comment: "Read Only")

    }

    else if isWriteOnly {

        localizedDescription = NSLocalizedString("Write Only", comment: "Write Only")

    }

    if let localizedDescription = localizedDescription {

```



```

        type = type + " (\(localizedDescription))"

    }

    return type

}

/// - returns: `true` if this characteristic has numeric values that are all integers; `false`
otherwise.

//  var isInteger: Bool {

//      return self.isNumeric && !self.isFloatingPoint

//  }

/**

- returns: `true` if this characteristic has numeric values;

        `false` otherwise.

*/

//  var isNumeric: Bool {

//      guard let metadata = metadata else { return false }

```

```

//    guard let format = metadata.format else { return false }

//    return Constants.numericFormats.contains(format)

// }

/// - returns: `true` if this characteristic is boolean; `false` otherwise.

var isBoolean: Bool {

    guard let metadata = metadata else { return false }

    return metadata.format == HMCharacteristicMetadataFormatBool

}

/**

    - returns: `true` if this characteristic is text-writable;

        `false` otherwise.

*/

var isTextWritable: Bool {

    guard let metadata = metadata else { return false }

    return    metadata.format    ==    HMCharacteristicMetadataFormatString    &&
properties.contains(HMCharacteristicPropertyWritable)

```

```
}
```

```
/**
```

```
- returns: `true` if this characteristic has numeric values
```

```
that are all floating point; `false` otherwise.
```

```
*/
```

```
var isFloatingPoint: Bool {
```

```
    guard let metadata = metadata else { return false }
```

```
    return metadata.format == HMCharacteristicMetadataFormatFloat
```

```
}
```

```
/// - returns: `true` if characteristic is read only; `false` otherwise.
```

```
var isReadOnly: Bool {
```

```
    return !properties.contains(HMCharacteristicPropertyWritable) &&
```

```
        properties.contains(HMCharacteristicPropertyReadable)
```

```
}
```

```

/**

- returns: `true` if this characteristic is write only;

        `false` otherwise.

*/

var isWriteOnly: Bool {

    return !properties.contains(HMCharacteristicPropertyReadable) &&

        properties.contains(HMCharacteristicPropertyWritable)

}

/**

- returns: `true` if this characteristic is the 'Identify'

        characteristic; `false` otherwise.

*/

var isIdentify: Bool {

    return self.characteristicType == HMCharacteristicTypeIdentify

}

```

/**

- returns: The number of possible values that this characteristic can contain.

The standard formula for the number of values between two numbers is

$((\text{greater} - \text{lesser}) + 1)$, and this takes step value into account.

*/

```
var numberOfChoices: Int {  
  
    guard let metadata = metadata, let minimumValue = metadata.minimumValue as? Int  
    else { return 0 }  
  
    guard let maximumValue = metadata.maximumValue as? Int else { return 0 }  
  
    var range = maximumValue - minimumValue  
  
    if let stepValue = metadata.stepValue as? Double {  
  
        range = Int(Double(range) / stepValue)  
  
    }  
  
    return range + 1  
  
}
```

/// - returns: All of the possible values that this characteristic can contain.

```
var allPossibleValues: [AnyObject]? {
```

```

// guard self.isInteger else { return nil }

guard let metadata = metadata, let stepValue = metadata.stepValue as? Double else
{ return nil }

let choices = Array(0..

```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The ``HMEventTrigger+Convenience`` methods are used for analyzing and deconstructing ``HMEventTrigger`` objects.

```
*/
```

```
import HomeKit
```

```
import MapKit
```

```
extension HMEventTrigger {
```

```
/// - returns: The first `CLCircularRegion` in the location event regions.
```

```

var firstCircularLocationEventRegion: CLCircularRegion? {

    for event in self.locationEvents {

        if let circularRegion = event.region as? CLCircularRegion {

            return circularRegion

        }

    }

    return nil

}

```

/// - returns: All events in the trigger which are `HMLocationEvent`s.

```

var locationEvents: [HMLocationEvent] {

    return events.filter { $0 is HMLocationEvent } as! [HMLocationEvent]

}

```

/// - returns: `true` if the trigger contains a location event, `false` otherwise.

```

var isLocationEvent: Bool {

    for event in events {

```



```

        if event is HMLocationEvent {

            return true

        }

    }

    return false

}

/**

- returns: The first `HMEvent` in the event list that is

        an `HMLocationEvent`, `nil` otherwise.

*/

var locationEvent: HMLocationEvent? {

    for event in events {

        if let locationEvent = event as? HMLocationEvent {

            return locationEvent

        }

    }

}

```

```

        return nil
    }

    /**
     - returns: All `HMEvent`s in the events list that are
        `HMCharacteristicEvent`s.

    */

    var characteristicEvents: [HMCharacteristicEvent<NSCopying>] {

        return events.filter { $0 is HMCharacteristicEvent<NSCopying> } as!
        [HMCharacteristicEvent<NSCopying>]

    }

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `HMHome+Properties` methods provide convenience methods for deconstructing `HMHome` objects.

```
*/
```

```
import HomeKit
```

```
extension HMHome {
```

```
    /// All the services within all the accessories within the home.
```

```
    var allServices: [HMService] {
```

```
        return accessories.reduce([], { (accumulator, accessory) -> [HMService] in
```

```
            return accumulator + accessory.services.filter { return !accumulator.contains($0) }
```

```
        })
```

```
    }
```

```
    /// All the characteristics within all of the services within the home.
```

```
    var allCharacteristics: [HMCharacteristic] {
```

```
        return allServices.reduce([], { (accumulator, service) -> [HMCharacteristic] in
```

```

        return accumulator + service.characteristics.filter
    { return !accumulator.contains($0) }

    })

}

```

```

/**

```

- returns: A dictionary mapping localized service types to an array
of all services of that type.

```

*/

```

```

var serviceTable: [String: [HMService]] {

```

```

    var serviceDictionary = [String: [HMService]]()

```

```

    for service in self.allServices {

```

```

        if !service.isControlType {

```

```

            continue

```

```

        }

```

```

        let serviceType = service.localizedDescription

```

```

        var existingServices: [HMService] = serviceDictionary[serviceType] ??
[HMService]()

```

```

        existingServices.append(service)

```

```

        serviceDictionary[service.localizedDescription] = existingServices

```

```

    }

```

```

    for (serviceType, services) in serviceDictionary {

```

```

        serviceDictionary[serviceType] = services.sorted {

```

```

            return $0.accessory!.name.localizedCompare($1.accessory!.name)
            == .orderedAscending

```

```

        }

```

```

    }

```

```

    return serviceDictionary

```

```

}

```

```

/// - returns: All rooms in the home, including `roomForEntireHome`.

```

```

var allRooms: [HMRoom] {

```

```

    let allRooms = [self.roomForEntireHome()] + self.rooms

```

```

    return allRooms.sortByLocalizedNames()

}

/// - returns: `true` if the current user is the admin of this home; `false` otherwise.

var isAdmin: Bool {

    return self.homeAccessControl(for: currentUser).isAdmin

}

/// - returns: All accessories which are 'control accessories'.

var sortedControlAccessories: [HMAccessory] {

    let filteredAccessories = self.accessories.filter { accessory -> Bool in

        for service in accessory.services {

            if service.isControlType {

                return true

            }

        }

        return false
    }
}

```

```

    }

    return filteredAccessories.sortByLocalizedNames()

}

/**

- parameter identifier: The UUID to look up.

- returns: The accessory within the receiver that matches the given UUID,

           or nil if there is no accessory with that UUID.

*/

func accessoryWithIdentifier(_ identifier: UUID) -> HMAccessory? {

    for accessory in self.accessories {

        if accessory.uniqueIdentifier == identifier {

            return accessory

        }

    }

    return nil
}

```

```
}
```

```
/**
```

- parameter identifiers: An array of `NSUUID`'s that match accessories in the receiver.

- returns: An array of `HMAccessory` instances corresponding to

the UUIDs passed in.

```
*/
```

```
func accessoriesWithIdentifiers(_ identifiers: [UUID]) -> [HMAccessory] {
```

```
    return self.accessories.filter { accessory in
```

```
        identifiers.contains(accessory.uniqueIdentifier)
```

```
    }
```

```
}
```

```
/**
```

Searches through the home's accessories to find the accessory

that is bridging the provided accessory.

- parameter accessory: The bridged accessory.

- returns: The accessory bridging the bridged accessory.

*/

```
func bridgeForAccessory(_ accessory: HMAccessory) -> HMAccessory? {  
  
    if !accessory.isBridged {  
  
        return nil  
  
    }  
  
    for bridge in self.accessories {  
  
        if let identifiers = bridge.uniqueIdentifiersForBridgedAccessories ,  
           identifiers.contains(accessory.uniqueIdentifier) {  
  
            return bridge  
  
        }  
  
    }  
  
    return nil  
  
}
```

/**

- parameter room: The room.

- returns: The name of the room, appending "Default Room" if the room

is the home's `roomForEntireHome`

*/

```
func nameForRoom(_ room: HMRoom) -> String {
```

```
    if room == self.roomForEntireHome() {
```

```
        let defaultRoom = NSLocalizedString("Default Room", comment: "Default Room")
```

```
        return room.name + " (\(defaultRoom))"
```

```
    }
```

```
    return room.name
```

```
}
```

/**

- parameter zone: The zone.

- parameter rooms: A list of rooms to add to the final list.

- returns: A list of rooms that exist in the home and have not yet been added to this zone.

*/

```
func roomsNotAlreadyInZone(_ zone: HMZone, includingRooms rooms: [HMRoom]?  
= nil) -> [HMRoom] {
```

```
    var filteredRooms = self.rooms.filter { room in
```

```
        return !zone.rooms.contains(room)
```

```
    }
```

```
    if let rooms = rooms {
```

```
        filteredRooms += rooms
```

```
    }
```

```
    return filteredRooms
```

```
}
```

/**

- parameter home: The home.

- parameter serviceGroup: The service group.

- parameter services: A list of services to add to the final list.

- returns: A list of services that exist in the home and have not yet been added to this service group.

*/

```
func servicesNotAlreadyInServiceGroup(_ serviceGroup: HMServiceGroup,
includingServices services: [HMService]? = nil) -> [HMService] {
```

```
    var filteredServices = self.allServices.filter { service in
```

```
        /*
```

```
            Exclude services that are already in the service group
```

```
            and the accessory information service.
```

```
        */
```

```
        return !serviceGroup.services.contains(service) && service.serviceType !=
HMServiceTypeAccessoryInformation
```

```
    }
```

```
    if let services = services {
```

```
        filteredServices += services
```

```
    }
```

```
    return filteredServices
```

```
}
```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The ``HMService+Properties`` methods provide convenience methods for deconstructing ``HMService`` objects.

```
*/
```

```
import HomeKit
```

```
extension HMService {
```

```
    struct Constants {
```

```
        static let serviceMap = [
```

```
            HMServiceTypeLightbulb:    NSLocalizedString("Lightbulb",    comment:
"Lightbulb"),
```

```

        HMSERVICE_TYPE_FAN: NSLocalizedString("Fan", comment: "Fan")

    ]

}

/**

- parameter serviceType: The service type.

- returns: A localized description of that service type or
           the original `type` string if one cannot be found.

*/

class func localizedDescriptionForServiceType(_ type: String) -> String {

    return Constants.serviceMap[type] ?? type

}

/// - returns: `true` if this service supports the `associatedServiceType` property; `false`
otherwise.

var supportsAssociatedServiceType: Bool {

```

```
        return self.serviceType == HMSERVICE_TYPE_OUTLET || self.serviceType ==  
HMSERVICE_TYPE_SWITCH
```

```
    }
```

```
/// - returns: `true` if this service is a 'control type'; `false` otherwise.
```

```
var isControlType: Bool {
```

```
    let noncontrolTypes = [HMSERVICE_TYPE_ACCESSORY_INFORMATION,  
HMSERVICE_TYPE_LOCK_MANAGEMENT]
```

```
    return !noncontrolTypes.contains(self.serviceType)
```

```
}
```

```
/**
```

```
    - returns: The valid associated service types for this service,
```

```
        e.g. `HMSERVICE_TYPE_FAN` or `HMSERVICE_TYPE_LIGHTBULB`
```

```
*/
```

```
class var validAssociatedServiceTypes: [String] {
```

```
    return [HMSERVICE_TYPE_FAN, HMSERVICE_TYPE_LIGHTBULB]
```

```
}
```

```
}

//

// HTTPHeaders.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,

// EXPRESS OR
```


// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

import Foundation

/// An order-preserving and case-insensitive representation of HTTP headers.

public struct HTTPHeaders {

private var headers: [HTTPHeader] = []

/// Creates an empty instance.

```
public init() {}
```

```
/// Creates an instance from an array of `HTTPHeader`s. Duplicate case-insensitive  
names are collapsed into the last
```

```
/// name and value encountered.
```

```
public init(_ headers: [HTTPHeader]) {
```

```
    self.init()
```

```
    headers.forEach { update($0) }
```

```
}
```

```
/// Creates an instance from a `[String: String]`. Duplicate case-insensitive names are  
collapsed into the last name
```

```
/// and value encountered.
```

```
public init(_ dictionary: [String: String]) {
```

```
    self.init()
```

```
    dictionary.forEach { update(HTTPHeader(name: $0.key, value: $0.value)) }
```

```
}
```

```
/// Case-insensitively updates or appends an `HTTPHeader` into the instance using the  
provided `name` and `value`.
```

```
///
```

```
/// - Parameters:
```

```
/// - name: The `HTTPHeader` name.
```

```
/// - value: The `HTTPHeader` value.
```

```
public mutating func add(name: String, value: String) {
```

```
    update(HTTPHeader(name: name, value: value))
```

```
}
```

```
/// Case-insensitively updates or appends the provided `HTTPHeader` into the instance.
```

```
///
```

```
/// - Parameter header: The `HTTPHeader` to update or append.
```

```
public mutating func add(_ header: HTTPHeader) {
```

```
    update(header)
```

```
}
```

```
/// Case-insensitively updates or appends an `HTTPHeader` into the instance using the
provided `name` and `value`.
```

```
///
```

```
/// - Parameters:
```

```
/// - name: The `HTTPHeader` name.
```

```
/// - value: The `HTTPHeader` value.
```

```
public mutating func update(name: String, value: String) {

    update(HTTPHeader(name: name, value: value))

}
```

```
/// Case-insensitively updates or appends the provided `HTTPHeader` into the instance.
```

```
///
```

```
/// - Parameter header: The `HTTPHeader` to update or append.
```

```
public mutating func update(_ header: HTTPHeader) {

    guard let index = headers.index(of: header.name) else {

        headers.append(header)

    }

    return
```

```
}
```

```
headers.replaceSubrange(index...index, with: [header])
```

```
}
```

```
/// Case-insensitively removes an `HTTPHeader`, if it exists, from the instance.
```

```
///
```

```
/// - Parameter name: The name of the `HTTPHeader` to remove.
```

```
public mutating func remove(name: String) {
```

```
    guard let index = headers.index(of: name) else { return }
```

```
    headers.remove(at: index)
```

```
}
```

```
/// Sort the current instance by header name.
```

```
public mutating func sort() {
```

```
    headers.sort { $0.name < $1.name }
```

```
}
```

```
/// Returns an instance sorted by header name.
```

```
///
```

```
/// - Returns: A copy of the current instance sorted by name.
```

```
public func sorted() -> HTTPHeaders {
```

```
    return HTTPHeaders(headers.sorted { $0.name < $1.name })
```

```
}
```

```
/// Case-insensitively find a header's value by name.
```

```
///
```

```
/// - Parameter name: The name of the header to search for, case-insensitively.
```

```
///
```

```
/// - Returns: The value of header, if it exists.
```

```
public func value(for name: String) -> String? {
```

```
    guard let index = headers.index(of: name) else { return nil }
```

```

        return headers[index].value
    }

    /// Case-insensitively access the header with the given name.

    ///
    /// - Parameter name: The name of the header.

    public subscript(_ name: String) -> String? {

        get { return value(for: name) }

        set {

            if let value = newValue {

                update(name: name, value: value)

            } else {

                remove(name: name)

            }

        }

    }

```

```

    /// The dictionary representation of all headers.

    ///

    /// This representation does not preserve the current order of the instance.

    public var dictionary: [String: String] {

        let namesAndValues = headers.map { ($0.name, $0.value) }

        return Dictionary(namesAndValues, uniquingKeysWith: { _, last in last })

    }

}

extension HTTPHeaders: ExpressibleByDictionaryLiteral {

    public init(dictionaryLiteral elements: (String, String)...) {

        self.init()

        elements.forEach { update(name: $0.0, value: $0.1) }

    }

}

```



```

extension HTTPHeaders: ExpressibleByArrayLiteral {

    public init(arrayLiteral elements: HTTPHeader...) {

        self.init(elements)

    }

}

extension HTTPHeaders: Sequence {

    public func makeIterator() -> IndexingIterator<[HTTPHeader]> {

        return headers.makeIterator()

    }

}

extension HTTPHeaders: Collection {

    public var startIndex: Int {

        return headers.startIndex

    }

}

```

```
public var endIndex: Int {  
  
    return headers.endIndex  
  
}
```

```
public subscript(position: Int) -> HTTPHeader {  
  
    return headers[position]  
  
}
```

```
public func index(after i: Int) -> Int {  
  
    return headers.index(after: i)  
  
}  
  
}
```

```
extension HTTPHeaders: CustomStringConvertible {  
  
    public var description: String {  
  
        return headers.map { $0.description }  
  
    }  
  
}
```

```

        .joined(separator: "\n")

    }

}

// MARK: - HTTPHeader

/// A representation of a single HTTP header's name / value pair.

public struct HTTPHeader: Hashable {

    /// Name of the header.

    public let name: String

    /// Value of the header.

    public let value: String

    /// Creates an instance from the given `name` and `value`.

    ///

    /// - Parameters:

```

```

    /// - name: The name of the header.

    /// - value: The value of the header.

    public init(name: String, value: String) {

        self.name = name

        self.value = value

    }

}

extension HTTPHeader: CustomStringConvertible {

    public var description: String {

        return "\\(name): \\(value)"

    }

}

extension HTTPHeader {

    /// Returns an `Accept` header.

    ///

```

/// - Parameter value: The `Accept` value.

/// - Returns: The header.

```
public static func accept(_ value: String) -> HTTPHeader {  
  
    return HTTPHeader(name: "Accept", value: value)  
  
}
```

/// Returns an `Accept-Charset` header.

///

/// - Parameter value: The `Accept-Charset` value.

/// - Returns: The header.

```
public static func acceptCharset(_ value: String) -> HTTPHeader {  
  
    return HTTPHeader(name: "Accept-Charset", value: value)  
  
}
```

/// Returns an `Accept-Language` header.

///

/// Alamofire offers a default Accept-Language header that accumulates and encodes the system's preferred languages.

```
/// Use `HTTPHeader.defaultAcceptLanguage`.
```

```
///
```

```
/// - Parameter value: The `Accept-Language` value.
```

```
///
```

```
/// - Returns: The header.
```

```
public static func acceptLanguage(_ value: String) -> HTTPHeader {  
  
    return HTTPHeader(name: "Accept-Language", value: value)  
  
}
```

```
/// Returns an `Accept-Encoding` header.
```

```
///
```

```
/// Alamofire offers a default accept encoding value that provides the most common  
values. Use
```

```
/// `HTTPHeader.defaultAcceptEncoding`.
```

```
///
```

```
/// - Parameter value: The `Accept-Encoding` value.
```

```
///
```

```
/// - Returns: The header
```

```

public static fun acceptEncoding(_ value: String) -> HTTPHeader {

    return HTTPHeader(name: "Accept-Encoding", value: value)

}

```

/// Returns a `Basic` `Authorization` header using the `username` and `password` provided.

```

///

```

```

/// - Parameters:

```

```

/// - username: The username of the header.

```

```

/// - password: The password of the header.

```

```

///

```

```

/// - Returns: The header.

```

```

public static fun authorization(username: String, password: String) -> HTTPHeader {

    let credential = Data("\(username):\(password)".utf8).base64EncodedString()

    return authorization("Basic \(credential)")

}

```

```
/// Returns a `Bearer` `Authorization` header using the `bearerToken` provided
```

```
///
```

```
/// - Parameter bearerToken: The bearer token.
```

```
///
```

```
/// - Returns: The header.
```

```
public static func authorization(bearerToken: String) -> HTTPHeader {
```

```
    return authorization("Bearer \(bearerToken)")
```

```
}
```

```
/// Returns an `Authorization` header.
```

```
///
```

```
/// Alamofire provides built-in methods to produce `Authorization` headers. For a Basic  
`Authorization` header use
```

```
/// `HTTPHeader.authorization(username:password:)` . For a Bearer `Authorization`  
header, use
```

```
/// `HTTPHeader.authorization(bearerToken:)` .
```

```
///
```

```
/// - Parameter value: The `Authorization` value.
```



```

///

/// - Returns:      The header.

public static func authorization(_ value: String) -> HTTPHeader {

    return HTTPHeader(name: "Authorization", value: value)

}


/// Returns a `Content-Disposition` header.

///

/// - Parameter value: The `Content-Disposition` value.

///

/// - Returns:      The header.

public static func contentDisposition(_ value: String) -> HTTPHeader {

    return HTTPHeader(name: "Content-Disposition", value: value)

}


/// Returns a `Content-Type` header.

///

```

/// All Alamofire `ParameterEncoding`s and `ParameterEncoder`s set the `Content-Type` of the request, so it may not be necessary to manually

/// set this value.

///

/// - Parameter value: The `Content-Type` value.

///

/// - Returns: The header.

```
public static func contentType(_ value: String) -> HTTPHeader {  
  
    return HTTPHeader(name: "Content-Type", value: value)  
  
}
```

/// Returns a `User-Agent` header.

///

/// - Parameter value: The `User-Agent` value.

///

/// - Returns: The header.

```
public static func userAgent(_ value: String) -> HTTPHeader {  
  
    return HTTPHeader(name: "User-Agent", value: value)  
  
}
```

```
}  
  
}
```

```
extension Array where Element == HTTPHeader {
```

```
  /// Case-insensitively finds the index of an `HTTPHeader` with the provided name, if it  
  exists.
```

```
  func index(of name: String) -> Int? {  
  
    let lowercasedName = name.lowercased()  
  
    return firstIndex { $0.name.lowercased() == lowercasedName }  
  
  }  
  
}
```

```
// MARK: - Defaults
```

```
public extension HTTPHeaders {
```

```
  /// The default set of `HTTPHeaders` used by Alamofire. Includes `Accept-Encoding`,  
  `Accept-Language`, and
```

```
  /// `User-Agent`.
```

```

static let `default`: HTTPHeaders = [.defaultAcceptEncoding,

                                     .defaultAcceptLanguage,

                                     .defaultUserAgent]

}

extension HTTPHeader {

    /// Returns Alamofire's default `Accept-Encoding` header, appropriate for the encodings
    supported by particular OS

    /// versions.

    ///

    /// See the [Accept-Encoding HTTP header
documentation](https://tools.ietf.org/html/rfc7230#section-4.2.3) .

    public static let defaultAcceptEncoding: HTTPHeader = {

        let encodings: [String]

        if #available(iOS 11.0, macOS 10.13, tvOS 11.0, watchOS 4.0, *) {

            encodings = ["br", "gzip", "deflate"]

        } else {

            encodings = ["gzip", "deflate"]

```

```
}
```

```
return .acceptEncoding(encodings.qualityEncoded())
```

```
}()
```

```
/// Returns Alamofire's default `Accept-Language` header, generated by querying  
`Locale` for the user's
```

```
/// `preferredLanguages`.
```

```
///
```

```
/// See the [Accept-Language HTTP header  
documentation](https://tools.ietf.org/html/rfc7231#section-5.3.5).
```

```
public static let defaultAcceptLanguage: HTTPHeader = {
```

```
    .acceptLanguage(Locale.preferredLanguages.prefix(6).qualityEncoded())
```

```
}()
```

```
/// Returns Alamofire's default `User-Agent` header.
```

```
///
```

```
/// See the [User-Agent header  
documentation](https://tools.ietf.org/html/rfc7231#section-5.5.3).
```

```

///

/// Example: `iOS Example/1.0 (org.alamofire.iOS-Example; build:1; iOS 13.0.0)
Alamofire/5.0.0`

public static let defaultUserAgent: HTTPHeader = {

    let userAgent: String = {

        if let info = Bundle.main.infoDictionary {

            let executable = info[kCFBundleExecutableKey as String] as? String ??
"Unknown"

            let bundle = info[kCFBundleIdentifierKey as String] as? String ?? "Unknown"

            let appVersion = info["CFBundleShortVersionString"] as? String ?? "Unknown"

            let appBuild = info[kCFBundleVersionKey as String] as? String ?? "Unknown"

            let osNameVersion: String = {

                let version = ProcessInfo.processInfo.operatingSystemVersion

                let versionString =
                =
                "\(version.majorVersion).\\(version.minorVersion).\\(version.patchVersion)"

                // swiftformat:disable indent

                let osName: String = {

                    #if os(iOS)

```

```
        return "iOS"

    #elseif os(watchOS)

        return "watchOS"

    #elseif os(tvOS)

        return "tvOS"

    #elseif os(macOS)

        return "macOS"

    #elseif os(Linux)

        return "Linux"

    #else

        return "Unknown"

    #endif

}()

// swiftformat:enable indent

return "\(osName) \(versionString)"

}()
```

```
let alamofireVersion = "Alamofire/(version)"
```

```
return      "\\(executable)^(appVersion)      (\\(bundle);      build:\\(appBuild);  
\\(osNameVersion)) \\(alamofireVersion)"  
  
}
```

```
return "Alamofire"
```

```
}()
```

```
return .userAgent(userAgent)
```

```
}()
```

```
}
```

```
extension Collection where Element == String {
```

```
func qualityEncoded() -> String {
```

```
return enumerated().map { index, encoding in
```

```
let quality = 1.0 - (Double(index) * 0.1)
```



```

        return "\\(encoding);q=\\(quality)"

    }.joined(separator: ", ")

}

}

```

// MARK: - System Type Extensions

```

extension URLRequest {

    /// Returns `allHTTPHeaderFields` as `HTTPHeaders`.

    public var headers: HTTPHeaders {

        get { return allHTTPHeaderFields.map(HTTPHeaders.init) ?? HTTPHeaders() }

        set { allHTTPHeaderFields = newValue.dictionary }

    }

}

```

```

extension HTTPURLResponse {

    /// Returns `allHeaderFields` as `HTTPHeaders`.

```

```

public var headers: HTTPHeaders {

    return (allHeaderFields as? [String: String]).map(HTTPHeaders.init) ??
HTTPHeaders()

}

}

```

```

public extension URLSessionConfiguration {

    /// Returns `httpAdditionalHeaders` as `HTTPHeaders`.

    var headers: HTTPHeaders {

        get { return (httpAdditionalHeaders as? [String: String]).map(HTTPHeaders.init) ??
HTTPHeaders() }

        set { httpAdditionalHeaders = newValue.dictionary }

    }

}

//

// HTTPMethod.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

```

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

/// Type representing HTTP methods. Raw `String` value is stored and compared case-
sensitively, so

/// `HTTPMethod.get != HTTPMethod(rawValue: "get")`.

///

/// See <https://tools.ietf.org/html/rfc7231#section-4.3>

public struct HTTPMethod: RawRepresentable, Equatable, Hashable {

/// `CONNECT` method.

public static let connect = HTTPMethod(rawValue: "CONNECT")

/// `DELETE` method.

public static let delete = HTTPMethod(rawValue: "DELETE")

/// `GET` method.

public static let get = HTTPMethod(rawValue: "GET")

```

/// `HEAD` method.

public static let head = HTTPMethod(rawValue: "HEAD")

/// `OPTIONS` method.

public static let options = HTTPMethod(rawValue: "OPTIONS")

/// `PATCH` method.

public static let patch = HTTPMethod(rawValue: "PATCH")

/// `POST` method.

public static let post = HTTPMethod(rawValue: "POST")

/// `PUT` method.

public static let put = HTTPMethod(rawValue: "PUT")

/// `TRACE` method.

public static let trace = HTTPMethod(rawValue: "TRACE")


public let rawValue: String


public init(rawValue: String) {

    self.rawValue = rawValue

```

```
}

}

//

// Highlight.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc(ChartHighlight)
```

```

open class Highlight: NSObject

{

    /// the x-value of the highlighted value

    fileprivate var _x = Double.nan


    /// the y-value of the highlighted value

    fileprivate var _y = Double.nan


    /// the x-pixel of the highlight

    private var _xPx = CGFloat.nan


    /// the y-pixel of the highlight

    private var _yPx = CGFloat.nan


    /// the index of the data object - in case it refers to more than one

    @objc open var dataIndex = Int(-1)

```

```
/// the index of the dataset the highlighted value is in
```

```
fileprivate var _dataSetIndex = Int(0)
```

```
/// index which value of a stacked bar entry is highlighted
```

```
///
```

```
/// default: -1
```

```
fileprivate var _stackIndex = Int(-1)
```

```
/// the axis the highlighted value belongs to
```

```
private var _axis: YAxis.AxisDependency = YAxis.AxisDependency.left
```

```
/// the x-position (pixels) on which this highlight object was last drawn
```

```
@objc open var drawX: CGFloat = 0.0
```

```
/// the y-position (pixels) on which this highlight object was last drawn
```

```
@objc open var drawY: CGFloat = 0.0
```



```

public override init()

{

    super.init()

}

/// - Parameters:

/// - x: the x-value of the highlighted value

/// - y: the y-value of the highlighted value

/// - xPx: the x-pixel of the highlighted value

/// - yPx: the y-pixel of the highlighted value

/// - dataIndex: the index of the Data the highlighted value belongs to

/// - dataSetIndex: the index of the DataSet the highlighted value belongs to

/// - stackIndex: references which value of a stacked-bar entry has been selected

/// - axis: the axis the highlighted value belongs to

@objc public init(

    x: Double, y: Double,

    xPx: CGFloat, yPx: CGFloat,

```

```
    dataIndex: Int,  
  
    dataSetIndex: Int,  
  
    stackIndex: Int,  
  
    axis: YAxis.AxisDependency)  
  
{  
  
    super.init()  
  
  
    _x = x  
  
    _y = y  
  
    _xPx = xPx  
  
    _yPx = yPx  
  
    self.dataIndex = dataIndex  
  
    _dataSetIndex = dataSetIndex  
  
    _stackIndex = stackIndex  
  
    _axis = axis  
  
}
```

```

/// - Parameters:

/// - x: the x-value of the highlighted value

/// - y: the y-value of the highlighted value

/// - xPx: the x-pixel of the highlighted value

/// - yPx: the y-pixel of the highlighted value

/// - dataSetIndex: the index of the DataSet the highlighted value belongs to

/// - stackIndex: references which value of a stacked-bar entry has been selected

/// - axis: the axis the highlighted value belongs to

@objc public convenience init(

    x: Double, y: Double,

    xPx: CGFloat, yPx: CGFloat,

    dataSetIndex: Int,

    stackIndex: Int,

    axis: YAxis.AxisDependency)

{

    self.init(x: x, y: y, xPx: xPx, yPx: yPx,

        dataIndex: 0,

```

```

        dataSetIndex: dataSetIndex,

        stackIndex: stackIndex,

        axis: axis)

    }

/// - Parameters:

///  - x: the x-value of the highlighted value

///  - y: the y-value of the highlighted value

///  - xPx: the x-pixel of the highlighted value

///  - yPx: the y-pixel of the highlighted value

///  - dataIndex: the index of the Data the highlighted value belongs to

///  - dataSetIndex: the index of the DataSet the highlighted value belongs to

///  - stackIndex: references which value of a stacked-bar entry has been selected

///  - axis: the axis the highlighted value belongs to

@objc public init(

    x: Double, y: Double,

    xPx: CGFloat, yPx: CGFloat,

```

```

    dataSetIndex: Int,

    axis: YAxis.AxisDependency)

{

    super.init()

    _x = x

    _y = y

    _xPx = xPx

    _yPx = yPx

    _dataSetIndex = dataSetIndex

    _axis = axis

}

/// - Parameters:

/// - x: the x-value of the highlighted value

/// - y: the y-value of the highlighted value

/// - dataSetIndex: the index of the DataSet the highlighted value belongs to

```

```
/// - dataIndex: The data index to search in (only used in CombinedChartView currently)
```

```
@objc public init(x: Double, y: Double, dataSetIndex: Int, dataIndex: Int = -1)
```

```
{
```

```
    _x = x
```

```
    _y = y
```

```
    _dataSetIndex = dataSetIndex
```

```
    self.dataIndex = dataIndex
```

```
}
```

```
/// - Parameters:
```

```
/// - x: the x-value of the highlighted value
```

```
/// - dataSetIndex: the index of the DataSet the highlighted value belongs to
```

```
/// - stackIndex: references which value of a stacked-bar entry has been selected
```

```
@objc public convenience init(x: Double, dataSetIndex: Int, stackIndex: Int)
```

```
{
```

```
    self.init(x: x, y: Double.nan, dataSetIndex: dataSetIndex)
```

```
    _stackIndex = stackIndex
```

```
}
```

```
@objc open var x: Double { return _x }
```

```
@objc open var y: Double { return _y }
```

```
@objc open var xPx: CGFloat { return _xPx }
```

```
@objc open var yPx: CGFloat { return _yPx }
```

```
@objc open var dataSetIndex: Int { return _dataSetIndex }
```

```
@objc open var stackIndex: Int { return _stackIndex }
```

```
@objc open var axis: YAxis.AxisDependency { return _axis }
```

```
@objc open var isStacked: Bool { return _stackIndex >= 0 }
```

```
/// Sets the x- and y-position (pixels) where this highlight was last drawn.
```

```
@objc open func setDraw(x: CGFloat, y: CGFloat)
```

```
{
```

```
    self.drawX = x
```

```
    self.drawY = y
```

```

    }

    /// Sets the x- and y-position (pixels) where this highlight was last drawn.

    @objc open func setDraw(pt: CGPoint)

    {

        self.drawX = pt.x

        self.drawY = pt.y

    }

    // MARK: NSObject

    open override var description: String

    {

        return "Highlight, x: \(_x), y: \(_y), dataIndex (combined charts): \(dataIndex),
        dataSetIndex: \(_dataSetIndex), stackIndex (only stacked barentry): \(_stackIndex)"

    }

}

```



```

// MARK: Equatable

extension Highlight /*: Equatable*/ {

    open override func isEqual(_ object: Any?) -> Bool {

        guard let object = object as? Highlight else { return false }

        if self === object

        {

            return true

        }

        return _x == object._x

        && _y == object._y

        && dataIndex == object.dataIndex

        && _dataSetIndex == object._dataSetIndex

        && _stackIndex == object._stackIndex

    }

```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `HomeKitObjectCollection` is a model object for the `HomeViewController`.

It manages arrays of HomeKit objects.

```
*/
```

```
import HomeKit
```

```
/// Represents the all different types of HomeKit objects.
```

```
enum HomeKitObjectSection: Int {
```

```
    case actionSet, trigger
```

```
    static let count = 2
```

```
}
```

```
/**
```

```
    Manages internal lists of HomeKit objects to allow for
```

```
    save insertion into a table view.
```

```
*/
```

```
class HomeKitObjectCollection {
```

```
    // MARK: Properties
```

```
    // var accessories = [HMAccessory]()
```

```
    //var rooms = [HMRoom]()
```

```
    //var zones = [HMZone]()
```

```
    var actionSets = [HMActionSet]()
```

```
    var triggers = [HMTrigger]()
```

```
    //var serviceGroups = [HMServiceGroup]()
```

```
/**
```

Adds an object to the collection by finding its corresponding array and appending the object to it.

- parameter object: The HomeKit object to append.

```
*/  
  
func append(_ object: AnyObject) {  
  
    switch object {  
  
        case let actionSet as HMActionSet:  
  
            actionSets.append(actionSet)  
  
            actionSets = actionSets.sortByTypeAndLocalizedString()  
  
//        case let accessory as HMAccessory:  
  
//            accessories.append(accessory)  
  
//            accessories = accessories.sortByLocalizedString()  
  
//        case let room as HMRoom:  
  
//            rooms.append(room)
```

```

//      rooms = rooms.sortByLocalizedNames()

//      case let zone as HMZone:

//          zones.append(zone)

//          zones = zones.sortByLocalizedNames()


case let trigger as HMTrigger:

    triggers.append(trigger)

    triggers = triggers.sortByLocalizedNames()


//      case let serviceGroup as HMServiceGroup:

//          serviceGroups.append(serviceGroup)

//          serviceGroups = serviceGroups.sortByLocalizedNames()


default:

    break

}

```

```
}
```

```
/**
```

Creates an `NSIndexPath` representing the location of the

HomeKit object in the table view.

- parameter object: The HomeKit object to find.

- returns: The `NSIndexPath` representing the location of

the HomeKit object in the table view.

```
*/
```

```
func indexPathOfObject(_ object: AnyObject) -> IndexPath? {
```

```
    switch object {
```

```
        case let actionSet as HMActionSet:
```

```
            if let index = actionSets.firstIndex(of: actionSet) {
```

```
                return IndexPath(row: index, section:  
HomeKitObjectSection.actionSet.rawValue)
```

```
            }
```

```

//      case let accessory as HMAccessory:

//          if let index = accessories.firstIndex(of: accessory) {

//                                  return IndexPath(row: index, section:
HomeKitObjectSection.accessory.rawValue)

//          }


//      case let room as HMRoom:

//          if let index = rooms.firstIndex(of: room) {

//              return IndexPath(row: index, section: HomeKitObjectSection.room.rawValue)

//          }


//      case let zone as HMZone:

//          if let index = zones.firstIndex(of: zone) {

//              return IndexPath(row: index, section: HomeKitObjectSection.zone.rawValue)

//          }

```

```

case let trigger as HMTrigger:

```

```

        if let index = triggers.firstIndex(of: trigger) {

            return IndexPath(row: index, section:
HomeKitObjectSection.trigger.rawValue)

        }


//      case let serviceGroup as HMServiceGroup:

//      if let index = serviceGroups.firstIndex(of: serviceGroup) {

//      return IndexPath(row: index, section:
HomeKitObjectSection.serviceGroup.rawValue)

//      }


        default: break

    }


    return nil

}


/**

```


Removes a HomeKit object from the collection.

- parameter object: The HomeKit object to remove.

*/

```
func remove(_ object: AnyObject) {  
  
    switch object {  
  
        case let actionSet as HMActionSet:  
  
            if let index = actionSets.firstIndex(of: actionSet) {  
  
                actionSets.remove(at: index)  
  
            }  
  
  
  
  
        // case let accessory as HMAccessory:  
  
        //     if let index = accessories.firstIndex(of: accessory) {  
  
        //         accessories.remove(at: index)  
  
        //     }  
  
  
  
  
        // case let room as HMRoom:
```

```

//      if let index = rooms.firstIndex(of: room) {

//      rooms.remove(at: index)

//      }


//      case let zone as HMZone:

//      if let index = zones.firstIndex(of: zone) {

//      zones.remove(at: index)

//      }


case let trigger as HMTrigger:

    if let index = triggers.firstIndex(of: trigger) {

        triggers.remove(at: index)

    }


//      case let serviceGroup as HMServiceGroup:

//      if let index = serviceGroups.firstIndex(of: serviceGroup) {

//      serviceGroups.remove(at: index)

```

```
//      }
```

```
    default:
```

```
        break
```

```
    }
```

```
}
```

```
/**
```

Provides the array of `NSObject`'s corresponding to the provided section.

- parameter section: A `HomeKitObjectSection`.

- returns: An array of `NSObject`'s corresponding to the provided section.

```
*/
```

```
func objectsForSection(_ section: HomeKitObjectSection) -> [NSObject] {
```

```
    switch section {
```

```
        case .actionSet:
```

```
        return actionSets

//      case .accessory:

//          return accessories


//      case .room:

//          return rooms


//      case .zone:

//          return zones


    case .trigger:

        return triggers


//      case .serviceGroup:

//          return serviceGroups
```

```

        default:

            return []

        }

    }

/**

Provides an `HomeKitObjectSection` for a given object.

- parameter object: A HomeKit object.

- returns: The corresponding `HomeKitObjectSection`

*/

class func sectionForObject(_ object: AnyObject?) -> HomeKitObjectSection? {

    switch object {

        case is HMActionSet:

            return .actionSet

```

```
//      case is HMAccessory:
```

```
//          return .accessory
```

```
//      case is HMZone:
```

```
//          return .zone
```

```
//      case is HMRoom:
```

```
//          return .room
```

```
case is HMTrigger:
```

```
    return .trigger
```

```
//      case is HMServiceGroup:
```

```
//          return .serviceGroup
```

```
default:
```

```
    return nil
```

```

    }

}

/**

Reloads all internal structures based on the provided home.

- parameter home: The `HMHome` with which to reset the collection.

*/

func resetWithHome(_ home: HMHome) {

//    accessories = home.accessories.sortByLocalizedNames()

//    rooms = home.allRooms

//    zones = home.zones.sortByLocalizedNames()

    actionSets = home.actionSets.sortByTypeAndLocalizedNames()

    triggers = home.triggers.sortByLocalizedNames()

//    serviceGroups = home.serviceGroups.sortByLocalizedNames()

}

}

```

/*

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The 'HomeListConfigurationViewController' allows for the creation and deletion of homes.

*/

import UIKit

import HomeKit

// Represents the sections in the 'HomeListConfigurationViewController'.

enum HomeListSection: Int {

case homes, primaryHome

static let count = 2

}


```
/**
```

A `HomeListViewController` subclass which allows the user to add and remove homes and set the primary home.

```
*/
```

```
class HomeListConfigurationViewController: HomeListViewController {
```

```
    // MARK: Types
```

```
    struct Identifiers {
```

```
        static let addHomeCell = "AddHomeCell"
```

```
        static let noHomesCell = "NoHomesCell"
```

```
        static let primaryHomeCell = "PrimaryHomeCell"
```

```
        static let homeCell = "HomeCell"
```

```
    }
```

```
    override func viewDidLoad() {
```

```
        navigationController?.navigationBar.prefersLargeTitles = true
```

```

if #available(iOS 13.0, *) {

    let navBarAppearance = UINavigationBarAppearance()

    navBarAppearance.configureWithOpaqueBackground()

    navBarAppearance.titleTextAttributes = [.foregroundColor: UIColor.white]

    navBarAppearance.largeTitleTextAttributes = [.foregroundColor: UIColor.black]

    navBarAppearance.backgroundColor = UIColor.init(hexString: "#a1edab")

    navigationController?.navigationBar.standardAppearance = navBarAppearance

    navigationController?.navigationBar.scrollEdgeAppearance = navBarAppearance

}

self.tableView.contentInset = UIEdgeInsets(top: 44,left: 0,bottom: 0,right: 0);


// Do any additional setup after loading the view.

}

```

```

// MARK: Table View Methods

```

```

/// - returns: The number of sections in the `HomeListSection` enum.

override func numberOfSections(in tableView: UITableView) -> Int {

    return HomeListSection.count

}

/// Provides the number of rows in the section using the internal home's list.

override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {

    switch HomeListSection(rawValue: section) {

        // Add row.

        case .homes?:

            return homes.count + 1

        // 'No homes' row.

        case .primaryHome?:

            return max(homes.count, 1)

```

```

        case nil: fatalError("Unexpected `HomeListSection` raw value.")

    }

}

/**

Generates and configures either a content cell or an add cell using the

provided index path.

*/

override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

    if indexPathIsAdd(indexPath) {

        return tableView.dequeueReusableCell(withIdentifier: Identifiers.addHomeCell,
for: indexPath)

    }

    else if homes.isEmpty {

        return tableView.dequeueReusableCell(withIdentifier: Identifiers.noHomesCell,
for: indexPath)

    }

```

```
let reuseIdentifier: String
```

```
switch HomeListSection(rawValue: indexPath.section) {
```

```
    case .homes?:
```

```
        reuseIdentifier = Identifiers.homeCell
```

```
    case .primaryHome?:
```

```
        reuseIdentifier = Identifiers.primaryHomeCell
```

```
    case nil: fatalError("Unexpected `HomeListSection` raw value.")
```

```
}
```

```
let cell = tableView.dequeueReusableCell(withIdentifier: reuseIdentifier, for:  
indexPath)
```

```
let home = homes[indexPath.row]
```

```
cell.textLabel!.text = home.name
```

```
cell.detailTextLabel?.text = sharedTextForHome(home)
```

```

// Mark the primary home with checkmark.

if HomeListSection(rawValue: indexPath.section) == .primaryHome {

    if home == homeManager.primaryHome {

        cell.accessoryType = .checkmark

    }

    else {

        cell.accessoryType = .none

    }

}

return cell

}

```

```

/// Homes in the list section can be deleted. The add row cannot be deleted.

```

```

override func tableView(_ tableView: UITableView, canEditRowAt indexPath:
IndexPath) -> Bool {

```

```
        return HomeListSection(rawValue: indexPath.section) == .homes
        && !indexPathIsAdd(indexPath)
```

```
    }
```

```
/// Only the 'primary home' section has a title.
```

```
override func tableView(_ tableView: UITableView, titleForHeaderInSection section:
Int) -> String? {
```

```
    if HomeListSection(rawValue: section) == .primaryHome {
```

```
        return NSLocalizedString("Primary Home", comment: "Primary Home")
```

```
    }
```

```
    return nil
```

```
}
```

```
/// Provides subtext about the use of designating a "primary home".
```

```
override func tableView(_ tableView: UITableView, titleForFooterInSection section:
Int) -> String? {
```

```
    if section == HomeListSection.primaryHome.rawValue {
```

```
        return NSLocalizedString("The primary home is used by Siri to route commands if  
the home is not specified.", comment: "Primary Home Description")
```

```
    }
```

```
    return nil
```

```
}
```

```
/**
```

If selecting a regular home, a segue will be performed.

If this method is called, the user either selected the 'add' row,

a primary home cell, or the 'No Homes' cell.

```
*/
```

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:  
IndexPath) {
```

```
    tableView.deselectRow(at: indexPath, animated: true)
```

```
    if indexPathIsAdd(indexPath) {
```

```
        addNewHome()
```

```
    }
```

```
    else if indexPathIsNone(indexPath) {
```



```

        return

    }

    else if HomeListSection(rawValue: indexPath.section) == .primaryHome {

        let newPrimaryHome = homes[indexPath.row]

        updatePrimaryHome(newPrimaryHome)

    }

}

/// Removes the home from HomeKit if the row is deleted.

override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {

    if editingStyle == .delete {

        removeHomeAtIndexPath(indexPath)

    }

}

// MARK: Helper Methods

```

```
/**
```

Updates the primary home in HomeKit and reloads the view.

If the home is already selected, no action is taken.

- parameter newPrimaryHome: The new `HMHome` to set as the primary home.

```
*/
```

```
private func updatePrimaryHome(_ newPrimaryHome: HMHome) {  
  
    guard newPrimaryHome != homeManager.primaryHome else { return }  
  
    homeManager.updatePrimaryHome(newPrimaryHome) { error in  
  
        if let error = error {  
  
            self.displayError(error)  
  
            return  
  
        }  
  
        self.didUpdatePrimaryHome()  
  
    }  
}
```

```
}
```

```
/// Reloads the 'primary home' section.
```

```
private func didUpdatePrimaryHome() {
```

```
    let primaryIndexSet = IndexSet(integer: HomeListSection.primaryHome.rawValue)
```

```
    tableView.reloadSections(primaryIndexSet, with: .automatic)
```

```
}
```

```
/**
```

```
    Removed the home at the specified index path from HomeKit and updates the view.
```

```
    - parameter indexPath: The `NSIndexPath` of the home to remove.
```

```
*/
```

```
private func removeHomeAtIndexPath(_ indexPath: IndexPath) {
```

```
    let home = homes[indexPath.row]
```

```

// Remove the home from the data structure. If it fails, put it back.

didRemoveHome(home)

homeManager.removeHome(home) { error in

    if let error = error {

        self.displayError(error)

        self.didAddHome(home)

        return

    }

}

}

/**

Presents an alert controller so the user can provide a name. If committed,

the home is created.

*/

private func addNewHome() {

    let attributedType = NSAttributedString("Home", comment: "Home")

```

```
let placeholder = NSLocalizedString("Apartment", comment: "Apartment")
```

```
        presentAddAlertWithType(attributedType, placeholder: placeholder)
    { name in

        self.addHomeWithName(name)

    }

}
```

```
/**
```

Removes a home from the internal structure and updates the view.

- parameter home: The `HMHome` to remove.

```
*/
```

```
override func didRemoveHome(_ home: HMHome) {
```

```
    guard let index = homes.firstIndex(of: home) else { return }
```

```
    let indexPath = IndexPath(row: index, section: HomeListSection.homes.rawValue)
```

```
    homes.remove(at: index)
```

```
        let primaryIndexPath = IndexPath(row: index, section:
HomeListSection.primaryHome.rawValue)
```

```
    /*
```

```
        If there aren't any homes, we still want one cell to display 'No Homes'.
```

```
        Just reload.
```

```
    */
```

```
    tableView.beginUpdates()
```

```
    if homes.isEmpty {
```

```
        tableView.reloadRows(at: [primaryIndexPath], with: .fade)
```

```
    }
```

```
    else {
```

```
        tableView.deleteRows(at: [primaryIndexPath], with: .automatic)
```

```
    }
```

```
    tableView.deleteRows(at: [indexPath], with: .automatic)
```

```
    tableView.endUpdates()
```

```
}
```

```

    /// Adds the home to the internal structure and updates the view.

    override func didAddHome(_ home: HMHome) {

        homes.append(home)

        sortHomes()

        guard let newHomeIndex = homes.firstIndex(of: home) else { return }

        let indexPath = IndexPath(row: newHomeIndex, section:
HomeListSection.homes.rawValue)

        let primaryIndexPath = IndexPath(row: newHomeIndex, section:
HomeListSection.primaryHome.rawValue)

        tableView.beginUpdates()

        if homes.count == 1 {

            tableView.reloadRows(at: [primaryIndexPath], with: .fade)

        }

```

```

else {

    tableView.insertRows(at: [primaryIndexPath], with: .automatic)

}

tableView.insertRows(at: [indexPath], with: .automatic)

tableView.endUpdates()

}

/**

Creates a new home with the provided name, adds the home to HomeKit

and reloads the view.

*/

private func addHomeWithName(_ name: String) {

    homeManager.addHome(withName: name) { newHome, error in

        if let error = error {

            self.displayError(error)

        }

        return
    }
}

```



```
}
```

```
self.didAddHome(newHome!)
```

```
}
```

```
}
```

```
/// - returns: `true` if the index path is the 'add row'; `false` otherwise.
```

```
private func indexPathIsAdd(_ indexPath: IndexPath) -> Bool {
```

```
    return HomeListSection(rawValue: indexPath.section) == .homes &&
```

```
        indexPath.row == homes.count
```

```
}
```

```
/// - returns: `true` if the index path is the 'No Homes' cell; `false` otherwise.
```

```
private func indexPathIsNone(_ indexPath: IndexPath) -> Bool {
```

```
    return HomeListSection(rawValue: indexPath.section) == .primaryHome &&  
        homes.isEmpty
```

```
}
```

```

// MARK: HMHomeDelegate Methods

/// Finds the home in the internal structure and reloads the corresponding row.

override func homeDidUpdateName(_ home: HMHome) {

    if let index = homes.firstIndex(of: home) {

        let listIndexPath = IndexPath(row: index, section:
HomeListSection.homes.rawValue)

        let primaryIndexPath = IndexPath(row: index, section:
HomeListSection.primaryHome.rawValue)

        tableView.reloadRows(at: [listIndexPath, primaryIndexPath], with: .automatic)

    }

    else {

        // Just reload the data since we don't know the index path.

        tableView.reloadData()

    }
}

```

```
}
```

```
// MARK: HMHomeManagerDelegate Methods
```

```
/// Reloads the 'primary home' section.
```

```
func homeManagerDidUpdatePrimaryHome(_ manager: HMHomeManager) {
```

```
    didUpdatePrimaryHome()
```

```
}
```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `HomeListViewController` is a superclass that lists the user's homes.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
/// A generic view controller for displaying a list of homes in a home manager.
```

```
class HomeListViewController: HMCatalogViewController, HMHomeManagerDelegate  
{
```

```
    // MARK: Types
```

```
    struct Identifiers {
```

```
        static let homeCell = "HomeCell"
```

```
        static let showHomeSegue = "Show Home"
```

```
    }
```

```
    // MARK: Properties
```

```
    var homes = [HMHome]()
```

```

var homeManager: HMHomeManager {

    return homeStore.homeManager

}

// MARK: View Methods

/// Configures the table view.

override func viewDidLoad() {

    super.viewDidLoad()

    tableView.estimatedRowHeight = 44.0

    tableView.rowHeight = UITableView.automaticDimension

}

/// Resets the list of homes (which will update the view).

override func viewWillAppear(_ animated: Bool) {

```

```

        super.viewWillAppear(animated)

        resetHomesList()
    }

    // MARK: Delegate Registration

    /**
     Registers as the delegate for the home manager and all homes in the internal
     homes list.
    */

    override func registerAsDelegate() {

        homeManager.delegate = self

        for home in homes {

            home.delegate = self
        }
    }

```

```

    /// Sets the home store's current home based on which cell was selected.

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

        super.prepare(for: segue, sender: sender)

        if segue.identifier == Identifiers.showHomeSegue {

            if sender as AnyObject? === self {

                // Don't update the selected home if we sent ourselves here.

                return

            }

            if let indexPath = tableView.indexPath(for: sender as! UITableViewCell) {

                homeStore.home = homes[indexPath.row]

            }

        }

    }

```

```
// MARK: Table View Methods
```

```
/**
```

Provides the number of sections based on the home array count.

Updates the background message for the table view.

- returns: The number of homes in the internal array.

```
*/
```

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
```

```
    let rows = homes.count
```

```
    if rows == 0 {
```

```
        let message = NSLocalizedString("No Homes", comment: "No Homes")
```

```
        setBackgroundMessage(message)
```

```
    }
```

```
    else {
```

```
        setBackgroundMessage(nil)
```



```
}
```

```
return rows
```

```
}
```

```
/**
```

```
    Generates a basic cell for a home.
```

```
    Subtext is provided to tell the user if the home is shared or owned by the user.
```

```
*/
```

```
    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
```

```
        let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.homeCell, for:
indexPath)
```

```
        let home = homes[indexPath.row]
```

```
        cell.textLabel?.text = home.name
```

```
        cell.detailTextLabel?.text = sharedTextForHome(home)
```

```

    return cell

}

// MARK: Helper Methods

/**

Provides an ordering for homes.

Homes are first ordered by their 'shared' status, then by name.

- parameter home1: The first `HMHome`.

- parameter home2: The second `HMHome`.

- returns: `true` if `home1` is ordered before `home2`; `false` otherwise.

*/

private func orderHomes(_ home1: HMHome, home2: HMHome) -> Bool {

    if home1.isAdmin == home2.isAdmin {

```

```
/*
```

```
    We are comparing two shared homes or two of our homes, just compare  
    names.
```

```
*/
```

```
    return home1.name.localizedCompare(home2.name) == .orderedAscending
```

```
    }
```

```
else {
```

```
/*
```

```
    We are comparing a shared home and one of our homes, if home1 is  
    ours, put it first.
```

```
*/
```

```
    return home1.isAdmin
```

```
    }
```

```
}
```

```
/**
```

```
    Regenerates the list of homes using list provided by the home manager.
```

The list is then sorted and the view is reloaded.

```
*/
```

```
private func resetHomesList() {
```

```
    homes = homeManager.homes.sorted(by: orderHomes)
```

```
    tableView.reloadData()
```

```
}
```

```
/// Sorts the list of homes (without reloading from the home manager).
```

```
func sortHomes() {
```

```
    homes.sort(by: orderHomes)
```

```
}
```

```
/**
```

Adds a new home into the internal homes array and inserts the new

row into the table view.

- parameter home: The new `HMHome` that's been added.

```
*/
```

```
func didAddHome(_ home: HMHome) {
```

```
    homes.append(home)
```

```
    sortHomes()
```

```
    if let newHomeIndex = homes.firstIndex(of: home) {
```

```
        let indexPathOfNewHome = IndexPath(row: newHomeIndex, section: 0)
```

```
        tableView.insertRows(at: [indexPathOfNewHome], with: .automatic)
```

```
    }
```

```
}
```

```
/**
```

Removes a home from the internal homes array (if it exists) and

deletes corresponding row from the table view.

- parameter home: The `HMHome` to remove.

*/

```
func didRemoveHome(_ home: HMHome) {
```

```
    guard let removedHomeIndex = homes.firstIndex(of: home) else { return }
```

```
    homes.remove(at: removedHomeIndex)
```

```
    let indexPath = IndexPath(row: removedHomeIndex, section: 0)
```

```
    tableView.deleteRows(at: [indexPath], with: .automatic)
```

```
}
```

/**

- returns: A localized description of who owns the provided home.

- parameter home: The `HMHome` to describe.

*/

```
func sharedTextForHome(_ home: HMHome) -> String {
```

```
    if !home.isAdmin {
```

```

        return NSLocalizedString("Shared with Me", comment: "Shared with Me")
    }

    else {

        return NSLocalizedString("My Home", comment: "My Home")

    }

}

```

// MARK: HMHomeDelegate Methods

/// Finds the cell with corresponds to the provided home and reloads it.

```

func homeDidUpdateName(_ home: HMHome) {

    if let homeIndex = homes.firstIndex(of: home) {

        let indexPath = IndexPath(row: homeIndex, section: 0)

        tableView.reloadRows(at: [indexPath], with: .automatic)

    }

}

```

```
// MARK: HMHomeManagerDelegate Methods
```

```
/**
```

```
    Reloads data and view.
```

```
    This view controller, in most cases, will remain the home manager delegate.
```

```
    For this reason, this method will close all modal views and pop all detail views
```

```
    if the home store's current home is no longer in the home manager's list of homes.
```

```
*/
```

```
func homeManagerDidUpdateHomes(_ manager: HMHomeManager) {
```

```
    registerAsDelegate()
```

```
    resetHomesList()
```

```
    if let home = homeStore.home , !manager.homes.contains(home) {
```

```
        // Close all modal and detail views.
```

```
        dismiss(animated: true, completion: nil)
```



```

        _ = navigationController?.popToRootViewController(animated: true)
    }
}

/// Registers for the delegate of the new home and updates the view.

func homeManager(_ manager: HMHomeManager, didAdd home: HMHome) {

    home.delegate = self

    didAddHome(home)

}

/**

    Removes the home from the current list of homes and updates the view.

    If the removed home was the current home, this view controller will dismiss

    all modals views and pop all detail views.

    */

```

```

func homeManager(_ manager: HMHomeManager, didRemove home: HMHome) {

    didRemoveHome(home)

    guard let selectedHome = homeStore.home , home == selectedHome else { return }

    homeStore.home = nil

    // Close all modal and detail views.

    dismiss(animated: true, completion: nil)

    _ = navigationController?.popToRootViewController(animated: true)

}

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

A central hub for accessory delegates.

```
*/
```

```
import HomeKit
```

```
extension HomeStore {
```

```
    /// Registers an object as an accessory delegate.
```

```
    func addAccessoryDelegate(_ delegate: NSObject) {
```

```
        accessoryDelegates.insert(delegate)
```

```
    }
```

```
    /// Deregisters a particular accessory delegate.
```

```
    func removeAccessoryDelegate(_ delegate: NSObject) {
```

```
        accessoryDelegates.remove(delegate)
```

```
    }
```

```
    /// Deregisters all accessory delegates.
```

```
    func removeAllAccessoryDelegates() {
```

```

        accessoryDelegates.removeAll()

    }

}

extension HomeStore: HMAccessoryDelegate {

    // The home store's only interest in the accessory updates is distributing them

    // to the objects that have registered as accessory delegates. Each of these

    // methods therefore simply passes along the call to all the items in the set,

    // after first ensuring that the item is in fact an accessory delegate.

    func accessory(_ accessory: HMAccessory, didUpdateNameFor service: HMService) {

        accessoryDelegates.forEach {

            guard let delegate = $0 as? HMAccessoryDelegate else { return }

            delegate.accessory?(accessory, didUpdateNameFor: service)

        }

    }

}

```

```

func accessoryDidUpdateReachability(_ accessory: HMAccessory) {

    accessoryDelegates.forEach {

        guard let delegate = $0 as? HMAccessoryDelegate else { return }

        delegate.accessoryDidUpdateReachability?(accessory)

    }

}

```

```

func accessoryDidUpdateServices(_ accessory: HMAccessory) {

    accessoryDelegates.forEach {

        guard let delegate = $0 as? HMAccessoryDelegate else { return }

        delegate.accessoryDidUpdateServices?(accessory)

    }

}

```

```

func accessory(_ accessory: HMAccessory, service: HMService, didUpdateValueFor
characteristic: HMCharacteristic) {

    accessoryDelegates.forEach {

```

```

guard let delegate = $0 as? HMAccessoryDelegate else { return }

delegate.accessory?(accessory, service: service, didUpdateValueFor: characteristic)

}

}

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

A central hub for home delegates.

```
*/
```

```
import HomeKit
```

```
extension HomeStore {
```

```
    /// Registers an object as a home delegate.
```

```

func addHomeDelegate(_ delegate: NSObject) {

    homeDelegates.insert(delegate)

}

/// Deregisters a particular home delegate.

func removeHomeDelegate(_ delegate: NSObject) {

    homeDelegates.remove(delegate)

}

/// Deregisters all home delegates.

func removeAllHomeDelegates() {

    homeDelegates.removeAll()

}

}

extension HomeStore: HMHomeDelegate {

```

```
// The home store's only interest in the home updates is distributing them

// to the objects that have registered as home delegates. Each of these

// methods therefore simply passes along the call to all the items in the set,

// after first ensuring that the item is in fact a home delegate.
```

```
func homeDidUpdateName(_ home: HMHome) {

    homeDelegates.forEach {

        guard let delegate = $0 as? HMHomeDelegate else { return }

        delegate.homeDidUpdateName?(home)

    }

}
```

```
func home(_ home: HMHome, didAdd accessory: HMAccessory) {

    homeDelegates.forEach {

        guard let delegate = $0 as? HMHomeDelegate else { return }

        delegate.home?(home, didAdd: accessory)

    }

}
```



```
}
```

```
func home(_ home: HMHome, didUpdate room: HMRoom, for accessory:
HMAccessory) {
```

```
    homeDelegates.forEach {
```

```
        guard let delegate = $0 as? HMHomeDelegate else { return }
```

```
        delegate.home?(home, didUpdate: room, for: accessory)
```

```
    }
```

```
}
```

```
func home(_ home: HMHome, didRemove accessory: HMAccessory) {
```

```
    homeDelegates.forEach {
```

```
        guard let delegate = $0 as? HMHomeDelegate else { return }
```

```
        delegate.home?(home, didRemove: accessory)
```

```
    }
```

```
}
```

```
func home(_ home: HMHome, didAdd room: HMRoom) {
```

```

homeDelegates.forEach {

    guard let delegate = $0 as? HMHomeDelegate else { return }

    delegate.home?(home, didAdd: room)

}

}

```

```

func home(_ home: HMHome, didUpdateNameFor room: HMRoom) {

    homeDelegates.forEach {

        guard let delegate = $0 as? HMHomeDelegate else { return }

        delegate.home?(home, didUpdateNameFor: room)

    }

}

```

```

func home(_ home: HMHome, didRemove room: HMRoom) {

    homeDelegates.forEach {

        guard let delegate = $0 as? HMHomeDelegate else { return }

        delegate.home?(home, didRemove: room)

    }

}

```

```

    }

}

func home(_ home: HMHome, didEncounterError error: Error, for accessory:
HMAccessory) {

    homeDelegates.forEach {

        guard let delegate = $0 as? HMHomeDelegate else { return }

        delegate.home?(home, didEncounterError: error, for: accessory)

    }

}

}

```

```

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

A singleton for holding the home manager.

```

*/

```

```

import UIKit

import HomeKit

class HomeStore: NSObject, HMHomeManagerDelegate {

    static let shared = HomeStore()

    // MARK: Properties

    /// The current 'selected' home.

    var home: HMHome?

    var homeDelegates = Set<NSObject>()

    var accessoryDelegates = Set<NSObject>()

    /// The singleton home manager.

    var homeManager = HMHomeManager()

```

```
}
```

```
extension HomeStore {
```

```
    /// Updates the name of a service and informs all accessory delegates.
```

```
    func updateService(_ service: HMService, name: String) {
```

```
        service.updateName(name) { error in
```

```
            if let error = error {
```

```
                print(error)
```

```
            } else if let accessory = service.accessory {
```

```
                self.accessory(accessory, didUpdateNameFor: service)
```

```
            }
```

```
        }
```

```
    }
```

```
    /// Moves an accessory to a given room and informs all the home delegates.
```

```
    func move(_ accessory: HMAccessory, in home: HMHome, to room: HMRoom) {
```

```

home.assignAccessory(accessory, to: room) { error in

    if let error = error {

        print(error)

    } else {

        self.home(home, didUpdate: room, for: accessory)

    }

}

}

```

/// Removes an accessory from a home and informs all the home delegates.

```

func remove(_ accessory: HMAccessory, from home: HMHome) {

    home.removeAccessory(accessory) { error in

        if let error = error {

            print(error)

        } else {

            self.home(home, didRemove: accessory)

        }

    }

}

```

```
    }  
  
    }  
  
}  
  
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `HomeController` displays all of the HomeKit objects in a selected home.

```
*/
```

```
import Foundation
```

```
import UIKit
```

```
import HomeKit
```

```
/// Distinguishes between the three types of cells in the `HomeController`.
```

```
enum HomeCellType {
```

```
/// Represents an actual object in HomeKit.
```

```
case object
```

```
/// Represents an "Add" row for users to select to create an object in HomeKit.
```

```
case add
```

```
/// The cell is displaying text to show the user that no objects exist in this section.
```

```
case none
```

```
}
```

```
/**
```

A view controller that displays all elements within a home.

It contains separate sections for Accessories, Rooms, Zones, Action Sets,

Triggers, and Service Groups.

```
*/
```

```
class HomeViewController: HMCatalogViewController, HMAccessoryDelegate {
```

```
  // MARK: Types
```



```

struct Identifiers {

    static let addCell = "AddCell"

    static let disabledAddCell = "DisabledAddCell"

    //    static let accessoryCell = "AccessoryCell"

    //    static let unreachableAccessoryCell = "UnreachableAccessoryCell"

    //    static let roomCell = "RoomCell"

    //static let zoneCell = "ZoneCell"

    //    static let userCell = "UserCell"

    static let actionSetCell = "ActionSetCell"

    static let triggerCell = "TriggerCell"

    //    static let serviceGroupCell = "ServiceGroupCell"

    static let addTimerTriggerSegue = "Add Timer Trigger"

    static let addCharacteristicTriggerSegue = "Add Characteristic Trigger"

    static let addLocationTriggerSegue = "Add Location Trigger"

    static let addActionSetSegue = "Add Action Set"

    static let addAccessoriesSegue = "Add Accessories"

```

```

//    static let showRoomSegue = "Show Room"

//static let showZoneSegue = "Show Zone"

static let showActionSetSegue = "Show Action Set"

static let showServiceGroupSegue = "Show Service Group"

//    static let showAccessorySegue = "Show Accessory"

static let modifyAccessorySegue = "Modify Accessory"

static let showTimerTriggerSegue = "Show Timer Trigger"

static let showLocationTriggerSegue = "Show Location Trigger"

static let showCharacteristicTriggerSegue = "Show Characteristic Trigger"

}


// MARK: Properties


/// A structure to maintain internal arrays of HomeKit objects.

private var objectCollection = HomeKitObjectCollection()


// MARK: View Methods

```

```

/**

Determines the destination of the segue and passes the correct

HomeKit object onto the next view controller.

*/

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

    guard let sender = sender as? UITableViewCell else { return }

    guard let indexPath = tableView.indexPath(for: sender) else { return }

    let homeKitObject = homeKitObjectAtIndexPath(indexPath)

    let destinationViewController = segue.intendedDestinationViewController

    switch segue.identifier! {

//        case Identifiers.showRoomSegue:

//            let roomVC = destinationViewController as! RoomViewController

//            roomVC.room = homeKitObject as? HMRoom

```

```
// case Identifiers.showZoneSegue:

//     let zoneViewController = destinationViewController as! ZoneViewController

//     zoneViewController.homeZone = homeKitObject as? HMZone
```

```
case Identifiers.showActionSetSegue:
```

```
    let actionSetVC = destinationViewController as! ActionSetViewController
```

```
    actionSetVC.actionSet = homeKitObject as? HMActionSet
```

```
case Identifiers.showServiceGroupSegue:
```

```
    let serviceGroupVC = destinationViewController as!
    ServiceGroupViewController
```

```
    serviceGroupVC.serviceGroup = homeKitObject as? HMServiceGroup
```

```
// case Identifiers.showAccessorySegue:
```

```
//     let detailVC = destinationViewController as! ServicesViewController
```

```
/*
```

The services view controller is generic, we need to provide

`showsFavorites` to display the stars next to characteristics.

```

        */

//        detailVC.accessory = homeKitObject as? HMAccessory

//        detailVC.showsFavorites = true

//        detailVC.cellDelegate = AccessoryUpdateController()

        case Identifiers.modifyAccessorySegue:

            let        addAccessoryVC        =        destinationViewController        as!
ModifyAccessoryViewController

            addAccessoryVC.accessory = homeKitObject as? HMAccessory

        case Identifiers.showTimerTriggerSegue:

            let triggerVC = destinationViewController as! TimerTriggerViewController

            triggerVC.trigger = homeKitObject as? HMTimerTrigger

        case Identifiers.showLocationTriggerSegue:

            let triggerVC = destinationViewController as! LocationTriggerViewController

            triggerVC.trigger = homeKitObject as? HMEventTrigger

```

```

        case Identifiers.showCharacteristicTriggerSegue:

            let triggerVC = destinationViewController as!
            CharacteristicTriggerViewController

            triggerVC.trigger = homeKitObject as? HMEventTrigger

        default:

            print("Received unknown segue identifier: \(segue.identifier ?? "nil").")

        }

    }

    /// Configures the table view.

    override func awakeFromNib() {

        super.awakeFromNib()

        tableView.estimatedRowHeight = 44.0

        tableView.rowHeight = UITableView.automaticDimension

    }

    /// Sets the navigation title and reloads view.

```

```

override func viewWillAppear(_ animated: Bool) {

    super.viewWillAppear(animated)

    self.navigationItem.setHidesBackButton(true, animated: true);

    navigationItem.title = home.name


    reloadTable()

}


// MARK: Delegate Registration


/**

Registers as the delegate for the home store's current home

and all accessories in the home.

*/

override func registerAsDelegate() {

    super.registerAsDelegate()

```

```

    for accessory in home.accessories {

        accessory.delegate = self

    }

}

// MARK: Helper Methods

/// Resets the object collection and reloads the view.

private func reloadData() {

    objectCollection.resetWithHome(home)

    tableView.reloadData()

}

/**

Determines the type of the cell based on the index path.

- parameter indexPath: The `NSIndexPath` of the cell.

```


- returns: The `HomeCellType` for cell.

*/

```
private func cellTypeForIndexPath(_ indexPath: IndexPath) -> HomeCellType {
```

```
    guard let section = HomeKitObjectSection(rawValue: indexPath.section) else  
    { return .none }
```

```
    let objectCount = objectCollection.objectsForSection(section).count
```

```
    if objectCount == 0 {
```

```
        // No objects -- this is either an 'Add Row' or a 'None Row'.
```

```
        return home.isAdmin ? .add : .none
```

```
    }
```

```
    else if indexPath.row == objectCount {
```

```
        return .add
```

```
    }
```

```
    else {
```

```
        return .object
```

```
}  
  
}
```

```
/// Reloads the trigger section.
```

```
private func updateTriggerAddRow() {
```

```
    let triggerSection = IndexSet(integer: HomeKitObjectSection.trigger.rawValue)
```

```
    tableView.reloadSections(triggerSection, with: .automatic)
```

```
}
```

```
/// Reloads the action set section.
```

```
private func updateActionSetSection() {
```

```
    let actionSetSection = IndexSet(integer: HomeKitObjectSection.actionSet.rawValue)
```

```
    tableView.reloadSections(actionSetSection, with: .automatic)
```

```
    updateTriggerAddRow()
```

```
}
```

```
/// - returns: `true` if there are accessories within the home; `false` otherwise.
```

```
// private var canAddActionSet: Bool {
```

```
//     return !objectCollection.accessories.isEmpty
```

```
// }
```

```
/// - returns: `true` if there are action sets (with actions) within the home; `false` otherwise.
```

```
private var canAddTrigger: Bool {
```

```
    return objectCollection.actionSets.contains { actionSet in
```

```
        return !actionSet.actions.isEmpty
```

```
    }
```

```
}
```

```
/**
```

Provides the 'HomeKit object' ('AnyObject?') at the specified index path.

- parameter indexPath: The `NSIndexPath` of the object.

- returns: The HomeKit object.

*/

```
private func homeKitObjectAtIndexPath(_ indexPath: IndexPath) -> AnyObject? {
```

```
    if cellTypeForIndexPath(indexPath) != .object {
```

```
        return nil
```

```
    }
```

```
    if let section = HomeKitObjectSection(rawValue: indexPath.section) {
```

```
        return objectCollection.objectsForSection(section)[indexPath.row]
```

```
    }
```

```
    return nil
```

```
}
```

```
// MARK: Table View Methods
```

```

    /// - returns: The number of `HomeKitObjectSection`s.

    override func numberOfSections(in tableView: UITableView) -> Int {

        return HomeKitObjectSection.count

    }

    /// - returns: Localized titles for each of the HomeKit sections.

    override func tableView(_ tableView: UITableView, titleForHeaderInSection section:
Int) -> String? {

        switch HomeKitObjectSection(rawValue: section) {

            //      case .accessory?:

            //          return NSLocalizedString("Accessories", comment: "Accessories")


            //      case .room?:

            //          return NSLocalizedString("Rooms", comment: "Rooms")


            //      case .zone?:

            //          return NSLocalizedString("Zones", comment: "Zones")

```

```

//      case .user?:

//          return NSLocalizedString("Users", comment: "Users")

//

case .actionSet?:

    return NSLocalizedString("Scenes", comment: "Scenes")


case .trigger?:

    return NSLocalizedString("Triggers", comment: "Triggers")


//      case .serviceGroup?:

//          return NSLocalizedString("Service Groups", comment: "Service Groups")

//

case nil:

    fatalError("Unexpected `HomeKitObjectSection` raw value.")

}

```

```

    }

    /// - returns: Localized text for the 'add row'.

    private func titleForAddRowInSection(_ section: HomeKitObjectSection) -> String {

        switch section {

        //      case .accessory:

        //          return NSLocalizedString("Add Accessory...", comment: "Add Accessory")

        //      case .room:

        //          return NSLocalizedString("Add Room...", comment: "Add Room")

        //      case .zone:

        //          return NSLocalizedString("Add Zone...", comment: "Add Zone")

        //      case .user:

        //          return NSLocalizedString("Manage Users...", comment: "Manage Users")

```

```

        case .actionSet:

            return NSLocalizedString("Add Scene...", comment: "Add Scene")

        case .trigger:

            return NSLocalizedString("Add Trigger...", comment: "Add Trigger")

        //      case .serviceGroup:

        //          return NSLocalizedString("Add Service Group...", comment: "Add Service
        Group")

        }

    }

    /// - returns: Localized text for the 'none row'.

    private func titleForNoneRowInSection(_ section: HomeKitObjectSection) -> String {

        switch section {

        //      case .accessory:

        //          return NSLocalizedString("No Accessories...", comment: "No Accessories")

```



```

//      case .room:

//          return NSLocalizedString("No Rooms...", comment: "No Rooms")


//      case .zone:

//          return NSLocalizedString("No Zones...", comment: "No Zones")


//      case .user:

//          // We only ever list 'Manage Users'.

//          return NSLocalizedString("Manage Users...", comment: "Manage Users")


case .actionSet:

    return NSLocalizedString("No Scenes...", comment: "No Scenes")


case .trigger:

    return NSLocalizedString("No Triggers...", comment: "No Triggers")


//      case .serviceGroup:

```

```
//          return NSLocalizedString("No Service Groups...", comment: "No Service
Groups")

    }

}
```

/// - returns: Localized descriptions for HomeKit object types.

```
override func tableView(_ tableView: UITableView, titleForFooterInSection section:
Int) -> String? {
```

```
    switch HomeKitObjectSection(rawValue: section) {
```

```
//        case .zone?:
```

```
//            return NSLocalizedString("Zones are optional collections of rooms.", comment:
"Zones Description")
```

```
//        case .user?:
```

```
//            return NSLocalizedString("Users can control the accessories in your home. You
can share your home with anybody with an iCloud account.", comment: "Users
Description")
```

```
        case .actionSet?:
```

```
        return NSLocalizedString("Scenes (action sets) represent a state of your home. You must have at least one paired accessory to create a scene.", comment: "Scenes Description")
```

```
    case .trigger?:
```

```
        return NSLocalizedString("Triggers set scenes at specific times, when you get to locations, or when a characteristic is in a specific state. You must have created at least one scene with an action to create a trigger.", comment: "Trigger Description")
```

```
//    case .serviceGroup?:
```

```
//        return NSLocalizedString("Service groups organize services in a custom way. For example, add a subset of lights in your living room to control them without controlling all the lights in the living room.", comment: "Service Group Description")
```

```
    case nil:
```

```
        fatalError("Unexpected `HomeKitObjectSection` raw value.")
```

```
    default:
```

```
        return nil
```

```
}
```

```
}
```

```
/**
```

Provides the number of rows in each HomeKit object section.

Most sections just return the object count, but we also handle special cases.

```
*/
```

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
```

```
    let sectionEnum = HomeKitObjectSection(rawValue: section)!
```

```
    // Only "Manage Users" button is in the Users section
```

```
    //    if sectionEnum == .user {
```

```
        //        return 1
```

```
    //    }
```

```
    let objectCount = objectCollection.objectsForSection(sectionEnum).count
```

```
    if home.isAdmin {
```

```
        // For add row.
```

```

        return objectCount + 1

    }

    else {

        // Always show at least one row in the section.

        return max(objectCount, 1)

    }

}

/// Generates a cell based on it's computed type.

override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

    switch cellTypeForIndexPath(indexPath) {

        case .add:

            return self.tableView(tableView, addCellForRowAtIndexPath: indexPath)

        case .object:

            return self.tableView(tableView, homeKitObjectCellForRowAtIndexPath:
indexPath)

```

```

        case .none:

            return self.tableView(tableView, noneCellForRowAtIndexPath: indexPath)

        }

    }

    /// Generates a 'none cell' with a localized title.

    private func tableView(_ tableView: UITableView, noneCellForRowAtIndexPath
indexPath: IndexPath) -> UITableViewCell {

        let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.disabledAddCell,
for: indexPath)

        let section = HomeKitObjectSection(rawValue: indexPath.section)!

        cell.textLabel!.text = titleForNoneRowInSection(section)

        return cell

    }

```

```

/**

Generates an 'add cell' with a localized title.

In some cases, the 'add cell' will be 'disabled' because the user is not

allowed to perform the action.

*/

private func tableView(_ tableView: UITableView, addCellForRowAtIndexPath
indexPath: IndexPath) -> UITableViewCell {

    var reuseIdentifier = Identifiers.addCell

    let section = HomeKitObjectSection(rawValue: indexPath.section)

    //    if (!canAddActionSet && section == .actionSet) ||

    //        (!canAddTrigger && section == .trigger) || !home.isAdmin {

    //        reuseIdentifier = Identifiers.disabledAddCell

    //    }

```

```
    let cell = tableView.dequeueReusableCell(withIdentifier: reuseIdentifier, for:
indexPath)
```

```
    cell.textLabel!.text = titleForAddRowInSection(section!)
```

```
    return cell
```

```
}
```

```
/**
```

```
    Produces the cell reuse identifier based on the section.
```

```
    - parameter indexPath: The `NSIndexPath` of the cell.
```

```
    - returns: The cell reuse identifier.
```

```
*/
```

```
private func reuseIdentifierForIndexPath(_ indexPath: IndexPath) -> String {
```

```
    switch HomeKitObjectSection(rawValue: indexPath.section) {
```

```
//        case .accessory?:
```



```

//      let accessory = homeKitObjectAtIndexPath(indexPath) as! HMAccessory

//      return accessory.isReachable ? Identifiers.accessoryCell :
Identifiers.unreachableAccessoryCell

//

//      case .room?:

//      return Identifiers.roomCell


//      case .zone?:

//      return Identifiers.zoneCell


//      case .user?:

//      return Identifiers.userCell


case .actionSet?:

    return Identifiers.actionSetCell


case .trigger?:

    return Identifiers.triggerCell

```

```

//      case .serviceGroup?:

//      return Identifiers.serviceGroupCell

    case nil:

        fatalError("Unexpected `HomeKitObjectSection` raw value.")

    }

}

/// Generates a cell for the HomeKit object at the specified index path.

private func tableView(_ tableView: UITableView,
homeKitObjectCellForRowAtIndexPath indexPath: IndexPath) -> UITableViewCell {

    // Grab the object associated with this indexPath.

    let homeKitObject = homeKitObjectAtIndexPath(indexPath)

    // Get the name of the object.

    let name: String

    switch HomeKitObjectSection(rawValue: indexPath.section) {

```

```

//      case .accessory?:

//          let accessory = homeKitObject as! HMAccessory

//          name = accessory.name


//      case .room?:

//          let room = homeKitObject as! HMRoom

//          name = self.home.nameForRoom(room)


//      case .zone?:

//          let zone = homeKitObject as! HMZone

//          name = zone.name


//      case .user?:

//          name = ""


case .actionSet?:

    let actionSet = homeKitObject as! HMActionSet

    name = actionSet.name

```

```
case .trigger?:
```

```
    let trigger = homeKitObject as! HMTrigger
```

```
    name = trigger.name
```

```
//     case .serviceGroup?:
```

```
//         let serviceGroup = homeKitObject as! HMServiceGroup
```

```
//         name = serviceGroup.name
```

```
case nil:
```

```
    fatalError("Unexpected `HomeKitObjectSection` raw value.")
```

```
}
```

```
// Grab the appropriate reuse identifier for this index path.
```

```
let reuseIdentifier = reuseIdentifierForIndexPath(indexPath)
```

```
        let cell = tableView.dequeueReusableCell(withIdentifier: reuseIdentifier, for:
indexPath)
```

```
        cell.textLabel?.text = name
```

```
        return cell
```

```
    }
```

```
/// Allows users to remove HomeKit object rows if they are the admin of the home.
```

```
override func tableView(_ tableView: UITableView, canEditRowAt indexPath:
IndexPath) -> Bool {
```

```
    let homeKitObject = homeKitObjectAtIndexPath(indexPath)
```

```
    if !home.isAdmin {
```

```
        return false
```

```
    }
```

```
    if let actionSet = homeKitObject as? HMActionSet, actionSet.isBuiltIn {
```

```
        // We cannot remove built-in action sets.
```

```

        return false
    }

    // Any row that is not an 'add' row, and is not the roomForEntireHome, can be removed.

    return !(homeKitObject as? NSObject == home.roomForEntireHome() ||
cellTypeForIndexPath(indexPath) == .add)

}

/// Removes the HomeKit object at the specified index path.

override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {

    if editingStyle == .delete {

        let homeKitObject = homeKitObjectAtIndexPath(indexPath)!

        // Remove the object from the data structure. If it fails put it back.

        didRemoveHomeKitObject(homeKitObject)

        removeHomeKitObject(homeKitObject) { error in

            guard let error = error else { return }

```

```

        self.displayError(error)

        self.didAddHomeKitObject(homeKitObject)

    }

}

}

/// Handles cell selection based on the cell type.

override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {

    tableView.deselectRow(at: indexPath, animated: true)

    let cell = tableView.cellForRow(at: indexPath)!

    guard cell.selectionStyle != .none else { return }

    guard let section = HomeKitObjectSection(rawValue: indexPath.section) else {

        fatalError("Unexpected `HomeKitObjectSection` raw value.")

    }

```

```

    if cellTypeForIndexPath(indexPath) == .add{

        switch section {

        //      case .accessory:

        //          browseForAccessories()


        //      case .room:

        //          addNewRoom()


        //      case .zone:

        //          addNewZone()


        //      case .user:

        //          manageUsers()


        case .actionSet:

            addNewActionSet()

```



```

        case .trigger:

            addNewTrigger()

//        case .serviceGroup:

//            addNewServiceGroup()

    }

}

else if section == .actionSet {

    let selectedActionSet = homeKitObjectAtIndexPath(indexPath) as! HMActionSet

    executeActionSet(selectedActionSet)

}

}

/// Handles an accessory button tap based on the section.

override func tableView(_ tableView: UITableView,
    accessoryButtonTappedForRowWith indexPath: IndexPath) {

    let cell = tableView.cellForRow(at: indexPath)

```

```

if HomeKitObjectSection(rawValue: indexPath.section) == .trigger {

    let trigger = homeKitObjectAtIndexPath(indexPath)

    switch trigger {

        case is HMTimerTrigger:

            performSegue(withIdentifier: Identifiers.showTimerTriggerSegue, sender:
cell)

        case let eventTrigger as HMEventTrigger:

            if eventTrigger.isLocationEvent {

                performSegue(withIdentifier: Identifiers.showLocationTriggerSegue,
sender: cell)

            }

            else {

                performSegue(withIdentifier: Identifiers.showCharacteristicTriggerSegue,
sender: cell)

            }

    }

}

```

```

        default: break
    }

}

}

```

```
// MARK: Action Methods
```

```
/// Presents an alert controller to allow the user to choose a trigger type.
```

```
private func addNewTrigger() {

    let title = NSLocalizedString("Add Trigger", comment: "Add Trigger")

    let alertController = UIAlertController(title: title, message: nil,
preferredStyle: .actionSheet)

    // Timer trigger

    let timeAction = UIAlertAction(title: NSLocalizedString("Time", comment: "Time"),
style: .default) { _ in

        self.performSegue(withIdentifier: Identifiers.addTimerTriggerSegue, sender: self)

    }
}

```

```

alertController.addAction(timeAction)

// Characteristic trigger

let eventAction = UIAlertAction(title: NSLocalizedString("Characteristic", comment:
"Characteristic"), style: .default) { _ in

    self.performSegue(withIdentifier: Identifiers.addCharacteristicTriggerSegue,
sender: self)

}

alertController.addAction(eventAction)

// Location trigger

let locationAction = UIAlertAction(title: NSLocalizedString("Location", comment:
"Location"), style: .default) { _ in

    self.performSegue(withIdentifier: Identifiers.addLocationTriggerSegue, sender:
self)

}

alertController.addAction(locationAction)

// Cancel

```

```
        let cancelAction = UIAlertAction(title: NSLocalizedString("Cancel", comment:
"Cancel"), style: .cancel, handler: nil)
```

```
        alertController.addAction(cancelAction)
```

```
        // Present alert
```

```
        if let presenter = alertController.popoverPresentationController {
```

```
            presenter.sourceView = self.view
```

```
            presenter.sourceRect = self.view.frame
```

```
            presenter.permittedArrowDirections = UIPopoverArrowDirection(rawValue: 0)
```

```
        }
```

```
        self.present(alertController, animated: true, completion: nil)
```

```
    }
```

```
    /// Navigates into the action set view controller.
```

```
    private func addNewActionSet() {
```

```
        performSegue(withIdentifier: Identifiers.addActionSetSegue, sender: self)
```

```
    }
```

```
/// Navigates into the browse accessory view controller.
```

```
private func browseForAccessories() {
```

```
    performSegue(withIdentifier: Identifiers.addAccessoriesSegue, sender: self)
```

```
}
```

```
// MARK: Dialog Creation Methods
```

```
/// Presents a dialog to name a new room and generates the HomeKit object if committed.
```

```
private func addNewRoom() {
```

```
    presentAddAlertWithType(NSLocalizedString("Room", comment:  
"Room"),
```

```
        placeholder: NSLocalizedString("Living Room", comment: "Living Room"))  
    { roomName in
```

```
        self.addRoomWithName(roomName)
```

```
    }
```

```
}
```

/// Presents a dialog to name a new service group and generates the HomeKit object if committed.

```
private func addNewServiceGroup() {  
  
    presentAddAlertWithAttributeType(NSLocalizedString("Service Group", comment:  
"Service Group"),  
  
        placeholder: NSLocalizedString("Group", comment: "Group")) { groupName in  
  
        self.addServiceGroupWithName(groupName)  
  
    }  
  
}
```

/// Presents a dialog to name a new zone and generates the HomeKit object if committed.

```
private func addNewZone() {  
  
    presentAddAlertWithAttributeType(NSLocalizedString("Zone", comment: "Zone"),  
  
        placeholder: NSLocalizedString("Upstairs", comment: "Upstairs")) { zoneName in  
  
        self.addZoneWithName(zoneName)  
  
    }  
  
}
```

```
// MARK: HomeKit Object Creation and Deletion
```

```
/**
```

Switches based on the type of object attempts to remove the HomeKit object

from the current home.

- parameter object: The HomeKit object to remove.

- parameter completionHandler: The closure to invoke when the removal has been completed.

```
*/
```

```
private func removeHomeKitObject(_ object: AnyObject, completionHandler:
@escaping (Error?) -> Void) {
```

```
    switch object {
```

```
        case let actionSet as HMActionSet:
```

```
            home.removeActionSet(actionSet) { error in
```

```
                completionHandler(error)
```

```
                self.updateActionSetSection()
```

```
        }
```


case let accessory as HMAccessory:

home.removeAccessory(accessory, completionHandler: completionHandler)

case let room as HMRoom:

home.removeRoom(room, completionHandler: completionHandler)

case let zone as HMZone:

home.removeZone(zone, completionHandler: completionHandler)

case let trigger as HMTrigger:

home.removeTrigger(trigger, completionHandler: completionHandler)

case let serviceGroup as HMServiceGroup:

home.removeServiceGroup(serviceGroup, completionHandler: completionHandler)

default:

```

        fatalError("Attempted to remove unknown HomeKit object.")
    }
}

/**

Adds a room to the current home.

- parameter name: The name of the new room.

*/

private func addRoomWithName(_ name: String) {

    home.addRoom(withName: name) { newRoom, error in

        if let error = error {

            self.displayError(error)

            return

        }

        self.didAddHomeKitObject(newRoom)
    }
}

```

```

    }

}

/**

Adds a service group to the current home.

- parameter name: The name of the new service group.

*/

private func addServiceGroupWithName(_ name: String) {

    home.addServiceGroup(withName: name) { newGroup, error in

        if let error = error {

            self.displayError(error)

            return

        }

        self.didAddHomeKitObject(newGroup)

    }

```

```
}
```

```
/**
```

Adds a zone to the current home.

- parameter name: The name of the new zone.

```
*/
```

```
private func addZoneWithName(_ name: String) {
```

```
    home.addZone(withName: name) { newZone, error in
```

```
        if let error = error {
```

```
            self.displayError(error)
```

```
            return
```

```
        }
```

```
        self.didAddHomeKitObject(newZone)
```

```
    }
```

```
}
```

```
/// Presents modal view for managing users.
```

```
private func manageUsers() {
```

```
    home.manageUsers { error in
```

```
        if let error = error {
```

```
            self.displayError(error)
```

```
        }
```

```
    }
```

```
}
```

```
/**
```

Checks to see if an action set has any actions.

If actions exists, the action set will be executed.

Otherwise, the user will be alerted.

- parameter actionSet: The `HMActionSet` to evaluate and execute.

```
*/
```

```

private func executeActionSet(_ actionSet: HMActionSet) {

    if actionSet.actions.isEmpty {

        let alertTitle = NSLocalizedString("Empty Scene", comment: "Empty Scene")

        let alertMessage = NSLocalizedString("This scene is empty. To set this scene, first
add some actions to it.", comment: "Empty Scene Description")

        displayMessage(alertTitle, message: alertMessage)

        return

    }

    home.executeActionSet(actionSet) { error in

        guard let error = error else { return }

        self.displayError(error)

    }

}

```

```
/**
```

Adds the HomeKit object into the object collection and inserts the new row into the section.

- parameter object: The HomeKit object to add.

```
*/
```

```
private func didAddHomeKitObject(_ object: AnyObject?) {  
  
    if let object = object {  
  
        objectCollection.append(object)  
  
        if let newIndexIndexPath = objectCollection.indexPathOfObject(object) {  
  
            tableView.insertRows(at: [newIndexPath], with: .automatic)  
  
        }  
  
    }  
  
}
```

```
/**
```

Finds the `NSIndexPath` of the specified object and reloads it in the table view.

- parameter object: The HomeKit object that was modified.

*/

```
private func didModifyHomeKitObject(_ object: AnyObject?) {  
  
    if let object = object,  
  
        let objectIndexPath = objectCollection.indexPathOfObject(object) {  
  
        tableView.reloadRows(at: [objectIndexPath], with: .automatic)  
  
    }  
  
}
```

/**

Removes the HomeKit object from the object collection and then deletes the row from the section.

- parameter object: The HomeKit object to remove.

*/

```
private func didRemoveHomeKitObject(_ object: AnyObject?) {  
  
    if let object = object,
```



```

        let indexPath = objectCollection.indexPathOfObject(object) {

            objectCollection.remove(object)

            tableView.deleteRows(at: [indexPath], with: .automatic)

        }
    }

```

```

/*

```

The following methods call the above helper methods to handle the addition, removal, and modification of HomeKit objects.

```

*/

```

```

// MARK: HMHomeDelegate Methods

```

```

func homeDidUpdateName(_ home: HMHome) {

    navigationItem.title = home.name

    reloadData()

}

```

```
func home(_ home: HMHome, didAddAccessory accessory: HMAccessory) {  
  
    didAddHomeKitObject(accessory)  
  
    accessory.delegate = self  
  
}
```

```
func home(_ home: HMHome, didRemoveAccessory accessory: HMAccessory) {  
  
    didRemoveHomeKitObject(accessory)  
  
}
```

```
// MARK: Triggers
```

```
func home(_ home: HMHome, didAddTrigger trigger: HMTrigger) {  
  
    didAddHomeKitObject(trigger)  
  
}
```

```
func home(_ home: HMHome, didRemoveTrigger trigger: HMTrigger) {
```

```
        didRemoveHomeKitObject(trigger)

    }
}
```

```
func home(_ home: HMHome, didUpdateNameFor trigger: HMTrigger) {

    didModifyHomeKitObject(trigger)

}
}
```

```
// MARK: Service Groups
```

```
func home(_ home: HMHome, didAddServiceGroup group: HMServiceGroup) {

    didAddHomeKitObject(group)

}
}
```

```
func home(_ home: HMHome, didRemoveServiceGroup group: HMServiceGroup) {

    didRemoveHomeKitObject(group)

}
}
```

```
@objc(home:didUpdateNameForGroup:)
```

```
func home(_ home: HMHome, didUpdateNameFor group: HMServiceGroup) {  
  
    didModifyHomeKitObject(group)  
  
}
```

```
// MARK: Action Sets
```

```
func home(_ home: HMHome, didAddActionSet actionSet: HMActionSet) {  
  
    didAddHomeKitObject(actionSet)  
  
}
```

```
func home(_ home: HMHome, didRemoveActionSet actionSet: HMActionSet) {  
  
    didRemoveHomeKitObject(actionSet)  
  
}
```

```
@objc(home:didUpdateNameForActionSet:)
```

```
func home(_ home: HMHome, didUpdateNameFor actionSet: HMActionSet) {
```

```

        didModifyHomeKitObject(actionSet)

    }

    // MARK: Zones

    func home(_ home: HMHome, didAddZone zone: HMZone) {

        didAddHomeKitObject(zone)

    }

    func home(_ home: HMHome, didRemoveZone zone: HMZone) {

        didRemoveHomeKitObject(zone)

    }

    @objc(home:didUpdateNameForZone:)

    func home(_ home: HMHome, didUpdateNameFor zone: HMZone) {

        didModifyHomeKitObject(zone)

    }

```

```
// MARK: Rooms
```

```
func home(_ home: HMHome, didAddRoom room: HMRoom) {  
  
    didAddHomeKitObject(room)  
  
}
```

```
func home(_ home: HMHome, didRemoveRoom room: HMRoom) {  
  
    didRemoveHomeKitObject(room)  
  
}
```

```
@objc(home:didUpdateNameForRoom:)
```

```
func home(_ home: HMHome, didUpdateNameFor room: HMRoom) {  
  
    didModifyHomeKitObject(room)  
  
}
```

```
// MARK: Accessories
```

```

func accessoryDidUpdateReachability(_ accessory: HMAccessory) {

    didModifyHomeKitObject(accessory)

}

func accessoryDidUpdateName(_ accessory: HMAccessory) {

    didModifyHomeKitObject(accessory)

}

}

//

// HorizontalBarChartRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

```

```

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

#if canImport(Uikit)

    import UIKit

#endif

#if canImport(Cocoa)

    import Cocoa

#endif

open class HorizontalBarChartRenderer: BarChartRenderer

{

    private class Buffer

```



```

{

    var rects = [CGRect]()

}

public override init(dataProvider: BarChartDataProvider, animator: Animator,
viewPortHandler: ViewPortHandler)

{

    super.init(dataProvider: dataProvider, animator: animator, viewPortHandler:
viewPortHandler)

}

// [CGRect] per dataset

private var _buffers = [Buffer]()

open override func initBuffers()

{

    if let barData = dataProvider?.barData

    {

```

```

// Matche buffers count to dataset count

if _buffers.count != barData.dataSetCount

{

    while _buffers.count < barData.dataSetCount

    {

        _buffers.append(Buffer())

    }

    while _buffers.count > barData.dataSetCount

    {

        _buffers.removeLast()

    }

}

for i in stride(from: 0, to: barData.dataSetCount, by: 1)

{

    let set = barData.dataSets[i] as! IBarChartDataSet

    let size = set.entryCount * (set.isStacked ? set.stackSize : 1)

```

```

        if _buffers[i].rects.count != size
        {
            _buffers[i].rects = [CGRect](repeating: CGRect(), count: size)
        }
    }
}

else
{
    _buffers.removeAll()
}
}

```

```

private func prepareBuffer(dataSet: IBarChartDataSet, index: Int)

```

```

{
    guard let
        dataProvider = dataProvider,

        let barData = dataProvider.barData

```

```
else { return }
```

```
let barWidthHalf = barData.barWidth / 2.0
```

```
let buffer = _buffers[index]
```

```
var bufferIndex = 0
```

```
let containsStacks = dataSet.isStacked
```

```
let isInverted = dataProvider.isInverted(axis: dataSet.axisDependency)
```

```
let phaseY = animator.phaseY
```

```
var barRect = CGRect()
```

```
var x: Double
```

```
var y: Double
```

```
for i in stride(from: 0, to: min(Int(ceil(Double(dataSet.entryCount) *  
animator.phaseX)), dataSet.entryCount), by: 1)
```

```
{
```

```
    guard let e = dataSet.entryForIndex(i) as? BarChartDataEntry else { continue }
```

```
let vals = e.yValues
```

```
x = e.x
```

```
y = e.y
```

```
if !containsStacks || vals == nil
```

```
{
```

```
    let bottom = CGFloat(x - barWidthHalf)
```

```
    let top = CGFloat(x + barWidthHalf)
```

```
    var right = isInverted
```

```
        ? (y <= 0.0 ? CGFloat(y) : 0)
```

```
        : (y >= 0.0 ? CGFloat(y) : 0)
```

```
    var left = isInverted
```

```
        ? (y >= 0.0 ? CGFloat(y) : 0)
```

```
        : (y <= 0.0 ? CGFloat(y) : 0)
```

```
// multiply the height of the rect with the phase
```

```
if right > 0
```

```
{
```

```
    right *= CGFloat(phaseY)
```

```
}
```

```
else
```

```
{
```

```
    left *= CGFloat(phaseY)
```

```
}
```

```
barRect.origin.x = left
```

```
barRect.size.width = right - left
```

```
barRect.origin.y = top
```

```
barRect.size.height = bottom - top
```

```
buffer.rects[bufferIndex] = barRect
```

```
bufferIndex += 1
```

```

    }

else

{

    var posY = 0.0

    var negY = -e.negativeSum

    var yStart = 0.0


    // fill the stack

    for k in 0 ..< vals!.count

    {

        let value = vals![k]


        if value == 0.0 && (posY == 0.0 || negY == 0.0)

        {

            // Take care of the situation of a 0.0 value, which overlaps a non-zero bar

            y = value

            yStart = y

```

```

    }

    else if value >= 0.0

    {

        y = posY

        yStart = posY + value

        posY = yStart

    }

    else

    {

        y = negY

        yStart = negY + abs(value)

        negY += abs(value)

    }

    let bottom = CGFloat(x - barWidthHalf)

    let top = CGFloat(x + barWidthHalf)

    var right = isInverted

```



```

        ? (y <= yStart ? CGFloat(y) : CGFloat(yStart))

        : (y >= yStart ? CGFloat(y) : CGFloat(yStart))

var left = isInverted

        ? (y >= yStart ? CGFloat(y) : CGFloat(yStart))

        : (y <= yStart ? CGFloat(y) : CGFloat(yStart))


// multiply the height of the rect with the phase

right *= CGFloat(phaseY)

left *= CGFloat(phaseY)


barRect.origin.x = left

barRect.size.width = right - left

barRect.origin.y = top

barRect.size.height = bottom - top


buffer.rects[bufferIndex] = barRect

bufferIndex += 1

```

```

    }

    }

}

}

```

```
private var _barShadowRectBuffer: CGRect = CGRect()
```

```
open override func drawDataSet(context: CGContext, dataSet: IBarChartDataSet, index:
Int)
```

```

{

    guard let dataProvider = dataProvider else { return }

```

```
let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)
```

```
prepareBuffer(dataSet: dataSet, index: index)
```

```
trans.rectValuesToPixel(&_buffers[index].rects)
```

```
let borderWidth = dataSet.barBorderWidth
```

```

let borderColor = dataSet.barBorderColor

let drawBorder = borderWidth > 0.0

context.saveGState()

// draw the bar shadow before the values

if dataProvider.isDrawBarShadowEnabled

{

    guard let barData = dataProvider.barData else { return }

    let barWidth = barData.barWidth

    let barWidthHalf = barWidth / 2.0

    var x: Double = 0.0

    for i in stride(from: 0, to: min(Int(ceil(Double(dataSet.entryCount) *
    animator.phaseX)), dataSet.entryCount), by: 1)

    {

        guard let e = dataSet.entryForIndex(i) as? BarChartDataEntry else { continue }

```

```
x = e.x
```

```
_barShadowRectBuffer.origin.y = CGFloat(x - barWidthHalf)
```

```
_barShadowRectBuffer.size.height = CGFloat(barWidth)
```

```
trans.rectValueToPixel(&_barShadowRectBuffer)
```

```
if !viewPortHandler.isInBoundsTop(_barShadowRectBuffer.origin.y +  
_barShadowRectBuffer.size.height)
```

```
{
```

```
    break
```

```
}
```

```
if !viewPortHandler.isInBoundsBottom(_barShadowRectBuffer.origin.y)
```

```
{
```

```
    continue
```

```
}
```

```

        _barShadowRectBuffer.origin.x = viewPortHandler.contentLeft

        _barShadowRectBuffer.size.width = viewPortHandler.contentWidth

        context.setFillColors(dataSet.barShadowColor.cgColor)

        context.fill(_barShadowRectBuffer)

    }

}

let buffer = _buffers[index]

let isSingleColor = dataSet.colors.count == 1

if isSingleColor

{

    context.setFillColors(dataSet.color(atIndex: 0).cgColor)

}

```

```
// In case the chart is stacked, we need to accomodate individual bars within  
accessibilityOrdereredElements
```

```
let isStacked = dataSet.isStacked
```

```
let stackSize = isStacked ? dataSet.stackSize : 1
```

```
for j in stride(from: 0, to: buffer.rects.count, by: 1)
```

```
{
```

```
    let barRect = buffer.rects[j]
```

```
    if (!viewPortHandler.isInBoundsTop(barRect.origin.y + barRect.size.height))
```

```
    {
```

```
        break
```

```
    }
```

```
    if (!viewPortHandler.isInBoundsBottom(barRect.origin.y))
```

```
    {
```

```
        continue
```

```

    }

    if !isSingleColor
    {

        // Set the color for the currently drawn value. If the index is out of bounds, reuse
colors.

        context.setFillColor(dataSet.color(atIndex: j).cgColor)

    }

    context.fill(barRect)

    if drawBorder
    {

        context.setStrokeColor(borderColor.cgColor)

        context.setLineWidth(borderWidth)

        context.stroke(barRect)

    }

```

```
        // Create and append the corresponding accessibility element to
        accessibilityOrderedElements (see BarChartRenderer)
```

```
        if let chart = dataProvider as? BarChartView
```

```
        {
```

```
            let element = createAccessibleElement(withIndex: j,
```

```
                                                    container: chart,
```

```
                                                    dataSet: dataSet,
```

```
                                                    dataSetIndex: index,
```

```
                                                    stackSize: stackSize)
```

```
            { (element) in
```

```
                element.accessibilityFrame = barRect
```

```
            }
```

```
            accessibilityOrderedElements[j/stackSize].append(element)
```

```
        }
```

```
    }
```

```
    context.restoreGState()
```



```
}
```

```
open override func prepareBarHighlight(
```

```
    x: Double,
```

```
    y1: Double,
```

```
    y2: Double,
```

```
    barWidthHalf: Double,
```

```
    trans: Transformer,
```

```
    rect: inout CGRect)
```

```
{
```

```
    let top = x - barWidthHalf
```

```
    let bottom = x + barWidthHalf
```

```
    let left = y1
```

```
    let right = y2
```

```
    rect.origin.x = CGFloat(left)
```

```
    rect.origin.y = CGFloat(top)
```

```
rect.size.width = CGFloat(right - left)
```

```
rect.size.height = CGFloat(bottom - top)
```

```
trans.rectValueToPixelHorizontal(&rect, phaseY: animator.phaseY)
```

```
}
```

```
open override func drawValues(context: CGContext)
```

```
{
```

```
    // if values are drawn
```

```
    if isDrawingValuesAllowed(dataProvider: dataProvider)
```

```
    {
```

```
        guard
```

```
            let dataProvider = dataProvider,
```

```
            let barData = dataProvider.barData
```

```
        else { return }
```

```
        let dataSets = barData.dataSets
```

```
let textAlign = NSTextAlignment.left
```

```
let valueOffsetPlus: CGFloat = 5.0
```

```
var posOffset: CGFloat
```

```
var negOffset: CGFloat
```

```
let drawValueAboveBar = dataProvider.isDrawValueAboveBarEnabled
```

```
for dataSetIndex in 0 ..< barData.dataSetCount
```

```
{
```

```
    guard let
```

```
        dataSet = dataSets[dataSetIndex] as? IBarChartDataSet,
```

```
        shouldDrawValues(forDataSet: dataSet)
```

```
    else { continue }
```

```
let isInverted = dataProvider.isInverted(axis: dataSet.axisDependency)
```

```
let valueFont = dataSet.valueFont
```

```
let yOffset = -valueFont.lineHeight / 2.0
```

```
guard let formatter = dataSet.valueFormatter else { continue }
```

```
let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)
```

```
let phaseY = animator.phaseY
```

```
let iconsOffset = dataSet.iconsOffset
```

```
let buffer = _buffers[dataSetIndex]
```

```
// if only single values are drawn (sum)
```

```
if !dataSet.isStacked
```

```
{
```

```
    for j in 0 ..< Int(ceil(Double(dataSet.entryCount) * animator.phaseX))
```

```

    {

        guard let e = dataSet.entryForIndex(j) as? BarChartDataEntry else
        { continue }

        let rect = buffer.rects[j]

        let y = rect.origin.y + rect.size.height / 2.0

        if !viewPortHandler.isInBoundsTop(rect.origin.y)

        {

            break

        }

        if !viewPortHandler.isInBoundsX(rect.origin.x)

        {

            continue

        }

```

```

        if !viewPortHandler.isInBoundsBottom(rect.origin.y)

        {

            continue

        }

        let val = e.y

        let valueText = formatter.stringForValue(

            val,

            entry: e,

            dataSetIndex: dataSetIndex,

            viewPortHandler: viewPortHandler)

        // calculate the correct offset depending on the draw position of the value

        let valueTextWidth = valueText.size(withAttributes:
[NSAttributedString.Key.font: valueFont]).width

        posOffset = (drawValueAboveBar ? valueOffsetPlus : -(valueTextWidth +
valueOffsetPlus))

        negOffset = (drawValueAboveBar ? -(valueTextWidth + valueOffsetPlus) :
valueOffsetPlus) - rect.size.width

```

```

if isInverted

{

    posOffset = -posOffset - valueTextWidth

    negOffset = -negOffset - valueTextWidth

}


if dataSet.isDrawValuesEnabled

{

    drawValue(

        context: context,

        value: valueText,

        xPos: (rect.origin.x + rect.size.width)

            + (val >= 0.0 ? posOffset : negOffset),

        yPos: y + yOffset,

        font: valueFont,

        align: textAlign,

```

```

        color: dataSet.valueTextColorAt(j))

    }

    if let icon = e.icon, dataSet.isDrawIconsEnabled

    {

        var px = (rect.origin.x + rect.size.width)

            + (val >= 0.0 ? posOffset : negOffset)

        var py = y

        px += iconsOffset.x

        py += iconsOffset.y

        ChartUtils.drawImage(

            context: context,

            image: icon,

            x: px,

            y: py,

```



```

        size: icon.size)

    }

}

}

else

{

    // if each value of a potential stack should be drawn

    var bufferIndex = 0

    for index in 0 ..< Int(ceil(Double(dataSet.entryCount) * animator.phaseX))

    {

        guard let e = dataSet.entryForIndex(index) as? BarChartDataEntry else
{ continue }

        let rect = buffer.rects[bufferIndex]

        let vals = e.yValues

```

```
// we still draw stacked bars, but there is one non-stacked in between
```

```
if vals == nil
```

```
{
```

```
    if !viewPortHandler.isInBoundsTop(rect.origin.y)
```

```
    {
```

```
        break
```

```
    }
```

```
    if !viewPortHandler.isInBoundsX(rect.origin.x)
```

```
    {
```

```
        continue
```

```
    }
```

```
    if !viewPortHandler.isInBoundsBottom(rect.origin.y)
```

```
    {
```

```
        continue
```

```
}
```

```
let val = e.y
```

```
let valueText = formatter.stringForValue(
```

```
    val,
```

```
    entry: e,
```

```
    dataSetIndex: dataSetIndex,
```

```
    viewPortHandler: viewPortHandler)
```

```
// calculate the correct offset depending on the draw position of the value
```

```
let valueTextWidth = valueText.size(withAttributes:
[NSAttributedString.Key.font: valueFont]).width
```

```
posOffset = (drawValueAboveBar ? valueOffsetPlus : -(valueTextWidth
+ valueOffsetPlus))
```

```
negOffset = (drawValueAboveBar ? -(valueTextWidth +
valueOffsetPlus) : valueOffsetPlus)
```

```
if isInverted
```

```
{
```

```

        posOffset = -posOffset - valueTextWidth

        negOffset = -negOffset - valueTextWidth

    }

    if dataSet.isDrawValuesEnabled

    {

        drawValue(

            context: context,

            value: valueText,

            xPos: (rect.origin.x + rect.size.width)

                + (val >= 0.0 ? posOffset : negOffset),

            yPos: rect.origin.y + yOffset,

            font: valueFont,

            align: textAlign,

            color: dataSet.valueTextColorAt(index))

    }

```

```

if let icon = e.icon, dataSet.isDrawIconsEnabled

{

    var px = (rect.origin.x + rect.size.width)

        + (val >= 0.0 ? posOffset : negOffset)

    var py = rect.origin.y


    px += iconsOffset.x

    py += iconsOffset.y


    ChartUtils.drawImage(

        context: context,

        image: icon,

        x: px,

        y: py,

        size: icon.size)

    }

}

```

```

else

{

    let vals = vals!

    var transformed = [CGPoint]()

    var posY = 0.0

    var negY = -e.negativeSum

    for k in 0 ..< vals.count

    {

        let value = vals[k]

        var y: Double

        if value == 0.0 && (posY == 0.0 || negY == 0.0)

        {

            // Take care of the situation of a 0.0 value, which overlaps a non-
zero bar

            y = value

```

```

    }

    else if value >= 0.0

    {

        posY += value

        y = posY

    }

    else

    {

        y = negY

        negY -= value

    }

    transformed.append(CGPoint(x: CGFloat(y * phaseY), y: 0.0))

}

trans.pointValuesToPixel(&transformed)

```

```

        for k in 0 ..< transformed.count

        {

            let val = vals[k]

            let valueText = formatter.stringForValue(

                val,

                entry: e,

                dataSetIndex: dataSetIndex,

                viewPortHandler: viewPortHandler)

            // calculate the correct offset depending on the draw position of the
            value

            let valueTextWidth = valueText.size(withAttributes:
[NSAttributedString.Key.font: valueFont]).width

            posOffset = (drawValueAboveBar ? valueOffsetPlus : -
(valueTextWidth + valueOffsetPlus))

            negOffset = (drawValueAboveBar ? -(valueTextWidth +
valueOffsetPlus) : valueOffsetPlus)

            if isInverted

```



```

{

    posOffset = -posOffset - valueTextWidth

    negOffset = -negOffset - valueTextWidth

}

```

```

0.0    let drawBelow = (val == 0.0 && negY == 0.0 && posY > 0.0) || val <

```

```

    let x = transformed[k].x + (drawBelow ? negOffset : posOffset)

```

```

    let y = rect.origin.y + rect.size.height / 2.0

```

```

    if (!viewPortHandler.isInBoundsTop(y))

```

```

    {

        break

    }

```

```

    if (!viewPortHandler.isInBoundsX(x))

```

```

    {

```

```

        continue
    }

    if (!viewPortHandler.isInBoundsBottom(y))

    {

        continue
    }

    if dataSet.isDrawValuesEnabled

    {

        drawValue(context: context,

            value: valueText,

            xPos: x,

            yPos: y + yOffset,

            font: valueFont,

            align: textAlign,

            color: dataSet.valueTextColorAt(index))
    }

```

```
}
```

```
if let icon = e.icon, dataSet.isDrawIconsEnabled
```

```
{
```

```
    ChartUtils.drawImage(
```

```
        context: context,
```

```
        image: icon,
```

```
        x: x + iconsOffset.x,
```

```
        y: y + iconsOffset.y,
```

```
        size: icon.size)
```

```
    }
```

```
}
```

```
}
```

```
bufferIndex = vals == nil ? (bufferIndex + 1) : (bufferIndex + vals!.count)
```

```
}
```

```
}
```

```

    }

}

}

open override func isDrawingValuesAllowed(dataProvider: ChartDataProvider?) ->
Bool

{

    guard let data = dataProvider?.data

    else { return false }

    return data.entryCount < Int(CGFloat(dataProvider?.maxVisibleCount ?? 0) *
self.viewPortHandler.scaleY)

}

/// Sets the drawing position of the highlight object based on the riven bar-rect.

internal override func setHighlightDrawPos(highlight high: Highlight, barRect: CGRect)

{

    high.setDraw(x: barRect.midY, y: barRect.origin.x + barRect.size.width)

}

```

```

}

//

// HorizontalBarChartView.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

/// BarChart with horizontal bar orientation. In this implementation, x- and y-axis are
switched.

open class HorizontalBarChartView: BarChartView

```

```

{

    internal override func initialize()

    {

        super.initialize()


        _leftAxisTransformer    =    TransformerHorizontalBarChart(viewPortHandler:
        _viewPortHandler)

        _rightAxisTransformer    =    TransformerHorizontalBarChart(viewPortHandler:
        _viewPortHandler)


        renderer = HorizontalBarChartRenderer(dataProvider: self, animator: _animator,
        viewPortHandler: _viewPortHandler)


        leftYAxisRenderer    =    YAxisRendererHorizontalBarChart(viewPortHandler:
        _viewPortHandler, yAxis: leftAxis, transformer: _leftAxisTransformer)

        rightYAxisRenderer    =    YAxisRendererHorizontalBarChart(viewPortHandler:
        _viewPortHandler, yAxis: rightAxis, transformer: _rightAxisTransformer)


        xAxisRenderer    =    XAxisRendererHorizontalBarChart(viewPortHandler:
        _viewPortHandler, xAxis: _xAxis, transformer: _leftAxisTransformer, chart: self)


        self.highlighter = HorizontalBarHighlighter(chart: self)

```

```
}
```

```
internal override fun calculateLegendOffsets(offsetLeft: inout CGFloat, offsetTop:  
inout CGFloat, offsetRight: inout CGFloat, offsetBottom: inout CGFloat)
```

```
{
```

```
    guard
```

```
        let legend = _legend,
```

```
        legend.isEnabled,
```

```
        !legend.drawInside
```

```
    else { return }
```

```
    // setup offsets for legend
```

```
    switch legend.orientation
```

```
    {
```

```
        case .vertical:
```

```
            switch legend.horizontalAlignment
```

```
            {
```

```
                case .left:
```

```
        offsetLeft += min(legend.neededWidth, _viewPortHandler.chartWidth *  
legend.maxSizePercent) + legend.xOffset
```

```
    case .right:
```

```
        offsetRight += min(legend.neededWidth, _viewPortHandler.chartWidth *  
legend.maxSizePercent) + legend.xOffset
```

```
    case .center:
```

```
        switch legend.verticalAlignment
```

```
        {
```

```
            case .top:
```

```
                offsetTop += min(legend.neededHeight, _viewPortHandler.chartHeight *  
legend.maxSizePercent) + legend.yOffset
```

```
            case .bottom:
```

```
                offsetBottom += min(legend.neededHeight, _viewPortHandler.chartHeight *  
legend.maxSizePercent) + legend.yOffset
```



```

        default:

            break

        }

    }

case .horizontal:

    switch legend.verticalAlignment

    {

        case .top:

            offsetTop += min(legend.neededHeight, _viewPortHandler.chartHeight *
legend.maxSizePercent) + legend.yOffset

            // left axis equals the top x-axis in a horizontal chart

            if leftAxis.isEnabled && leftAxis.isDrawLabelsEnabled

            {

                offsetTop += leftAxis.getRequiredHeightSpace()

            }

```

```

        case .bottom:

            offsetBottom += min(legend.neededHeight, _viewPortHandler.chartHeight *
legend.maxSizePercent) + legend.yOffset


            // right axis equals the bottom x-axis in a horizontal chart

            if rightAxis.isEnabled && rightAxis.isDrawLabelsEnabled

            {

                offsetBottom += rightAxis.getRequiredHeightSpace()

            }

        default:

            break

    }

}

}

}

internal override func calculateOffsets()

{

    var offsetLeft: CGFloat = 0.0,

```

```
offsetRight: CGFloat = 0.0,
```

```
offsetTop: CGFloat = 0.0,
```

```
offsetBottom: CGFloat = 0.0
```

```
calculateLegendOffsets(offsetLeft: &offsetLeft,
```

```
    offsetTop: &offsetTop,
```

```
    offsetRight: &offsetRight,
```

```
    offsetBottom: &offsetBottom)
```

```
// offsets for y-labels
```

```
if leftAxis.needsOffset
```

```
{
```

```
    offsetTop += leftAxis.getRequiredHeightSpace()
```

```
}
```

```
if rightAxis.needsOffset
```

```
{
```

```
        offsetBottom += rightAxis.getRequiredHeightSpace()

    }
```

```
    let xlabelwidth = _xAxis.labelRotatedWidth
```

```
    if _xAxis.isEnabled
```

```
    {
```

```
        // offsets for x-labels
```

```
        if _xAxis.labelPosition == .bottom
```

```
        {
```

```
            offsetLeft += xlabelwidth
```

```
        }
```

```
        else if _xAxis.labelPosition == .top
```

```
        {
```

```
            offsetRight += xlabelwidth
```

```
        }
```

```
        else if _xAxis.labelPosition == .bothSided
```

```

{

    offsetLeft += xlabelwidth

    offsetRight += xlabelwidth

}

}

offsetTop += self.extraTopOffset

offsetRight += self.extraRightOffset

offsetBottom += self.extraBottomOffset

offsetLeft += self.extraLeftOffset


_viewPortHandler.restrainViewPort(

    offsetLeft: max(self.minOffset, offsetLeft),

    offsetTop: max(self.minOffset, offsetTop),

    offsetRight: max(self.minOffset, offsetRight),

    offsetBottom: max(self.minOffset, offsetBottom))

```

```

prepareOffsetMatrix()

prepareValuePxMatrix()

}

```

```

internal override func prepareValuePxMatrix()

{

    _rightAxisTransformer.prepareMatrixValuePx(chartXMin:
rightAxis._axisMinimum,    deltaX:    CGFloat(rightAxis.axisRange),    deltaY:
CGFloat(_xAxis.axisRange), chartYMin: _xAxis._axisMinimum)

    _leftAxisTransformer.prepareMatrixValuePx(chartXMin: leftAxis._axisMinimum,
deltaX: CGFloat(leftAxis.axisRange), deltaY: CGFloat(_xAxis.axisRange), chartYMin:
_xAxis._axisMinimum)

}

```

```

open override func getMarkerPosition(highlight: Highlight) -> CGPoint

{

    return CGPoint(x: highlight.drawY, y: highlight.drawX)

}

```

```
open override func getBarBounds(entry e: BarChartDataEntry) -> CGRect
```

```
{
```

```
    guard
```

```
        let data = _data as? BarChartData,
```

```
        let set = data.getDataSetForEntry(e) as? IBarChartDataSet
```

```
    else { return CGRect.null }
```

```
    let y = e.y
```

```
    let x = e.x
```

```
    let barWidth = data.barWidth
```

```
    let top = x - 0.5 + barWidth / 2.0
```

```
    let bottom = x + 0.5 - barWidth / 2.0
```

```
    let left = y >= 0.0 ? y : 0.0
```

```
    let right = y <= 0.0 ? y : 0.0
```

```
var bounds = CGRect(x: left, y: top, width: right - left, height: bottom - top)
```

```
getTransformer(forAxis: set.axisDependency).rectValueToPixel(&bounds)
```

```
return bounds
```

```
}
```

```
open override func getPosition(entry e: ChartDataEntry, axis: YAxis.AxisDependency)  
-> CGPoint
```

```
{
```

```
var vals = CGPoint(x: CGFloat(e.y), y: CGFloat(e.x))
```

```
getTransformer(forAxis: axis).pointValueToPixel(&vals)
```

```
return vals
```

```
}
```

```
open override func getHighlightByTouchPoint(_ pt: CGPoint) -> Highlight?
```



```

{

    if _data === nil

    {

        Swift.print("Can't select by touch. No data set.", terminator: "\n")

        return nil

    }

    return self.highlighter?.getHighlight(x: pt.y, y: pt.x)

}

```

/// The lowest x-index (value on the x-axis) that is still visible on he chart.

open override var lowestVisibleX: Double

```

{

    var pt = CGPoint(

        x: viewPortHandler.contentLeft,

        y: viewPortHandler.contentBottom)

```

```

    getTransformer(forAxis: .left).pixelToValues(&pt)

    return max(xAxis._axisMinimum, Double(pt.y))

}

/// The highest x-index (value on the x-axis) that is still visible on the chart.

open override var highestVisibleX: Double

{

    var pt = CGPoint(

        x: viewPortHandler.contentLeft,

        y: viewPortHandler.contentTop)

    getTransformer(forAxis: .left).pixelToValues(&pt)

    return min(xAxis._axisMaximum, Double(pt.y))

}

```

```
// MARK: - Viewport
```

```
open override func setVisibleXRangeMaximum(_ maxXRange: Double)
```

```
{
```

```
    let xScale = xAxis.axisRange / maxXRange
```

```
    viewPortHandler.setMinimumScaleY(CGFloat(xScale))
```

```
}
```

```
open override func setVisibleXRangeMinimum(_ minXRange: Double)
```

```
{
```

```
    let xScale = xAxis.axisRange / minXRange
```

```
    viewPortHandler.setMaximumScaleY(CGFloat(xScale))
```

```
}
```

```
open override func setVisibleXRange(minXRange: Double, maxXRange: Double)
```

```
{
```

```
    let minScale = xAxis.axisRange / minXRange
```

```

        let maxScale = xAxis.axisRange / maxXRange

        viewPortHandler.setMinMaxScaleY(minScaleY: CGFloat(minScale), maxScaleY:
CGFloat(maxScale))

    }

    open override func setVisibleYRangeMaximum(_ maxYRange: Double, axis:
YAxis.AxisDependency)

    {

        let yScale = getAxisRange(axis: axis) / maxYRange

        viewPortHandler.setMinimumScaleX(CGFloat(yScale))

    }

    open override func setVisibleYRangeMinimum(_ minYRange: Double, axis:
YAxis.AxisDependency)

    {

        let yScale = getAxisRange(axis: axis) / minYRange

        viewPortHandler.setMaximumScaleX(CGFloat(yScale))

    }

```

```

    open override func setVisibleYRange(minYRange: Double, maxYRange: Double, axis:
YAxis.AxisDependency)

    {

        let minScale = getAxisRange(axis: axis) / minYRange

        let maxScale = getAxisRange(axis: axis) / maxYRange

        viewPortHandler.setMinMaxScaleX(minScaleX: CGFloat(minScale), maxScaleX:
CGFloat(maxScale))

    }

}

//

// HorizontalBarHighlighter.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
@objc(HorizontalBarChartHighlighter)
```

```
open class HorizontalBarHighlighter: BarHighlighter
```

```
{
```

```
    open override func getHighlight(x: CGFloat, y: CGFloat) -> Highlight?
```

```
    {
```

```
        guard let barData = self.chart?.data as? BarChartData else { return nil }
```

```
        let pos = getValsForTouch(x: y, y: x)
```

```
        guard let high = getHighlight(xValue: Double(pos.y), x: y, y: x) else { return nil }
```

```
        if let set = barData.getDataSetByIndex(high.dataSetIndex) as? IBarChartDataSet,
```

```
           set.isStacked
```

```

    {

        return getStackedHighlight(high: high,

                                   set: set,

                                   xValue: Double(pos.y),

                                   yValue: Double(pos.x))

    }

    return high

}

internal override fun buildHighlights(

    dataSet set: IChartDataSet,

    dataSetIndex: Int,

    xValue: Double,

    rounding: ChartDataSetRounding) -> [Highlight]

{

    guard let chart = self.chart as? BarLineScatterCandleBubbleChartDataProvider else
    { return [] }

```

```

var entries = set.entriesForXValue(xValue)

if entries.count == 0, let closest = set.entryForXValue(xValue, closestToY: .nan,
rounding: rounding)

{

    // Try to find closest x-value and take all entries for that x-value

    entries = set.entriesForXValue(closest.x)

}

return entries.map { e in

    let px = chart.getTransformer(forAxis: set.axisDependency)

    .pixelForValues(x: e.y, y: e.x)

    return Highlight(x: e.x, y: e.y, xPx: px.x, yPx: px.y, dataSetIndex: dataSetIndex,
axis: set.axisDependency)

}

}

internal override func getDistance(x1: CGFloat, y1: CGFloat, x2: CGFloat, y2: CGFloat)
-> CGFloat

```



```
{  
  
    return abs(y1 - y2)  
  
}  
  
}  
  
//  
  
// IAxisValueFormatter.swift  
  
// Charts  
  
//  
  
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda  
  
// A port of MPAndroidChart for iOS  
  
// Licensed under Apache License 2.0  
  
//  
  
// https://github.com/danielgindi/Charts  
  
//  
  
import Foundation
```

```

/// An interface for providing custom axis Strings.

@objc(IChartAxisValueFormatter)

public protocol IAxisValueFormatter: class

{

    /// Called when a value from an axis is formatted before being drawn.

    ///

    /// For performance reasons, avoid excessive calculations and memory allocations inside
    this method.

    ///

    /// - Parameters:

    /// - value:      the value that is currently being drawn

    /// - axis:       the axis that the value belongs to

    /// - Returns: The customized label that is drawn on the x-axis.

    func stringForValue(_ value: Double,

                        axis: AxisBase?) -> String

}

```

```

//

// IBarChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc

public protocol IBarChartDataSet: IBarLineScatterCandleBubbleChartDataSet

{

```

```
// MARK: - Data functions and accessors
```

```
// MARK: - Styling functions and accessors
```

```
/// `true` if this DataSet is stacked (stacksize > 1) or not.
```

```
var isStacked: Bool { get }
```

```
/// The maximum number of bars that can be stacked upon another in this DataSet.
```

```
var stackSize: Int { get }
```

```
/// the color used for drawing the bar-shadows. The bar shadows is a surface behind the  
bar that indicates the maximum value
```

```
var barShadowColor: NSUIColor { get set }
```

```
/// the width used for drawing borders around the bars. If borderWidth == 0, no border  
will be drawn.
```

```
var barBorderWidth : CGFloat { get set }
```

```

    /// the color drawing borders around the bars.

    var barBorderColor: NSUIColor { get set }

    /// the alpha value (transparency) that is used for drawing the highlight indicator bar. min
    = 0.0 (fully transparent), max = 1.0 (fully opaque)

    var highlightAlpha: CGFloat { get set }

    /// array of labels used to describe the different values of the stacked bars

    var stackLabels: [String] { get set }

}

//

// IBarLineScatterCandleBubbleChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

```

```

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc

public protocol IBarLineScatterCandleBubbleChartDataSet: IChartDataSet

{

    // MARK: - Data functions and accessors


    // MARK: - Styling functions and accessors


    var highlightColor: NSUIColor { get set }

    var highlightLineWidth: CGFloat { get set }

    var highlightLineDashPhase: CGFloat { get set }

    var highlightLineDashLengths: [CGFloat]? { get set }

```

```
}

//

// IBubbleChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc

public protocol IBubbleChartDataSet: IBarLineScatterCandleBubbleChartDataSet
```

```

{

    // MARK: - Data functions and accessors


    var maxSize: CGFloat { get }

    var isNormalizeSizeEnabled: Bool { get }


    // MARK: - Styling functions and accessors


    /// Sets/gets the width of the circle that surrounds the bubble when highlighted

    var highlightCircleWidth: CGFloat { get set }

}

//

// ICandleChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

```



```

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc

public protocol ICandleChartDataSet: ILineScatterCandleRadarChartDataSet

{

    // MARK: - Data functions and accessors


    // MARK: - Styling functions and accessors


    /// the space that is left out on the left and right side of each candle,

    /// default: 0.1 (10%), max 0.45, min 0.0

```

```
var barSpace: CGFloat { get set }

/// should the candle bars show?

/// when false, only "ticks" will show

///

/// **default**: true

var showCandleBar: Bool { get set }

/// the width of the candle-shadow-line in pixels.

///

/// **default**: 3.0

var shadowWidth: CGFloat { get set }

/// the color of the shadow line

var shadowColor: UIColor? { get set }

/// use candle color for the shadow
```

```
var shadowColorSameAsCandle: Bool { get set }
```

```
/// Is the shadow color same as the candle color?
```

```
var isShadowColorSameAsCandle: Bool { get }
```

```
/// color for open == close
```

```
var neutralColor: NSUIColor? { get set }
```

```
/// color for open > close
```

```
var increasingColor: NSUIColor? { get set }
```

```
/// color for open < close
```

```
var decreasingColor: NSUIColor? { get set }
```

```
/// Are increasing values drawn as filled?
```

```
var increasingFilled: Bool { get set }
```

```

    /// Are increasing values drawn as filled?

    var isIncreasingFilled: Bool { get }


    /// Are decreasing values drawn as filled?

    var decreasingFilled: Bool { get set }


    /// Are decreasing values drawn as filled?

    var isDecreasingFilled: Bool { get }

}

//

// IChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
@objc
```

```
public protocol IChartDataSet
```

```
{
```

```
    // MARK: - Data functions and accessors
```

```
    /// Use this method to tell the data set that the underlying data has changed
```

```
    func notifyDataSetChanged()
```

```
    /// Calculates the minimum and maximum x and y values (_xMin, _xMax, _yMin, _yMax).
```

```
    func calcMinMax()
```

```
/// Calculates the min and max y-values from the Entry closest to the given fromX to the  
Entry closest to the given toX value.
```

```
/// This is only needed for the autoScaleMinMax feature.
```

```
func calcMinMaxY(fromX: Double, toX: Double)
```

```
/// The minimum y-value this DataSet holds
```

```
var yMin: Double { get }
```

```
/// The maximum y-value this DataSet holds
```

```
var yMax: Double { get }
```

```
/// The minimum x-value this DataSet holds
```

```
var xMin: Double { get }
```

```
/// The maximum x-value this DataSet holds
```

```
var xMax: Double { get }
```

```
/// The number of y-values this DataSet represents
```

```

var entryCount: Int { get }

/// - Throws: out of bounds

/// if `i` is out of bounds, it may throw an out-of-bounds exception

/// - Returns: The entry object found at the given index (not x-value!)

func entryForIndex(_ i: Int) -> ChartDataEntry?

/// - Parameters:

/// - xValue: the x-value

/// - closestToY: If there are multiple y-values for the specified x-value,

/// - rounding: determine whether to round up/down/closest if there is no Entry matching
the provided x-value

/// - Returns: The first Entry object found at the given x-value with binary search.

/// If the no Entry at the specified x-value is found, this method returns the Entry at the
closest x-value according to the rounding.

/// nil if no Entry object at that x-value.

func entryForXValue(
    _ xValue: Double,

```

closestToY yValue: Double,

rounding: ChartDataSetRounding) -> ChartDataEntry?

/// - Parameters:

/// - xValue: the x-value

/// - closestToY: If there are multiple y-values for the specified x-value,

/// - Returns: The first Entry object found at the given x-value with binary search.

/// If the no Entry at the specified x-value is found, this method returns the Entry at the closest x-value.

/// nil if no Entry object at that x-value.

func entryForXValue(

_ xValue: Double,

closestToY yValue: Double) -> ChartDataEntry?

/// - Returns: All Entry objects found at the given x-value with binary search.

/// An empty array if no Entry object at that x-value.

func entriesForXValue(_ xValue: Double) -> [ChartDataEntry]


```

/// - Parameters:

/// - xValue: x-value of the entry to search for

/// - closestToY: If there are multiple y-values for the specified x-value,

/// - rounding: Rounding method if exact value was not found

/// - Returns: The array-index of the specified entry.

/// If the no Entry at the specified x-value is found, this method returns the index of the
Entry at the closest x-value according to the rounding.

```

```

func entryIndex(

    x xValue: Double,

    closestToY yValue: Double,

    rounding: ChartDataSetRounding) -> Int

```

```

/// - Parameters:

/// - e: the entry to search for

/// - Returns: The array-index of the specified entry

func entryIndex(entry e: ChartDataEntry) -> Int

```

```

/// Adds an Entry to the DataSet dynamically.

```

```

///

/// *optional feature, can return `false` if not implemented*

///

/// Entries are added to the end of the list.

///

/// - Parameters:

/// - e: the entry to add

/// - Returns: `true` if the entry was added successfully, `false` if this feature is not
supported

func addEntry(_ e: ChartDataEntry) -> Bool

/// Adds an Entry to the DataSet dynamically.

/// Entries are added to their appropriate index in the values array respective to their x-
position.

/// This will also recalculate the current minimum and maximum values of the DataSet
and the value-sum.

///

/// *optional feature, can return `false` if not implemented*

///

```

```

/// Entries are added to the end of the list.

///

/// - Parameters:

/// - e: the entry to add

/// - Returns: `true` if the entry was added successfully, `false` if this feature is not
supported

func addEntryOrdered(_ e: ChartDataEntry) -> Bool

/// Removes an Entry from the DataSet dynamically.

///

/// *optional feature, can return `false` if not implemented*

///

/// - Parameters:

/// - entry: the entry to remove

/// - Returns: `true` if the entry was removed successfully, `false` if the entry does not
exist or if this feature is not supported

func removeEntry(_ entry: ChartDataEntry) -> Bool

```

```

/// Removes the Entry object at the given index in the values array from the DataSet.

///

/// *optional feature, can return `false` if not implemented*

///

/// - Parameters:

/// - index: the index of the entry to remove

/// - Returns: `true` if the entry was removed successfully, `false` if the entry does not
exist or if this feature is not supported

func removeEntry(index: Int) -> Bool

/// Removes the Entry object closest to the given x-value from the DataSet.

///

/// *optional feature, can return `false` if not implemented*

///

/// - Parameters:

/// - x: the x-value to remove

/// - Returns: `true` if the entry was removed successfully, `false` if the entry does not
exist or if this feature is not supported

```

```

func removeEntry(x: Double) -> Bool

/// Removes the first Entry (at index 0) of this DataSet from the entries array.

///

/// *optional feature, can return `false` if not implemented*

///

/// - Returns: `true` if the entry was removed successfully, `false` if the entry does not
exist or if this feature is not supported

func removeFirst() -> Bool

/// Removes the last Entry (at index 0) of this DataSet from the entries array.

///

/// *optional feature, can return `false` if not implemented*

///

/// - Returns: `true` if the entry was removed successfully, `false` if the entry does not
exist or if this feature is not supported

func removeLast() -> Bool

```

```

/// Checks if this DataSet contains the specified Entry.

///

/// - Returns: `true` if contains the entry, `false` if not.

func contains(_ e: ChartDataEntry) -> Bool


/// Removes all values from this DataSet and does all necessary recalculations.

///

/// *optional feature, could throw if not implemented*

func clear()


// MARK: - Styling functions and accessors


/// The label string that describes the DataSet.

var label: String? { get }


/// The axis this DataSet should be plotted against.

var axisDependency: YAxis.AxisDependency { get }

```

```

/// List representing all colors that are used for drawing the actual values for this DataSet

var valueColors: [NSUIColor] { get }


/// All the colors that are used for this DataSet.

/// Colors are reused as soon as the number of Entries the DataSet represents is higher
than the size of the colors array.

var colors: [NSUIColor] { get }


/// - Returns: The color at the given index of the DataSet's color array.

/// This prevents out-of-bounds by performing a modulus on the color index, so colours
will repeat themselves.

func color(atIndex: Int) -> NSUIColor


func resetColors()


func addColor(_ color: NSUIColor)

```

```
func setColor(_ color: NSUIColor)

/// if true, value highlighting is enabled

var highlightEnabled: Bool { get set }

/// `true` if value highlighting is enabled for this dataset

var isHighlightEnabled: Bool { get }

/// Custom formatter that is used instead of the auto-formatter if set

var valueFormatter: IValueFormatter? { get set }

/// `true` if the valueFormatter object of this DataSet is null.

var needsFormatter: Bool { get }

/// Sets/get a single color for value text.

/// Setting the color clears the colors array and adds a single color.

/// Getting will return the first color in the array.
```



```
var valueTextColor: NSUIColor { get set }
```

/// - Returns: The color at the specified index that is used for drawing the values inside the chart. Uses modulus internally.

```
func valueTextColorAt(_ index: Int) -> NSUIColor
```

/// the font for the value-text labels

```
var valueFont: NSUIFont { get set }
```

/// The form to draw for this dataset in the legend.

///

/// Return `.Default` to use the default legend form.

```
var form: Legend.Form { get }
```

/// The form size to draw for this dataset in the legend.

///

/// Return `NaN` to use the default legend form size.

```
var formSize: CGFloat { get }
```

```
/// The line width for drawing the form of this dataset in the legend
```

```
///
```

```
/// Return `NaN` to use the default legend form line width.
```

```
var formLineWidth: CGFloat { get }
```

```
/// Line dash configuration for legend shapes that consist of lines.
```

```
///
```

```
/// This is how much (in pixels) into the dash pattern are we starting from.
```

```
var formLineDashPhase: CGFloat { get }
```

```
/// Line dash configuration for legend shapes that consist of lines.
```

```
///
```

```
/// This is the actual dash pattern.
```

```
/// I.e. [2, 3] will paint [-- -- ]
```

```
/// [1, 3, 4, 2] will paint [- ---- - ---- ]
```

```
var formLineDashLengths: [CGFloat]? { get }
```

```
/// Set this to true to draw y-values on the chart.
```

```
///
```

```
/// - Note: For bar and line charts: if `maxVisibleCount` is reached, no values will be  
drawn even if this is enabled.
```

```
var drawValuesEnabled: Bool { get set }
```

```
/// `true` if y-value drawing is enabled, `false` if not
```

```
var isDrawValuesEnabled: Bool { get }
```

```
/// Set this to true to draw y-icons on the chart
```

```
///
```

```
/// - Note: For bar and line charts: if `maxVisibleCount` is reached, no icons will be  
drawn even if this is enabled.
```

```
var drawIconsEnabled: Bool { get set }
```

```
/// Returns true if y-icon drawing is enabled, false if not
```

```
var isDrawIconsEnabled: Bool { get }
```

```

    /// Offset of icons drawn on the chart.

    ///

    /// For all charts except Pie and Radar it will be ordinary (x offset, y offset).

    ///

    /// For Pie and Radar chart it will be (y offset, distance from center offset); so if you want
    icon to be rendered under value, you should increase X component of CGPoint, and if you
    want icon to be rendered closet to center, you should decrease height component of
    CGPoint.

    var iconsOffset: CGPoint { get set }

    /// Set the visibility of this DataSet. If not visible, the DataSet will not be drawn to the
    chart upon refreshing it.

    var visible: Bool { get set }

    /// `true` if this DataSet is visible inside the chart, or `false` if it is currently hidden.

    var isVisible: Bool { get }

}

//

```

```

// IFillFormatter.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

/// Protocol for providing a custom logic to where the filling line of a LineDataSet should
end. This of course only works if setFillEnabled(...) is set to true.

@objc(IChartFillFormatter)

public protocol IFillFormatter

{

```

/// - Returns: The vertical (y-axis) position where the filled-line of the LineDataSet should end.

```
func getFillLinePosition(dataSet: ILineChartDataSet, dataProvider:
LineChartDataProvider) -> CGFloat
```

```
}
```

```
//
```

```
// IHighlighter.swift
```

```
// Charts
```

```
//
```

```
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```
// A port of MPAndroidChart for iOS
```

```
// Licensed under Apache License 2.0
```

```
//
```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```

@objc(IChartHighlighter)

public protocol IHighlighter: class

{

    /// - Parameters:

    /// - x:

    /// - y:

    /// - Returns: A Highlight object corresponding to the given x- and y- touch positions in
    pixels.

    func getHighlight(x: CGFloat, y: CGFloat) -> Highlight?

}

//

// ILineChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

```

```
//  
  
// https://github.com/danielgindi/Charts  
  
//  
  
import Foundation  
  
import CoreGraphics  
  
  
@objc  
  
public protocol ILineChartDataSet: ILineRadarChartDataSet  
  
{  
  
    // MARK: - Data functions and accessors  
  
  
    // MARK: - Styling functions and accessors  
  
  
    /// The drawing mode for this line dataset  
  
    ///
```



```

/// **default**: Linear

var mode: LineChartDataSet.Mode { get set }


/// Intensity for cubic lines (min = 0.05, max = 1)

///

/// **default**: 0.2

var cubicIntensity: CGFloat { get set }


/// The radius of the drawn circles.

var circleRadius: CGFloat { get set }


/// The hole radius of the drawn circles.

var circleHoleRadius: CGFloat { get set }


var circleColors: [UIColor] { get set }


/// - Returns: The color at the given index of the DataSet's circle-color array.

```

```
/// Performs a IndexOutOfBounds check by modulus.
```

```
func getCircleColor(atIndex: Int) -> NSUIColor?
```

```
/// Sets the one and ONLY color that should be used for this DataSet.
```

```
/// Internally, this recreates the colors array and adds the specified color.
```

```
func setCircleColor(_ color: NSUIColor)
```

```
/// Resets the circle-colors array and creates a new one
```

```
func resetCircleColors(_ index: Int)
```

```
/// If true, drawing circles is enabled
```

```
var drawCirclesEnabled: Bool { get set }
```

```
/// `true` if drawing circles for this DataSet is enabled, `false` if not
```

```
var isDrawCirclesEnabled: Bool { get }
```

```
/// The color of the inner circle (the circle-hole).
```

```
var circleHoleColor: NSUIColor? { get set }
```

```
/// `true` if drawing circles for this DataSet is enabled, `false` ifnot
```

```
var drawCircleHoleEnabled: Bool { get set }
```

```
/// `true` if drawing the circle-holes is enabled, `false` ifnot.
```

```
var isDrawCircleHoleEnabled: Bool { get }
```

```
/// This is how much (in pixels) into the dash pattern are we starting from.
```

```
var lineDashPhase: CGFloat { get }
```

```
/// This is the actual dash pattern.
```

```
/// I.e. [2, 3] will paint [-- -- ]
```

```
/// [1, 3, 4, 2] will paint [- ---- - ---- ]
```

```
var lineDashLengths: [CGFloat]? { get set }
```

```
/// Line cap type, default is CGLineCap.Butt
```

```

var lineCapType: CGLineCap { get set }

/// Sets a custom IFillFormatter to the chart that handles the position of the filled-line for
each DataSet. Set this to null to use the default logic.

var fillFormatter: IFillFormatter? { get set }

}

//

// ILineRadarChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

```

```
import CoreGraphics
```

```
@objc
```

```
public protocol ILineRadarChartDataSet: ILineScatterCandleRadarChartDataSet
```

```
{
```

```
    // MARK: - Data functions and accessors
```

```
    // MARK: - Styling functions and accessors
```

```
    /// The color that is used for filling the line surface area.
```

```
    var fillColor: NSUIColor { get set }
```

```
    /// - Returns: The object that is used for filling the area below the line.
```

```
    /// default: nil
```

```
    var fill: Fill? { get set }
```

```
    /// The alpha value that is used for filling the line surface.
```

```

    /// **default**: 0.33

    var fillAlpha: CGFloat { get set }

    /// line width of the chart (min = 0.0, max = 10)

    ///

    /// **default**: 1

    var lineWidth: CGFloat { get set }

    /// Set to `true` if the DataSet should be drawn filled (surface), and not just as a line.

    /// Disabling this will give great performance boost.

    /// Please note that this method uses the path clipping for drawing the filled area (with
    images, gradients and layers).

    var drawFilledEnabled: Bool { get set }

    /// `true` if filled drawing is enabled, `false` if not

    var isDrawFilledEnabled: Bool { get }

}

//

```

```

// ILineScatterCandleRadarChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

@objc

public protocol ILineScatterCandleRadarChartDataSet:
    IBarLineScatterCandleBubbleChartDataSet

{

    // MARK: - Data functions and accessors

```

```
// MARK: - Styling functions and accessors
```

```
/// Enables / disables the horizontal highlight-indicator. If disabled, the indicator is not drawn.
```

```
var drawHorizontalHighlightIndicatorEnabled: Bool { get set }
```

```
/// Enables / disables the vertical highlight-indicator. If disabled, the indicator is not drawn.
```

```
var drawVerticalHighlightIndicatorEnabled: Bool { get set }
```

```
/// `true` if horizontal highlight indicator lines are enabled (drawn)
```

```
var isHorizontalHighlightIndicatorEnabled: Bool { get }
```

```
/// `true` if vertical highlight indicator lines are enabled (drawn)
```

```
var isVerticalHighlightIndicatorEnabled: Bool { get }
```

```
/// Enables / disables both vertical and horizontal highlight-indicators.
```

```
/// :param: enabled
```



```
func setDrawHighlightIndicators(_ enabled: Bool)

}

//

// ChartMarker.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc(IChartMarker)
```

```

public protocol IMarker: class

{

    /// - Returns: The desired (general) offset you wish the IMarker to have on the x-axis.

    /// By returning x: -(width / 2) you will center the IMarker horizontally.

    /// By returning y: -(height / 2) you will center the IMarker vertically.

    var offset: CGPoint { get }


    /// - Parameters:

    /// - point: This is the point at which the marker wants to be drawn. You can adjust the
    offset conditionally based on this argument.

    /// - Returns: The offset for drawing at the specific `point`.

    ///      This allows conditional adjusting of the Marker position.

    ///      If you have no adjustments to make, return self.offset().

    func offsetForDrawing(atPoint: CGPoint) -> CGPoint


    /// This method enables a custom IMarker to update it's content every time the IMarker
    is redrawn according to the data entry it points to.

    ///

```

```

/// - Parameters:

/// - entry: The Entry the IMarker belongs to. This can also be any subclass of Entry,
like BarEntry or CandleEntry, simply cast it at runtime.

/// - highlight: The highlight object contains information about the highlighted value
such as it's dataset-index, the selected range or stack-index (only stacked bar entries).

func refreshContent(entry: ChartDataEntry, highlight: Highlight)

/// Draws the IMarker on the given position on the given context

func draw(context: CGContext, point: CGPoint)

}

//

// IPieChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
@objc
```

```
public protocol IPieChartDataSet: IChartDataSet
```

```
{
```

```
    // MARK: - Styling functions and accessors
```

```
    /// the space in pixels between the pie-slices
```

```
    /// default: 0
```

```
    /// maximum: 20
```

```
    var sliceSpace: CGFloat { get set }
```

```
    /// When enabled, slice spacing will be 0.0 when the smallest value is going to be smaller than the slice spacing itself.
```

```
var automaticallyDisableSliceSpacing: Bool { get set }
```

```
/// indicates the selection distance of a pie slice
```

```
var selectionShift: CGFloat { get set }
```

```
var xValuePosition: PieChartDataSet.ValuePosition { get set }
```

```
var yValuePosition: PieChartDataSet.ValuePosition { get set }
```

```
/// When valuePosition is OutsideSlice, indicates line color
```

```
var valueLineColor: NSUIColor? { get set }
```

```
/// When valuePosition is OutsideSlice and enabled, line will have the same color as the  
slice
```

```
var useValueColorForLine: Bool { get set }
```

```
/// When valuePosition is OutsideSlice, indicates line width
```

```
var valueLineWidth: CGFloat { get set }
```

/// When valuePosition is OutsideSlice, indicates offset as percentage out of the slice size

var valueLinePart1OffsetPercentage: CGFloat { get set }

/// When valuePosition is OutsideSlice, indicates length of first half of the line

var valueLinePart1Length: CGFloat { get set }

/// When valuePosition is OutsideSlice, indicates length of second half of the line

var valueLinePart2Length: CGFloat { get set }

/// When valuePosition is OutsideSlice, this allows variable line length

var valueLineVariableLength: Bool { get set }

/// the font for the slice-text labels

var entryLabelFont: NSUIFont? { get set }

/// the color for the slice-text labels

var entryLabelColor: NSUIColor? { get set }

```
/// get/sets the color for the highlighted sector

var highlightColor: NSUIColor? { get set }

}

//

// IRadarChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation
```

```
import CoreGraphics
```

```
@objc
```

```
public protocol IRadarChartDataSet: ILineRadarChartDataSet
```

```
{
```

```
    // MARK: - Data functions and accessors
```

```
    // MARK: - Styling functions and accessors
```

```
    /// flag indicating whether highlight circle should be drawn or not
```

```
    var drawHighlightCircleEnabled: Bool { get set }
```

```
    var isDrawHighlightCircleEnabled: Bool { get }
```

```
    var highlightCircleFillColor: NSUIColor? { get set }
```

```
    /// The stroke color for highlight circle.
```



```
/// If `nil`, the color of the dataset is taken.

var highlightCircleStrokeColor: NSUIColor? { get set }

var highlightCircleStrokeAlpha: CGFloat { get set }

var highlightCircleInnerRadius: CGFloat { get set }

var highlightCircleOuterRadius: CGFloat { get set }

var highlightCircleStrokeWidth: CGFloat { get set }

}

//

// IScatterChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS
```

```

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc

public protocol IScatterChartDataSet: ILineScatterCandleRadarChartDataSet

{

    // MARK: - Data functions and accessors


    // MARK: - Styling functions and accessors


    /// The size the scatter shape will have

    var scatterShapeSize: CGFloat { get }

```

```

    /// - Returns: The radius of the hole in the shape (applies to Square, Circle and Triangle)

    /// Set this to <= 0 to remove holes.

    /// **default**: 0.0

    var scatterShapeHoleRadius: CGFloat { get }

    /// - Returns: Color for the hole in the shape. Setting to `nil` will behave as transparent.

    /// **default**: nil

    var scatterShapeHoleColor: NSUIColor? { get }

    /// The IShapeRenderer responsible for rendering this DataSet.

    var shapeRenderer: IShapeRenderer? { get }

}

//

// IShapeRenderer.swift

// Charts

//

```

```
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```
// A port of MPAndroidChart for iOS
```

```
// Licensed under Apache License 2.0
```

```
//
```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
@objc
```

```
public protocol IShapeRenderer: class
```

```
{
```

```
    /// Renders the provided ScatterDataSet with a shape.
```

```
    ///
```

```
    /// - Parameters:
```

```
    /// - context:    CGContext for drawing on
```

```

/// - dataSet:      The DataSet to be drawn

/// - viewPortHandler: Contains information about the current state of the view

/// - point:        Position to draw the shape at

/// - color:         Color to draw the shape

func renderShape(

    context: CGContext,

    dataSet: IScatterChartDataSet,

    viewPortHandler: ViewPortHandler,

    point: CGPoint,

    color: NSUIColor)

}

//

// IValueFormatter.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

```

```

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

/// Interface that allows custom formatting of all values inside the chart before they are
drawn to the screen.

///

/// Simply create your own formatting class and let it implement ValueFormatter. Then
override the stringForValue()

/// method and return whatever you want.

@objc(IChartValueFormatter)

public protocol IValueFormatter: class

{

```

```

/// Called when a value (from labels inside the chart) is formatted before being drawn.

///

/// For performance reasons, avoid excessive calculations and memory allocations inside
this method.

///

/// - Parameters:

/// - value:          The value to be formatted

/// - dataSetIndex:   The index of the DataSet the entry in focus belongs to

/// - viewPortHandler: provides information about the current chart state (scale,
translation, ...)

/// - Returns:         The formatted label ready to be drawn

func stringForValue(_ value: Double,

                    entry: ChartDataEntry,

                    dataSetIndex: Int,

                    viewPortHandler: ViewPortHandler?) -> String

}

//

// IndexAxisValueFormatter.swift

```

```

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

/// This formatter is used for passing an array of x-axis labels, on whole x steps.

@objc(ChartIndexAxisValueFormatter)

open class IndexAxisValueFormatter: NSObject, IAxisValueFormatter

{

    private var _values: [String] = [String]()

    private var _valueCount: Int = 0

```



```

@objc public var values: [String]

{

    get

    {

        return _values

    }

    set

    {

        _values = newValue

        _valueCount = _values.count

    }

}

```

```

public override init()

{

    super.init()

```

```
}
```

```
@objc public init(values: [String])
```

```
{
```

```
    super.init()
```

```
    self.values = values
```

```
}
```

```
@objc public static func with(values: [String]) -> IndexAxisValueFormatter?
```

```
{
```

```
    return IndexAxisValueFormatter(values: values)
```

```
}
```

```
open func stringForValue(_ value: Double,
```

```
    axis: AxisBase?) -> String
```

```

{

    let index = Int(value.rounded())

    guard values.indices.contains(index), index == Int(value) else { return "" }

    return _values[index]

}

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

Properties and behaviors specific to Kilgo service characteristics.

```
*/
```

```
import HomeKit
```

```
extension HMCharacteristic {
```

```
    var stateNames: [String] {
```

```

switch characteristicType {

    case HMCharacteristicTypeCurrentLockMechanismState: return ["Unsecured",
"Secured", "Jammed", "Unknown"]

    case HMCharacteristicTypeTargetLockMechanismState: return ["Not Locked",
"Locked"]

    case HMCharacteristicTypeCurrentDoorState: return ["Open", "Closed", "Opening",
"Closing", "Stopped"]

    case HMCharacteristicTypeTargetDoorState: return ["Open", "Closed"]

    default: return []

}

}

```

/// Indicates if you can write to the characteristic.

```

var isWriteable: Bool {

    return properties.contains(HMCharacteristicPropertyWritable)

}

```

/// Indicates if you can read from the characteristic.

```

var isReadable: Bool {

```

```
        return properties.contains(HMCharacteristicPropertyReadable)

    }
```

/// Indicates if the characteristic is both readable and writable.

```
var isReadWrite: Bool {

    return isReadable && isWritable

}
```

/// Indicates that the characteristic value is a floating point number.

```
var isFloat: Bool {

    return metadata?.format == HMCharacteristicMetadataFormatFloat

}
```

/// Indicates that the characteristic value is a signed integer.

```
var isInt: Bool {

    return metadata?.format == HMCharacteristicMetadataFormatInt

}
```

```

/// Indicates that the characteristic value is an unsigned integer.

var isUInt: Bool {

    return metadata?.format == HMCharacteristicMetadataFormatUInt8

        || metadata?.format == HMCharacteristicMetadataFormatUInt16

        || metadata?.format == HMCharacteristicMetadataFormatUInt32

        || metadata?.format == HMCharacteristicMetadataFormatUInt64

}

```

```

/// Indicates that the characteristic value is a number.

//  var isNumeric: Bool {

//      return isInt || isFloat || isUInt

//  }

```

```

/// Indicates that the characteristic value is a Boolean.

var isBool: Bool {

```

```

    return metadata?.format == HMCharacteristicMetadataFormatBool
}

/// The difference between the characteristic's maximum and minimum values.

var span: Decimal {

    guard let max = metadata?.maximumValue?.decimalValue,

        let min = metadata?.minimumValue?.decimalValue else {

            return 1.0

        }

    return max - min

}

/// The characteristic value as a float.

var floatValue: Float? {

    return (value as? NSNumber)?.floatValue

}

```

```

/// The characteristic value as a decimal value.

var decimalValue: Decimal? {

    return (value as? NSNumber)?.decimalValue

}

/// Returns the value that you get by mapping the given fraction

/// to the characteristic's span, accounting for step size.

func valueFor(fraction: Float) -> Decimal {

    let interval = metadata?.stepValue?.decimalValue ?? 1.0

    var inVal = (Decimal(Double(fraction)) * span) / interval

    var outVal: Decimal = 0.0

    NSDecimalRound(&outVal, &inVal, 0, .plain)

    return outVal * interval

}

/// A name that best represents the characteristic in the UI.

var displayName: String {

```



```

        return metadata?.manufacturerDescription ?? localizedDescription
    }

```

/// A string that represents the characteristic's value.

```

var formattedValueString: String {

    guard

        let value = value,

        let metadata = metadata else { return "—" }

    // Use the metadata to drive the string formatting.

    switch metadata.format {

    case HMCharacteristicMetadataFormatString:

        return value as? String ?? "—"

    case HMCharacteristicMetadataFormatInt,

        HMCharacteristicMetadataFormatUInt8,

        HMCharacteristicMetadataFormatUInt16,

```

```

HMCharacteristicMetadataFormatUInt32,

HMCharacteristicMetadataFormatUInt64:

guard let intValue = value as? Int else { return "—" }

guard stateNames.isEmpty else { return stateNames[intValue] }

return String(intValue)


case HMCharacteristicMetadataFormatFloat:

guard let floatValue = floatValue else { return "—" }

return String(format: "%.1f", floatValue)


case HMCharacteristicMetadataFormatBool:

guard let boolValue = value as? Bool else { return "—" }

return boolValue ? "YES" : "NO"


default:

return "—"

}

```

```
}
```

```
}
```

```
/*
```

See LICENSE folder for this sample's licensing information.

Abstract:

Properties and behaviors specific to Kilgo services.

```
*/
```

```
import HomeKit
```

```
extension HMService {
```

```
/// The service types that Kilgo garage doors support.
```

```
enum KilgoServiceType {
```

```
    case lightBulb, garageDoor, unknown
```

```
    case hmswitch
```

```
}
```

```
/// The Kilgo service type for this service.
```

```
var kilgoServiceType: KilgoServiceType {
```

```
    switch serviceType {
```

```
        case HMServiceTypeLightbulb: return .lightBulb
```

```
        case HMServiceTypeGarageDoorOpener: return .garageDoor
```

```
        case HMServiceTypeSwitch: return .hmswitch
```

```
        default: return .unknown
```

```
    }
```

```
}
```

```
/// The primary characteristic type to be controlled, given the service type.
```

```
var primaryControlCharacteristicType: String? {
```

```
    switch kilgoServiceType {
```

```
        case .lightBulb: return HMCharacteristicTypePowerState
```

```
case .garageDoor: return HMCharacteristicTypeTargetDoorState
```

```
case .hmswitch: return HMCharacteristicTypePowerState
```

```
case .unknown: return nil
```

```
}
```

```
}
```

/// The primary characteristic controlled by tapping the accessory cell in the accessory list.

```
var primaryControlCharacteristic: HMCharacteristic? {
```

```
    return characteristics.first { $0.characteristicType ==  
primaryControlCharacteristicType }
```

```
}
```

/// The primary characteristic type to be displayed, given the service type.

```
var primaryDisplayCharacteristicType: String? {
```

```
    switch kilgoServiceType {
```

```

        case .lightBulb: return HMCharacteristicTypePowerState

        case .garageDoor: return HMCharacteristicTypeCurrentDoorState

        case .hmswitch: return HMCharacteristicTypePowerState

        case .unknown: return nil

    }

}

/// The primary characteristic visible in the accessory cell in the accessory list.

var primaryDisplayCharacteristic: HMCharacteristic? {

    return characteristics.first { $0.characteristicType ==
primaryDisplayCharacteristicType }

}

/// The custom displayable characteristic types specific to Kilgo devices.

enum KilgoCharacteristicTypes: String {

    case fadeRate = "7E536242-341C-4862-BE90-272CE15BD633"

}

```

```

/// The list of characteristics to display in the UI.

var displayableCharacteristics: [HMCharacteristic] {

    let characteristicTypes = [HMCharacteristicTypePowerState,

                                HMCharacteristicTypeBrightness,

                                HMCharacteristicTypeHue,

                                HMCharacteristicTypeSaturation,

                                HMCharacteristicTypeTargetDoorState,

                                HMCharacteristicTypeCurrentDoorState,

                                HMCharacteristicTypeObstructionDetected,

                                HMCharacteristicTypeTargetLockMechanismState,

                                HMCharacteristicTypeCurrentLockMechanismState,

                                KilgoCharacteristicTypes.fadeRate.rawValue]

    return characteristics.filter { characteristicTypes.contains($0.characteristicType) }

}

/// A graphical representation of the service, given its current state.

```

```

var icon: UIImage? {

    let (_, icon) = stateAndIcon

    return icon

}

```

/// A textual representation of the current state of the service.

```

var state: String {

    let (state, _) = stateAndIcon

    return state

}

```

/// A tuple containing a string and icon representing the current state of the service.

```

private var stateAndIcon: (String, UIImage?) {

    switch kilgoServiceType {

    case .garageDoor:

        if let value = primaryDisplayCharacteristic?.value as? Int,

            let doorState = HMCharacteristicValueDoorState(rawValue: value) {

```



```

switch doorState {

case .open: return ("Open", #imageLiteral(resourceName: "door-open"))

case .closed: return ("Closed", #imageLiteral(resourceName: "door-closed"))

case .opening: return ("Opening", #imageLiteral(resourceName: "door-
opening"))

case .closing: return ("Closing", #imageLiteral(resourceName: "door-closing"))

case .stopped: return ("Stopped", #imageLiteral(resourceName: "door-closed"))

@unknown default: return ("Unknown", nil)

}

} else {

return ("Unknown", #imageLiteral(resourceName: "door-closed"))

}

case .lightBulb:

if let value = primaryDisplayCharacteristic?.value as? Bool {

if value {

return ("On", #imageLiteral(resourceName: "bulb-on"))

} else {

return ("Off", #imageLiteral(resourceName: "bulb-off"))

```

```

    }

    } else {

        return ("Unknown", #imageLiteral(resourceName: "bulb-off"))

    }

case .unknown:

    return ("Unknown", nil)

case .hmswitch:

    if let value = primaryDisplayCharacteristic?.value as? Bool {

        if value {

            return ("On", #imageLiteral(resourceName: "bulb-on"))

        } else {

            return ("Off", #imageLiteral(resourceName: "bulb-off"))

        }

    } else {

        return ("Unknown", #imageLiteral(resourceName: "bulb-off"))

    }

}

```

```
}

}

//

// Legend.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc(ChartLegend)
```

```

open class Legend: ComponentBase

{

    @objc(ChartLegendForm)

    public enum Form: Int

    {

        /// Avoid drawing a form

        case none


        /// Do not draw the a form, but leave space for it

        case empty


        /// Use default (default dataset's form to the legend's form)

        case `default`


        /// Draw a square

        case square
    }
}

```

```
/// Draw a circle
```

```
case circle
```

```
/// Draw a horizontal line
```

```
case line
```

```
}
```

```
@objc(CharLegendHorizontalAlignment)
```

```
public enum HorizontalAlignment: Int
```

```
{
```

```
case left
```

```
case center
```

```
case right
```

```
}
```

```
@objc(CharLegendVerticalAlignment)
```

```
public enum VerticalAlignment: Int
```

```
{  
  
    case top  
  
    case center  
  
    case bottom  
  
}
```

@objc(CharLegendOrientation)

public enum Orientation: Int

```
{  
  
    case horizontal  
  
    case vertical  
  
}
```

@objc(CharLegendDirection)

public enum Direction: Int

```
{  
  
    case leftToRight
```

```
case rightToLeft
```

```
}
```

```
/// The legend entries array
```

```
@objc open var entries = [LegendEntry]()
```

```
/// Entries that will be appended to the end of the auto calculated entries after calculating  
the legend.
```

```
/// (if the legend has already been calculated, you will need to call  
notifyDataSetChanged() to let the changes take effect)
```

```
@objc open var extraEntries = [LegendEntry]()
```

```
/// Are the legend labels/colors a custom value or auto calculated? If false, then it's auto,  
if true, then custom.
```

```
///
```

```
/// default: false (automatic legend)
```

```
private var _isLegendCustom = false
```

```
/// The horizontal alignment of the legend
```

```
@objc open var horizontalAlignment: HorizontalAlignment = HorizontalAlignment.left
```

```
/// The vertical alignment of the legend
```

```
@objc open var verticalAlignment: VerticalAlignment = VerticalAlignment.bottom
```

```
/// The orientation of the legend
```

```
@objc open var orientation: Orientation = Orientation.horizontal
```

```
/// Flag indicating whether the legend will draw inside the chart or outside
```

```
@objc open var drawInside: Bool = false
```

```
/// Flag indicating whether the legend will draw inside the chart or outside
```

```
@objc open var isDrawInsideEnabled: Bool { return drawInside }
```

```
/// The text direction of the legend
```

```
@objc open var direction: Direction = Direction.leftToRight
```



```
@objc open var font: NSUIFont = NSUIFont.systemFont(ofSize: 10.0)
```

```
@objc open var textColor = NSUIColor.labelOrBlack
```

```
/// The form/shape of the legend forms
```

```
@objc open var form = Form.square
```

```
/// The size of the legend forms
```

```
@objc open var formSize = CGFloat(8.0)
```

```
/// The line width for forms that consist of lines
```

```
@objc open var formLineWidth = CGFloat(3.0)
```

```
/// Line dash configuration for shapes that consist of lines.
```

```
///
```

```
/// This is how much (in pixels) into the dash pattern are we starting from.
```

```
@objc open var formLineDashPhase: CGFloat = 0.0
```

```
/// Line dash configuration for shapes that consist of lines.
```

```
///
```

```
/// This is the actual dash pattern.
```

```
/// I.e. [2, 3] will paint [-- -- ]
```

```
/// [1, 3, 4, 2] will paint [- ---- - ---- ]
```

```
@objc open var formLineDashLengths: [CGFloat]?
```

```
@objc open var xEntrySpace = CGFloat(6.0)
```

```
@objc open var yEntrySpace = CGFloat(0.0)
```

```
@objc open var formToTextSpace = CGFloat(5.0)
```

```
@objc open var stackSpace = CGFloat(3.0)
```

```
@objc open var calculatedLabelSizes = [CGSize]()
```

```
@objc open var calculatedLabelBreakPoints = [Bool]()
```

```
@objc open var calculatedLineSizes = [CGSize]()
```

```
public override init()
```

```
{  
  
    super.init()  
  
    self.xOffset = 5.0  
  
    self.yOffset = 3.0  
  
}
```

```
@objc public init(entries: [LegendEntry])  
  
{  
  
    super.init()  
  
    self.entries = entries  
  
}
```

```
@objc open func getMaximumEntrySize(withFont font: NSUIFont) -> CGSize  
  
{  
  
    var maxW = CGFloat(0.0)
```

```
var maxH = CGFloat(0.0)
```

```
var maxFormSize: CGFloat = 0.0
```

```
for entry in entries
```

```
{
```

```
    let formSize = entry.formSize.isNaN ? self.formSize : entry.formSize
```

```
    if formSize > maxFormSize
```

```
    {
```

```
        maxFormSize = formSize
```

```
    }
```

```
guard let label = entry.label
```

```
    else { continue }
```

```
let size = (label as NSString).size(withAttributes: [.font: font])
```

```

        if size.width > maxW
        {
            maxW = size.width
        }

        if size.height > maxH
        {
            maxH = size.height
        }
    }

    return CGSize(

        width: maxW + maxFormSize + formToTextSpace,

        height: maxH

    )
}

```

```

@objc open var neededWidth = CGFloat(0.0)

```

```
@objc open var neededHeight = CGFloat(0.0)
```

```
@objc open var textWidthMax = CGFloat(0.0)
```

```
@objc open var textHeightMax = CGFloat(0.0)
```

```
/// flag that indicates if word wrapping is enabled
```

```
/// this is currently supported only for `orientation == Horizontal`.
```

```
/// you may want to set maxSizePercent when word wrapping, to set the point where the  
text wraps.
```

```
///
```

```
/// default: true
```

```
@objc open var wordWrapEnabled = true
```

```
/// if this is set, then word wrapping the legend is enabled.
```

```
@objc open var isWordWrapEnabled: Bool { return wordWrapEnabled }
```

```
/// The maximum relative size out of the whole chart view in percent.
```

```
/// If the legend is to the right/left of the chart, then this affects the width of the legend.
```

```
/// If the legend is to the top/bottom of the chart, then this affects the height of the legend.
```

```

///

/// **default**: 0.95 (95%)

@objc open var maxSizePercent: CGFloat = 0.95


@objc open func calculateDimensions(labelFont: NSUIFont, viewPortHandler:
ViewPortHandler)

{

    let maxEntrySize = getMaximumEntrySize(withFont: labelFont)

    let defaultFormSize = self.formSize

    let stackSpace = self.stackSpace

    let formToTextSpace = self.formToTextSpace

    let xEntrySpace = self.xEntrySpace

    let yEntrySpace = self.yEntrySpace

    let wordWrapEnabled = self.wordWrapEnabled

    let entries = self.entries

    let entryCount = entries.count


    textWidthMax = maxEntrySize.width

```

```
textHeightMax = maxEntrySize.height
```

```
switch orientation
```

```
{
```

```
case .vertical:
```

```
    var maxWidth = CGFloat(0.0)
```

```
    var width = CGFloat(0.0)
```

```
    var maxHeight = CGFloat(0.0)
```

```
    let labelLineHeight = labelFont.lineHeight
```

```
    var wasStacked = false
```

```
    for i in 0 ..< entryCount
```

```
    {
```

```
        let e = entries[i]
```

```
        let drawingForm = e.form != .none
```



```
let formSize = e.formSize.isNaN ? defaultFormSize : e.formSize
```

```
let label = e.label
```

```
if !wasStacked
```

```
{
```

```
    width = 0.0
```

```
}
```

```
if drawingForm
```

```
{
```

```
    if wasStacked
```

```
    {
```

```
        width += stackSpace
```

```
    }
```

```
    width += formSize
```

```
}
```

```

if label != nil

{

    let size = (label! as NSString).size(withAttributes: [.font: labelFont])


    if drawingForm && !wasStacked

    {

        width += formToTextSpace

    }

    else if wasStacked

    {

        maxWidth = max(maxWidth, width)

        maxHeight += labelLineHeight + yEntrySpace

        width = 0.0

        wasStacked = false

    }


    width += size.width

```

```

        maxHeight += labelLineHeight + yEntrySpace

    }

    else

    {

        wasStacked = true

        width += formSize

        if i < entryCount - 1

        {

            width += stackSpace

        }

    }

    maxWidth = max(maxWidth, width)

}

neededWidth = maxWidth

```

```
neededHeight = maxHeight
```

```
case .horizontal:
```

```
let labelLineHeight = labelFont.lineHeight
```

```
let contentWidth: CGFloat = viewPortHandler.contentWidth * maxSizePercent
```

```
// Prepare arrays for calculated layout
```

```
if calculatedLabelSizes.count != entryCount
```

```
{
```

```
    calculatedLabelSizes = [CGSize](repeating: CGSize(), count: entryCount)
```

```
}
```

```
if calculatedLabelBreakPoints.count != entryCount
```

```
{
```

```
    calculatedLabelBreakPoints = [Bool](repeating: false, count: entryCount)
```

```
}
```

```
calculatedLineSizes.removeAll(keepingCapacity: true)
```

```
// Start calculating layout
```

```
let labelAttrs = [NSAttributedString.Key.font: labelFont]
```

```
var maxLineWidth: CGFloat = 0.0
```

```
var currentLineWidth: CGFloat = 0.0
```

```
var requiredWidth: CGFloat = 0.0
```

```
var stackedStartIndex: Int = -1
```

```
for i in 0 ..< entryCount
```

```
{
```

```
    let e = entries[i]
```

```
    let drawingForm = e.form != .none
```

```
    let label = e.label
```

```
calculatedLabelBreakPoints[i] = false
```

```
if stackedStartIndex == -1
```

```
{
```

```
    // we are not stacking, so required width is for this label only
```

```
    requiredWidth = 0.0
```

```
}
```

```
else
```

```
{
```

```
    // add the spacing appropriate for stacked labels/forms
```

```
    requiredWidth += stackSpace
```

```
}
```

```
// grouped forms have null labels
```

```
if label != nil
```

```
{
```

```

        calculatedLabelSizes[i] = (label! as NSString).size(withAttributes: labelAttrs)

        requiredWidth += drawingForm ? formToTextSpace + formSize : 0.0

        requiredWidth += calculatedLabelSizes[i].width

    }

    else

    {

        calculatedLabelSizes[i] = CGSize()

        requiredWidth += drawingForm ? formSize : 0.0


        if stackedStartIndex == -1

        {

            // mark this index as we might want to break here later

            stackedStartIndex = i

        }

    }

    if label != nil || i == entryCount - 1

```

```

{

    let requiredSpacing = currentLineWidth == 0.0 ? 0.0 : xEntrySpace

    if (!wordWrapEnabled || // No word wrapping, it must fit.

        currentLineWidth == 0.0 || // The line is empty, it must fit.

        (contentWidth - currentLineWidth >= requiredSpacing + requiredWidth))
// It simply fits

    {

        // Expand current line

        currentLineWidth += requiredSpacing + requiredWidth

    }

    else

    { // It doesn't fit, we need to wrap a line

        // Add current line size to array

        calculatedLineSizes.append(CGSize(width:  currentLineWidth,  height:
labelLineHeight))

        maxLineWidth = max(maxLineWidth, currentLineWidth)

```



```

        // Start a new line

        calculatedLabelBreakPoints[stackedStartIndex > -1 ? stackedStartIndex : i]
= true

        currentLineWidth = requiredWidth

    }

    if i == entryCount - 1

    { // Add last line size to array

        calculatedLineSizes.append(CGSize(width:  currentLineWidth,  height:
labelLineHeight))

        maxLineWidth = max(maxLineWidth, currentLineWidth)

    }

}

stackedStartIndex = label != nil ? -1 : stackedStartIndex

}

```

```

        neededWidth = maxLineWidth

        neededHeight = labelLineHeight * CGFloat(calculatedLineSizes.count) +

            yEntrySpace * CGFloat(calculatedLineSizes.count == 0 ? 0 :
(calculatedLineSizes.count - 1))

    }

    neededWidth += xOffset

    neededHeight += yOffset

}

/// MARK: - Custom legend

/// Sets a custom legend's entries array.

/// * A nil label will start a group.

/// This will disable the feature that automatically calculates the legend entries from the
datasets.

/// Call `resetCustom(...)` to re-enable automatic calculation (and then
`notifyDataSetChanged()` is needed).

@objc open func setCustom(entries: [LegendEntry])

```

```

{

    self.entries = entries

    _isLegendCustom = true

}

```

/// Calling this will disable the custom legend entries (set by `setLegend(...)`). Instead, the entries will again be calculated automatically (after `notifyDataSetChanged()` is called).

```

@objc open func resetCustom()

{

    _isLegendCustom = false

}

```

/// **default**: false (automatic legend)

/// `true` if a custom legend entries has been set

```

@objc open var isLegendCustom: Bool

{

    return _isLegendCustom

}

```

```

}

//

// LegendEntry.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc(CharLegendEntry)

open class LegendEntry: NSObject

```

```

{

    public override init()

    {

        super.init()

    }

    /// - Parameters:

    /// - label:          The legend entry text.

    ///                A `nil` label will start a group.

    /// - form:          The form to draw for this entry.

    /// - formSize:       Set to NaN to use the legend's default.

    /// - formLineWidth:  Set to NaN to use the legend's default.

    /// - formLineDashPhase:  Line dash configuration.

    /// - formLineDashLengths: Line dash configurationas NaN to use the legend's default.

    /// - formColor:      The color for drawing the form.

    @objc public init(label: String?,

        form: Legend.Form,

```

```

        formSize: CGFloat,

        formLineWidth: CGFloat,

        formLineDashPhase: CGFloat,

        formLineDashLengths: [CGFloat]?,

        formColor: NSUIColor?)

{

    self.label = label

    self.form = form

    self.formSize = formSize

    self.formLineWidth = formLineWidth

    self.formLineDashPhase = formLineDashPhase

    self.formLineDashLengths = formLineDashLengths

    self.formColor = formColor

}

/// The legend entry text.

/// A `nil` label will start a group.

```

@objc open var label: String?

/// The form to draw for this entry.

///

/// `None` will avoid drawing a form, and any related space.

/// `Empty` will avoid drawing a form, but keep its space.

/// `Default` will use the Legend's default.

@objc open var form: Legend.Form = .default

/// Form size will be considered except for when .None is used

///

/// Set as NaN to use the legend's default

@objc open var formSize: CGFloat = CGFloat.nan

/// Line width used for shapes that consist of lines.

///

/// Set to NaN to use the legend's default.

```
@objc open var formLineWidth: CGFloat = CGFloat.nan
```

```
/// Line dash configuration for shapes that consist of lines.
```

```
///
```

```
/// This is how much (in pixels) into the dash pattern are we starting from.
```

```
///
```

```
/// Set to NaN to use the legend's default.
```

```
@objc open var formLineDashPhase: CGFloat = 0.0
```

```
/// Line dash configuration for shapes that consist of lines.
```

```
///
```

```
/// This is the actual dash pattern.
```

```
/// I.e. [2, 3] will paint [-- -- ]
```

```
/// [1, 3, 4, 2] will paint [- ---- - ---- ]
```

```
///
```

```
/// Set to nil to use the legend's default.
```

```
@objc open var formLineDashLengths: [CGFloat]?
```



```
/// The color for drawing the form

@objc open var formColor: NSUIColor?

}

//

// LegendRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics
```

```

@objc(ChartLegendRenderer)

open class LegendRenderer: Renderer

{

    /// the legend object this renderer renders

    @objc open var legend: Legend?

    @objc public init(viewPortHandler: ViewPortHandler, legend: Legend?)

    {

        super.init(viewPortHandler: viewPortHandler)

        self.legend = legend

    }

    /// Prepares the legend and calculates all needed forms, labels and colors.

    @objc open func computeLegend(data: ChartData)

    {

```

```

guard let legend = legend else { return }

if !legend.isLegendCustom
{

    var entries: [LegendEntry] = []

    // loop for building up the colors and labels used in the legend

    for i in 0..

```

```

{

    let bds = dataSet as! IBarChartDataSet

    let sLabels = bds.stackLabels

    let minEntries = min(clrs.count, bds.stackSize)

    for j in 0..

```

```

entries.append(

    LegendEntry(

        label: label,

        form: dataSet.form,

        formSize: dataSet.formSize,

        formLineWidth: dataSet.formLineWidth,

        formLineDashPhase: dataSet.formLineDashPhase,

        formLineDashLengths: dataSet.formLineDashLengths,

        formColor: clr[j]

    )

)

}

if dataSet.label != nil

{

    // add the legend description label

```

```

entries.append(

    LegendEntry(

        label: dataSet.label,

        form: .none,

        formSize: CGFloat.nan,

        formLineWidth: CGFloat.nan,

        formLineDashPhase: 0.0,

        formLineDashLengths: nil,

        formColor: nil

    )

)

}

}

else if dataSet is IPieChartDataSet

{

    let pds = dataSet as! IPieChartDataSet

```

```

for j in 0..

```

```

{

    // add the legend description label

    entries.append(

        LegendEntry(

            label: dataSet.label,

            form: .none,

            formSize: CGFloat.nan,

            formLineWidth: CGFloat.nan,

            formLineDashPhase: 0.0,

            formLineDashLengths: nil,

            formColor: nil

        )

    )

}

}

else if dataSet is ICandleChartDataSet &&

```



```

        (dataSet as! ICandleChartDataSet).decreasingColor != nil

    {

        let candleDataSet = dataSet as! ICandleChartDataSet

        entries.append(

            LegendEntry(

                label: nil,

                form: dataSet.form,

                formSize: dataSet.formSize,

                formLineWidth: dataSet.formLineWidth,

                formLineDashPhase: dataSet.formLineDashPhase,

                formLineDashLengths: dataSet.formLineDashLengths,

                formColor: candleDataSet.decreasingColor

            )

        )

        entries.append(

```

```

        LegendEntry(

            label: dataSet.label,

            form: dataSet.form,

            formSize: dataSet.formSize,

            formLineWidth: dataSet.formLineWidth,

            formLineDashPhase: dataSet.formLineDashPhase,

            formLineDashLengths: dataSet.formLineDashLengths,

            formColor: candleDataSet.increasingColor

        )

    )

}

else

{ // all others

    for j in 0..

```

```
// if multiple colors are set for a DataSet, group them
```

```
if j < clr.count - 1 && j < entryCount - 1
```

```
{
```

```
    label = nil
```

```
}
```

```
else
```

```
{ // add label to the last entry
```

```
    label = dataSet.label
```

```
}
```

```
entries.append(
```

```
    LegendEntry(
```

```
        label: label,
```

```
        form: dataSet.form,
```

```
        formSize: dataSet.formSize,
```

```
        formLineWidth: dataSet.formLineWidth,
```

```

        formLineDashPhase: dataSet.formLineDashPhase,

        formLineDashLengths: dataSet.formLineDashLengths,

        formColor: clr[j]

    )

)

}

}

}

    legend.entries = entries + legend.extraEntries

}

// calculate all dimensions of the legend

    legend.calculateDimensions(labelFont:    legend.font,    viewPortHandler:
viewPortHandler)

}

```

```

@objc open func renderLegend(context: CGContext)

```

```
{  
  
    guard let legend = legend else { return }  
  
    if !legend.enabled  
  
    {  
  
        return  
  
    }  
  
    let labelFont = legend.font  
  
    let labelTextColor = legend.textColor  
  
    let labelLineHeight = labelFont.lineHeight  
  
    let formYOffset = labelLineHeight / 2.0  
  
    let entries = legend.entries  
  
    let defaultFormSize = legend.formSize  
  
    let formToTextSpace = legend.formToTextSpace
```

```
let xEntrySpace = legend.xEntrySpace
```

```
let yEntrySpace = legend.yEntrySpace
```

```
let orientation = legend.orientation
```

```
let horizontalAlignment = legend.horizontalAlignment
```

```
let verticalAlignment = legend.verticalAlignment
```

```
let direction = legend.direction
```

```
// space between the entries
```

```
let stackSpace = legend.stackSpace
```

```
let yoffset = legend.yOffset
```

```
let xoffset = legend.xOffset
```

```
var originPosX: CGFloat = 0.0
```

```
switch horizontalAlignment
```

```
{
```

```
case .left:
```

```
    if orientation == .vertical
```

```
    {
```

```
        originPosX = xoffset
```

```
    }
```

```
    else
```

```
    {
```

```
        originPosX = viewPortHandler.contentLeft + xoffset
```

```
    }
```

```
    if direction == .rightToLeft
```

```
    {
```

```
        originPosX += legend.neededWidth
```

```
    }
```

```
case .right:
```

```

if orientation == .vertical

{

    originPosX = viewPortHandler.chartWidth - xoffset

}

else

{

    originPosX = viewPortHandler.contentRight - xoffset

}


if direction == .leftToRight

{

    originPosX -= legend.neededWidth

}


case .center:

```



```

if orientation == .vertical

{

    originPosX = viewPortHandler.chartWidth / 2.0

}

else

{

    originPosX = viewPortHandler.contentLeft

        + viewPortHandler.contentWidth / 2.0

}


originPosX += (direction == .leftToRight

    ? +xoffset

    : -xoffset)


// Horizontally layed out legends do the center offset on a line basis,

// So here we offset the vertical ones only.

if orientation == .vertical

```

```

{

    if direction == .leftToRight

    {

        originPosX -= legend.neededWidth / 2.0 - xoffset

    }

    else

    {

        originPosX += legend.neededWidth / 2.0 - xoffset

    }

}

```

```

switch orientation

```

```

{

case .horizontal:

```

```

    let calculatedLineSizes = legend.calculatedLineSizes

```

```
let calculatedLabelSizes = legend.calculatedLabelSizes
```

```
let calculatedLabelBreakPoints = legend.calculatedLabelBreakPoints
```

```
var posX: CGFloat = originPosX
```

```
var posY: CGFloat
```

```
switch verticalAlignment
```

```
{
```

```
case .top:
```

```
    posY = yoffset
```

```
case .bottom:
```

```
    posY = viewPortHandler.chartHeight - yoffset - legend.neededHeight
```

```
case .center:
```

```
    posY = (viewPortHandler.chartHeight - legend.neededHeight) / 2.0 + yoffset
```

```
}
```

```

var lineIndex: Int = 0

for i in 0 ..< entries.count

{

    let e = entries[i]

    let drawingForm = e.form != .none

    let formSize = e.formSize.isNaN ? defaultFormSize : e.formSize

    if i < calculatedLabelBreakPoints.count &&

        calculatedLabelBreakPoints[i]

    {

        posX = originPosX

        posY += labelLineHeight + yEntrySpace

    }

    if posX == originPosX &&

```

```

horizontalAlignment == .center &&

lineIndex < calculatedLineSizes.count

{

    posX += (direction == .rightToLeft

        ? calculatedLineSizes[lineIndex].width

        : -calculatedLineSizes[lineIndex].width) / 2.0

    lineIndex += 1

}

let isStacked = e.label == nil // grouped forms have null labels

if drawingForm

{

    if direction == .rightToLeft

    {

        posX -= formSize

    }

}

```

```
drawForm(  
  
    context: context,  
  
    x: posX,  
  
    y: posY + formYOffset,  
  
    entry: e,  
  
    legend: legend)
```

```
if direction == .leftToRight  
  
    {  
  
        posX += formSize  
  
    }  
  
}
```

```
if !isStacked  
  
    {  
  
        if drawingForm
```

```
{  
  
    posX += direction == .rightToLeft ? -formToTextSpace : formToTextSpace  
  
}
```

```
if direction == .rightToLeft
```

```
{  
  
    posX -= calculatedLabelSizes[i].width  
  
}
```

```
drawLabel(  
  
    context: context,  
  
    x: posX,  
  
    y: posY,  
  
    label: e.label!,  
  
    font: labelFont,  
  
    textColor: labelTextColor)
```

```

        if direction == .leftToRight

        {

            posX += calculatedLabelSizes[i].width

        }

        posX += direction == .rightToLeft ? -xEntrySpace : xEntrySpace

    }

    else

    {

        posX += direction == .rightToLeft ? -stackSpace : stackSpace

    }

}

case .vertical:

    // contains the stacked legend size in pixels

    var stack = CGFloat(0.0)

```



```
var wasStacked = false
```

```
var posY: CGFloat = 0.0
```

```
switch verticalAlignment
```

```
{
```

```
case .top:
```

```
    posY = (horizontalAlignment == .center
```

```
        ? 0.0
```

```
        : viewPortHandler.contentTop)
```

```
    posY += yoffset
```

```
case .bottom:
```

```
    posY = (horizontalAlignment == .center
```

```
        ? viewPortHandler.chartHeight
```

```
        : viewPortHandler.contentBottom)
```

```
    posY -= legend.neededHeight + yoffset
```

```

case .center:

    posY = viewPortHandler.chartHeight / 2.0 - legend.neededHeight / 2.0 +
legend.yOffset

    }

for i in 0 ..< entries.count

{

    let e = entries[i]

    let drawingForm = e.form != .none

    let formSize = e.formSize.isNaN ? defaultFormSize : e.formSize

    var posX = originPosX

    if drawingForm

    {

        if direction == .leftToRight

```

```
{  
  
    posX += stack  
  
}  
  
else  
  
{  
  
    posX -= formSize - stack  
  
}
```

```
drawForm(  
  
    context: context,  
  
    x: posX,  
  
    y: posY + formYOffset,  
  
    entry: e,  
  
    legend: legend)
```

```
if direction == .leftToRight  
  
{
```

```

        posX += formSize

    }

}

if e.label != nil

{

    if drawingForm && !wasStacked

    {

        posX += direction == .leftToRight ? formToTextSpace : -formToTextSpace

    }

    else if wasStacked

    {

        posX = originPosX

    }

    if direction == .rightToLeft

    {

```

```

        posX -= (e.label! as NSString).size(withAttributes: [.font: labelFont]).width
    }

    if !wasStacked

    {

        drawLabel(context: context, x: posX, y: posY, label: e.label!, font:
labelFont, textColor: labelTextColor)

    }

    else

    {

        posY += labelLineHeight + yEntrySpace

        drawLabel(context: context, x: posX, y: posY, label: e.label!, font:
labelFont, textColor: labelTextColor)

    }

    // make a step down

    posY += labelLineHeight + yEntrySpace

    stack = 0.0

```

```

    }

    else

    {

        stack += formSize + stackSpace

        wasStacked = true

    }

}

}

}

```

```
private var _formLineSegmentsBuffer = [CGPoint](repeating: CGPoint(), count: 2)
```

```
/// Draws the Legend-form at the given position with the color at the given index.
```

```
@objc open func drawForm(
```

```
    context: CGContext,
```

```
    x: CGFloat,
```

```
    y: CGFloat,
```

```

        entry: LegendEntry,

        legend: Legend)

{

    guard

        let formColor = entry.formColor,

        formColor != NSUIColor.clear

        else { return }

    var form = entry.form

    if form == .default

    {

        form = legend.form

    }

    let formSize = entry.formSize.isNaN ? legend.formSize : entry.formSize

    context.saveGState()

```

```

defer { context.restoreGState() }

switch form
{

case .none:

    // Do nothing

    break


case .empty:

    // Do not draw, but keep space for the form

    break


case .default: fallthrough

case .circle:


    context.setFillColor(formColor.cgColor)

    context.fillEllipse(in: CGRect(x: x, y: y - formSize / 2.0, width: formSize, height:
formSize))

```



```
case .square:
```

```
    context.setFillColor(formColor.cgColor)
```

```
    context.fill(CGRect(x: x, y: y - formSize / 2.0, width: formSize, height: formSize))
```

```
case .line:
```

```
    let formLineWidth = entry.formLineWidth.isNaN ? legend.formLineWidth :  
    entry.formLineWidth
```

```
    let formLineDashPhase = entry.formLineDashPhase.isNaN ?  
    legend.formLineDashPhase : entry.formLineDashPhase
```

```
    let formLineDashLengths = entry.formLineDashLengths == nil ?  
    legend.formLineDashLengths : entry.formLineDashLengths
```

```
    context.setLineWidth(formLineWidth)
```

```
    if formLineDashLengths != nil && formLineDashLengths!.count > 0
```

```
    {
```

```

        context.setLineDash(phase:          formLineDashPhase,          lengths:
formLineDashLengths!)

    }

    else

    {

        context.setLineDash(phase: 0.0, lengths: [])

    }


    context.setStrokeColor(formColor.cgColor)


    _formLineSegmentsBuffer[0].x = x

    _formLineSegmentsBuffer[0].y = y

    _formLineSegmentsBuffer[1].x = x + formSize

    _formLineSegmentsBuffer[1].y = y

    context.strokeLineSegments(between: _formLineSegmentsBuffer)

}

}

```

```

    /// Draws the provided label at the given position.

    @objc open func drawLabel(context: CGContext, x: CGFloat, y: CGFloat, label: String,
font: UIFont, textColor: UIColor)

    {

        ChartUtils.drawText(context: context, text: label, point: CGPoint(x: x, y: y),
align: .left, attributes: [NSAttributedString.Key.font: font,
NSAttributedString.Key.foregroundColor: textColor])

    }

}

//

// LineChartData.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

```

```
import Foundation
```

```
/// Data object that encapsulates all data associated with a LineChart.
```

```
open class LineChartData: ChartData
```

```
{
```

```
    public override init()
```

```
    {
```

```
        super.init()
```

```
    }
```

```
    public override init(dataSets: [IChartDataSet]?)
```

```
    {
```

```
        super.init(dataSets: dataSets)
```

```
    }
```

```
}
```

```
//
```

```
// LineChartDataProvider.swift
```

```
// Charts
```

```
//
```

```
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```
// A port of MPAndroidChart for iOS
```

```
// Licensed under Apache License 2.0
```

```
//
```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
@objc
```

```
public protocol LineChartDataProvider: BarLineScatterCandleBubbleChartDataProvider
```

```
{
```

```
    var lineData: LineChartData? { get }
```

```
func getAxis(_ axis: YAxis.AxisDependency) -> YAxis

}

//

// LineChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics
```

```

open class LineChartDataSet: LineRadarChartDataSet, ILineChartDataSet

{

    @objc(LineChartMode)

    public enum Mode: Int

    {

        case linear

        case stepped

        case cubicBezier

        case horizontalBezier

    }

    private func initialize()

    {

        // default color

        circleColors.append(NSUIColor(red: 140.0/255.0, green: 234.0/255.0, blue:
255.0/255.0, alpha: 1.0))

    }

```

```
public required init()
```

```
{
```

```
    super.init()
```

```
    initialize()
```

```
}
```

```
public override init(entries: [ChartDataEntry]?, label: String?)
```

```
{
```

```
    super.init(entries: entries, label: label)
```

```
    initialize()
```

```
}
```

```
// MARK: - Data functions and accessors
```

```
// MARK: - Styling functions and accessors
```



```

/// The drawing mode for this line dataset

///

/// **default**: Linear

open var mode: Mode = Mode.linear


private var _cubicIntensity = CGFloat(0.2)


/// Intensity for cubic lines (min = 0.05, max = 1)

///

/// **default**: 0.2

open var cubicIntensity: CGFloat

{

    get

    {

        return _cubicIntensity

    }

    set

```

```

    {

        _cubicIntensity = newValue.clamped(to: 0.05...1)

    }

}

```

/// The radius of the drawn circles.

```
open var circleRadius = CGFloat(8.0)
```

/// The hole radius of the drawn circles

```
open var circleHoleRadius = CGFloat(4.0)
```

```
open var circleColors = [NSUIColor]()
```

/// - Returns: The color at the given index of the DataSet's circle-color array.

/// Performs a IndexOutOfBounds check by modulus.

```
open func getCircleColor(atIndex index: Int) -> NSUIColor?
```

```

{

```

```

let size = circleColors.count

let index = index % size

if index >= size

{

    return nil

}

return circleColors[index]

}

```

/// Sets the one and ONLY color that should be used for this DataSet.

/// Internally, this recreates the colors array and adds the specified color.

```

open func setCircleColor(_ color: NSUIColor)

{

    circleColors.removeAll(keepingCapacity: false)

    circleColors.append(color)

}

```

```
open func setCircleColors(_ colors: NSUIColor...)
```

```
{
```

```
    circleColors.removeAll(keepingCapacity: false)
```

```
    circleColors.append(contentsOf: colors)
```

```
}
```

```
/// Resets the circle-colors array and creates a new one
```

```
open func resetCircleColors(_ index: Int)
```

```
{
```

```
    circleColors.removeAll(keepingCapacity: false)
```

```
}
```

```
/// If true, drawing circles is enabled
```

```
open var drawCirclesEnabled = true
```

```
/// `true` if drawing circles for this DataSet is enabled, `false` ifnot
```

```
open var isDrawCirclesEnabled: Bool { return drawCirclesEnabled }
```

```

/// The color of the inner circle (the circle-hole).

open var circleHoleColor: NSUIColor? = NSUIColor.white


/// `true` if drawing circles for this DataSet is enabled, `false` ifnot

open var drawCircleHoleEnabled = true


/// `true` if drawing the circle-holes is enabled, `false` ifnot.

open var isDrawCircleHoleEnabled: Bool { return drawCircleHoleEnabled }


/// This is how much (in pixels) into the dash pattern are we starting from.

open var lineDashPhase = CGFloat(0.0)


/// This is the actual dash pattern.

/// I.e. [2, 3] will paint [-- -- ]

/// [1, 3, 4, 2] will paint [- ---- - ---- ]

open var lineDashLengths: [CGFloat]?

```

```

/// Line cap type, default is CGLineCap.Butt

open var lineCapType = CGLineCap.butt


/// formatter for customizing the position of the fill-line

private var _fillFormatter: IFillFormatter = DefaultFillFormatter()


/// Sets a custom IFillFormatter to the chart that handles the position of the filled-line for
each DataSet. Set this to null to use the default logic.

open var fillFormatter: IFillFormatter?

{

    get

    {

        return _fillFormatter

    }

    set

    {

        _fillFormatter = newValue ?? DefaultFillFormatter()
    }
}

```

```
}
```

```
}
```

```
// MARK: NSCopying
```

```
open override func copy(with zone: NSZone? = nil) -> Any
```

```
{
```

```
    let copy = super.copy(with: zone) as! LineChartDataSet
```

```
    copy.circleColors = circleColors
```

```
    copy.circleHoleColor = circleHoleColor
```

```
    copy.circleRadius = circleRadius
```

```
    copy.circleHoleRadius = circleHoleRadius
```

```
    copy.cubicIntensity = cubicIntensity
```

```
    copy.lineDashPhase = lineDashPhase
```

```
    copy.lineDashLengths = lineDashLengths
```

```
    copy.lineCapType = lineCapType
```

```
    copy.drawCirclesEnabled = drawCirclesEnabled
```

```
        copy.drawCircleHoleEnabled = drawCircleHoleEnabled

        copy.mode = mode

        copy._fillFormatter = _fillFormatter

        return copy

    }

}

//

// LineChartRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//
```



```
import Foundation
```

```
import CoreGraphics
```

```
open class LineChartRenderer: LineRadarRenderer
```

```
{
```

```
    // TODO: Currently, this nesting isn't necessary for LineCharts. However, it will make  
    it much easier to add a custom rotor
```

```
    // that navigates between datasets.
```

```
    // NOTE: Unlike the other renderers, LineChartRenderer populates  
    accessibleChartElements in drawCircles due to the nature of its drawing options.
```

```
    /// A nested array of elements ordered logically (i.e not in visual/drawing order) for use  
    with VoiceOver.
```

```
    private lazy var accessibilityOrderedElements: [[NSUIAccessibilityElement]] =  
    accessibilityCreateEmptyOrderedElements()
```

```
    @objc open weak var dataProvider: LineChartDataProvider?
```

```
    @objc public init(dataProvider: LineChartDataProvider, animator: Animator,  
    viewportHandler: ViewPortHandler)
```

```
{
```

```

super.init(animator: animator, viewPortHandler: viewPortHandler)

self.dataProvider = dataProvider
}

open override func drawData(context: CGContext)
{
    guard let lineData = dataProvider?.lineData else { return }

    for i in 0 ..< lineData.dataSetCount
    {
        guard let set = lineData.getDataSetByIndex(i) else { continue }

        if set.isVisible
        {
            if !(set is ILineChartDataSet)
            {

```

```
fatalError("Datasets for LineChartRenderer must conform to
ILineChartDataSet")
```

```
}
```

```
drawDataSet(context: context, dataSet: set as! ILineChartDataSet)
```

```
}
```

```
}
```

```
}
```

```
@objc open func drawDataSet(context: CGContext, dataSet: ILineChartDataSet)
```

```
{
```

```
    if dataSet.entryCount < 1
```

```
    {
```

```
        return
```

```
    }
```

```
    context.saveGState()
```

```

context.setLineWidth(dataSet.lineWidth)

if dataSet.lineDashLengths != nil

{

    context.setLineDash(phase:          dataSet.lineDashPhase,          lengths:
dataSet.lineDashLengths!)

}

else

{

    context.setLineDash(phase: 0.0, lengths: [])

}


context.setLineCap(dataSet.lineCapType)


// if drawing cubic lines is enabled

switch dataSet.mode

{

case .linear: fallthrough

case .stepped:

```

```
drawLinear(context: context, dataSet: dataSet)
```

```
case .cubicBezier:
```

```
drawCubicBezier(context: context, dataSet: dataSet)
```

```
case .horizontalBezier:
```

```
drawHorizontalBezier(context: context, dataSet: dataSet)
```

```
}
```

```
context.restoreGState()
```

```
}
```

```
@objc open func drawCubicBezier(context: CGContext, dataSet: ILineChartDataSet)
```

```
{
```

```
guard let dataProvider = dataSet.dataProvider else { return }
```

```
let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)
```

```
let phaseY = animator.phaseY
```

```
_xBounds.set(chart: dataProvider, dataSet: dataSet, animator: animator)
```

```
// get the color that is specified for this position from the DataSet
```

```
let drawingColor = dataSet.colors.first!
```

```
let intensity = dataSet.cubicIntensity
```

```
// the path for the cubic-spline
```

```
let cubicPath = CGMutablePath()
```

```
let valueToPixelMatrix = trans.valueToPixelMatrix
```

```
if _xBounds.range >= 1
```

```
{
```

```
var prevDx: CGFloat = 0.0
```

```
var prevDy: CGFloat = 0.0
```

```
var curDx: CGFloat = 0.0
```

```
var curDy: CGFloat = 0.0
```

```
// Take an extra point from the left, and an extra from the right.
```

```
// That's because we need 4 points for a cubic bezier (cubic=4), otherwise we get  
lines moving and doing weird stuff on the edges of the chart.
```

```
// So in the starting `prev` and `cur`, go -2, -1
```

```
let firstIndex = _xBounds.min + 1
```

```
var prevPrev: ChartDataEntry! = nil
```

```
var prev: ChartDataEntry! = dataSet.entryForIndex(max(firstIndex - 2, 0))
```

```
var cur: ChartDataEntry! = dataSet.entryForIndex(max(firstIndex - 1, 0))
```

```
var next: ChartDataEntry! = cur
```

```
var nextIndex: Int = -1
```

```

if cur == nil { return }

// let the spline start

cubicPath.move(to: CGPoint(x: CGFloat(cur.x), y: CGFloat(cur.y * phaseY)),
transform: valueToPixelMatrix)

for j in _xBounds.dropFirst() // same as firstIndex

{

    prevPrev = prev

    prev = cur

    cur = nextIndex == j ? next : dataSet.entryForIndex(j)


    nextIndex = j + 1 < dataSet.entryCount ? j + 1 : j

    next = dataSet.entryForIndex(nextIndex)


    if next == nil { break }


    prevDx = CGFloat(cur.x - prevPrev.x) * intensity

```



```
prevDy = CGFloat(cur.y - prevPrev.y) * intensity
```

```
curDx = CGFloat(next.x - prev.x) * intensity
```

```
curDy = CGFloat(next.y - prev.y) * intensity
```

```
cubicPath.addCurve(
```

```
    to: CGPoint(
```

```
        x: CGFloat(cur.x),
```

```
        y: CGFloat(cur.y) * CGFloat(phaseY)),
```

```
    control1: CGPoint(
```

```
        x: CGFloat(prev.x) + prevDx,
```

```
        y: (CGFloat(prev.y) + prevDy) * CGFloat(phaseY)),
```

```
    control2: CGPoint(
```

```
        x: CGFloat(cur.x) - curDx,
```

```
        y: (CGFloat(cur.y) - curDy) * CGFloat(phaseY)),
```

```
    transform: valueToPixelMatrix)
```

```
    }
```

```
}
```

```

context.saveGState()

if dataSet.isDrawFilledEnabled
{

    // Copy this path because we make changes to it

    let fillPath = cubicPath.mutableCopy()

    drawCubicFill(context: context, dataSet: dataSet, spline: fillPath!, matrix:
valueToPixelMatrix, bounds: _xBounds)

}

context.beginPath()

context.addPath(cubicPath)

context.setStrokeColor(drawingColor.cgColor)

context.strokePath()

context.restoreGState()

```

```
}
```

```
@objc open func drawHorizontalBezier(context: CGContext, dataSet:  
ILineChartDataSet)
```

```
{
```

```
    guard let dataProvider = dataProvider else { return }
```

```
    let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)
```

```
    let phaseY = animator.phaseY
```

```
    _xBounds.set(chart: dataProvider, dataSet: dataSet, animator: animator)
```

```
    // get the color that is specified for this position from the DataSet
```

```
    let drawingColor = dataSet.colors.first!
```

```
    // the path for the cubic-spline
```

```
    let cubicPath = CGMutablePath()
```

```

let valueToPixelMatrix = trans.valueToPixelMatrix

if _xBounds.range >= 1

{

    var prev: ChartDataEntry! = dataSet.entryForIndex(_xBounds.min)

    var cur: ChartDataEntry! = prev

    if cur == nil { return }

    // let the spline start

    cubicPath.move(to: CGPoint(x: CGFloat(cur.x), y: CGFloat(cur.y * phaseY)),
transform: valueToPixelMatrix)

    for j in _xBounds.dropFirst()

    {

        prev = cur

        cur = dataSet.entryForIndex(j)

```

```
let cpx = CGFloat(prev.x + (cur.x - prev.x) / 2.0)
```

```
cubicPath.addCurve(
```

```
  to: CGPoint(
```

```
    x: CGFloat(cur.x),
```

```
    y: CGFloat(cur.y * phaseY)),
```

```
  control1: CGPoint(
```

```
    x: cpx,
```

```
    y: CGFloat(prev.y * phaseY)),
```

```
  control2: CGPoint(
```

```
    x: cpx,
```

```
    y: CGFloat(cur.y * phaseY)),
```

```
  transform: valueToPixelMatrix)
```

```
  }
```

```
}
```

```

context.saveGState()

if dataSet.isDrawFilledEnabled
{

    // Copy this path because we make changes to it

    let fillPath = cubicPath.mutableCopy()

    drawCubicFill(context: context, dataSet: dataSet, spline: fillPath!, matrix:
valueToPixelMatrix, bounds: _xBounds)

}

context.beginPath()

context.addPath(cubicPath)

context.setStrokeColor(drawingColor.cgColor)

context.strokePath()

context.restoreGState()

}

```

```

open func drawCubicFill(

    context: CGContext,

    dataSet: ILineChartDataSet,

    spline: CGMutablePath,

    matrix: CGAffineTransform,

    bounds: XBounds)

{

    guard

        let dataProvider = dataProvider

    else { return }

    if bounds.range <= 0

    {

        return

    }

```

```
let fillMin = dataSet.fillFormatter?.getFillLinePosition(dataSet: dataSet, dataProvider:
dataProvider) ?? 0.0
```

```
var pt1 = CGPoint(x: CGFloat(dataSet.entryForIndex(bounds.min +
bounds.range)?.x ?? 0.0), y: fillMin)
```

```
var pt2 = CGPoint(x: CGFloat(dataSet.entryForIndex(bounds.min)?.x ?? 0.0), y:
fillMin)
```

```
pt1 = pt1.applying(matrix)
```

```
pt2 = pt2.applying(matrix)
```

```
spline.addLine(to: pt1)
```

```
spline.addLine(to: pt2)
```

```
spline.closeSubpath()
```

```
if dataSet.fill != nil
```

```
{
```

```
drawFilledPath(context: context, path: spline, fill: dataSet.fill!, fillAlpha:
dataSet.fillAlpha)
```

```
}
```



```

else

{

    drawFilledPath(context: context, path: spline, fillColor: dataSet.fillColor, fillAlpha:
dataSet.fillAlpha)

}

}

```

```

private var _lineSegments = [CGPoint](repeating: CGPoint(), count: 2)

```

```

@objc open func drawLinear(context: CGContext, dataSet: ILineChartDataSet)

```

```

{

    guard let dataProvider = dataProvider else { return }

```

```

    let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)

```

```

    let valueToPixelMatrix = trans.valueToPixelMatrix

```

```

    let entryCount = dataSet.entryCount

```

```
let isDrawSteppedEnabled = dataSet.mode == .stepped
```

```
let pointsPerEntryPair = isDrawSteppedEnabled ? 4 : 2
```

```
let phaseY = animator.phaseY
```

```
_xBounds.set(chart: dataProvider, dataSet: dataSet, animator: animator)
```

```
// if drawing filled is enabled
```

```
if dataSet.isDrawFilledEnabled && entryCount > 0
```

```
{
```

```
    drawLinearFill(context: context, dataSet: dataSet, trans: trans, bounds: _xBounds)
```

```
}
```

```
context.saveGState()
```

```
if _lineSegments.count != pointsPerEntryPair
```

```
{
```

```

        // Allocate once in correct size

        _lineSegments = [CGPoint](repeating: CGPoint(), count: pointsPerEntryPair)

    }

    for j in _xBounds.dropLast()

    {

        var e: ChartDataEntry! = dataSet.entryForIndex(j)

        if e == nil { continue }

        _lineSegments[0].x = CGFloat(e.x)

        _lineSegments[0].y = CGFloat(e.y * phaseY)

        if j < _xBounds.max

        {

            // TODO: remove the check.

            // With the new XBounds iterator, j is always smaller than _xBounds.max

```

// Keeping this check for a while, if xBounds have no further breaking changes,
it should be safe to remove the check

```
e = dataSet.entryForIndex(j + 1)
```

```
if e == nil { break }
```

```
if isDrawSteppedEnabled
```

```
{
```

```
    _lineSegments[1] = CGPoint(x: CGFloat(e.x), y: _lineSegments[0].y)
```

```
    _lineSegments[2] = _lineSegments[1]
```

```
    _lineSegments[3] = CGPoint(x: CGFloat(e.x), y: CGFloat(e.y * phaseY))
```

```
}
```

```
else
```

```
{
```

```
    _lineSegments[1] = CGPoint(x: CGFloat(e.x), y: CGFloat(e.y * phaseY))
```

```
}
```

```
}
```

```
else
```

```

{

    _lineSegments[1] = _lineSegments[0]

}

for i in 0..<_lineSegments.count

{

    _lineSegments[i] = _lineSegments[i].applying(valueToPixelMatrix)

}

if !viewPortHandler.isInBoundsRight(_lineSegments[0].x)

{

    break

}

// Determine the start and end coordinates of the line, and make sure they differ.

guard

let firstCoordinate = _lineSegments.first,

```

```

        let lastCoordinate = _lineSegments.last,

        firstCoordinate != lastCoordinate else { continue }

// make sure the lines don't do shitty things outside bounds

if !viewPortHandler.isInBoundsLeft(lastCoordinate.x) ||

    !viewPortHandler.isInBoundsTop(max(firstCoordinate.y, lastCoordinate.y)) ||

    !viewPortHandler.isInBoundsBottom(min(firstCoordinate.y, lastCoordinate.y))

{

    continue

}

// get the color that is set for this line-segment

context.setStrokeColor(dataSet.color(atIndex: j).cgColor)

context.strokeLineSegments(between: _lineSegments)

}

context.restoreGState()

```

```

    }

    open func drawLinearFill(context: CGContext, dataSet: ILineChartDataSet, trans:
Transformer, bounds: XBounds)

    {

        guard let dataProvider = dataSet.dataProvider else { return }

        let filled = generateFilledPath(

            dataSet: dataSet,

            fillMin: dataSet.fillFormatter?.getFillLinePosition(dataSet: dataSet, dataProvider:
dataProvider) ?? 0.0,

            bounds: bounds,

            matrix: trans.valueToPixelMatrix)

        if dataSet.fill != nil

        {

            drawFilledPath(context: context, path: filled, fill: dataSet.fill!, fillAlpha:
dataSet.fillAlpha)

        }
    }

```

```

else

{

    drawFilledPath(context: context, path: filled, fillColor: dataSet.fillColor, fillAlpha:
dataSet.fillAlpha)

}

}

```

/// Generates the path that is used for filled drawing.

```

private func generateFilledPath(dataSet: ILineChartDataSet, fillMin: CGFloat, bounds:
XBounds, matrix: CGAffineTransform) -> CGPath

```

```

{

    let phaseY = animator.phaseY

    let isDrawSteppedEnabled = dataSet.mode == .stepped

    let matrix = matrix

    var e: ChartDataEntry!

    let filled = CGMutablePath()

```



```

e = dataSet.entryForIndex(bounds.min)

if e != nil

{

    filled.move(to: CGPoint(x: CGFloat(e.x), y: fillMin), transform: matrix)

    filled.addLine(to: CGPoint(x: CGFloat(e.x), y: CGFloat(e.y * phaseY)), transform:
matrix)

}

// create a new path

for x in stride(from: (bounds.min + 1), through: bounds.range + bounds.min, by: 1)

{

    guard let e = dataSet.entryForIndex(x) else { continue }

    if isDrawSteppedEnabled

    {

        guard let ePrev = dataSet.entryForIndex(x-1) else { continue }

```

```
        filled.addLine(to: CGPoint(x: CGFloat(e.x), y: CGFloat(ePrev.y * phaseY)),
transform: matrix)
```

```
    }
```

```
        filled.addLine(to: CGPoint(x: CGFloat(e.x), y: CGFloat(e.y * phaseY)), transform:
matrix)
```

```
    }
```

```
// close up
```

```
e = dataSet.entryForIndex(bounds.range + bounds.min)
```

```
if e != nil
```

```
{
```

```
    filled.addLine(to: CGPoint(x: CGFloat(e.x), y: fillMin), transform: matrix)
```

```
}
```

```
filled.closeSubpath()
```

```
return filled
```

```
}
```

```

open override func drawValues(context: CGContext)

{

    guard

        let dataProvider = dataProvider,

        let lineData = dataProvider.lineData

        else { return }

    if isDrawingValuesAllowed(dataProvider: dataProvider)

    {

        let dataSets = lineData.dataSets

        let phaseY = animator.phaseY

        var pt = CGPoint()

        for i in 0 ..< dataSets.count

```

```

{

    guard let

        dataSet = dataSets[i] as? ILineChartDataSet,

        shouldDrawValues(forDataSet: dataSet)

        else { continue }

    let valueFont = dataSet.valueFont

    guard let formatter = dataSet.valueFormatter else { continue }

    let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)

    let valueToPixelMatrix = trans.valueToPixelMatrix

    let iconsOffset = dataSet.iconsOffset

    // make sure the values do not interfere with the circles

    var valOffset = Int(dataSet.circleRadius * 1.75)

```

```

if !dataSet.isDrawCirclesEnabled

{

    valOffset = valOffset / 2

}

_xBounds.set(chart: dataProvider, dataSet: dataSet, animator: animator)

for j in _xBounds

{

    guard let e = dataSet.entryForIndex(j) else { break }

    pt.x = CGFloat(e.x)

    pt.y = CGFloat(e.y * phaseY)

    pt = pt.applying(valueToPixelMatrix)

    if (!viewPortHandler.isInBoundsRight(pt.x))

```

```

        {

            break

        }

        if (!viewPortHandler.isInBoundsLeft(pt.x)
|| !viewPortHandler.isInBoundsY(pt.y))

        {

            continue

        }

        if dataSet.isDrawValuesEnabled {

            ChartUtils.drawText(

                context: context,

                text: formatter.stringForValue(

                    e.y,

                    entry: e,

                    dataSetIndex: i,

                    viewPortHandler: viewPortHandler),

```

```

        point: CGPoint(

            x: pt.x,

            y: pt.y - CGFloat(valOffset) - valueFont.lineHeight),

        align: .center,

        attributes: [NSAttributedString.Key.font: valueFont,
NSAttributedString.Key.foregroundColor: dataSet.valueTextColorAt(j)]

    }

    if let icon = e.icon, dataSet.isDrawIconsEnabled

    {

        ChartUtils.drawImage(context: context,

            image: icon,

            x: pt.x + iconsOffset.x,

            y: pt.y + iconsOffset.y,

            size: icon.size)

    }

}

}

```

```
}
```

```
}
```

```
open override fun drawExtras(context: CGContext)
```

```
{
```

```
    drawCircles(context: context)
```

```
}
```

```
private fun drawCircles(context: CGContext)
```

```
{
```

```
    guard
```

```
        let dataProvider = dataProvider,
```

```
        let lineData = dataProvider.lineData
```

```
    else { return }
```

```
    let phaseY = animator.phaseY
```



```
let dataSets = lineData.dataSets
```

```
var pt = CGPoint()
```

```
var rect = CGRect()
```

```
// If we redraw the data, remove and repopulate accessible elements to update label  
values and frames
```

```
accessibleChartElements.removeAll()
```

```
accessibilityOrderedElements = accessibilityCreateEmptyOrderedElements()
```

```
// Make the chart header the first element in the accessible elements array
```

```
if let chart = dataProvider as? LineChartView {
```

```
    let element = createAccessibleHeader(usingChart: chart,
```

```
        andData: lineData,
```

```
        withDefaultDescription: "Line Chart")
```

```
    accessibleChartElements.append(element)
```

```
}
```

```
context.saveGState()
```

```
for i in 0 ..< dataSets.count
```

```
{
```

```
    guard let dataSet = lineData.getDataSetByIndex(i) as? ILineChartDataSet else  
    { continue }
```

```
    if !dataSet.isVisible || dataSet.entryCount == 0
```

```
    {
```

```
        continue
```

```
    }
```

```
let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)
```

```
let valueToPixelMatrix = trans.valueToPixelMatrix
```

```
_xBounds.set(chart: dataProvider, dataSet: dataSet, animator: animator)
```

```
let circleRadius = dataSet.circleRadius
```

```

let circleDiameter = circleRadius * 2.0

let circleHoleRadius = dataSet.circleHoleRadius

let circleHoleDiameter = circleHoleRadius * 2.0


let drawCircleHole = dataSet.isDrawCircleHoleEnabled &&

    circleHoleRadius < circleRadius &&

    circleHoleRadius > 0.0

let drawTransparentCircleHole = drawCircleHole &&

    (dataSet.circleHoleColor == nil ||

        dataSet.circleHoleColor == NSUIColor.clear)


for j in _xBounds

{

    guard let e = dataSet.entryForIndex(j) else { break }


    pt.x = CGFloat(e.x)

    pt.y = CGFloat(e.y * phaseY)

```

```

    pt = pt.applying(valueToPixelMatrix)

    if (!viewPortHandler.isInBoundsRight(pt.x))

    {

        break

    }

    // make sure the circles don't do shitty things outside bounds

    if                                     (!viewPortHandler.isInBoundsLeft(pt.x)
|| !viewPortHandler.isInBoundsY(pt.y))

    {

        continue

    }

    // Skip Circles and Accessibility if not enabled,

    // reduces CPU significantly if not needed

    if !dataSet.isDrawCirclesEnabled

```

```

    {

        continue

    }

    // Accessibility element geometry

    let scaleFactor: CGFloat = 3

    let accessibilityRect = CGRect(x: pt.x - (scaleFactor * circleRadius),

                                    y: pt.y - (scaleFactor * circleRadius),

                                    width: scaleFactor * circleDiameter,

                                    height: scaleFactor * circleDiameter)

    // Create and append the corresponding accessibility element to
    accessibilityOrderedElements

    if let chart = dataProvider as? LineChartView

    {

        let element = createAccessibleElement(withIndex: j,

                                              container: chart,

                                              dataSet: dataSet,

                                              dataSetIndex: i)

```

```

    { (element) in

        element.accessibilityFrame = accessibilityRect

    }

    accessibilityOrderedElements[i].append(element)

}

context.setFillColor(dataSet.getCircleColor(atIndex: j)!.cgColor)

rect.origin.x = pt.x - circleRadius

rect.origin.y = pt.y - circleRadius

rect.size.width = circleDiameter

rect.size.height = circleDiameter

if drawTransparentCircleHole

{

    // Begin path for circle with hole

```

```

context.beginPath()

context.addEllipse(in: rect)


// Cut hole in path

rect.origin.x = pt.x - circleHoleRadius

rect.origin.y = pt.y - circleHoleRadius

rect.size.width = circleHoleDiameter

rect.size.height = circleHoleDiameter

context.addEllipse(in: rect)


// Fill in-between

context.fillPath(using: .evenOdd)

}

else

{

context.fillEllipse(in: rect)

```

```

        if drawCircleHole

        {

            context.setFill_color(datasets.circleHoleColor!.cgColor)


            // The hole rect

            rect.origin.x = pt.x - circleHoleRadius

            rect.origin.y = pt.y - circleHoleRadius

            rect.size.width = circleHoleDiameter

            rect.size.height = circleHoleDiameter


            context.fillEllipse(in: rect)

        }

    }

}

context.restoreGState()

```



```

// Merge nested ordered arrays into the single accessibleChartElements.

accessibleChartElements.append(contentsOf: accessibilityOrderedElements.flatMap
{ $0 } )

accessibilityPostLayoutChangedNotification()

}

open override func drawHighlighted(context: CGContext, indices: [Highlight])

{

    guard

        let dataProvider = dataProvider,

        let lineData = dataProvider.lineData

        else { return }

    let chartXMax = dataProvider.chartXMax

    context.saveGState()

```

```

for high in indices

{

    guard let set = lineData.getDataSetByIndex(high.dataSetIndex) as?
    ILineChartDataSet

        , set.isHighlightEnabled

        else { continue }

    guard let e = set.entryForXValue(high.x, closestToY: high.y) else { continue }

    if !isInBoundsX(entry: e, dataSet: set)

    {

        continue

    }

    context.setStrokeColor(set.highlightColor.cgColor)

    context.setLineWidth(set.highlightLineWidth)

    if set.highlightLineDashLengths != nil

    {

```

```
        context.setLineDash(phase: set.highlightLineDashPhase, lengths:
set.highlightLineDashLengths!)
```

```
    }
```

```
    else
```

```
    {
```

```
        context.setLineDash(phase: 0.0, lengths: [])
```

```
    }
```

```
    let x = e.x // get the x-position
```

```
    let y = e.y * Double(animator.phaseY)
```

```
    if x > chartXMax * animator.phaseX
```

```
    {
```

```
        continue
```

```
    }
```

```
    let trans = dataProvider.getTransformer(forAxis: set.axisDependency)
```

```
let pt = trans.pixelForValues(x: x, y: y)
```

```
high.setDraw(pt: pt)
```

```
// draw the lines
```

```
drawHighlightLines(context: context, point: pt, set: set)
```

```
}
```

```
context.restoreGState()
```

```
}
```

```
/// Creates a nested array of empty subarrays each of which will be populated with  
NSUIAccessibilityElements.
```

```
/// This is marked internal to support HorizontalBarChartRenderer as well.
```

```
private      func      accessibilityCreateEmptyOrderedElements()      ->  
[[NSUIAccessibilityElement]]
```

```
{
```

```
guard let chart = dataProvider as? LineChartView else { return [] }
```

```

let dataSetCount = chart.lineData?.dataSetCount ?? 0

return Array(repeating: [NSUIAccessibilityElement](),
                    count: dataSetCount)

}

/// Creates an NSUIAccessibleElement representing the smallest meaningful bar of the
chart

/// i.e. in case of a stacked chart, this returns each stack, not the combined bar.

/// Note that it is marked internal to support subclass modification in the
HorizontalBarChart.

private func createAccessibleElement(withIndex idx: Int,
                                    container: LineChartView,
                                    dataSet: ILineChartDataSet,
                                    dataSetIndex: Int,
                                    modifier: (NSUIAccessibilityElement) -> () ->
NSUIAccessibilityElement
                                {

```

```
let element = NSUIAccessibilityElement(accessibilityContainer: container)
```

```
let xAxis = container.xAxis
```

```
guard let e = dataSet.entryForIndex(idx) else { return element }
```

```
guard let dataProvider = dataProvider else { return element }
```

```
// NOTE: The formatter can cause issues when the x-axis labels are consecutive ints.
```

```
// i.e. due to the Double conversion, if there are more than one data set that are grouped,
```

```
// there is the possibility of some labels being rounded up. A floor() might fix this, but  
seems to be a brute force solution.
```

```
let label = xAxis.valueFormatter?.stringForValue(e.x, axis: xAxis) ?? "\(e.x)"
```

```
let elementValueText = dataSet.valueFormatter?.stringForValue(e.y,
```

```
    entry: e,
```

```
    dataSetIndex: dataSetIndex,
```

```
    viewportHandler:    viewportHandler)    ??
```

```
"\(e.y)"
```

```

        let dataSetCount = dataProvider.lineData?.dataSetCount ?? -1

        let doesContainMultipleDataSets = dataSetCount > 1

        element.accessibilityLabel = "\(\(doesContainMultipleDataSets ? (dataSet.label ?? "")
+ ", " : "") \((label): \((elementValueText))"

        modifier(element)

        return element
    }

}

//

// LineChartView.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

```

```

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

/// Chart that draws lines, surfaces, circles, ...

open class LineChartView: BarLineChartViewBase, LineChartDataProvider

{

    internal override func initialize()

    {

        super.initialize()

        renderer = LineChartRenderer(dataProvider: self, animator: _animator,
viewPortHandler: _viewPortHandler)

    }

```



```
// MARK: - LineChartDataProvider

open var lineData: LineChartData? { return _data as? LineChartData }

}

//

// LineRadarChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics
```

```

open class LineRadarChartDataSet: LineScatterCandleRadarChartDataSet,
ILineRadarChartDataSet

{

    // MARK: - Data functions and accessors


    // MARK: - Styling functions and accessors


    /// The color that is used for filling the line surface area.

    private var _fillColor = NSUIColor(red: 140.0/255.0, green: 234.0/255.0, blue:
255.0/255.0, alpha: 1.0)


    /// The color that is used for filling the line surface area.

    open var fillColor: NSUIColor

    {

        get { return _fillColor }

        set

```

```

    {

        _fillColor = newValue

        fill = nil

    }

}

```

/// The object that is used for filling the area below the line.

/// **default**: nil

open var fill: Fill?

/// The alpha value that is used for filling the line surface,

/// **default**: 0.33

open var fillAlpha = CGFloat(0.33)

private var _lineWidth = CGFloat(1.0)

/// line width of the chart (min = 0.0, max = 10)

```
///
```

```
/// **default**: 1
```

```
open var lineWidth: CGFloat
```

```
{
```

```
    get
```

```
    {
```

```
        return _lineWidth
```

```
    }
```

```
    set
```

```
    {
```

```
        _lineWidth = newValue.clamped(to: 0...10)
```

```
    }
```

```
}
```

```
/// Set to `true` if the DataSet should be drawn filled (surface), and not just as a line.
```

```
/// Disabling this will give great performance boost.
```

```
/// Please note that this method uses the path clipping for drawing the filled area (with  
images, gradients and layers).
```

```
open var drawFilledEnabled = false
```

```
/// `true` if filled drawing is enabled, `false` if not
```

```
open var isDrawFilledEnabled: Bool
```

```
{  
  
    return drawFilledEnabled  
  
}
```

```
// MARK: NSCopying
```

```
open override func copy(with zone: NSZone? = nil) -> Any
```

```
{  
  
    let copy = super.copy(with: zone) as! LineRadarChartDataSet  
  
    copy.fill = fill  
  
    copy.fillAlpha = fillAlpha  
  
    copy._fillColor = _fillColor  
  
    copy._lineWidth = _lineWidth
```

```
        copy.drawFilledEnabled = drawFilledEnabled

        return copy

    }

}

//

// LineRadarRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation
```

```
import CoreGraphics
```

```
@objc(LineRadarChartRenderer)
```

```
open class LineRadarRenderer: LineScatterCandleRadarRenderer
```

```
{
```

```
    public override init(animator: Animator, viewPortHandler: ViewPortHandler)
```

```
    {
```

```
        super.init(animator: animator, viewPortHandler: viewPortHandler)
```

```
    }
```

```
    /// Draws the provided path in filled mode with the provided drawable.
```

```
    @objc open func drawFilledPath(context: CGContext, path: CGPath, fill: Fill, fillAlpha: CGFloat)
```

```
    {
```

```
        context.saveGState()
```

```
        context.beginPath()
```

```
        context.addPath(path)
```

```

// filled is usually drawn with less alpha

context.setAlpha(fillAlpha)

fill.fillPath(context: context, rect: viewPortHandler.contentRect)

context.restoreGState()

}

/// Draws the provided path in filled mode with the provided color and alpha.

@objc open func drawFilledPath(context: CGContext, path: CGPath, fillColor:
NSUIColor, fillAlpha: CGFloat)

{

    context.saveGState()

    context.beginPath()

    context.addPath(path)

    // filled is usually drawn with less alpha

```



```
context.setAlpha(fillAlpha)

context.setFillColor(fillColor.cgColor)

context.fillPath()

context.restoreGState()

}

}

//

// LineScatterCandleRadarChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
open class LineScatterCandleRadarChartDataSet:  
BarLineScatterCandleBubbleChartDataSet, ILineScatterCandleRadarChartDataSet
```

```
{
```

```
    // MARK: - Data functions and accessors
```

```
    // MARK: - Styling functions and accessors
```

```
    /// Enables / disables the horizontal highlight-indicator. If disabled, the indicator is not  
drawn.
```

```
    open var drawHorizontalHighlightIndicatorEnabled = true
```

```
    /// Enables / disables the vertical highlight-indicator. If disabled, the indicator is not  
drawn.
```

```
    open var drawVerticalHighlightIndicatorEnabled = true
```

```

    /// `true` if horizontal highlight indicator lines are enabled (drawn)

    open    var    isHorizontalHighlightIndicatorEnabled:    Bool    {    return
drawHorizontalHighlightIndicatorEnabled }

    /// `true` if vertical highlight indicator lines are enabled (drawn)

    open    var    isVerticalHighlightIndicatorEnabled:    Bool    {    return
drawVerticalHighlightIndicatorEnabled }

    /// Enables / disables both vertical and horizontal highlight-indicators.

    /// :param: enabled

    open func setDrawHighlightIndicators(_ enabled: Bool)

    {

        drawHorizontalHighlightIndicatorEnabled = enabled

        drawVerticalHighlightIndicatorEnabled = enabled

    }

    // MARK: NSCopying

```

```

open override func copy(with zone: NSZone? = nil) -> Any

{

    let copy = super.copy(with: zone) as! LineScatterCandleRadarChartDataSet

    copy.drawHorizontalHighlightIndicatorEnabled =
drawHorizontalHighlightIndicatorEnabled

    copy.drawVerticalHighlightIndicatorEnabled =
drawVerticalHighlightIndicatorEnabled

    return copy

}

}

//

// LineScatterCandleRadarRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

```

```

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc(LineScatterCandleRadarChartRenderer)

open class LineScatterCandleRadarRenderer: BarLineScatterCandleBubbleRenderer

{

    public override init(animator: Animator, viewPortHandler: ViewPortHandler)

    {

        super.init(animator: animator, viewPortHandler: viewPortHandler)

    }

}

/// Draws vertical & horizontal highlight-lines if enabled.

```

```

/// :param: context

/// :param: points

/// :param: horizontal

/// :param: vertical

@objc open func drawHighlightLines(context: CGContext, point: CGPoint, set:
ILineScatterCandleRadarChartDataSet)

{

    // draw vertical highlight lines

    if set.isVerticalHighlightIndicatorEnabled

    {

        context.beginPath()

        context.move(to: CGPoint(x: point.x, y: viewPortHandler.contentTop))

        context.addLine(to: CGPoint(x: point.x, y: viewPortHandler.contentBottom))

        context.strokePath()

    }

    // draw horizontal highlight lines

```

```

    if set.isHorizontalHighlightIndicatorEnabled

    {

        context.beginPath()

        context.move(to: CGPoint(x: viewPortHandler.contentLeft, y: point.y))

        context.addLine(to: CGPoint(x: viewPortHandler.contentRight, y: point.y))

        context.strokePath()

    }

}

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The 'LocationTriggerCreator' creates Location triggers.

```

*/

```

```
import HomeKit
```

```
import MapKit
```

```
/**
```

```
    An `EventTriggerCreator` subclass which allows for the creation  
    of location triggers.
```

```
*/
```

```
class LocationTriggerCreator: EventTriggerCreator, MapViewControllerDelegate {
```

```
    // MARK: Properties
```

```
    var eventTrigger: HMEventTrigger? {  
        return trigger as? HMEventTrigger  
    }  
}
```

```
    var locationEvent: HMLocationEvent?
```

```
    var targetRegion: CLCircularRegion?
```

```
    var targetRegionStateIndex = 0
```



```
// MARK: Trigger Creator Methods
```

```
/// Initializes location event, target region, and region state.
```

```
required init(trigger: HMTrigger?, home: HMHome) {
```

```
    super.init(trigger: trigger, home: home)
```

```
    if let eventTrigger = eventTrigger {
```

```
        self.locationEvent = eventTrigger.locationEvent
```

```
        if let region = locationEvent?.region as? CLCircularRegion {
```

```
            self.targetRegion = region
```

```
        }
```

```
        self.targetRegionStateIndex = (self.targetRegion?.notifyOnEntry ?? true) ? 0 : 1
```

```
    }
```

```
}
```

```
/// Generates a new region and updates the location event.
```

```
override func updateTrigger() {
```

```

if let region = targetRegion {

    prepareRegion()

    if let locationEvent = locationEvent {

        saveTriggerGroup.enter()

        locationEvent.updateRegion(region) { error in

            if let error = error {

                self.errors.append(error)

            }

            self.saveTriggerGroup.leave()

        }

    }

}

self.savePredicate()

}

/**

```

- returns: A new `HMEventTrigger` with a new generated

location event and predicate.

*/

```
override fun newTrigger() -> HMTrigger? {
```

```
    var events = [HMLocationEvent]()
```

```
    if let region = targetRegion {
```

```
        prepareRegion()
```

```
        events.append(HMLocationEvent(region: region))
```

```
    }
```

```
    return HMEventTrigger(name: name, events: events, predicate: newPredicate())
```

```
}
```

```
// MARK: Helper Methods
```

```
/**
```

Sets the `notifyOnEntry` and `notifyOnExit` region

properties based on the selected state.

```
*/
```

```
private func prepareRegion() {  
  
    if let region = targetRegion {  
  
        region.notifyOnEntry = (targetRegionStateIndex == 0)  
  
        region.notifyOnExit = !region.notifyOnEntry  
  
    }  
  
}
```

```
/**
```

Updates the target region from the one provided

by the delegate.

- parameter region: A new `CLCircularRegion`, provided by the delegate.

```
*/
```

```
func mapViewDidUpdateRegion(_ region: CLCircularRegion) {  
  
    targetRegion = region  
  
}
```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `LocationTriggerViewController` allows the user to modify and create Location triggers.

```
*/
```

```
import UIKit
```

```
import MapKit
```

```
import HomeKit
```

```
import AddressBookUI
```

```
import Contacts
```

```
/// A view controller which facilitates the creation of a location trigger.
```

```
class LocationTriggerViewController: EventTriggerViewController {
```

```

struct Identifiers {

    static let locationCell = "LocationCell"

    static let regionStatusCell = "RegionStatusCell"

    static let selectLocationSegue = "Select Location"

}

static let geocoder = CLGeocoder()

static let regionStatusTitles = [

    NSLocalizedString("When I Enter The Area", comment: "When I Enter The Area"),

    NSLocalizedString("When I Leave The Area", comment: "When I Leave The Area")

]

var locationTriggerCreator: LocationTriggerCreator {

    return triggerCreator as! LocationTriggerCreator

}

```

```
var localizedAddress: String?
```

```
var viewIsDisplayed = false
```

```
// MARK: View Methods
```

```
/// Initializes a trigger creator and registers for table view cells.
```

```
override func viewDidLoad() {
```

```
    super.viewDidLoad()
```

```
    triggerCreator = LocationTriggerCreator(trigger: trigger, home: home)
```

```
    tableView.register(UITableViewCell.self,                       forCellReuseIdentifier:
Identifiers.locationCell)
```

```
    tableView.register(UITableViewCell.self,                       forCellReuseIdentifier:
Identifiers.regionStatusCell)
```

```
}
```

```
/**
```

Generates an address string for the current region location and

reloads the table view.

*/

```
override func viewDidLoad(_ animated: Bool) {
```

```
    super.viewDidLoad(animated)
```

```
    viewIsDisplayed = true
```

```
    if let region = locationTriggerCreator.targetRegion {
```

```
        let centerLocation = CLLocation(latitude: region.center.latitude, longitude:
region.center.longitude)
```

```
LocationTriggerViewController.geocoder.reverseGeocodeLocation(centerLocation)
{ placemarks, error in
```

```
    if !self.viewIsDisplayed {
```

```
        // The geocoder took too long, we're not on this view any more.
```

```
        return
```

```
    }
```

```
    if let error = error {
```

```
        self.displayError(error)
```

```
        return
```



```

    }

    if let mostLikelyPlacemark = placemarks?.first {

        let address = CNMutablePostalAddress(placemark: mostLikelyPlacemark)

        let addressFormatter = CNPostalAddressFormatter()

        let addressString = addressFormatter.string(from: address)

        self.localizedAddress = addressString.replacingOccurrences(of: "\n", with: ",
    ")

        let section = IndexSet(integer: 2)

        self.tableView.reloadSections(section, with: .automatic)

    }

}

}

tableView.reloadData()

}

```

/// Passes the trigger creator and region into the `MapViewController`.

```

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

    super.prepare(for: segue, sender: sender)

```

```

    if segue.identifier == Identifiers.selectLocationSegue {

        guard let destinationVC = segue.intendedDestinationViewController as?
MapViewController else { return }

        // Give the map the previous target region (if exists).

        destinationVC.targetRegion = locationTriggerCreator.targetRegion

        destinationVC.delegate = locationTriggerCreator

    }

}

```

```

override func viewWillDisappear(_ animated: Bool) {

    super.viewWillDisappear(animated)

    viewIsDisplayed = false

}

```

// MARK: Table View Methods

/**

- returns: The number of rows in the Region section;

defaults to the super implementation for other sections.

```
*/
```

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
```

```
    switch sectionForIndex(section) {
```

```
        case .region?:
```

```
            return 2
```

```
        case nil:
```

```
            fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
        default:
```

```
            return super.tableView(tableView, numberOfRowsInSection: section)
```

```
    }
```

```
}
```

```
/**
```

Generates a cell based on the section.

Handles Region and Location sections, defaults to

super implementations for other sections.

*/

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
```

```
    switch sectionForIndex(indexPath.section) {
```

```
        case .region?:
```

```
            return self.tableView(tableView, regionStatusCellForRowAtIndexPath:
indexPath)
```

```
        case .location?:
```

```
            return self.tableView(tableView, locationCellForRowAtIndexPath: indexPath)
```

```
        case nil:
```

```
            fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
        default:
```

```
            return super.tableView(tableView, cellForRowAt: indexPath)
```

```

    }

}

/// Generates the single location cell.

private func tableView(_ tableView: UITableView, locationCellForRowAtIndexPath
indexPath: IndexPath) -> UITableViewCell {

    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.locationCell, for:
indexPath)

    cell.accessoryType = .disclosureIndicator

    if locationTriggerCreator.targetRegion != nil {

        cell.textLabel?.text = localizedAddress ?? NSLocalizedString("Update Location",
comment: "Update Location")

    }

    else {

        cell.textLabel?.text = NSLocalizedString("Set Location", comment: "Set Location")

    }

    return cell

}

```

/// Generates the cell which allow the user to select either 'on enter' or 'on exit'.

```
private func tableView(_ tableView: UITableView,
regionStatusCellForRowAtIndexPath indexPath: IndexPath) -> UITableViewCell {

    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.regionStatusCell,
for: indexPath)

    cell.textLabel?.text =
LocationTriggerViewController.regionStatusTitles[indexPath.row]

    cell.accessoryType = (locationTriggerCreator.targetRegionStateIndex ==
indexPath.row) ? .checkmark : .none

    return cell

}
```

/**

Allows the user to select a location or change the region status.

Defaults to the super implmentation for other sections.

*/

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
```

```
    switch sectionForIndex(indexPath.section) {
```

```
case .location?:
```

```
    performSegue(withIdentifier: Identifiers.selectLocationSegue, sender: self)
```

```
case .region?:
```

```
    locationTriggerCreator.targetRegionStateIndex = indexPath.row
```

```
    let reloadIndexSet = IndexSet(integer: indexPath.section)
```

```
    tableView.reloadSections(reloadIndexSet, with: .automatic)
```

```
case nil:
```

```
    fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
default:
```

```
    super.tableView(tableView, didSelectRowAt: indexPath)
```

```
    }
```

```
}
```

```
/**
```

- returns: A localized title for the Location and Region sections.

Defaults to the super implementation for other sections.

*/

```
override func tableView(_ tableView: UITableView, titleForHeaderInSection section:
Int) -> String? {
```

```
    switch sectionForIndex(section) {
```

```
        case .location?:
```

```
            return NSLocalizedString("Location", comment: "Location")
```

```
        case .region?:
```

```
            return NSLocalizedString("Region Status", comment: "Region Status")
```

```
        case nil:
```

```
            fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
        default:
```

```
            return super.tableView(tableView, titleForHeaderInSection: section)
```

```
    }
```



```
}
```

```
/**
```

```
- returns: A localized description of the region status.
```

```
Defaults to the super implementation for other sections.
```

```
*/
```

```
override func tableView(_ tableView: UITableView, titleForFooterInSection section:  
Int) -> String? {
```

```
    switch sectionForIndex(section) {
```

```
        case .region?:
```

```
            return NSLocalizedString("This trigger can activate when you enter or leave a  
region. For example, when you arrive at home or when you leave work.", comment:  
"Location Region Description")
```

```
        case nil:
```

```
            fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
        default:
```

```
            return super.tableView(tableView, titleForFooterInSection: section)
```

```
}
```

```
}
```

```
// MARK: Trigger Controller Methods
```

```
/**
```

```
- parameter index: The section index.
```

```
- returns: The `TriggerTableViewSection` for the given index.
```

```
*/
```

```
override func sectionForIndex(_ index: Int) -> TriggerTableViewSection? {
```

```
    switch index {
```

```
        case 0:
```

```
            return .name
```

```
        case 1:
```

```
            return .enabled
```

case 2:

return .location

case 3:

return .region

case 4:

return .conditions

case 5:

return .actionSets

default:

return nil

}

}

}

// Copyright (c) 2016-present, Facebook, Inc. All rights reserved.

//

// You are hereby granted a non-exclusive, worldwide, royalty-free license to use,

// copy, modify, and distribute this software in source code or binary form for use

// in connection with the web services and APIs provided by Facebook.

//

// As with any software that integrates with the Facebook platform, your use of

// this software is subject to the Facebook Developer Principles and Policies

// [<http://developers.facebook.com/policy/>]. This copyright notice shall be

// included in all copies or substantial portions of the software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS

// FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR

```
// COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER  
LIABILITY, WHETHER
```

```
// IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT  
OF OR IN
```

```
// CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN  
THE SOFTWARE.
```

```
import FacebookCore
```

```
import Foundation
```

```
import UIKit
```

```
/// Login Result Block
```

```
public typealias LoginResultBlock = (LoginResult) -> Void
```

```
/**
```

```
Describes the result of a login attempt.
```

```
*/
```

```
public enum LoginResult {
```

```
/// User succesfully logged in. Contains granted, declined permissions and access token.
```

```
case success(granted: Set<Permission>, declined: Set<Permission>, token: AccessToken)
```

```
/// Login attempt was cancelled by the user.
```

```
case cancelled
```

```
/// Login attempt failed.
```

```
case failed(Error)
```

```
internal init(result: LoginManagerLoginResult?, error: Error?) {
```

```
    guard let result = result, error == nil else {
```

```
        self = .failed(error ?? LoginError(.unknown))
```

```
        return
```

```
    }
```

```
    guard !result.isCancelled, let token = result.token else {
```

```
        self = .cancelled
```

```
        return
```

```
    }
```

```

        let granted: Set<Permission> = Set(result.grantedPermissions.map
{ Permission(stringLiteral: $0) })

        let declined: Set<Permission> = Set(result.declinedPermissions.map
{ Permission(stringLiteral: $0) })

        self = .success(granted: granted, declined: declined, token: token)

    }

}

/**

```

This class provides methods for logging the user in and out.

It works directly with ``AccessToken.current`` and

sets the "current" token upon successful authorizations (or sets ``nil`` in case of ``logout``).

You should check ``AccessToken.current`` before calling ``login`` to see if there is

a cached token available (typically in your ``viewDidLoad``).

If you are managing your own token instances outside of ``AccessToken.current``, you will need to set

``current`` before calling ``login`` to authorize further permissions on your tokens.

```
*/
```

```
public extension LoginManager {
```

```
/**
```

```
Initialize an instance of `LoginManager`.
```

```
- parameter loginBehavior: Optional login behavior to use. Default: `.Native`.
```

```
- parameter defaultAudience: Optional default audience to use. Default: `.Friends`.
```

```
*/
```

```
convenience init(loginBehavior: LoginBehavior = .browser, defaultAudience:  
DefaultAudience = .friends) {
```

```
    self.init()
```

```
    self.loginBehavior = loginBehavior
```

```
    self.defaultAudience = defaultAudience
```

```
}
```

```
/**
```

```
Logs the user in or authorizes additional permissions.
```


Use this method when asking for read permissions. You should only ask for permissions when they

are needed and explain the value to the user. You can inspect the `'declinedPermissions'` in the result to also

provide more information to the user if they decline permissions.

This method will present UI to the user. You typically should check if `'AccessToken.current'` already

contains the permissions you need before asking to reduce unnecessary app switching.

- parameter `permissions`: Array of read permissions. Default: `'[.PublicProfile]'`

- parameter `viewController`: Optional view controller to present from. Default: topmost view controller.

- parameter `completion`: Optional callback.

`*/`

```
func login(permissions: [Permission] = [.publicProfile],
```

```
    viewController: UIViewController? = nil,
```

```
    completion: LoginResultBlock? = nil) {
```

```
    self.login(permissions: permissions.map { $0.name }, from: viewController, handler:
sdkCompletion(completion))
```

```
}
```

```
private func sdkCompletion(_ completion: LoginResultBlock?) ->
LoginManagerLoginResultBlock? {

    guard let completion = completion else {

        return nil

    }

    return { (result: LoginManagerLoginResult?, error: Error?) in

        let result = LoginResult(result: result, error: error)

        completion(result)

    }

}

/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `MapViewOverlayView` draws the circle over the `MapViewController`.

```
*/
```

```
import MapKit
```

```
/**
```

A simple `UIView` subclass to draw a selection circle over

a `MKMapView` of the same size.

```
*/
```

```
class MapOverlayView: UIView {
```

```
/**
```

Draws a dashed circle in the center of the `rect` with

a radius 1/4th of the `rect`'s smallest side.

```
*/
```

```
override func draw(_ rect: CGRect) {
```

```
    super.draw(rect)
```

```

let context = UIGraphicsGetCurrentContext()

let strokeColor = UIColor.blue

let circleDiameter: CGFloat = min(rect.width, rect.height) / 2.0

let circleRadius = circleDiameter / 2.0

let circleRect = CGRect(x: rect.midX - circleRadius, y: rect.midY - circleRadius,
width: circleDiameter, height: circleDiameter)

let circlePath = UIBezierPath(ovalIn: circleRect)

strokeColor.setStroke()

circlePath.lineWidth = 3

context!.saveGState()

context!.setLineDash(phase: 0, lengths: [6, 6])

circlePath.stroke()

context!.restoreGState()

}

```

```
/// - returns: `false` to accept no touches.
```

```
override func point(inside point: CGPoint, with event: UIEvent?) -> Bool {
```

```
    return false
```

```
}
```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `MapViewController` allow the user to select a location using the map.

This location will be passed back to the sender when the user saves the view.

```
*/
```

```
import UIKit
```

```
import MapKit
```

```
/**
```

Allows the sender to get notified when there
have been changes to the region.

```
*/
```

```
protocol MapViewControllerDelegate {
```

```
/**
```

Notifies the delegate that the `MapViewController`'s
region has been updated.

```
*/
```

```
func mapViewDidUpdateRegion(_ region: CLCircularRegion)
```

```
}
```

```
/**
```

A view controller which allows the selection of a
circular region on a map.

```
*/
```

```
class MapViewController: UIViewController, UISearchBarDelegate,  
CLLocationManagerDelegate, MKMapViewDelegate {
```

```
// MARK: Types
```

```
struct Identifiers {
```

```
    static let circularRegion = "MapViewController.Region"
```

```
}
```

```
/// When the view loads, we'll zoom to this longitude/latitude span delta.
```

```
static let InitialZoomDelta: Double = 0.0015
```

```
/// When the view loads, we'll zoom into this span.
```

```
static let InitialZoomSpan = MKCoordinateSpan(latitudeDelta:
MapViewController.InitialZoomDelta, longitudeDelta:
MapViewController.InitialZoomDelta)
```

```
// The inverse of the percentage of the map view that should be captured in the region.
```

```
static let MapRegionFraction: Double = 4.0
```

```
// The size of the query region with respect to the map's zoom.
```

```
static let RegionQueryDegreeMultiplier: Double = 5.0
```

```
// MARK: Properties
```

```
@IBOutlet weak var overlayView: MapOverlayView!
```

```
@IBOutlet weak var searchBar: UISearchBar!
```

```
@IBOutlet weak var mapView: MKMapView!
```

```
var delegate: MapViewControllerDelegate?
```

```
var targetRegion: CLCircularRegion?
```

```
var circleOverlay: MKCircle? {
```

```
    didSet {
```

```
        // Remove the old overlay (if exists)
```

```
        if let oldOverlay = oldValue {
```

```
            mapView.removeOverlay(oldOverlay)
```



```

    }

    // Add the new overlay (if exists)

    if let overlay = circleOverlay {

        mapView.addOverlay(overlay)

    }

}

}

}

var locationManager = CLLocationManager()


// MARK: View Methods


/// Configures the map view, search bar and location manager.

override func viewDidLoad() {

    super.viewDidLoad()

    searchBar.delegate = self

```

```

    mapView.delegate = self

    mapView.showsUserLocation = true

    mapView.isPitchEnabled = false

    locationManager.delegate = self

}

/// Loads the user's location and zooms the target region.

override func viewDidAppear(_ animated: Bool) {

    super.viewDidAppear(animated)

    locationManager.requestWhenInUseAuthorization()

    locationManager.requestLocation()

    if let region = targetRegion {

        annotateAndZoomToRegion(region)

    }

}

```

```

/// Updates the overlay when the orientation changes.

override func didRotate(from fromInterfaceOrientation: UIInterfaceOrientation) {

    overlayView.setNeedsDisplay()

}

// MARK: Button Actions

/**

    Generates a map region based on the map's position

    and zoom, then notifies the delegate that the region has changed.

    This will dismiss the view.

*/

@IBAction func didTapSaveButton(_ sender: UIBarButtonItem) {

    let circleDegreeDelta: CLLocationDegrees

    let pointOnCircle: CLLocation

    if mapView.region.span.latitudeDelta > mapView.region.span.longitudeDelta {

```

```

        circleDegreeDelta = mapView.region.span.longitudeDelta /
MapViewController.MapRegionFraction

```

```

        pointOnCircle = CLLocation(latitude: mapView.region.center.latitude, longitude:
mapView.region.center.longitude - circleDegreeDelta)

```

```

    }

```

```

    else {

```

```

        circleDegreeDelta = mapView.region.span.latitudeDelta /
MapViewController.MapRegionFraction

```

```

        pointOnCircle = CLLocation(latitude: mapView.region.center.latitude -
circleDegreeDelta, longitude: mapView.region.center.longitude)

```

```

    }

```

```

    let mapCenterLocation = CLLocation(latitude: mapView.region.center.latitude,
longitude: mapView.region.center.longitude)

```

```

    let distance = pointOnCircle.distance(from: mapCenterLocation)

```

```

    let genericRegion = CLCircularRegion(center: mapView.region.center, radius:
distance, identifier: Identifiers.circularRegion)

```

```

    circleOverlay = MKCircle(center: genericRegion.center, radius:
genericRegion.radius)

```

```

        delegate?.mapViewDidUpdateRegion(genericRegion)

        dismiss(animated: true, completion: nil)

    }

    /// Dismisses the view without notifying the delegate.

    @IBAction func didTapCancelButton(_ sender: UIBarButtonItem) {

        dismiss(animated: true, completion: nil)

    }

    // MARK: Search Bar Methods

    /**

    Dismisses the keyboard and runs a new search from the

    search bar.

    */

    func searchBarSearchButtonClicked(_ searchBar: UISearchBar) {

        searchBar.resignFirstResponder()

```

```

        mapView.removeAnnotations(mapView.annotations)

        performSearch()

    }

// MARK: Location Manager Methods

/// Zooms to the user's location if the region is not set.

func locationManager(_ manager: CLLocationManager, didUpdateLocations locations:
[CLLocation]) {

    guard let lastLocation = locations.last else { return }

    if targetRegion != nil {

        // Do not zoom to the user's location if there is already a target region.

        return

    }

    let newRegion = MKCoordinateRegion(center: lastLocation.coordinate, span:
MapViewController.InitialZoomSpan)

    mapView.setRegion(newRegion, animated: true)

}

```

```
/**
```

The method is required.

Simply logs the error.

```
*/
```

```
func locationManager(_ manager: CLLocationManager, didFailWithError error: Error)
{
    print("System: Location Manager Error: \(error)")
}
```

```
/**
```

When the user updates the authorization status, we want to

zoom to their current location by asking for it.

```
*/
```

```
func locationManager(_ manager: CLLocationManager, didChangeAuthorization status:
CLLocationAuthorizationStatus) {
    locationManager.requestLocation()
}
```

```
// MARK: Helper Methods
```

```
/**
```

Updates the region overlay and zooms the map region

- parameter region: The new `CLCircularRegion`.

```
*/
```

```
private func annotateAndZoomToRegion(_ region: CLCircularRegion) {
```

```
    circleOverlay = MKCircle(center: region.center, radius: region.radius)
```

```
    let multiplier = MapViewController.MapRegionFraction
```

```
    let mapRegion = MKCoordinateRegion.init(center: region.center, latitudinalMeters:
region.radius*multiplier, longitudinalMeters: region.radius*multiplier)
```

```
    mapView.setRegion(mapRegion, animated: false)
```

```
}
```

```
/**
```

Performs a natural language search for locations

in the map's region that match the `searchBar`'s text.

```
*/
```

```
private func performSearch() {
```

```
    let request = MKLocalSearch.Request()
```

```
    request.naturalLanguageQuery = searchBar.text
```

```
    let multiplier = MapViewController.RegionQueryDegreeMultiplier
```

```
    let querySpan = MKCoordinateSpan(latitudeDelta:
mapView.region.span.latitudeDelta*multiplier, longitudeDelta:
mapView.region.span.longitudeDelta*multiplier)
```

```
    request.region = MKCoordinateRegion(center: mapView.region.center, span:
querySpan)
```

```
    let search = MKLocalSearch(request: request)
```

```
    var matchingItems = [MKMapItem]()
```

```
    search.start { response, error in
```

```
        let mapItems: [MKMapItem] = response?.mapItems ?? []
```

```
        for item in mapItems {
```

```

        matchingItems.append(item)

        let annotation = MKPointAnnotation()

        annotation.coordinate = item.placemark.coordinate

        annotation.title = item.name

        self.mapView.addAnnotation(annotation)

    }

}

}

```

/// - returns: An `MKOverlayRenderer` with our custom stroke and fill.

```

func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) ->
MKOverlayRenderer {

    let circleRenderer = MKCircleRenderer(overlay: overlay)

    circleRenderer.fillColor = UIColor.blue.withAlphaComponent(0.2)

    circleRenderer.strokeColor = UIColor.black

    circleRenderer.lineWidth = 2.0

    return circleRenderer

}

```

```
}

//

// ChartMarkerImage.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc(ChartMarkerImage)

open class MarkerImage: NSObject, IMarker
```

```

{

    /// The marker image to render

    @objc open var image: NSUIImage?

    open var offset: CGPoint = CGPoint()

    @objc open weak var chartView: ChartViewBase?

    /// As long as size is 0.0/0.0 - it will default to the image's size

    @objc open var size: CGSize = CGSize()

    public override init()
    {
        super.init()
    }

    open func offsetForDrawing(atPoint point: CGPoint) -> CGPoint

```

```
{  
  
    var offset = self.offset  
  
  
    let chart = self.chartView  
  
  
    var size = self.size  
  
  
    if size.width == 0.0 && image != nil  
    {  
  
        size.width = image?.size.width ?? 0.0  
  
    }  
  
    if size.height == 0.0 && image != nil  
    {  
  
        size.height = image?.size.height ?? 0.0  
  
    }  
  
  
    let width = size.width
```

```
let height = size.height
```

```
if point.x + offset.x < 0.0
```

```
{
```

```
    offset.x = -point.x
```

```
}
```

```
else if chart != nil && point.x + width + offset.x > chart!.bounds.size.width
```

```
{
```

```
    offset.x = chart!.bounds.size.width - point.x - width
```

```
}
```

```
if point.y + offset.y < 0
```

```
{
```

```
    offset.y = -point.y
```

```
}
```

```
else if chart != nil && point.y + height + offset.y > chart!.bounds.size.height
```

```
{
```

```

        offset.y = chart!.bounds.size.height - point.y - height
    }

    return offset
}

open func refreshContent(entry: ChartDataEntry, highlight: Highlight)
{
    // Do nothing here...
}

open func draw(context: CGContext, point: CGPoint)
{
    guard let image = image else { return }

    let offset = offsetForDrawing(atPoint: point)

```

```
var size = self.size

if size.width == 0.0

{

    size.width = image.size.width

}

if size.height == 0.0

{

    size.height = image.size.height

}


let rect = CGRect(

    x: point.x + offset.x,

    y: point.y + offset.y,

    width: size.width,

    height: size.height)
```



```

        NSUIGraphicsPushContext(context)

        image.draw(in: rect)

        NSUIGraphicsPopContext()

    }

}

//

// ChartMarkerView.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

```

```
import CoreGraphics
```

```
#if canImport(AppKit)
```

```
import AppKit
```

```
#endif
```

```
@objc(CharMarkerView)
```

```
open class MarkerView: NSUIView, IMarker
```

```
{
```

```
    open var offset: CGPoint = CGPoint()
```

```
    @objc open weak var chartView: ChartViewBase?
```

```
    open func offsetForDrawing(atPoint point: CGPoint) -> CGPoint
```

```
{
```

```
    guard let chart = chartView else { return self.offset }
```

```
var offset = self.offset
```

```
let width = self.bounds.size.width
```

```
let height = self.bounds.size.height
```

```
if point.x + offset.x < 0.0
```

```
{
```

```
    offset.x = -point.x
```

```
}
```

```
else if point.x + width + offset.x > chart.bounds.size.width
```

```
{
```

```
    offset.x = chart.bounds.size.width - point.x - width
```

```
}
```

```
if point.y + offset.y < 0
```

```
{
```

```
    offset.y = -point.y
```

```
}
```

```
else if point.y + height + offset.y > chart.bounds.size.height
```

```
{
```

```
    offset.y = chart.bounds.size.height - point.y - height
```

```
}
```

```
return offset
```

```
}
```

```
open fun refreshContent(entry: ChartDataEntry, highlight: Highlight)
```

```
{
```

```
    // Do nothing here...
```

```
}
```

```
open fun draw(context: CGContext, point: CGPoint)
```

```
{
```

```
    let offset = self.offsetForDrawing(atPoint: point)
```

```

context.saveGState()

context.translateBy(x: point.x + offset.x,
                    y: point.y + offset.y)

NSUIGraphicsPushContext(context)

self.nsuiLayer?.render(in: context)

NSUIGraphicsPopContext()

context.restoreGState()
}

@objc

open class func viewFromXib(in bundle: Bundle = .main) -> MarkerView?

{

    #if !os(OSX)

return bundle.loadNibNamed(

    String(describing: self),

```

```

        owner: nil,

        options: nil)?[0] as? MarkerView

    #else

    var loadedObjects = NSArray()

    let                                loadedObjectsPointer                                =
    AutoreleasingUnsafeMutablePointer<NSArray?>(&loadedObjects)

    if bundle.loadNibNamed(

        NSNib.Name(String(describing: self)),

        owner: nil,

        topLevelObjects: loadedObjectsPointer)

    {

        return loadedObjects[0] as? MarkerView

    }

    return nil

    #endif

```

```
}
```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `ModifyAccessoryViewController` allows the user to modify a HomeKit accessory.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
/// Represents the sections in the `ModifyAccessoryViewController`.
```

```
enum AddAccessoryTableViewSection: Int {
```

```
    case name, rooms, identify
```

```

static let count = 3

}

/// Contains a method for notifying the delegate that the accessory was saved.

protocol ModifyAccessoryDelegate {

    func          accessoryViewController(_          accessoryViewController:
ModifyAccessoryViewController, didSaveAccessory accessory: HMAccessory)

}

/// A view controller that allows for renaming, reassigning, and identifying accessories
before and after they've been added to a home.

class          ModifyAccessoryViewController:          HMCatalogViewController,
HMAccessoryDelegate {

    // MARK: Types

    struct Identifiers {

        static let roomCell = "RoomCell"

    }

```



```
// MARK: Properties
```

```
// Update this if the accessory failed in any way.
```

```
private var didEncounterError = false
```

```
private var selectedIndexPath: IndexPath?
```

```
private var selectedRoom: HMRoom!
```

```
@IBOutlet weak var nameField: UITextField!
```

```
private lazy var activityIndicator = UIActivityIndicatorView(style: .gray)
```

```
private let saveAccessoryGroup = DispatchGroup()
```

```
private var editingExistingAccessory = false
```

```
// Strong reference, because we will replace the button with an activity indicator.
```

```
@IBOutlet /* strong */ var addButton: UIBarButtonItem!
```

```
var delegate: ModifyAccessoryDelegate?
```

```
var rooms = [HMRoom]()
```

```
var accessory: HMAccessory!
```

```
// MARK: View Methods
```

```
/// Configures the table view and initializes view elements.
```

```
override func viewDidLoad() {
```

```
    super.viewDidLoad()
```

```
    tableView.estimatedRowHeight = 44.0
```

```
    tableView.rowHeight = UITableView.automaticDimension
```

```
    selectedRoom = accessory.room ?? home.roomForEntireHome()
```

```

// If the accessory belongs to the home already, we are in 'edit' mode.

editingExistingAccessory = accessoryHasBeenAddedToHome()

if editingExistingAccessory {

    // Show 'save' instead of 'add.'

    addButton.title = NSLocalizedString("Save", comment: "Save")

}

else {

    /*

        If we're not editing an existing accessory, then let the back

        button show in the left.

    */

    navigationItem.leftBarButtonItem = nil

}

// Put the accessory's name in the 'name' field.

resetNameField()

```

```

        // Register a cell for the rooms.

        tableView.register(UITableViewCell.self,                       forCellReuseIdentifier:
Identifiers.roomCell)

    }

    /**
     Registers as the delegate for the current home
     and the accessory.

    */

    override func registerAsDelegate() {

        super.registerAsDelegate()

        accessory.delegate = self

    }

    /// Replaces the activity indicator with the 'Add' or 'Save' button.

    func hideActivityIndicator() {

        activityIndicator.stopAnimating()

        navigationItem.rightBarButtonItem = addButton

```

```
}
```

```
/// Temporarily replaces the 'Add' or 'Save' button with an activity indicator.
```

```
func showActivityIndicator() {
```

```
    navigationItem.rightBarButtonItem = UIBarButtonItem(customView:
activityIndicator)
```

```
    activityIndicator.startAnimating()
```

```
}
```

```
/**
```

```
    Called whenever the user taps the 'add' button.
```

```
    This method:
```

1. Adds the accessory to the home, if not already added.
2. Updates the accessory's name, if necessary.
3. Assigns the accessory to the selected room, if necessary.

```
*/
```

```
@IBAction func didTapAddButton() {
```

```

let name = trimmedName

showActivityIndicator()

if editingExistingAccessory {

    home(home, assignAccessory: accessory, toRoom: selectedRoom)

    updateName(name, forAccessory: accessory)

}

else {

    saveAccessoryGroup.enter()

    home.addAccessory(accessory) { error in

        if let error = error {

            self.hideActivityIndicator()

            self.displayError(error)

            self.didEncounterError = true

        }

        else {

            // Once it's successfully added to the home, add it to the room that's selected.

```

```
                self.home(self.home,          assignAccessory:self.accessory,      toRoom:
self.selectedRoom)
```

```
                self.updateName(name, forAccessory: self.accessory)
```

```
            }
```

```
            self.saveAccessoryGroup.leave()
```

```
        }
```

```
    }
```

```
saveAccessoryGroup.notify(queue: DispatchQueue.main) {
```

```
    self.hideActivityIndicator()
```

```
    if !self.didEncounterError {
```

```
        self.dismiss(nil)
```

```
    }
```

```
}
```

```
}
```

```
/**
```

Informs the delegate that the accessory has been saved, and

dismisses the view controller.

*/

```
@IBAction func dismiss(_ sender: AnyObject?) {
```

```
    delegate?.accessoryViewController(self, didSaveAccessory: accessory)
```

```
    if editingExistingAccessory {
```

```
        presentingViewController?.dismiss(animated: true, completion: nil)
```

```
    }
```

```
    else {
```

```
        _ = navigationController?.popViewController(animated: true)
```

```
    }
```

```
}
```

/**

- returns: `true` if the accessory has already been added to

the home; `false` otherwise.

*/

```
func accessoryHasBeenAddedToHome() -> Bool {
```



```

        return home.accessories.contains(accessory)

    }

```

```

/**

```

Updates the accessories name. This function will enter and leave the saved dispatch group.

If the accessory's name is already equal to the passed-in name, this method does nothing.

- parameter name: The new name for the accessory.

- parameter accessory: The accessory to rename.

```

*/

```

```

func updateName(_ name: String, forAccessory accessory: HMAccessory) {

```

```

    if accessory.name == name {

```

```

        return

```

```

    }

```

```

    saveAccessoryGroup.enter()

```

```

    accessory.updateName(name) { error in

```

```

        if let error = error {

            self.displayError(error)

            self.didEncounterError = true

        }

        self.saveAccessoryGroup.leave()

    }
}

```

```

/**

```

Assigns the given accessory to the provided room. This method will enter and leave the saved dispatch group.

- parameter home: The home to assign.
- parameter accessory: The accessory to be assigned.
- parameter room: The room to which to assign the accessory.

```

*/

```

```

func home(_ home: HMHome, assignAccessory accessory: HMAccessory, toRoom
room: HMRoom) {

```

```

    if accessory.room == room {

        return

    }

    saveAccessoryGroup.enter()

    home.assignAccessory(accessory, to: room) { error in

        if let error = error {

            self.displayError(error)

            self.didEncounterError = true

        }

        self.saveAccessoryGroup.leave()

    }

}

```

/// Tells the current accessory to identify itself.

```

func identifyAccessory() {

    accessory.identify { error in

        if let error = error {

```

```

        self.displayError(error)

    }

}

}

/// Enables the name field if the accessory's name changes.

func resetNameField() {

    var action: String

    if editingExistingAccessory {

        action = NSLocalizedString("Edit %@", comment: "Edit Accessory")

    }

    else {

        action = NSLocalizedString("Add %@", comment: "Add Accessory")

    }

    navigationItem.title = String(format: action, accessory.name) as String

    nameField.text = accessory.name

    nameField.isEnabled = home.isAdmin

```

```

        enableAddButtonIfApplicable()

    }

    /// Enables the save button if the name field is not empty.

    func enableAddButtonIfApplicable() {

        addButton.isEnabled = home.isAdmin && trimmedName.count > 0

    }

    /// - returns: The `nameField`'s text, trimmed of newline and whitespace characters.

    var trimmedName: String {

        return nameField.text!.trimmingCharacters(in:
CharacterSet.whitespacesAndNewlines)

    }

    /// Enables or disables the add button.

    @IBAction func nameFieldDidChange(_ sender: AnyObject) {

        enableAddButtonIfApplicable()

    }

```

```
// MARK: Table View Methods
```

```
/// - returns: The number of `AddAccessoryTableViewSection`s.
```

```
override func numberOfSections(in tableView: UITableView) -> Int {
```

```
    return AddAccessoryTableViewSection.count
```

```
}
```

```
/// - returns: The number rows for the rooms section. All other sections are static.
```

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section:  
Int) -> Int {
```

```
    switch AddAccessoryTableViewSection(rawValue: section) {
```

```
        case .rooms?:
```

```
            return home.allRooms.count
```

```
        case nil:
```

```
            fatalError("Unexpected `AddAccessoryTableViewSection` raw value.")
```

default:

```
return super.tableView(tableView, numberOfRowsInSection: section)
```

```
}
```

```
}
```

/// - returns: `UITableViewAutomaticDimension` for dynamic cell, super otherwise.

```
override func tableView(_ tableView: UITableView, heightForRowAt indexPath:
IndexPath) -> CGFloat {
```

```
switch AddAccessoryTableViewSection(rawValue: indexPath.section) {
```

```
case .rooms?:
```

```
return UITableView.automaticDimension
```

```
case nil:
```

```
fatalError("Unexpected `AddAccessoryTableViewSection` raw value.")
```

default:

```
return super.tableView(tableView, heightForRowAt: indexPath)
```

```
}
```

```
}
```

```
/// - returns: A 'room cell' for the rooms section, super otherwise.
```

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
```

```
    switch AddAccessoryTableViewSection(rawValue: indexPath.section) {
```

```
        case .rooms?:
```

```
            return self.tableView(tableView, roomCellForRowAtIndexPath: indexPath)
```

```
        case nil:
```

```
            fatalError("Unexpected `AddAccessoryTableViewSection` raw value.")
```

```
        default:
```

```
            return super.tableView(tableView, cellForRowAt: indexPath)
```

```
    }
```

```
}
```

```
/**
```


Creates a cell with the name of each room within the home, displaying a checkmark if the room

is the currently selected room.

*/

```
func tableView(_ tableView: UITableView, cellForRowAtIndexPath indexPath:
IndexPath) -> UITableViewCell {
```

```
    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.roomCell, for:
indexPath)
```

```
    let room = home.allRooms[indexPath.row] as HMRoom
```

```
    cell.textLabel?.text = home.nameForRoom(room)
```

```
    // Put a checkmark on the selected room.
```

```
    cell.accessoryType = room == selectedRoom ? .checkmark : .none
```

```
    if !home.isAdmin {
```

```
        cell.selectionStyle = .none
```

```
    }
```

```
    return cell
```

```
}
```

```

/// Handles row selection based on the section.

override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {

    tableView.deselectRow(at: indexPath, animated: true)

    switch AddAccessoryTableViewSection(rawValue: indexPath.section) {

        case .rooms?:

            guard home.isAdmin else { return }

            selectedRoom = home.allRooms[indexPath.row]

            let sections = IndexSet(integer:
AddAccessoryTableViewSection.rooms.rawValue)

            tableView.reloadSections(sections, with: .automatic)

        case .identify?:

```

```
identifyAccessory()
```

```
case nil:
```

```
fatalError("Unexpected `AddAccessoryTableViewSection` raw value.")
```

```
default: break
```

```
}
```

```
}
```

```
/// Required override.
```

```
override func tableView(_ tableView: UITableView, indentationLevelForRowAt  
indexPath: IndexPath) -> Int {
```

```
    return super.tableView(tableView, indentationLevelForRowAt: IndexPath(row: 0,  
section: indexPath.section))
```

```
}
```

```
// MARK: HMHomeDelegate Methods
```

```
// All home changes reload the view.
```

```
func home(_ home: HMHome, didUpdateNameFor room: HMRoom) {  
  
    tableView.reloadData()  
  
}
```

```
func home(_ home: HMHome, didAddRoom room: HMRoom) {  
  
    tableView.reloadData()  
  
}
```

```
func home(_ home: HMHome, didRemoveRoom room: HMRoom) {  
  
    if selectedRoom == room {  
  
        // Reset the selected room if ours was deleted.  
  
        selectedRoom = homeStore.home!.roomForEntireHome()  
  
    }  
  
    tableView.reloadData()  
  
}
```

```

func home(_ home: HMHome, didAddAccessory accessory: HMAccessory) {

    /*

    Bridged accessories don't call the original completion handler if their

    bridges are added to the home. We must respond to `HMHomeDelegate`s

    `home(_:didAddAccessory:)` and assign bridged accessories properly.

    */

    if selectedRoom != nil {

        self.home(home, assignAccessory: accessory, toRoom: selectedRoom)

    }

}

func home(_ home: HMHome, didUnblockAccessory accessory: HMAccessory) {

    tableView.reloadData()

}

// MARK: HMAccessoryDelegate Methods

```

```

    /// If the accessory's name changes, we update the name field.

    func accessoryDidUpdateName(_ accessory: HMAccessory) {

        resetNameField()

    }

}

//

// MornitorCell.swift

// FirebaseStarterApp

//

// Created by Ruengruj Vivatthanamukda on 5/5/20.

// Copyright © 2020 Instamobile. All rights reserved.

//

```

```
import UIKit
```

```
import FoldingCell
```

```

class MornitorCell: FoldingCell {

    override func awakeFromNib() {

        super.awakeFromNib()

        // Initialization code

    }

    override func setSelected(_ selected: Bool, animated: Bool) {

        super.setSelected(selected, animated: animated)

        // Configure the view for the selected state

    }

    override func animationDuration(_ itemIndex: NSInteger, type _:
FoldingCell.AnimationType) -> TimeInterval {

        let durations = [0.26, 0.2, 0.2]

        return durations[itemIndex]

    }

```

```

    }

    //

    // MornitorTableViewController.swift

    // FirebaseStarterApp

    //

    // Created by Ruengruj Vivatthanamukda on 5/5/20.

    // Copyright © 2020 Instamobile. All rights reserved.

    //

import UIKit

import FoldingCell

class MornitorTableViewController: UITableViewController {

    fileprivate struct C {

        struct CellHeight {

```



```

static let close: CGFloat = 75 // equal or greater foregroundView height

static let open: CGFloat = 300 // equal or greater containerView height

}

}

var cellHeights: [CGFloat] = []

override func viewDidLoad() {

    super.viewDidLoad()

    // Uncomment the following line to preserve selection between presentations

    // self.clearsSelectionOnViewWillAppear = false

    // Uncomment the following line to display an Edit button in the navigation bar for
    this view controller.

    // self.navigationItem.rightBarButtonItem = self.editButtonItem

}

```

```
// MARK: - Table view data source
```

```
override func numberOfSections(in tableView: UITableView) -> Int {
```

```
    // #warning Incomplete implementation, return the number of sections
```

```
    return 0
```

```
}
```

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section:  
Int) -> Int {
```

```
    // #warning Incomplete implementation, return the number of rows
```

```
    return 0
```

```
}
```

```
// override func tableView(_ tableView: UITableView, cellForRowAt indexPath:  
IndexPath) -> UITableViewCell {
```

```
    // let cell = tableView.dequeueReusableCell(withIdentifier: "reuseIdentifier", for:  
indexPath)
```

```
//

//    // Configure the cell...

//

//    return cell

// }
```

```
override func tableView(_ : UITableView, heightForRowAt indexPath: IndexPath) ->
CGFloat {

    return cellHeights[indexPath.row]

}
```

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

    let cell = tableView.dequeueReusableCell(withIdentifier: "FoldingCell", for:
indexPath) as! FoldingCell

    let durations: [TimeInterval] = [0.26, 0.2, 0.2]

    cell.durationsForExpandedState = durations

    cell.durationsForCollapsedState = durations

    return cell
}
```

```
}
```

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:  
IndexPath) {
```

```
    let cell = tableView.cellForRow(at: indexPath) as! FoldingCell
```

```
    if cell.isAnimating() {
```

```
        return
```

```
    }
```

```
    var duration = 0.0
```

```
    let cellIsCollapsed = cellHeights[indexPath.row] == C.CellHeight.close
```

```
    if cellIsCollapsed {
```

```
        cellHeights[indexPath.row] = C.CellHeight.open
```

```
        cell.unfold(true, animated: true, completion: nil)
```

```
        duration = 0.5
```

```
    } else {
```

```
        cellHeights[indexPath.row] = C.CellHeight.close
```

```
cell.unfold(false, animated: true, completion: nil)
```

```
duration = 0.8
```

```
}
```

```
UIView.animate(withDuration: duration, delay: 0, options: .curveEaseOut,  
animations: { () -> Void in
```

```
tableView.beginUpdates()
```

```
tableView.endUpdates()
```

```
// fix https://github.com/Ramotion/folding-cell/issues/169
```

```
if cell.frame.maxY > tableView.frame.maxY {
```

```
tableView.scrollToRow(at: indexPath, at: UITableView.ScrollPosition.bottom,  
animated: true)
```

```
}
```

```
}, completion: nil)
```

```
}
```

```
/*
```

```

// Override to support conditional editing of the table view.

override func tableView(_ tableView: UITableView, canEditRowAt indexPath:
IndexPath) -> Bool {

    // Return false if you do not want the specified item to be editable.

    return true

}

*/

/*

// Override to support editing the table view.

override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {

    if editingStyle == .delete {

        // Delete the row from the data source

        tableView.deleteRows(at: [indexPath], with: .fade)

    } else if editingStyle == .insert {

        // Create a new instance of the appropriate class, insert it into the array, and add a
new row to the table view

    }

```

```

    }

    */

    /*

    // Override to support rearranging the table view.

    override func tableView(_ tableView: UITableView, moveRowAt fromIndexPath:
IndexPath, to: IndexPath) {

    }

    */

    /*

    // Override to support conditional rearranging of the table view.

    override func tableView(_ tableView: UITableView, canMoveRowAt indexPath:
IndexPath) -> Bool {

        // Return false if you do not want the item to be re-orderable.

        return true

    }

```

```

*/

/*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before
navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

    // Get the new view controller using segue.destination.

    // Pass the selected object to the new view controller.

}

*/

}

//

// MornitorViewController.swift

// FirebaseStarterApp

//

```



```
// Created by Ruengruj Vivatthanamukda on 3/5/20.
```

```
// Copyright © 2020 Instamobile. All rights reserved.
```

```
//
```

```
import UIKit
```

```
import Charts
```

```
class MornitorViewController: UIViewController, ChartViewDelegate{
```

```
    var shouldHideData: Bool = false
```

```
    @IBOutlet var chartView: LineChartView!
```

```
    @IBOutlet var chartView2: LineChartView!
```

```
    @IBOutlet var sliderX: UISlider!
```

```
    @IBOutlet var sliderTextX: UITextField!
```

```
override func viewDidLoad() {  
  
    super.viewDidLoad()  
  
    navigationController?.navigationBar.prefersLargeTitles = true  
  
    if #available(iOS 13.0, *) {  
  
        let navBarAppearance = UINavigationBarAppearance()  
  
        navBarAppearance.configureWithOpaqueBackground()  
  
        navBarAppearance.titleTextAttributes = [.foregroundColor: UIColor.white]  
  
        navBarAppearance.largeTitleTextAttributes = [.foregroundColor: UIColor.black]  
  
        navBarAppearance.backgroundColor = UIColor.init(hexString: "#a1edab")  
  
        navigationController?.navigationBar.standardAppearance = navBarAppearance  
  
        navigationController?.navigationBar.scrollEdgeAppearance = navBarAppearance  
  
        chartView.layer.cornerRadius = 10  
  
        chartView.clipsToBounds = true  
  
        chartView2.layer.cornerRadius = 10  
  
        chartView2.clipsToBounds = true  
  
    }  
}
```

```
// Do any additional setup after loading the view.
```

```
self.title = "Mornitor"
```

```
chartView.delegate = self
```

```
chartView.chartDescription?.enabled = false
```

```
chartView.dragEnabled = true
```

```
chartView.setScaleEnabled(true)
```

```
chartView.pinchZoomEnabled = false
```

```
chartView.highlightPerDragEnabled = true
```

```
chartView.backgroundColor = .systemPink
```

```
//chartView.legend.enabled = false
```

```
let xAxis = chartView.xAxis
```

```
xAxis.labelPosition = .top
```

```
xAxis.labelFont = .systemFont(ofSize: 10, weight: .light)
```

```
xAxis.labelTextColor = UIColor(red: 255/255, green: 192/255, blue: 56/255, alpha:
```

1)

```
xAxis.drawAxisLineEnabled = false
```

```
xAxis.drawGridLinesEnabled = false
```

```
xAxis.centerAxisLabelsEnabled = false
```

```
xAxis.granularity = 3600
```

```
xAxis.valueFormatter = DateValueFormatter()
```

```
xAxis.setLabelCount(3, force: false)
```

```
let leftAxis = chartView.leftAxis
```

```
leftAxis.labelPosition = .outsideChart
```

```
leftAxis.labelFont = .systemFont(ofSize: 12, weight: .light)
```

```
leftAxis.drawGridLinesEnabled = false
```

```
leftAxis.granularityEnabled = true
```

```
leftAxis.granularity = 1
```

```
leftAxis.axisMinimum = 0
```

```
leftAxis.axisMaximum = 500
```

```
leftAxis.yOffset = -1
```

```
leftAxis.labelTextColor = UIColor(red: 255/255, green: 192/255, blue: 56/255,  
alpha: 1)
```

```
chartView.rightAxis.enabled = false
```

```
chartView.legend.form = .line
```

```
sliderX.value = 100
```

```
slidersValueChanged(nil)
```

```
chartView.animate(xAxisDuration: 2.5)
```

```
chartView2.delegate = self
```

```
chartView2.chartDescription?.enabled = false
```

```
chartView2.dragEnabled = true
```

```
chartView2.setScaleEnabled(true)
```

```
chartView2.pinchZoomEnabled = false
```

```
chartView2.highlightPerDragEnabled = true
```

```
chartView2.backgroundColor = UIColor(red: 30/255, green: 144/255, blue:  
255/255, alpha: 1)
```

```
//chartView2.legend.enabled = false
```

```
let xAxis2 = chartView2.xAxis
```

```
xAxis2.labelPosition = .top
```

```
xAxis2.labelFont = .systemFont(ofSize: 10, weight: .light)
```

```
xAxis2.labelTextColor = UIColor(red: 255/255, green: 192/255, blue: 56/255,  
alpha: 1)
```

```
xAxis2.drawAxisLineEnabled = false
```

```
xAxis2.drawGridLinesEnabled = false
```

```
xAxis2.centerAxisLabelsEnabled = false
```

```
xAxis2.granularity = 3600
```

```
xAxis2.valueFormatter = DateValueFormatter()
```

```
xAxis2.setLabelCount(3, force: false)
```

```
let leftAxis2 = chartView2.leftAxis
```

```
leftAxis2.labelPosition = .outsideChart
```

```
leftAxis2.labelFont = .systemFont(ofSize: 12, weight: .light)
```

```
leftAxis2.drawGridLinesEnabled = false
```

```
leftAxis2.granularityEnabled = true
```

```
leftAxis2.granularity = 1
```

```
leftAxis2.axisMinimum = 0
```

```
leftAxis2.axisMaximum = 50
```

```
leftAxis2.yOffset = -1
```

```
leftAxis2.labelTextColor = UIColor(red: 255/255, green: 192/255, blue: 56/255,  
alpha: 1)
```

```
chartView2.rightAxis.enabled = false
```

```
chartView2.legend.form = .line
```

```
sliderX.value = 100
```

```
slidersValueChanged(nil)
```

```
chartView2.animate(xAxisDuration: 2.5)
```

```
}
```

```
func updateChartData() {
```

```
    if self.shouldHideData {
```

```
        chartView.data = nil
```

```
        chartView2.data = nil
```

```
        return
```

```
    }
```

```
    self.setDataCount(Int(sliderX.value), range: 30)
```

```
}
```



```

func setDataCount(_ count: Int, range: UInt32) {

    let now = Date().timeIntervalSince1970

    let hourSeconds: TimeInterval = 3600

    let from = now - (Double(count) / 2) * hourSeconds

    let to = now + (Double(count) / 2) * hourSeconds

    let values1 = stride(from: from, to: to, by: hourSeconds).map { (x) ->
ChartDataEntry in

        let y = Double.random(in: 150 ... 450)

        return ChartDataEntry(x: x, y: Double(y))

    }

    let values3 = stride(from: from, to: to, by: hourSeconds).map { (x) ->
ChartDataEntry in

        let y = Double.random(in: 30 ... 35)

```

```
        return ChartDataEntry(x: x, y: Double(y))
    }
}
```

```
let set1 = LineChartDataSet(entries: values1, label: "Unit")
```

```
set1.axisDependency = .left
```

```
set1.setColor(UIColor(red: 255/255, green: 255/255, blue: 255/255, alpha: 1))
```

```
set1.lineWidth = 1.5
```

```
set1.drawCirclesEnabled = false
```

```
set1.drawValuesEnabled = false
```

```
set1.fillAlpha = 0.26
```

```
set1.fillColor = UIColor(red: 51/255, green: 181/255, blue: 229/255, alpha: 1)
```

```
set1.highlightColor = UIColor(red: 244/255, green: 117/255, blue: 117/255, alpha:
```

1)

```
set1.drawCircleHoleEnabled = false
```

```
let data = LineChartData(dataSets: [set1])
```

```
data.setValueTextColor(.white)
```

```
data.setValueFont(.systemFont(ofSize: 9, weight: .light))
```

```
let set3 = LineChartDataSet(entries: values3, label: "Temperature")
```

```
set3.axisDependency = .left
```

```
set3.setColor(UIColor(red: 255/255, green: 255/255, blue: 255/255, alpha: 1))
```

```
set3.lineWidth = 1.5
```

```
set3.drawCirclesEnabled = false
```

```
set3.drawValuesEnabled = false
```

```
set3.fillAlpha = 0.26
```

```
set3.fillColor = UIColor(red: 51/255, green: 181/255, blue: 229/255, alpha: 1)
```

```
set3.highlightColor = .white
```

```
set3.drawCircleHoleEnabled = false
```

```
let data2 = LineChartData(dataSets: [set3])
```

```
data2.setValueTextColor(.white)
```

```
data2.setValueFont(.systemFont(ofSize: 9, weight: .light))
```

```
chartView.data = data
```

```
chartView2.data = data2
```

```
}
```

```
@IBAction func slidersValueChanged(_ sender: Any?) {
```

```
    sliderTextX.text = "\(Int(sliderX.value))"
```

```
    self.updateChartData()
```

```
}
```

```
}
```

```
//
```

```
// MoveViewJob.swift
```

```
// Charts
```

```
//  
  
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda  
  
// A port of MPAndroidChart for iOS  
  
// Licensed under Apache License 2.0  
  
//  
  
// https://github.com/danielgindi/Charts  
  
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
@objc(MoveChartViewJob)
```

```
open class MoveViewJob: ViewPortJob
```

```
{
```

```
    open override func doJob()
```

```
    {
```

```
        guard
```

```

        let viewPortHandler = viewPortHandler,

        let transformer = transformer,

        let view = view

        else { return }

    var pt = CGPoint(

        x: xValue,

        y: yValue

    )

    transformer.pointValueToPixel(&pt)

    viewPortHandler.centerViewPort(pt: pt, chart: view)

}

}

//

// MultipartFormData.swift

//

```

// Copyright (c) 2014-2018 Alamofire Software Foundation (<http://alamofire.org/>)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

import Foundation

#if os(iOS) || os(watchOS) || os(tvOS)

import MobileCoreServices

#elseif os(macOS)

import CoreServices

#endif

/// Constructs `multipart/form-data` for uploads within an HTTP or HTTPS body. There
are currently two ways to encode

/// multipart form data. The first way is to encode the data directly in memory. This is very efficient, but can lead

/// to memory issues if the dataset is too large. The second way is designed for larger datasets and will write all the

/// data to a single file on disk with all the proper boundary segmentation. The second approach MUST be used for

/// larger datasets such as video content, otherwise your app may run out of memory when trying to encode the dataset.

///

/// For more information on `multipart/form-data` in general, please refer to the RFC-2388 and RFC-2045 specs as well

/// and the w3 form documentation.

///

/// - <https://www.ietf.org/rfc/rfc2388.txt>

/// - <https://www.ietf.org/rfc/rfc2045.txt>

/// - <https://www.w3.org/TR/html401/interact/forms.html#h-17.13>

open class MultipartFormData {

// MARK: - Helper Types

struct EncodingCharacters {

```

static let crlf = "\r\n"

}

struct BoundaryGenerator {

    enum BoundaryType {

        case initial, encapsulated, final

    }

    static func randomBoundary() -> String {

        return String(format: "alamofire.boundary.%08x%08x", arc4random(),
arc4random())

    }

    static func boundaryData(forBoundaryType boundaryType: BoundaryType,
boundary: String) -> Data {

        let boundaryText: String

        switch boundaryType {

```

```

    case .initial:

        boundaryText = "--\{(boundary)\}(EncodingCharacters.crlf)"

    case .encapsulated:

        boundaryText = "\{(EncodingCharacters.crlf)--
\{(boundary)\}(EncodingCharacters.crlf)"

    case .final:

        boundaryText = "\{(EncodingCharacters.crlf)--\{(boundary)--
\{(EncodingCharacters.crlf)"

    }

    return Data(boundaryText.utf8)

}

}

```

```

class BodyPart {

    let headers: HTTPHeaders

    let bodyStream: InputStream

    let bodyContentLength: UInt64

```

```

var hasInitialBoundary = false

var hasFinalBoundary = false

init(headers: HTTPHeaders, bodyStream: InputStream, bodyContentLength: UInt64)
{

    self.headers = headers

    self.bodyStream = bodyStream

    self.bodyContentLength = bodyContentLength

}

}

// MARK: - Properties

/// Default memory threshold used when encoding `MultipartFormData`, in bytes.

public static let encodingMemoryThreshold: UInt64 = 10_000_000

/// The `Content-Type` header value containing the boundary used to generate the
`multipart/form-data`.

```

```
open lazy var contentType: String = "multipart/form-data; boundary=\(self.boundary)"
```

```
/// The content length of all body parts used to generate the `multipart/form-data` not  
including the boundaries.
```

```
public var contentLength: UInt64 { return bodyParts.reduce(0) { $0 +  
$1.bodyContentLength } }
```

```
/// The boundary used to separate the body parts in the encoded form data.
```

```
public let boundary: String
```

```
let fileManager: FileManager
```

```
private var bodyParts: [BodyPart]
```

```
private var bodyPartError: AFError?
```

```
private let streamBufferSize: Int
```

```
// MARK: - Lifecycle
```

```

/// Creates an instance.

///

/// - Parameters:

/// - fileManager: `FileManager` to use for file operations, if needed.

/// - boundary: Boundary `String` used to separate body parts.

public init(fileManager: FileManager = .default, boundary: String? = nil) {

    self.fileManager = fileManager

    self.boundary = boundary ?? BoundaryGenerator.randomBoundary()

    bodyParts = []

    //

    // The optimal read/write buffer size in bytes for input and output streams is 1024
    (1KB). For more

    // information, please refer to the following article:

    //
    -
https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/Streams/Articles/ReadingInputStreams.html

    //

    streamBufferSize = 1024

```

```
}
```

```
// MARK: - Body Parts
```

```
/// Creates a body part from the data and appends it to the instance.
```

```
///
```

```
/// The body part data will be encoded using the following format:
```

```
///
```

```
/// - `Content-Disposition: form-data; name=#{name}; filename=#{filename}` (HTTP  
Header)
```

```
/// - `Content-Type: #{mimeType}` (HTTP Header)
```

```
/// - Encoded file data
```

```
/// - Multipart form boundary
```

```
///
```

```
/// - Parameters:
```

```
/// - data: `Data` to encoding into the instance.
```

```
/// - name: Name to associate with the `Data` in the `Content-Disposition` HTTP  
header.
```

```
/// - fileName: Filename to associate with the `Data` in the `Content-Disposition` HTTP header.
```

```
/// - mimeType: MIME type to associate with the data in the `Content-Type` HTTP header.
```

```
public func append(_ data: Data, withName name: String, fileName: String? = nil, mimeType: String? = nil) {
```

```
    let headers = contentHeaders(withName: name, fileName: fileName, mimeType: mimeType)
```

```
    let stream = InputStream(data: data)
```

```
    let length = UInt64(data.count)
```

```
    append(stream, withLength: length, headers: headers)
```

```
}
```

```
/// Creates a body part from the file and appends it to the instance.
```

```
///
```

```
/// The body part data will be encoded using the following format:
```

```
///
```

```
/// - `Content-Disposition: form-data; name=#{name}; filename=#{generated filename}` (HTTP Header)
```



```

/// - `Content-Type: #{generated mimeType}` (HTTP Header)

/// - Encoded file data

/// - Multipart form boundary

///

/// The filename in the `Content-Disposition` HTTP header is generated from the last
path component of the

/// `fileURL`. The `Content-Type` HTTP header MIME type is generated by mapping
the `fileURL` extension to the

/// system associated MIME type.

///

/// - Parameters:

/// - fileURL: `URL` of the file whose content will be encoded into the instance.

/// - name: Name to associate with the file content in the `Content-Disposition` HTTP
header.

public func append(_ fileURL: URL, withName name: String) {

    let fileName = fileURL.lastPathComponent

    let pathExtension = fileURL.pathExtension

    if !fileName.isEmpty && !pathExtension.isEmpty {

```

```

        let mime = mimeType(forPathExtension: pathExtension)

        append(fileURL, withName: name, fileName: fileName, mimeType: mime)

    } else {

        setBodyPartError(withReason: .bodyPartFilenameInvalid(in: fileURL))

    }

}

```

/// Creates a body part from the file and appends it to the instance.

///

/// The body part data will be encoded using the following format:

///

/// - Content-Disposition: form-data; name=#{name}; filename=#{filename} (HTTP Header)

/// - Content-Type: #{mimeType} (HTTP Header)

/// - Encoded file data

/// - Multipart form boundary

///

/// - Parameters:

```

    /// - fileURL: `URL` of the file whose content will be encoded into the instance.

    /// - name: Name to associate with the file content in the `Content-Disposition` HTTP
    header.

    /// - fileName: Filename to associate with the file content in the `Content-Disposition`
    HTTP header.

    /// - mimeType: MIME type to associate with the file content in the `Content-Type`
    HTTP header.

    public func append(_ fileURL: URL, withName name: String, fileName: String,
    mimeType: String) {

        let headers = contentHeaders(withName: name, fileName: fileName, mimeType:
    mimeType)

    //=====

    //          Check 1 - is file URL?

    //=====

    guard fileURL.isFileURL else {

        setBodyPartError(withReason: .bodyPartURLInvalid(url: fileURL))

        return

```

```
}
```

```
//=====
```

```
//      Check 2 - is file URL reachable?
```

```
//=====
```

```
do {
```

```
    let isReachable = try fileURL.checkPromisedItemIsReachable()
```

```
    guard isReachable else {
```

```
        setBodyPartError(withReason: .bodyPartFileNotReachable(at: fileURL))
```

```
        return
```

```
    }
```

```
} catch {
```

```
    setBodyPartError(withReason:      .bodyPartFileNotReachableWithError(atURL:
fileURL, error: error))
```

```
    return
```

```
}
```

```
//=====
```

```
//      Check 3 - is file URL a directory?
```

```
//=====
```

```
var isDirectory: ObjCBool = false
```

```
let path = fileURL.path
```

```
guard    fileManager.fileExists(atPath:    path,    isDirectory:    &isDirectory)  
&& !isDirectory.boolValue else {
```

```
    setBodyPartError(withReason: .bodyPartFileIsDirectory(at: fileURL))
```

```
    return
```

```
}
```

```
//=====
```

```
//      Check 4 - can the file size be extracted?
```

```
//=====
```

```
let bodyContentLength: UInt64
```

```
do {
```

```
    guard let fileSize = try fileManager.attributesOfItem(atPath: path)[.size] as?
    NSNumber else {
```

```
        setBodyPartError(withReason: .bodyPartFileSizeNotAvailable(at: fileURL))
```

```
        return
```

```
    }
```

```
    bodyContentLength = fileSize.uint64Value
```

```
    } catch {
```

```
        setBodyPartError(withReason: .bodyPartFileSizeQueryFailedWithError(forURL:
    fileURL, error: error))
```

```
        return
```

```
    }
```

```
//=====
```

```
//    Check 5 - can a stream be created from file URL?
```

```
//=====
```

```
guard let stream = InputStream(url: fileURL) else {
```

```
    setBodyPartError(withReason: .bodyPartInputStreamCreationFailed(for: fileURL))
```

```
    return
```

```
}
```

```
append(stream, withLength: bodyContentLength, headers: headers)
```

```
}
```

```
/// Creates a body part from the stream and appends it to the instance.
```

```
///
```

```
/// The body part data will be encoded using the following format:
```

```
///
```

/// - `Content-Disposition: form-data; name=#{name}; filename=#{filename}` (HTTP Header)

/// - `Content-Type: #{mimeType}` (HTTP Header)

/// - Encoded stream data

/// - Multipart form boundary

///

/// - Parameters:

/// - stream: `InputStream` to encode into the instance.

/// - length: Length, in bytes, of the stream.

/// - name: Name to associate with the stream content in the `Content-Disposition` HTTP header.

/// - fileName: Filename to associate with the stream content in the `Content-Disposition` HTTP header.

/// - mimeType: MIME type to associate with the stream content in the `Content-Type` HTTP header.

public func append(_ stream: InputStream,

 withLength length: UInt64,

 name: String,

 fileName: String,


```

        mimeType: String) {

            let headers = contentHeaders(withName: name, fileName: fileName, mimeType:
mimeType)

            append(stream, withLength: length, headers: headers)

        }

```

/// Creates a body part with the stream, length, and headers and appends it to the instance.

///

/// The body part data will be encoded using the following format:

///

/// - HTTP headers

/// - Encoded stream data

/// - Multipart form boundary

///

/// - Parameters:

/// - stream: `InputStream` to encode into the instance.

/// - length: Length, in bytes, of the stream.

/// - headers: `HTTPHeaders` for the body part.

```
public func append(_ stream: InputStream, withLength length: UInt64, headers:
HTTPHeaders) {
```

```
    let bodyPart = BodyPart(headers: headers, bodyStream: stream, bodyContentLength:
length)
```

```
    bodyParts.append(bodyPart)
```

```
}
```

```
// MARK: - Data Encoding
```

```
/// Encodes all appended body parts into a single `Data` value.
```

```
///
```

```
/// - Note: This method will load all the appended body parts into memory all at the same
time. This method should
```

```
///      only be used when the encoded data will have a small memory footprint. For
large data cases, please use
```

```
///      the `writeEncodedData(to:)` method.
```

```
///
```

```
/// - Returns: The encoded `Data`, if encoding is successful.
```

```
/// - Throws: An `NSError` if encoding encounters an error.
```

```

public func encode() throws -> Data {

    if let bodyPartError = bodyPartError {

        throw bodyPartError

    }

    var encoded = Data()

    bodyParts.first?.hasInitialBoundary = true

    bodyParts.last?.hasFinalBoundary = true

    for bodyPart in bodyParts {

        let encodedData = try encode(bodyPart)

        encoded.append(encodedData)

    }

    return encoded

}

```

```

/// Writes all appended body parts to the given file `URL`.

///

/// This process is facilitated by reading and writing with input and output streams,
respectively. Thus,

/// this approach is very memory efficient and should be used for large body part data.

///

/// - Parameter fileURL: File `URL` to which to write the form data.

/// - Throws:      An `AFError` if encoding encounters an error.

public func writeEncodedData(to fileURL: URL) throws {

    if let bodyPartError = bodyPartError {

        throw bodyPartError

    }

    if fileManager.fileExists(atPath: fileURL.path) {

        throw

        AFError.multipartEncodingFailed(reason: .outputStreamFileAlreadyExists(at: fileURL))

    } else if !fileURL.isFileURL {

```

```
        throw AFError.multipartEncodingFailed(reason: .outputStreamURLInvalid(url:
fileURL))
```

```
    }
```

```
    guard let outputStream = OutputStream(url: fileURL, append: false) else {
```

```
        throw AFError.multipartEncodingFailed(reason: .outputStreamCreationFailed(for:
fileURL))
```

```
    }
```

```
    outputStream.open()
```

```
    defer { outputStream.close() }
```

```
    bodyParts.first?.hasInitialBoundary = true
```

```
    bodyParts.last?.hasFinalBoundary = true
```

```
    for bodyPart in bodyParts {
```

```
        try write(bodyPart, to: outputStream)
```

```
    }
```

```
}
```

```
// MARK: - Private - Body Part Encoding
```

```
private func encode(_ bodyPart: BodyPart) throws -> Data {
```

```
    var encoded = Data()
```

```
    let initialData = bodyPart.hasInitialBoundary ? initialBoundaryData() :  
    encapsulatedBoundaryData()
```

```
    encoded.append(initialData)
```

```
    let headerData = encodeHeaders(for: bodyPart)
```

```
    encoded.append(headerData)
```

```
    let bodyStreamData = try encodeBodyStream(for: bodyPart)
```

```
    encoded.append(bodyStreamData)
```

```
    if bodyPart.hasFinalBoundary {
```

```

        encoded.append(finalBoundaryData())

    }

    return encoded

}

private fun encodeHeaders(for bodyPart: BodyPart) -> Data {

    let      headerText      =      bodyPart.headers.map      {      "\($0.name):
\($0.value)\(EncodingCharacters.crlf)" }

    .joined()

    + EncodingCharacters.crlf

    return Data(headerText.utf8)

}

private fun encodeBodyStream(for bodyPart: BodyPart) throws -> Data {

    let inputStream = bodyPart.bodyStream

    inputStream.open()

```

```

defer { inputStream.close() }

var encoded = Data()

while inputStream.hasBytesAvailable {

    var buffer = [UInt8](repeating: 0, count: streamBufferSize)

    let bytesRead = inputStream.read(&buffer, maxLength: streamBufferSize)

    if let error = inputStream.streamError {

        throw AFError.multipartEncodingFailed(reason: .inputStreamReadFailed(error:
error))

    }

    if bytesRead > 0 {

        encoded.append(buffer, count: bytesRead)

    } else {

        break

    }
}

```



```
}
```

```
return encoded
```

```
}
```

```
// MARK: - Private - Writing Body Part to Output Stream
```

```
private func write(_ bodyPart: BodyPart, to outputStream: OutputStream) throws {
```

```
    try writeInitialBoundaryData(for: bodyPart, to: outputStream)
```

```
    try writeHeaderData(for: bodyPart, to: outputStream)
```

```
    try writeBodyStream(for: bodyPart, to: outputStream)
```

```
    try writeFinalBoundaryData(for: bodyPart, to: outputStream)
```

```
}
```

```
private func writeInitialBoundaryData(for bodyPart: BodyPart, to outputStream:
OutputStream) throws {
```

```
    let initialData = bodyPart.hasInitialBoundary ? initialBoundaryData() :
encapsulatedBoundaryData()
```

```

        return try write(initialData, to: outputStream)

    }

    private func writeHeaderData(for bodyPart: BodyPart, to outputStream: OutputStream)
    throws {

        let headerData = encodeHeaders(for: bodyPart)

        return try write(headerData, to: outputStream)

    }

    private func writeBodyStream(for bodyPart: BodyPart, to outputStream: OutputStream)
    throws {

        let inputStream = bodyPart.bodyStream

        inputStream.open()

        defer { inputStream.close() }

        while inputStream.hasBytesAvailable {

            var buffer = [UInt8](repeating: 0, count: streamBufferSize)

```

```

let bytesRead = inputStream.read(&buffer, maxLength: streamBufferSize)

if let streamError = inputStream.streamError {

    throw AFError.multipartEncodingFailed(reason: .inputStreamReadFailed(error:
streamError))

}

if bytesRead > 0 {

    if buffer.count != bytesRead {

        buffer = Array(buffer[0..

```

```
private func writeFinalBoundaryData(for bodyPart: BodyPart, to outputStream:
OutputStream) throws {
```

```
    if bodyPart.hasFinalBoundary {
```

```
        return try write(finalBoundaryData(), to: outputStream)
```

```
    }
```

```
}
```

```
// MARK: - Private - Writing Buffered Data to Output Stream
```

```
private func write(_ data: Data, to outputStream: OutputStream) throws {
```

```
    var buffer = [UInt8](repeating: 0, count: data.count)
```

```
    data.copyBytes(to: &buffer, count: data.count)
```

```
    return try write(&buffer, to: outputStream)
```

```
}
```

```
private func write(_ buffer: inout [UInt8], to outputStream: OutputStream) throws {
```

```

var bytesToWrite = buffer.count

while bytesToWrite > 0, outputStream.hasSpaceAvailable {

    let bytesWritten = outputStream.write(buffer, maxLength: bytesToWrite)

    if let error = outputStream.streamError {

        throw
        NSError.multipartEncodingFailed(reason: .outputStreamWriteFailed(error: error))

    }

    bytesToWrite -= bytesWritten

    if bytesToWrite > 0 {

        buffer = Array(buffer[bytesWritten..

```

```
// MARK: - Private - Mime Type
```

```
private func mimeType(forPathExtension pathExtension: String) -> String {  
  
    if  
  
        let id = UTTypeCreatePreferredIdentifierForTag(kUTTagClassFilenameExtension,  
pathExtension as CFString, nil)?.takeRetainedValue(),  
  
        let contentType = UTTypeCopyPreferredTagWithClass(id,  
kUTTagClassMIMETYPE)?.takeRetainedValue() {  
  
            return contentType as String  
  
        }  
  
        return "application/octet-stream"  
  
    }
```

```
// MARK: - Private - Content Headers
```

```
private func contentHeaders(withName name: String, fileName: String? = nil,  
mimeType: String? = nil) -> HTTPHeaders {  
  
    var disposition = "form-data; name=\"\"(name)\"\""
```

```
if let fileName = fileName { disposition += "; filename=\"\"(fileName)\"\" } }
```

```
var headers: HTTPHeaders = [.contentDisposition(disposition)]
```

```
if let mimeType = mimeType { headers.add(.contentType(mimeType)) }
```

```
return headers
```

```
}
```

```
// MARK: - Private - Boundary Encoding
```

```
private func initialBoundaryData() -> Data {
```

```
    return BoundaryGenerator.boundaryData(forBoundaryType: .initial, boundary: boundary)
```

```
}
```

```
private func encapsulatedBoundaryData() -> Data {
```

```
    return BoundaryGenerator.boundaryData(forBoundaryType: .encapsulated, boundary: boundary)
```

```

    }

    private func finalBoundaryData() -> Data {

        return BoundaryGenerator.boundaryData(forBoundaryType: .final, boundary:
boundary)

    }

    // MARK: - Private - Errors

    private func setBodyPartError(withReason reason:
AFError.MultipartEncodingFailureReason) {

        guard bodyPartError == nil else { return }

        bodyPartError = AFError.multipartEncodingFailed(reason: reason)

    }

}

//

// MultipartUpload.swift

//

```


// Copyright (c) 2014-2018 Alamofire Software Foundation (<http://alamofire.org/>)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

import Foundation

/// Internal type which encapsulates a `MultipartFormData` upload.

final class MultipartUpload {

lazy var result = Result { try build() }

let isInBackgroundSession: Bool

let multipartFormData: MultipartFormData

let encodingMemoryThreshold: UInt64

let request: URLRequestConvertible

```
let fileManager: FileManager
```

```
init(isInBackgroundSession: Bool,
```

```
    encodingMemoryThreshold: UInt64,
```

```
    request: URLRequestConvertible,
```

```
    multipartFormData: MultipartFormData) {
```

```
    self.isInBackgroundSession = isInBackgroundSession
```

```
    self.encodingMemoryThreshold = encodingMemoryThreshold
```

```
    self.request = request
```

```
    fileManager = multipartFormData.fileManager
```

```
    self.multipartFormData = multipartFormData
```

```
}
```

```
func build() throws -> (request: URLRequest, uploadable: UploadRequest.Uploadable)  
{
```

```
    var urlRequest = try request.asURLRequest()
```

```
    urlRequest.setValue(multipartFormData.contentType, forHTTPHeaderField:  
        "Content-Type")
```

```

let uploadable: UploadRequest.Uploadable

if multipartFormData.contentLength < encodingMemoryThreshold
&& !isInBackgroundSession {

    let data = try multipartFormData.encode()

    uploadable = .data(data)

} else {

    let tempDirectoryURL = fileManager temporaryDirectory

    let directoryURL =
tempDirectoryURL.appendingPathComponent("org.alamofire.manager/multipart.form.da
ta")

    let fileName = UUID().uuidString

    let fileURL = directoryURL.appendingPathComponent(fileName)

    try fileManager.createDirectory(at: directoryURL, withIntermediateDirectories:
true, attributes: nil)

    do {

```

```

        try multipartFormData.writeEncodedData(to: fileURL)

    } catch {

        // Cleanup after attempted write if it fails.

        try? fileManager.removeItem(at: fileURL)

    }

    uploadable = .file(fileURL, shouldRemove: true)

}

return (request: urlRequest, uploadable: uploadable)

}

}

extension MultipartUpload: UploadConvertible {

    func asURLRequest() throws -> URLRequest {

        return try result.get().request

    }

}

```

```

func createUploadable() throws -> UploadRequest.Uploadable {

    return try result.get().uploadable

}

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `NSPredicate+Condition` properties and methods are used to parse the conditions used in `HMEventTrigger`'s.

```

*/

```

```

import HomeKit

```

```

/// Represents condition type in HomeKit with associated values.

```

```

enum HomeKitConditionType {

```

/**

Represents a characteristic condition.

The tuple represents the `HMCharacteristic` and its condition value.

For example, "Current gargage door is set to 'Open'".

*/

case characteristic(HMCharacteristic, NSCopying)

/**

Represents a time condition.

The tuple represents the time ordering and the sun state.

For example, "Before sunset".

*/

case sunTime(TimeConditionOrder, TimeConditionSunState)

/**

Represents an exact time condition.

The tuple represents the time ordering and time.

For example, "At 12:00pm".

```
*/
```

```
case exactTime(TimeConditionOrder, DateComponents)
```

```
/// The predicate is not a HomeKit condition.
```

```
case unknown
```

```
}
```

```
extension NSPredicate {
```

```
/**
```

```
    Parses the predicate and attempts to generate a characteristic-value  
    `HomeKitConditionType`.
```

```
- returns: An optional characteristic-value tuple.
```


*/

```
private func characteristic() -> HomeKitConditionType? {
```

```
    guard let predicate = self as? NSCompoundPredicate else { return nil }
```

```
    guard let subpredicates = predicate.subpredicates as? [NSPredicate] else { return nil }
```

```
    guard subpredicates.count == 2 else { return nil }
```

```
    var characteristicPredicate: NSComparisonPredicate? = nil
```

```
    var valuePredicate: NSComparisonPredicate? = nil
```

```
    for subpredicate in subpredicates {
```

```
        if let comparison = subpredicate as? NSComparisonPredicate,
           comparison.leftExpression.expressionType == .keyPath &&
           comparison.rightExpression.expressionType == .constantValue {
```

```
            switch comparison.leftExpression.keyPath {
```

```
                case HMCharacteristicKeyPath:
```

```
                    characteristicPredicate = comparison
```

```
                case HMCharacteristicValueKeyPath:
```

```
valuePredicate = comparison
```

```
default:
```

```
break
```

```
}
```

```
}
```

```
}
```

```
if let characteristic = characteristicPredicate?.rightExpression.constantValue as?  
HMCharacteristic,
```

```
let characteristicValue = valuePredicate?.rightExpression.constantValue as?  
NSCopying {
```

```
return .characteristic(characteristic, characteristicValue)
```

```
}
```

```
return nil
```

```
}
```

```
/**
```

Parses the predicate and attempts to generate an order-sunstate
`HomeKitConditionType`.

- returns: An optional order-sunstate tuple.

*/

```
private func sunState() -> HomeKitConditionType? {
```

```
    guard let comparison = self as? NSComparisonPredicate else { return nil }
```

```
    guard comparison.leftExpression.expressionType == .keyPath else { return nil }
```

```
    guard comparison.rightExpression.expressionType == .function else { return nil }
```

```
    guard comparison.rightExpression.function == "now" else { return nil }
```

```
    guard comparison.rightExpression.arguments?.count == 0 else { return nil }
```

```
    switch (comparison.leftExpression.keyPath, comparison.predicateOperatorType) {
```

```
        case (HMSignificantEvent.sunrise.rawValue, .lessThan):
```

```
            return .sunTime(.after, .sunrise)
```

```
        case (HMSignificantEvent.sunrise.rawValue, .lessThanOrEqualTo):
```

```
            return .sunTime(.after, .sunrise)
```

case (HMSignificantEvent.sunrise.rawValue, .greaterThan):

return .sunTime(.before, .sunrise)

case (HMSignificantEvent.sunrise.rawValue, .greaterThanOrEqualTo):

return .sunTime(.before, .sunrise)

case (HMSignificantEvent.sunset.rawValue, .lessThan):

return .sunTime(.after, .sunset)

case (HMSignificantEvent.sunset.rawValue, .lessThanOrEqualTo):

return .sunTime(.after, .sunset)

case (HMSignificantEvent.sunset.rawValue, .greaterThan):

return .sunTime(.before, .sunset)

case (HMSignificantEvent.sunset.rawValue, .greaterThanOrEqualTo):

```
return .sunTime(.before, .sunset)
```

```
default:
```

```
return nil
```

```
}
```

```
}
```

```
/**
```

```
    Parses the predicate and attempts to generate an order-exacttime  
    `HomeKitConditionType`.
```

```
- returns: An optional order-exacttime tuple.
```

```
*/
```

```
private func exactTime() -> HomeKitConditionType? {
```

```
    guard let comparison = self as? NSComparisonPredicate else { return nil }
```

```
    guard comparison.leftExpression.expressionType == .function else { return nil }
```

```
    guard comparison.leftExpression.function == "now" else { return nil }
```

```
guard comparison.rightExpression.expressionType == .constantValue else { return  
nil }
```

```
guard let dateComponents = comparison.rightExpression.constantValue as?  
DateComponents else { return nil }
```

```
switch comparison.predicateOperatorType {
```

```
case .lessThan, .lessThanOrEqualTo:
```

```
    return .exactTime(.before, dateComponents)
```

```
case .greaterThan, .greaterThanOrEqualTo:
```

```
    return .exactTime(.after, dateComponents)
```

```
case .equalTo:
```

```
    return .exactTime(.at, dateComponents)
```

```
default:
```

```
    return nil
```

```
}
```

```
}
```

/// - returns: The 'type' of HomeKit condition, with associated value, if applicable.

```
var homeKitConditionType: HomeKitConditionType {
```

```
    if let characteristic = characteristic() {
```

```
        return characteristic
```

```
    }
```

```
    else if let sunState = sunState() {
```

```
        return sunState
```

```
    }
```

```
    else if let exactTime = exactTime() {
```

```
        return exactTime
```

```
    }
```

```
    else {
```

```
        return .unknown
```

```
    }
```

```
}
```

```
}
```

```
/*
```

See LICENSE folder for this sample's licensing information.

Abstract:

Enables editing a name field.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
class NameEditor: UITableViewController {
```

```
    @IBOutlet weak var nameField: UITextField!
```

```
    var service: HMService?
```



```

override func viewDidLoad() {

    super.viewDidLoad()

    nameField.text = service?.name

}

override func viewWillAppear(_ animated: Bool) {

    super.viewWillAppear(animated)

    if let name = nameField.text,

        let service = service {

        HomeStore.shared.updateService(service, name: name)

    }

}

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

Enables editing a name field.

*/

import UIKit

import HomeKit

class NameEditor: UITableViewController {

 @IBOutlet weak var nameField: UITextField!

 var service: HMService?

 override func viewDidLoad() {

 super.viewDidLoad()

 nameField.text = service?.name

 }

```

override func viewWillDisappear(_ animated: Bool) {

    super.viewWillDisappear(animated)

    if let name = nameField.text,

        let service = service {

            HomeStore.shared.updateService(service, name: name)

        }

    }

}

//

// NetworkReachabilityManager.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

```

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

```
// THE SOFTWARE.
```

```
//
```

```
#if !(os(watchOS) || os(Linux))
```

```
import Foundation
```

```
import SystemConfiguration
```

```
/// The `NetworkReachabilityManager` class listens for reachability changes of hosts and  
addresses for both cellular and
```

```
/// WiFi network interfaces.
```

```
///
```

```
/// Reachability can be used to determine background information about why a network  
operation failed, or to retry
```

```
/// network requests when a connection is established. It should not be used to prevent a  
user from initiating a network
```

```
/// request, as it's possible that an initial request may be required to establish reachability.
```

```
open class NetworkReachabilityManager {
```

```
    /// Defines the various states of network reachability.
```

```

public enum NetworkReachabilityStatus {

    /// It is unknown whether the network is reachable.

    case unknown

    /// The network is not reachable.

    case notReachable

    /// The network is reachable on the associated `ConnectionType`.

    case reachable(ConnectionType)

    init(_ flags: SCNetworkReachabilityFlags) {

        guard flags.isActuallyReachable else { self = .notReachable; return }

        var networkStatus: NetworkReachabilityStatus = .reachable(.ethernetOrWiFi)

        if flags.isCellular { networkStatus = .reachable(.cellular) }

        self = networkStatus

    }

```

/// Defines the various connection types detected by reachability flags.

```
public enum ConnectionType {
```

```
    /// The connection type is either over Ethernet or WiFi.
```

```
    case ethernetOrWiFi
```

```
    /// The connection type is a cellular connection.
```

```
    case cellular
```

```
}
```

```
}
```

/// A closure executed when the network reachability status changes. The closure takes a single argument: the

/// network reachability status.

```
public typealias Listener = (NetworkReachabilityStatus) -> Void
```

/// Default `NetworkReachabilityManager` for the zero address and a `listenerQueue` of `.main`.

```
public static let `default` = NetworkReachabilityManager()
```

```
// MARK: - Properties
```

```
/// Whether the network is currently reachable.
```

```
open var isReachable: Bool { return isReachableOnCellular ||  
isReachableOnEthernetOrWiFi }
```

```
/// Whether the network is currently reachable over the cellular interface.
```

```
///
```

```
/// - Note: Using this property to decide whether to make a high or low bandwidth request  
is not recommended.
```

```
/// Instead, set the `allowsCellularAccess` on any `URLRequest`'s being issued.
```

```
///
```

```
open var isReachableOnCellular: Bool { return status == .reachable(.cellular) }
```

```
/// Whether the network is currently reachable over Ethernet or WiFi interface.
```

```
open var isReachableOnEthernetOrWiFi: Bool { return status  
== .reachable(.ethernetOrWiFi) }
```



```

/// `DispatchQueue` on which reachability will update.

public let reachabilityQueue = DispatchQueue(label: "org.alamofire.reachabilityQueue")

/// Flags of the current reachability type, if any.

open var flags: SCNetworkReachabilityFlags? {

    var flags = SCNetworkReachabilityFlags()

    return (SCNetworkReachabilityGetFlags(reachability, &flags)) ? flags : nil

}

/// The current network reachability status.

open var status: NetworkReachabilityStatus {

    return flags.map(NetworkReachabilityStatus.init) ?? .unknown

}

/// Mutable state storage.

struct MutableState {

```

```

    /// A closure executed when the network reachability status changes.

    var listener: Listener?

    /// `DispatchQueue` on which listeners will be called.

    var listenerQueue: DispatchQueue?

    /// Previously calculated status.

    var previousStatus: NetworkReachabilityStatus?

}

/// `SCNetworkReachability` instance providing notifications.

private let reachability: SCNetworkReachability

/// Protected storage for mutable state.

private let mutableState = Protector(MutableState())

// MARK: - Initialization

/// Creates an instance with the specified host.

```

```

///

/// - Note: The `host` value must *not* contain a scheme, just the hostname.

///

/// - Parameters:

///   - host:      Host used to evaluate network reachability. Must *not* include the
scheme (e.g. `https`).

public convenience init?(host: String) {

    guard let reachability = SCNetworkReachabilityCreateWithName(nil, host) else
{ return nil }

    self.init(reachability: reachability)

}

/// Creates an instance that monitors the address 0.0.0.0.

///

/// Reachability treats the 0.0.0.0 address as a special token that causes it to monitor the
general routing

/// status of the device, both IPv4 and IPv6.

public convenience init?() {

```

```

var zero = sockaddr()

zero.sa_len = UInt8(MemoryLayout<sockaddr>.size)

zero.sa_family = sa_family_t(AF_INET)


guard let reachability = SCNetworkReachabilityCreateWithAddress(nil, &zero) else
{ return nil }


self.init(reachability: reachability)

}


private init(reachability: SCNetworkReachability) {

    self.reachability = reachability

}


deinit {

    stopListening()

}

```

```
// MARK: - Listening
```

```
/// Starts listening for changes in network reachability status.
```

```
///
```

```
/// - Note: Stops and removes any existing listener.
```

```
///
```

```
/// - Parameters:
```

```
/// - queue: `DispatchQueue` on which to call the `listener` closure. `.main` by default.
```

```
/// - listener: `Listener` closure called when reachability changes.
```

```
///
```

```
/// - Returns: `true` if listening was started successfully, `false` otherwise.
```

```
@discardableResult
```

```
open func startListening(onQueue queue: DispatchQueue = .main,
```

```
                        onUpdatePerforming listener: @escaping Listener) -> Bool {
```

```
    stopListening()
```

```
    mutableState.write { state in
```

```

state.listenerQueue = queue

state.listener = listener

}

```

```

var context = SCNetworkReachabilityContext(version: 0,

                                           info: Unmanaged.passRetained(self).toOpaque(),

                                           retain: nil,

                                           release: nil,

                                           copyDescription: nil)

```

```

let callback: SCNetworkReachabilityCallBack = { _, flags, info in

    guard let info = info else { return }

```

```

        let instance =
Unmanaged<NetworkReachabilityManager>.fromOpaque(info).takeUnretainedValue()

        instance.notifyListener(flags)

    }

```

```
        let queueAdded = SCNetworkReachabilitySetDispatchQueue(reachability,
reachabilityQueue)
```

```
        let callbackAdded = SCNetworkReachabilitySetCallback(reachability, callback,
&context)
```

```
// Manually call listener to give initial state, since the framework may not.
```

```
if let currentFlags = flags {
```

```
    reachabilityQueue.async {
```

```
        self.notifyListener(currentFlags)
```

```
    }
```

```
}
```

```
return callbackAdded && queueAdded
```

```
}
```

```
/// Stops listening for changes in network reachability status.
```

```
open func stopListening() {
```

```
    SCNetworkReachabilitySetCallback(reachability, nil, nil)
```

```

SCNetworkReachabilitySetDispatchQueue(reachability, nil)

mutableState.write { state in

    state.listener = nil

    state.listenerQueue = nil

    state.previousStatus = nil

}

}

// MARK: - Internal - Listener Notification

/// Calls the `listener` closure of the `listenerQueue` if the computed status hasn't changed.

///

/// - Note: Should only be called from the `reachabilityQueue`.

///

/// - Parameter flags: `SCNetworkReachabilityFlags` to use to calculate the status.

func notifyListener(_ flags: SCNetworkReachabilityFlags) {

    let newStatus = NetworkReachabilityStatus(flags)

```



```

mutableState.write { state in

    guard state.previousStatus != newStatus else { return }

    state.previousStatus = newStatus

    let listener = state.listener

    state.listenerQueue?.async { listener?(newStatus) }

}

}

}

```

// MARK: -

```

extension NetworkReachabilityManager.NetworkReachabilityStatus: Equatable {}

```

```

extension SCNetworkReachabilityFlags {

```

```

var isReachable: Bool { return contains(.reachable) }

var isConnectedRequired: Bool { return contains(.connectionRequired) }

var canConnectAutomatically: Bool { return contains(.connectionOnDemand) ||
contains(.connectionOnTraffic) }

var canConnectWithoutUserInteraction: Bool { return canConnectAutomatically
&& !contains(.interventionRequired) }

var isActuallyReachable: Bool { return isReachable && (!isConnectionRequired ||
canConnectWithoutUserInteraction) }

var isCellular: Bool {

    #if os(iOS) || os(tvOS)

    return contains(.isWWAN)

    #else

    return false

    #endif

}

/// Human readable `String` for all states, to help with debugging.

var readableDescription: String {

    let W = isCellular ? "W" : "-"

```

```

let R = isReachable ? "R" : "-"

let c = isConnectionRequired ? "c" : "-"

let t = contains(.transientConnection) ? "t" : "-"

let i = contains(.interventionRequired) ? "i" : "-"

let C = contains(.connectionOnTraffic) ? "C" : "-"

let D = contains(.connectionOnDemand) ? "D" : "-"

let l = contains(.isLocalAddress) ? "l" : "-"

let d = contains(.isDirect) ? "d" : "-"

let a = contains(.connectionAutomatic) ? "a" : "-"


return "\ (W)\ (R) \ (c)\ (t)\ (i)\ (C)\ (D)\ (l)\ (d)\ (a)"

}

}

#endif

//

// Notifications.swift

//

```

// Copyright (c) 2014-2018 Alamofire Software Foundation (<http://alamofire.org/>)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

```
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,  
DAMAGES OR OTHER
```

```
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,  
ARISING FROM,
```

```
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER  
DEALINGS IN
```

```
// THE SOFTWARE.
```

```
//
```

```
import Foundation
```

```
public extension Request {
```

```
    /// Posted when a `Request` is resumed. The `Notification` contains the resumed  
    `Request`.
```

```
    static let didResumeNotification = Notification.Name(rawValue:  
"org.alamofire.notification.name.request.didResume")
```

```
    /// Posted when a `Request` is suspended. The `Notification` contains the suspended  
    `Request`.
```

```
    static let didSuspendNotification = Notification.Name(rawValue:  
"org.alamofire.notification.name.request.didSuspend")
```

/// Posted when a `Request` is cancelled. The `Notification` contains the cancelled `Request`.

```
static let didCancelNotification = Notification.Name(rawValue:
"org.alamofire.notification.name.request.didCancel")
```

/// Posted when a `Request` is finished. The `Notification` contains the completed `Request`.

```
static let didFinishNotification = Notification.Name(rawValue:
"org.alamofire.notification.name.request.didFinish")
```

/// Posted when a `URLSessionTask` is resumed. The `Notification` contains the `Request` associated with the `URLSessionTask`.

```
static let didResumeTaskNotification = Notification.Name(rawValue:
"org.alamofire.notification.name.request.didResumeTask")
```

/// Posted when a `URLSessionTask` is suspended. The `Notification` contains the `Request` associated with the `URLSessionTask`.

```
static let didSuspendTaskNotification = Notification.Name(rawValue:
"org.alamofire.notification.name.request.didSuspendTask")
```

/// Posted when a `URLSessionTask` is cancelled. The `Notification` contains the `Request` associated with the `URLSessionTask`.

```
static let didCancelTaskNotification = Notification.Name(rawValue:
"org.alamofire.notification.name.request.didCancelTask")
```

/// Posted when a `URLSessionTask` is completed. The `Notification` contains the `Request` associated with the `URLSessionTask`.

```

        static let didCompleteTaskNotification = Notification.Name(rawValue:
"org.alamofire.notification.name.request.didCompleteTask")

    }

```

// MARK: -

```

extension Notification {

```

```

    /// The `Request` contained by the instance's `userInfo`, `nil` otherwise.

```

```

    public var request: Request? {

```

```

        return userInfo?[String.requestKey] as? Request

```

```

    }

```

```

    /// Convenience initializer for a `Notification` containing a `Request` payload.

```

```

    ///

```

```

    /// - Parameters:

```

```

    /// - name: The name of the notification.

```

```

    /// - request: The `Request` payload.

```

```

    init(name: Notification.Name, request: Request) {

```

```

        self.init(name: name, object: nil, userInfo: [String.requestKey: request])

    }

}

extension NotificationCenter {

    /// Convenience function for posting notifications with `Request` payloads.

    ///

    /// - Parameters:

    ///   - name: The name of the notification.

    ///   - request: The `Request` payload.

    func postNotification(named name: Notification.Name, with request: Request) {

        let notification = Notification(name: name, request: request)

        post(notification)

    }

}

```

```

extension String {

```



```

    /// User info dictionary key representing the `Request` associated with the notification.

    fileprivate static let requestKey = "org.alamofire.notification.key.request"

}

/// `EventManager` that provides Alamofire's notifications.

public final class AlamofireNotifications: EventMonitor {

    public func requestDidResume(_ request: Request) {

        NotificationCenter.default.postNotification(named: Request.didResumeNotification,
        with: request)

    }

    public func requestDidSuspend(_ request: Request) {

        NotificationCenter.default.postNotification(named: Request.didSuspendNotification,
        with: request)

    }

    public func requestDidCancel(_ request: Request) {

        NotificationCenter.default.postNotification(named: Request.didCancelNotification,
        with: request)
    }
}

```

```
}
```

```
public func requestDidFinish(_ request: Request) {
```

```
    NotificationCenter.default.postNotification(named: Request.didFinishNotification,  
with: request)
```

```
}
```

```
public func request(_ request: Request, didResumeTask task: URLSessionTask) {
```

```
    NotificationCenter.default.postNotification(named:  
Request.didResumeTaskNotification, with: request)
```

```
}
```

```
public func request(_ request: Request, didSuspendTask task: URLSessionTask) {
```

```
    NotificationCenter.default.postNotification(named:  
Request.didSuspendTaskNotification, with: request)
```

```
}
```

```
public func request(_ request: Request, didCancelTask task: URLSessionTask) {
```

```
        NotificationCenter.default.postNotification(named:
Request.didCancelTaskNotification, with: request)

    }
```

```
    public func request(_ request: Request, didCompleteTask task: URLSessionTask, with
error: NSError?) {
```

```
        NotificationCenter.default.postNotification(named:
Request.didCompleteTaskNotification, with: request)

    }
```

```
}
```

```
//
```

```
// OperationQueue+Alamofire.swift
```

```
//
```

```
// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)
```

```
//
```

```
// Permission is hereby granted, free of charge, to any person obtaining a copy
```

```
// of this software and associated documentation files (the "Software"), to deal
```

```
// in the Software without restriction, including without limitation the rights
```

```
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
```

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

```
import Foundation
```

```
extension OperationQueue {
```

```
    /// Creates an instance using the provided parameters.
```

```
    ///
```

```
    /// - Parameters:
```

```
    /// - qualityOfService:      `QualityOfService` to be applied to the queue. `.default`  
    by default.
```

```
    /// - maxConcurrentOperationCount: Maximum concurrent operations.
```

```
    ///                                `OperationQueue.defaultMaxConcurrentOperationCount` by  
    default.
```

```
    /// - underlyingQueue: Underlying `DispatchQueue`. `nil` by default.
```

```
    /// - name:                  Name for the queue. `nil` by default.
```

```
    /// - startSuspended:       Whether the queue starts suspended. `false` by default.
```

```
    convenience init(qualityOfService: QualityOfService = .default,
```

```
                     maxConcurrentOperationCount: Int =  
    OperationQueue.defaultMaxConcurrentOperationCount,
```

```
                     underlyingQueue: DispatchQueue? = nil,
```

```

        name: String? = nil,

        startSuspended: Bool = false) {

    self.init()

    self.qualityOfService = qualityOfService

    self.maxConcurrentOperationCount = maxConcurrentOperationCount

    self.underlyingQueue = underlyingQueue

    self.name = name

    isSuspended = startSuspended

}

}

//

// ParameterEncoder.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

```

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

```
// THE SOFTWARE.
```

```
//
```

```
import Foundation
```

```
/// A type that can encode any `Encodable` type into a `URLRequest`.
```

```
public protocol ParameterEncoder {
```

```
    /// Encode the provided `Encodable` parameters into `request`.
```

```
    ///
```

```
    /// - Parameters:
```

```
    /// - parameters: The `Encodable` parameter value.
```

```
    /// - request: The `URLRequest` into which to encode the parameters.
```

```
    ///
```

```
    /// - Returns: A `URLRequest` with the result of the encoding.
```

```
    /// - Throws: An `Error` when encoding fails. For Alamofire provided encoders, this  
will be an instance of
```

```
    ///                                     `AFError.parameterEncoderFailed` with an associated  
`ParameterEncoderFailureReason`.
```



```

    func encode<Parameters: Encodable>(_ parameters: Parameters?, into request:
URLRequest) throws -> URLRequest

}

```

```

/// A `ParameterEncoder` that encodes types as JSON body data.

```

```

///

```

```

/// If no `Content-Type` header is already set on the provided `URLRequest`s, it's set to
`application/json`.

```

```

open class JSONParameterEncoder: ParameterEncoder {

```

```

    /// Returns an encoder with default parameters.

```

```

    public static var `default`: JSONParameterEncoder { return JSONParameterEncoder() }

```

```

    /// Returns an encoder with `JSONEncoder.outputFormatting` set to `.prettyPrinted`.

```

```

    public static var prettyPrinted: JSONParameterEncoder {

```

```

        let encoder = JSONEncoder()

```

```

        encoder.outputFormatting = .prettyPrinted

```

```

        return JSONParameterEncoder(encoder: encoder)
    }

```

```
}
```

```
/// Returns an encoder with `JSONEncoder.outputFormatting` set to `.sortedKeys`.
```

```
@available(macOS 10.13, iOS 11.0, tvOS 11.0, watchOS 4.0, *)
```

```
public static var sortedKeys: JSONParameterEncoder {
```

```
    let encoder = JSONEncoder()
```

```
    encoder.outputFormatting = .sortedKeys
```

```
    return JSONParameterEncoder(encoder: encoder)
```

```
}
```

```
/// `JSONEncoder` used to encode parameters.
```

```
public let encoder: JSONEncoder
```

```
/// Creates an instance with the provided `JSONEncoder`.
```

```
///
```

```
/// - Parameter encoder: The `JSONEncoder`. `JSONEncoder()` by default.
```

```

public init(encoder: JSONEncoder = JSONEncoder()) {

    self.encoder = encoder

}

open func encode<Parameters: Encodable>(_ parameters: Parameters?,

                                     into request: URLRequest) throws -> URLRequest {

    guard let parameters = parameters else { return request }

    var request = request

    do {

        let data = try encoder.encode(parameters)

        request.httpBody = data

        if request.headers["Content-Type"] == nil {

            request.headers.update(.contentType("application/json"))

        }

    } catch {

```

```

        throw NSError.parameterEncodingFailed(reason: NSJSONEncodingFailed(error:
error))

    }

```

```

        return request

    }

}

```

/// A `ParameterEncoder` that encodes types as URL-encoded query strings to be set on the URL or as body data, depending

/// on the `Destination` set.

///

/// If no `Content-Type` header is already set on the provided `URLRequest`'s, it will be set to

/// `application/x-www-form-urlencoded; charset=utf-8`.

///

/// Encoding behavior can be customized by passing an instance of `URLEncodedFormEncoder` to the initializer.

```

open class URLEncodedFormParameterEncoder: ParameterEncoder {

```

```

/// Defines where the URL-encoded string should be set for each `URLRequest`.

public enum Destination {

    /// Applies the encoded query string to any existing query string for `.get`, `.head`, and
    /// `.delete` request.

    /// Sets it to the `httpBody` for all other methods.

    case methodDependent

    /// Applies the encoded query string to any existing query string from the
    /// `URLRequest`.

    case queryString

    /// Applies the encoded query string to the `httpBody` of the `URLRequest`.

    case httpBody

    /// Determines whether the URL-encoded string should be applied to the
    /// `URLRequest`'s `url`.

    ///

    /// - Parameter method: The `HTTPMethod`.

    ///

    /// - Returns: Whether the URL-encoded string should be applied to a `URL`.

    func encodesParametersInURL(for method: HTTPMethod) -> Bool {

```

```

switch self {

case .methodDependent: return [.get, .head, .delete].contains(method)

case .queryString: return true

case .httpBody: return false

}

}

}

```

```

/// Returns an encoder with default parameters.

```

```

public static var `default`: URLEncodedFormParameterEncoder { return
URLEncodedFormParameterEncoder() }

```

```

/// The `URLEncodedFormEncoder` to use.

```

```

public let encoder: URLEncodedFormEncoder

```

```

/// The `Destination` for the URL-encoded string.

```

```

public let destination: Destination

```

```
/// Creates an instance with the provided `URLEncodedFormEncoder` instance and  
`Destination` value.
```

```
///
```

```
/// - Parameters:
```

```
/// - encoder: The `URLEncodedFormEncoder`. `URLEncodedFormEncoder()` by  
default.
```

```
/// - destination: The `Destination`. `.methodDependent` by default.
```

```
public init(encoder: URLEncodedFormEncoder = URLEncodedFormEncoder(),  
destination: Destination = .methodDependent) {
```

```
    self.encoder = encoder
```

```
    self.destination = destination
```

```
}
```

```
open func encode<Parameters: Encodable>(_ parameters: Parameters?,
```

```
    into request: URLRequest) throws -> URLRequest {
```

```
    guard let parameters = parameters else { return request }
```

```
    var request = request
```

```

guard let url = request.url else {

    throw
    NSError.parameterEncoderFailed(reason: .missingRequiredComponent(.url))

}

guard let method = request.method else {

    let rawValue = request.method?.rawValue ?? "nil"

    throw
    NSError.parameterEncoderFailed(reason: .missingRequiredComponent(.httpMethod(rawValue: rawValue)))

}

if destination.encodesParametersInURL(for: method),

    var components = URLComponents(url: url, resolvingAgainstBaseURL: false) {

        let query: String = try Result<String, Error> { try encoder.encode(parameters) }

        .mapError { NSError.parameterEncoderFailed(reason: .encoderFailed(error: $0)) }.get()

        let newQueryString = [components.percentEncodedQuery, query].compactMap
        { $0 }.joinedWithAmpersands()

```



```
        components.percentEncodedQuery = newQueryString.isEmpty ? nil :
newQueryString
```

```
        guard let newURL = components.url else {

            throw
            AFError.parameterEncoderFailed(reason: .missingRequiredComponent(.url))

        }
```

```
        request.url = newURL
```

```
    } else {
```

```
        if request.headers["Content-Type"] == nil {
```

```
            request.headers.update(.contentType("application/x-www-form-urlencoded;
charset=utf-8"))

        }
```

```
        request.httpBody = try Result<Data, Error> { try encoder.encode(parameters) }
```

```
        .mapError { AFError.parameterEncoderFailed(reason: .encoderFailed(error:
$0)) }.get()

    }
```

```
        return request

    }

}

//

// ParameterEncoding.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in
```

```
// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
// EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
// MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
// EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
// ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
// DEALINGS IN

// THE SOFTWARE.

//
```

```
import Foundation
```

```
/// A dictionary of parameters to apply to a `URLRequest`.
```

```
public typealias Parameters = [String: Any]
```

```
/// A type used to define how a set of parameters are applied to a `URLRequest`.
```

```
public protocol ParameterEncoding {
```

```
    /// Creates a `URLRequest` by encoding parameters and applying them on the passed request.
```

```
    ///
```

```
    /// - Parameters:
```

```
    ///   - URLRequest: `URLRequestConvertible` value onto which parameters will be encoded.
```

```
    ///   - parameters: `Parameters` to encode onto the request.
```

```
    ///
```

```
    /// - Returns: The encoded `URLRequest`.
```

```
    /// - Throws: Any `Error` produced during parameter encoding.
```

```
    func encode(_ URLRequest: URLRequestConvertible, with parameters: Parameters?) throws -> URLRequest
}
```

```
// MARK: -
```

/// Creates a url-encoded query string to be set as or appended to any existing URL query string or set as the HTTP

/// body of the URL request. Whether the query string is set or appended to any existing URL query string or set as

/// the HTTP body depends on the destination of the encoding.

///

/// The `Content-Type` HTTP header field of an encoded request with HTTP body is set to

/// `application/x-www-form-urlencoded; charset=utf-8`.

///

/// There is no published specification for how to encode collection types. By default the convention of appending

/// `[ ]` to the key for array values (`foo[]=1&foo[]=2`), and appending the key surrounded by square brackets for

/// nested dictionary values (`foo[bar]=baz`) is used. Optionally, `ArrayEncoding` can be used to omit the

/// square brackets appended to array keys.

///

/// `BoolEncoding` can be used to configure how boolean values are encoded. The default behavior is to encode

```

/// `true` as 1 and `false` as 0.

public struct URLEncoding: ParameterEncoding {

    // MARK: Helper Types

    /// Defines whether the url-encoded query string is applied to the existing query string
    or HTTP body of the

    /// resulting URL request.

    public enum Destination {

        /// Applies encoded query string result to existing query string for `GET`, `HEAD`
        and `DELETE` requests and

        /// sets as the HTTP body for requests with any other HTTP method.

        case methodDependent

        /// Sets or appends encoded query string result to existing query string.

        case queryString

        /// Sets encoded query string result as the HTTP body of the URL request.

        case httpBody

    }

    func encodesParametersInURL(for method: HTTPMethod) -> Bool {

```

```

switch self {

case .methodDependent: return [.get, .head, .delete].contains(method)

case .queryString: return true

case .httpBody: return false

}

}

}

```

/// Configures how `Array` parameters are encoded.

```
public enum ArrayEncoding {
```

/// An empty set of square brackets is appended to the key for every value. This is the default behavior.

```
case brackets
```

/// No brackets are appended. The key is encoded as is.

```
case noBrackets
```

```
func encode(key: String) -> String {
```

```
switch self {
```

```

    case .brackets:

        return "\\(key)[]"

    case .noBrackets:

        return key

    }

}

}

```

/// Configures how `Bool` parameters are encoded.

```
public enum BoolEncoding {
```

```
    /// Encode `true` as `1` and `false` as `0`. This is the default behavior.
```

```
    case numeric
```

```
    /// Encode `true` and `false` as string literals.
```

```
    case literal
```

```
func encode(value: Bool) -> String {
```

```
    switch self {
```



```
case .numeric:
```

```
    return value ? "1" : "0"
```

```
case .literal:
```

```
    return value ? "true" : "false"
```

```
    }
```

```
  }
```

```
}
```

```
// MARK: Properties
```

```
/// Returns a default `URLEncoding` instance with a `.methodDependent` destination.
```

```
public static var `default`: URLEncoding { return URLEncoding() }
```

```
/// Returns a `URLEncoding` instance with a `.queryString` destination.
```

```
public static var queryString: URLEncoding { return  
URLEncoding(destination: .queryString) }
```

```
/// Returns a `URLEncoding` instance with an `.httpBody` destination.
```

```
public static var httpBody: URLEncoding { return
URLEncoding(destination: .httpBody) }
```

```
/// The destination defining where the encoded query string is to be applied to the URL
request.
```

```
public let destination: Destination
```

```
/// The encoding to use for `Array` parameters.
```

```
public let arrayEncoding: ArrayEncoding
```

```
/// The encoding to use for `Bool` parameters.
```

```
public let boolEncoding: BoolEncoding
```

```
// MARK: Initialization
```

```
/// Creates an instance using the specified parameters.
```

```
///
```

```
/// - Parameters:
```

```
/// - destination: `Destination` defining where the encoded query string will be applied.  
`.methodDependent` by
```

```
///          default.
```

```
/// - arrayEncoding: `ArrayEncoding` to use. `.brackets` by default.
```

```
/// - boolEncoding: `BoolEncoding` to use. `.numeric` by default.
```

```
public init(destination: Destination = .methodDependent,
```

```
            arrayEncoding: ArrayEncoding = .brackets,
```

```
            boolEncoding: BoolEncoding = .numeric) {
```

```
    self.destination = destination
```

```
    self.arrayEncoding = arrayEncoding
```

```
    self.boolEncoding = boolEncoding
```

```
}
```

```
// MARK: Encoding
```

```
public func encode(_ urlRequest: URLRequestConvertible, with parameters:  
Parameters?) throws -> URLRequest {
```

```
    var urlRequest = try urlRequest.asURLRequest()
```

```

guard let parameters = parameters else { return urlRequest }

if let method = urlRequest.method, destination.encodesParametersInURL(for:
method) {

    guard let url = urlRequest.url else {

        throw AFError.parameterEncodingFailed(reason: .missingURL)

    }

    if var urlComponents = URLComponents(url: url, resolvingAgainstBaseURL:
false), !parameters.isEmpty {

        let percentEncodedQuery = (urlComponents.percentEncodedQuery.map { $0 +
"&" } ?? "") + query(parameters)

        urlComponents.percentEncodedQuery = percentEncodedQuery

        urlRequest.url = urlComponents.url

    }

} else {

    if urlRequest.value(forHTTPHeaderField: "Content-Type") == nil {

```

```
        urlRequest.setValue("application/x-www-form-urlencoded; charset=utf-8",  
forHTTPHeaderField: "Content-Type")
```

```
    }
```

```
    urlRequest.httpBody = Data(query(parameters).utf8)
```

```
    }
```

```
    return urlRequest
```

```
}
```

/// Creates a percent-escaped, URL encoded query string components from the given key-value pair recursively.

///

/// - Parameters:

/// - key: Key of the query component.

/// - value: Value of the query component.

///

/// - Returns: The percent-escaped, URL encoded query string components.

```

public func queryComponents(fromKey key: String, value: Any) -> [(String, String)] {

    var components: [(String, String)] = []

    if let dictionary = value as? [String: Any] {

        for (nestedKey, value) in dictionary {

            components += queryComponents(fromKey: "\\(key)[\\(nestedKey)]", value:
value)

        }

    } else if let array = value as? [Any] {

        for value in array {

            components += queryComponents(fromKey: arrayEncoding.encode(key: key),
value: value)

        }

    } else if let value = value as? NSNumber {

        if value.isBool {

            components.append((escape(key),          escape(boolEncoding.encode(value:
value.boolValue))))

        } else {

            components.append((escape(key), escape("\\(value)")))

        }

    }

```

```

    }

    } else if let bool = value as? Bool {

        components.append((escape(key), escape(boolEncoding.encode(value: bool))))

    } else {

        components.append((escape(key), escape("\(value)")))

    }

    return components

}

```

/// Creates a percent-escaped string following RFC 3986 for a query string key or value.

///

/// - Parameter string: `String` to be percent-escaped.

///

/// - Returns: The percent-escaped `String`.

```

public func escape(_ string: String) -> String {

    return
    string.addingPercentEncoding(withAllowedCharacters: .afURLQueryAllowed) ?? string
}

```

```

    }

    private fun query(_ parameters: [String: Any]) -> String {

        var components: [(String, String)] = []

        for key in parameters.keys.sorted(by: <) {

            let value = parameters[key]!

            components += queryComponents(fromKey: key, value: value)

        }

        return components.map { "\($0)=\($1)" }.joined(separator: "&")

    }

}

```

// MARK: -

/// Uses `JSONSerialization` to create a JSON representation of the parameters object,
which is set as the body of the

/// request. The `Content-Type` HTTP header field of an encoded request is set to `application/json`.

```
public struct JSONEncoding: ParameterEncoding {
```

```
    // MARK: Properties
```

```
    /// Returns a `JSONEncoding` instance with default writing options.
```

```
    public static var `default`: JSONEncoding { return JSONEncoding() }
```

```
    /// Returns a `JSONEncoding` instance with `.prettyPrinted` writing options.
```

```
    public static var prettyPrinted: JSONEncoding { return  
    JSONEncoding(options: .prettyPrinted) }
```

```
    /// The options for writing the parameters as JSON data.
```

```
    public let options: JSONSerialization.WritingOptions
```

```
    // MARK: Initialization
```

```
    /// Creates an instance using the specified `WritingOptions`.
```

```

///

/// - Parameter options: `JSONSerialization.WritingOptions` to use.

public init(options: JSONSerialization.WritingOptions = []) {

    self.options = options

}

// MARK: Encoding

public func encode(_ urlRequest: URLRequestConvertible, with parameters:
Parameters?) throws -> URLRequest {

    var urlRequest = try urlRequest.asURLRequest()

    guard let parameters = parameters else { return urlRequest }

    do {

        let data = try JSONSerialization.data(withJSONObject: parameters, options:
options)

```

```

        if urlRequest.value(forHTTPHeaderField: "Content-Type") == nil {

            urlRequest.setValue("application/json", forHTTPHeaderField: "Content-Type")

        }

        urlRequest.httpBody = data

    } catch {

        throw NSError.parameterEncodingFailed(reason: NSError.jsonEncodingFailed(error:
error))

    }

    return urlRequest

}

/// Encodes any JSON compatible object into a `URLRequest`.

///

/// - Parameters:

/// - urlRequest: `URLRequestConvertible` value into which the object will be encoded.

```

```
/// - jsonObject: `Any` value (must be JSON compatible` to be encoded into the  
`URLRequest`. `nil` by default.
```

```
///
```

```
/// - Returns: The encoded `URLRequest`.
```

```
/// - Throws: Any `Error` produced during encoding.
```

```
public func encode(_ urlRequest: URLRequestConvertible, withObject  
jsonObject: Any? = nil) throws -> URLRequest {
```

```
    var urlRequest = try urlRequest.asURLRequest()
```

```
    guard let jsonObject = jsonObject else { return urlRequest }
```

```
    do {
```

```
        let data = try JSONSerialization.data(withJSONObject: jsonObject, options:  
options)
```

```
        if urlRequest.value(forHTTPHeaderField: "Content-Type") == nil {
```

```
            urlRequest.setValue("application/json", forHTTPHeaderField: "Content-Type")
```

```
        }
```

```

        urlRequest.httpBody = data

    } catch {

        throw NSError.parameterEncodingFailed(reason: NSError.jsonEncodingFailed(error:
error))

    }

    return urlRequest

}

}

```

// MARK: -

```

extension NSNumber {

    fileprivate var isBool: Bool { return CFBooleanGetTypeID() == CFGetTypeID(self) }

}

```

// Copyright (c) 2016-present, Facebook, Inc. All rights reserved.

//

// You are hereby granted a non-exclusive, worldwide, royalty-free license to use,

// copy, modify, and distribute this software in source code or binary form for use

// in connection with the web services and APIs provided by Facebook.

//

// As with any software that integrates with the Facebook platform, your use of

// this software is subject to the Facebook Developer Principles and Policies

// [<http://developers.facebook.com/policy/>]. This copyright notice shall be

// included in all copies or substantial portions of the software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS

// FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR

// COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER

// IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN

// CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

```
import Foundation
```

```
/**
```

Represents a Graph API permission.

Each permission has its own set of requirements and suggested use cases.

See a full list at <https://developers.facebook.com/docs/facebook-login/permissions>

```
*/
```

```
public enum Permission: Hashable, ExpressibleByStringLiteral {
```

```
    /// Provides access to a subset of items that are part of a person's public profile.
```

```
    case publicProfile
```

```
    /// Provides access the list of friends that also use your app.
```

```
    case userFriends
```

```
    /// Provides access to the person's primary email address.
```

```
    case email
```

```
    /// Provides access to a person's personal description (the 'About Me' section on their Profile)
```

```
    case userAboutMe
```

/// Provides access to all common books actions published by any app the person has used.

/// This includes books they've read, want to read, rated or quoted.

case userActionsBooks

/// Provides access to all common Open Graph fitness actions published by any app the person has used.

/// This includes runs, walks and bikes actions.

case userActionsFitness

/// Provides access to all common Open Graph music actions published by any app the person has used.

/// This includes songs they've listened to, and playlists they've created.

case userActionsMusic

/// Provides access to all common Open Graph news actions published by any app the person

/// has used which publishes these actions.

/// This includes news articles they've read or news articles they've published.

case userActionsNews

/// Provides access to all common Open Graph video actions published by any app the person

/// has used which publishes these actions.

case userActionsVideo

/// Access the date and month of a person's birthday. This may or may not include the person's year of birth,

/// dependent upon their privacy settings and the access token being used to query this field.

case userBirthday

/// Provides access to a person's education history through the education field on the User object.

case userEducationHistory

/// Provides read-only access to the Events a person is hosting or has RSVP'd to.

case userEvents

/// Provides access to read a person's game activity (scores, achievements) in any game the person has played.

case userGamesActivity

/// Provides access to a person's gender.

case userGender

/// Provides access to a person's hometown location through the hometown field on the User object.

case userHometown

/// Provides access to the list of all Facebook Pages and Open Graph objects that a person has liked.

case userLikes

/// Provides access to a person's current city through the location field on the User object.

case userLocation

/// Lets your app read the content of groups a person is an admin of through the Groups edge on the User object.

case userManagedGroups

/// Provides access to the photos a person has uploaded or been tagged in.

case userPhotos

/// Provides access to the posts on a person's Timeline. Includes their own posts, posts they are tagged in,

/// and posts other people make on their Timeline.

case userPosts

/// Provides access to a person's relationship status,

/// significant other and family members as fields on the User object.

case userRelationships

/// Provides access to a person's relationship interests as the interested_in field on the User object.

case userRelationshipDetails

/// Provides access to a person's religious and political affiliations.

case userReligionPolitics

/// Provides access to the Places a person has been tagged at in photos, videos, statuses and links.

case userTaggedPlaces

/// Provides access to the videos a person has uploaded or been tagged in.

case userVideos

/// Provides access to the person's personal website URL via the website field on the User object.

case userWebsite

/// Provides access to a person's work history and list of employers via the work field on the User object.

case userWorkHistory

/// Provides access to the names of custom lists a person has created to organize their friends.

case readCustomFriendlists

/// Provides read-only access to the Insights data for Pages, Apps and web domains the person owns.

case readInsights

/// Provides read-only access to the Audience Network Insights data for Apps the person owns.

case readAudienceNetworkInsights

/// Provides the ability to read from the Page Inboxes of the Pages managed by a person.

case readPageMailboxes

/// Provides the access to show the list of the Pages that you manage.

case pagesShowList

/// Provides the access to manage call to actions of the Pages that you manage.

case pagesManageCta

/// Lets your app manage Instant Articles on behalf of Facebook Pages administered by people using your app.

case pagesManageInstantArticles

/// Provides the access to Ads Insights API to pull ads report information for ad accounts you have access to.

case adsRead

/**

Permission with a custom string value.

See <https://developers.facebook.com/docs/facebook-login/permissions> for full list of available permissions.

```
*/
```

```
case custom(String)
```

```
public init(stringLiteral value: String) {
```

```
    guard let permission = StringPermission(rawValue: value)?.permission else {
```

```
        self = .custom(value)
```

```
        return
```

```
    }
```

```
    self = permission
```

```
}
```

```
public var name: String {
```

```
    if let permission = stringPermission {
```

```
        return permission.rawValue
```

```
    }
```

```

if case let .custom(permission) = self {

    return permission

}

return ""

}

private var stringPermission: StringPermission? {

    switch self {

    case .publicProfile: return .publicProfile

    case .userFriends: return .userFriends

    case .email: return .email

    case .userAboutMe: return .userAboutMe

    case .userActionsBooks: return .userActionsBooks

    case .userActionsFitness: return .userActionsFitness

    case .userActionsMusic: return .userActionsMusic

    case .userActionsNews: return .userActionsNews

```

case .userActionsVideo: return .userActionsVideo

case .userBirthday: return .userBirthday

case .userEducationHistory: return .userEducationHistory

case .userEvents: return .userEvents

case .userGamesActivity: return .userGamesActivity

case .userGender: return .userGender

case .userHometown: return .userHometown

case .userLikes: return .userLikes

case .userLocation: return .userLocation

case .userManagedGroups: return .userManagedGroups

case .userPhotos: return .userPhotos

case .userPosts: return .userPosts

case .userRelationships: return .userRelationships

case .userRelationshipDetails: return .userRelationshipDetails

case .userReligionPolitics: return .userReligionPolitics

case .userTaggedPlaces: return .userTaggedPlaces

case .userVideos: return .userVideos

```

    case .userWebsite: return .userWebsite

    case .userWorkHistory: return .userWorkHistory

    case .readCustomFriendlists: return .readCustomFriendlists

    case .readInsights: return .readInsights

    case .readAudienceNetworkInsights: return .readAudienceNetworkInsights

    case .readPageMailboxes: return .readPageMailboxes

    case .pagesShowList: return .pagesShowList

    case .pagesManageCta: return .pagesManageCta

    case .pagesManageInstantArticles: return .pagesManageInstantArticles

    case .adsRead: return .adsRead

    case .custom: return nil

    }

}

}

private enum StringPermission: String {

    case publicProfile = "public_profile"

```


case userFriends = "user_friends"

case email = "email"

case userAboutMe = "user_about_me"

case userActionsBooks = "user_actions.books"

case userActionsFitness = "user_action.fitness"

case userActionsMusic = "user_actions.music"

case userActionsNews = "user_actions.news"

case userActionsVideo = "user_actions.video"

case userBirthday = "user_birthday"

case userEducationHistory = "user_education_history"

case userEvents = "user_events"

case userGamesActivity = "user_games_activity"

case userGender = "user_gender"

case userHometown = "user_hometown"

case userLikes = "user_likes"

case userLocation = "user_location"

case userManagedGroups = "user_managed_groups"

case userPhotos = "user_photos"

case userPosts = "user_posts"

case userRelationships = "user_relationships"

case userRelationshipDetails = "user_relationship_details"

case userReligionPolitics = "user_religion_politics"

case userTaggedPlaces = "user_tagged_places"

case userVideos = "user_videos"

case userWebsite = "user_website"

case userWorkHistory = "user_work_history"

case readCustomFriendlists = "read_custom_friendlists"

case readInsights = "read_insights"

case readAudienceNetworkInsights = "read_audience_network_insights"

case readPageMailboxes = "read_page_mailboxes"

case pagesShowList = "pages_show_list"

case pagesManageCta = "pages_manage_cta"

case pagesManageInstantArticles = "pages_manage_instant_articles"

case adsRead = "ads_read"

```
var permission: Permission {  
  
    switch self {  
  
        case .publicProfile: return .publicProfile  
  
        case .userFriends: return .userFriends  
  
        case .email: return .email  
  
        case .userAboutMe: return .userAboutMe  
  
        case .userActionsBooks: return .userActionsBooks  
  
        case .userActionsFitness: return .userActionsFitness  
  
        case .userActionsMusic: return .userActionsMusic  
  
        case .userActionsNews: return .userActionsNews  
  
        case .userActionsVideo: return .userActionsVideo  
  
        case .userBirthday: return .userBirthday  
  
        case .userEducationHistory: return .userEducationHistory  
  
        case .userEvents: return .userEvents  
  
        case .userGamesActivity: return .userGamesActivity  
  
        case .userGender: return .userGender
```

case .userHometown: return .userHometown

case .userLikes: return .userLikes

case .userLocation: return .userLocation

case .userManagedGroups: return .userManagedGroups

case .userPhotos: return .userPhotos

case .userPosts: return .userPosts

case .userRelationships: return .userRelationships

case .userRelationshipDetails: return .userRelationshipDetails

case .userReligionPolitics: return .userReligionPolitics

case .userTaggedPlaces: return .userTaggedPlaces

case .userVideos: return .userVideos

case .userWebsite: return .userWebsite

case .userWorkHistory: return .userWorkHistory

case .readCustomFriendlists: return .readCustomFriendlists

case .readInsights: return .readInsights

case .readAudienceNetworkInsights: return .readAudienceNetworkInsights

case .readPageMailboxes: return .readPageMailboxes

```
case .pagesShowList: return .pagesShowList

case .pagesManageCta: return .pagesManageCta

case .pagesManageInstantArticles: return .pagesManageInstantArticles

case .adsRead: return .adsRead

}

}

}

//

// PieData.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//
```

```
import Foundation
```

```
open class PieChartData: ChartData
```

```
{
```

```
    public override init()
```

```
    {
```

```
        super.init()
```

```
    }
```

```
    public override init(dataSets: [IChartDataSet]?)
```

```
    {
```

```
        super.init(dataSets: dataSets)
```

```
    }
```

```
    /// All DataSet objects this ChartData object holds.
```

```
    @objc open override var dataSets: [IChartDataSet]
```

```

{

    get

    {

        assert(super.dataSets.count <= 1, "Found multiple data sets while pie chart only
allows one")

        return super.dataSets

    }

    set

    {

        super.dataSets = newValue

    }

}

```

@objc var dataSet: IPieChartDataSet?

```

{

    get

    {

        return dataSet.count > 0 ? dataSet[0] as? IPieChartDataSet : nil

    }

}

```

```

    }

    set

    {

        if let newValue = newValue

        {

            dataSets = [newValue]

        }

        else

        {

            dataSets = []

        }

    }

}

```

open override func getDataSetByIndex(_ index: Int) -> IChartDataSet?

```

{

    if index != 0

```



```

{

    return nil

}

return super.getDataSetByIndex(index)

}

```

open override fun getDataSetByLabel(_ label: String, ignorecase: Bool) ->
IChartDataSet?

```

{

    if dataSets.count == 0 || dataSets[0].label == nil

    {

        return nil

    }

    if ignorecase

    {

        if let label = dataSets[0].label, label.caseInsensitiveCompare(label)
        == .orderedSame

```

```

    {

        return dataSets[0]

    }

}

else

{

    if label == dataSets[0].label

    {

        return dataSets[0]

    }

}

return nil

}

```

open override func entryForHighlight(_ highlight: Highlight) -> ChartDataEntry?

```

{

    return dataSet?.entryForIndex(Int(highlight.x))

}

```

```
}
```

```
open override func addDataSet(_ d: IChartDataSet!)
```

```
{
```

```
    super.addDataSet(d)
```

```
}
```

```
/// Removes the DataSet at the given index in the DataSet array from the data object.
```

```
/// Also recalculates all minimum and maximum values.
```

```
///
```

```
/// - Returns: `true` if a DataSet was removed, `false` if no DataSet could be removed.
```

```
open override func removeDataSetByIndex(_ index: Int) -> Bool
```

```
{
```

```
    if index >= _dataSets.count || index < 0
```

```
    {
```

```
        return false
```

```
    }
```

```

        return false
    }

    /// The total y-value sum across all DataSet objects the this object represents.

    @objc open var yValueSum: Double

    {

        guard let dataSet = dataSet else { return 0.0 }

        return (0..

```

```
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```
// A port of MPAndroidChart for iOS
```

```
// Licensed under Apache License 2.0
```

```
//
```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
open class PieChartDataEntry: ChartDataEntry
```

```
{
```

```
    public required init()
```

```
    {
```

```
        super.init()
```

```
    }
```

```
/// - Parameters:
```

```
/// - value: The value on the y-axis
```

```
@objc public init(value: Double)
```

```
{
```

```
    super.init(x: .nan, y: value)
```

```
}
```

```
/// - Parameters:
```

```
/// - value: The value on the y-axis
```

```
/// - label: The label for the x-axis
```

```
@objc public convenience init(value: Double, label: String?)
```

```
{
```

```
    self.init(value: value)
```

```
    self.label = label
```

```
}
```

```
/// - Parameters:
```

```
/// - value: The value on the y-axis
```

```
/// - label: The label for the x-axis
```

```
/// - data: Spot for additional data this Entry represents
```

```
@objc public convenience init(value: Double, label: String?, data: Any?)
```

```
{
```

```
    self.init(value: value, label: label, icon: nil, data: data)
```

```
}
```

```
/// - Parameters:
```

```
/// - value: The value on the y-axis
```

```
/// - label: The label for the x-axis
```

```
/// - icon: icon image
```

```
@objc public convenience init(value: Double, label: String?, icon: NSUIImage?)
```

```
{
```

```
    self.init(value: value)
```

```
    self.label = label
```

```

        self.icon = icon

    }

    /// - Parameters:

    /// - value: The value on the y-axis

    /// - label: The label for the x-axis

    /// - icon: icon image

    /// - data: Spot for additional data this Entry represents

    @objc public convenience init(value: Double, label: String?, icon: NSUIImage?, data:
Any?)

    {

        self.init(value: value)

        self.label = label

        self.icon = icon

        self.data = data

    }

    /// - Parameters:

```



```
/// - value: The value on the y-axis

/// - data: Spot for additional data this Entry represents

@objc public convenience init(value: Double, data: Any?)

{

    self.init(value: value)

    self.data = data

}
```

```
/// - Parameters:
```

```
/// - value: The value on the y-axis

/// - icon: icon image

@objc public convenience init(value: Double, icon: NSUIImage?)

{

    self.init(value: value)

    self.icon = icon

}
```

```

/// - Parameters:

/// - value: The value on the y-axis

/// - icon: icon image

/// - data: Spot for additional data this Entry represents

@objc public convenience init(value: Double, icon: NSUIImage?, data: Any?)

{

    self.init(value: value)

    self.icon = icon

    self.data = data

}


// MARK: Data property accessors


@objc open var label: String?


@objc open var value: Double

{

```

```
    get { return y }

    set { y = newValue }

}
```

```
// MARK: NSCopying
```

```
open override func copy(with zone: NSZone? = nil) -> Any

{

    let copy = super.copy(with: zone) as! PieChartDataEntry

    copy.label = label

    return copy

}

}

//

// PieChartDataSet.swift

// Charts

//
```

```
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```
// A port of MPAndroidChart for iOS
```

```
// Licensed under Apache License 2.0
```

```
//
```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
open class PieChartDataSet: ChartDataSet, IPieChartDataSet
```

```
{
```

```
    @objc(PieChartValuePosition)
```

```
    public enum ValuePosition: Int
```

```
{
```

```
    case insideSlice
```

```
    case outsideSlice
```

```
}
```

```
private fun initialize()
```

```
{
```

```
    self.valueTextColor = NSUIColor.white
```

```
    self.valueFont = NSUIFont.systemFont(ofSize: 13.0)
```

```
}
```

```
public required init()
```

```
{
```

```
    super.init()
```

```
    initialize()
```

```
}
```

```
public override init(entries: [ChartDataEntry]?, label: String?)
```

```
{
```

```
    super.init(entries: entries, label: label)
```

```

        initialize()

    }

    internal override fun calcMinMax(entry e: ChartDataEntry)

    {

        calcMinMaxY(entry: e)

    }


    // MARK: - Styling functions and accessors


    private var _sliceSpace = CGFloat(0.0)


    /// the space in pixels between the pie-slices


    /// **default**: 0


    /// **maximum**: 20


    open var sliceSpace: CGFloat

    {

```

```
get

{

    return _sliceSpace

}

set

{

    var space = newValue

    if space > 20.0

    {

        space = 20.0

    }

    if space < 0.0

    {

        space = 0.0

    }

    _sliceSpace = space

}
```

```
}
```

```
/// When enabled, slice spacing will be 0.0 when the smallest value is going to be smaller  
than the slice spacing itself.
```

```
open var automaticallyDisableSliceSpacing: Bool = false
```

```
/// indicates the selection distance of a pie slice
```

```
open var selectionShift = CGFloat(18.0)
```

```
open var xValuePosition: ValuePosition = .insideSlice
```

```
open var yValuePosition: ValuePosition = .insideSlice
```

```
/// When valuePosition is OutsideSlice, indicates line color
```

```
open var valueLineColor: NSUIColor? = NSUIColor.black
```

```
/// When valuePosition is OutsideSlice and enabled, line will have the same color as the  
slice
```

```
open var useValueColorForLine: Bool = false
```


/// When valuePosition is OutsideSlice, indicates line width

open var valueLineWidth: CGFloat = 1.0

/// When valuePosition is OutsideSlice, indicates offset as percentage out of the slice size

open var valueLinePart1OffsetPercentage: CGFloat = 0.75

/// When valuePosition is OutsideSlice, indicates length of first half of the line

open var valueLinePart1Length: CGFloat = 0.3

/// When valuePosition is OutsideSlice, indicates length of second half of the line

open var valueLinePart2Length: CGFloat = 0.4

/// When valuePosition is OutsideSlice, this allows variable line length

open var valueLineVariableLength: Bool = true

/// the font for the slice-text labels

```
open var entryLabelFont: NSUIFont? = nil
```

```
/// the color for the slice-text labels
```

```
open var entryLabelColor: NSUIColor? = nil
```

```
/// the color for the highlighted sector
```

```
open var highlightColor: NSUIColor? = nil
```

```
// MARK: - NSCopying
```

```
open override func copy(with zone: NSZone? = nil) -> Any
```

```
{
```

```
    let copy = super.copy(with: zone) as! PieChartDataSet
```

```
    copy._sliceSpace = _sliceSpace
```

```
    copy.automaticallyDisableSliceSpacing = automaticallyDisableSliceSpacing
```

```
    copy.selectionShift = selectionShift
```

```
    copy.xValuePosition = xValuePosition
```

```

        copy.yValuePosition = yValuePosition

        copy.valueLineColor = valueLineColor

        copy.valueLineWidth = valueLineWidth

        copy.valueLinePart1OffsetPercentage = valueLinePart1OffsetPercentage

        copy.valueLinePart1Length = valueLinePart1Length

        copy.valueLinePart2Length = valueLinePart2Length

        copy.valueLineVariableLength = valueLineVariableLength

        copy.entryLabelFont = entryLabelFont

        copy.entryLabelColor = entryLabelColor

        copy.highlightColor = highlightColor

        return copy

    }

}

//

// PieChartRenderer.swift

// Charts

//

```

```
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```
// A port of MPAndroidChart for iOS
```

```
// Licensed under Apache License 2.0
```

```
//
```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
#if canImport(Uikit)
```

```
    import UIKit
```

```
#endif
```

```
#if canImport(Cocoa)
```

```
import Cocoa
```

```
#endif
```

```
open class PieChartRenderer: DataRenderer
```

```
{
```

```
    @objc open weak var chart: PieChartView?
```

```
    @objc public init(chart: PieChartView, animator: Animator, viewPortHandler:  
ViewPortHandler)
```

```
{
```

```
    super.init(animator: animator, viewPortHandler: viewPortHandler)
```

```
    self.chart = chart
```

```
}
```

```
open override func drawData(context: CGContext)
```

```
{
```

```
    guard let chart = chart else { return }
```

```
    let pieData = chart.data
```

```

if pieData != nil

{

    // If we redraw the data, remove and repopulate accessible elements to update label
    values and frames

    accessibleChartElements.removeAll()

    for set in pieData!.dataSets as! [IPieChartDataSet]

        where set.isVisible && set.entryCount > 0

        {

            drawDataSet(context: context, dataSet: set)

        }

    }

}

```

```

@objc open func calculateMinimumRadiusForSpacedSlice(

```

```

    center: CGPoint,

```

```

    radius: CGFloat,

```

```

    angle: CGFloat,

    arcStartPointX: CGFloat,

    arcStartPointY: CGFloat,

    startAngle: CGFloat,

    sweepAngle: CGFloat) -> CGFloat

{

    let angleMiddle = startAngle + sweepAngle / 2.0


    // Other point of the arc

    let arcEndPointX = center.x + radius * cos((startAngle + sweepAngle).DEG2RAD)

    let arcEndPointY = center.y + radius * sin((startAngle + sweepAngle).DEG2RAD)


    // Middle point on the arc

    let arcMidPointX = center.x + radius * cos(angleMiddle.DEG2RAD)

    let arcMidPointY = center.y + radius * sin(angleMiddle.DEG2RAD)


    // This is the base of the contained triangle

```

```

let basePointsDistance = sqrt(

    pow(arcEndPointX - arcStartPointX, 2) +

        pow(arcEndPointY - arcStartPointY, 2))

// After reducing space from both sides of the "slice",

// the angle of the contained triangle should stay the same.

// So let's find out the height of that triangle.

let containedTriangleHeight = (basePointsDistance / 2.0 *

    tan((180.0 - angle).DEG2RAD / 2.0))

// Now we subtract that from the radius

var spacedRadius = radius - containedTriangleHeight

// And now subtract the height of the arc that's between the triangle and the outer circle

spacedRadius -= sqrt(

    pow(arcMidPointX - (arcEndPointX + arcStartPointX) / 2.0, 2) +

        pow(arcMidPointY - (arcEndPointY + arcStartPointY) / 2.0, 2))

```



```

        return spacedRadius
    }

    /// Calculates the sliceSpace to use based on visible values and their size compared to
    the set sliceSpace.

    @objc open func getSliceSpace(dataSet: IPieChartDataSet) -> CGFloat

    {

        guard

            dataSet AutomaticallyDisableSliceSpacing,

            let data = chart?.data as? PieChartData

        else { return dataSet.sliceSpace }

        let spaceSizeRatio = dataSet.sliceSpace / min(viewPortHandler.contentWidth,
viewPortHandler.contentHeight)

        let minValueRatio = dataSet.yMin / data.yValueSum * 2.0

        let sliceSpace = spaceSizeRatio > CGFloat(minValueRatio)

```

? 0.0

: dataSet.sliceSpace

return sliceSpace

}

@objc open func drawDataSet(context: CGContext, dataSet: IPieChartDataSet)

{

guard let chart = chart else {return }

var angle: CGFloat = 0.0

let rotationAngle = chart.rotationAngle

let phaseX = animator.phaseX

let phaseY = animator.phaseY

let entryCount = dataSet.entryCount

```

let drawAngles = chart.drawAngles

let center = chart.centerCircleBox

let radius = chart.radius

let drawInnerArc = chart.drawHoleEnabled && !chart.drawSlicesUnderHoleEnabled

let userInnerRadius = drawInnerArc ? radius * chart.holeRadiusPercent : 0.0


var visibleAngleCount = 0

for j in 0 ..< entryCount

{

    guard let e = dataSet.entryForIndex(j) else { continue }

    if ((abs(e.y) > Double.ulpOfOne))

    {

        visibleAngleCount += 1

    }

}

let sliceSpace = visibleAngleCount <= 1 ? 0.0 : getSliceSpace(dataSet: dataSet)

```

```
context.saveGState()
```

```
// Make the chart header the first element in the accessible elements array
```

```
// We can do this in drawDataSet, since we know PieChartView can have only 1  
dataSet
```

```
// Also since there's only 1 dataset, we don't use the typical createAccessibleHeader()  
here.
```

```
// NOTE: - Since we want to summarize the total count of slices/portions/elements,  
use a default string here
```

```
// This is unlike when we are naming individual slices, wherein it's alright to not use  
a prefix as descriptor.
```

```
// i.e. We want to VO to say "3 Elements" even if the developer didn't specify an  
accessibility prefix
```

```
// If prefix is unspecified it is safe to assume they did not want to use "Element 1", so  
that uses a default empty string
```

```
let prefix: String = chart.data?.accessibilityEntryLabelPrefix ?? "Element"
```

```
let description = chart.chartDescription?.text ?? dataSet.label ?? chart.centerText ??  
"Pie Chart"
```

```
let
```

```

element = NSUIAccessibilityElement(accessibilityContainer: chart)

element.accessibilityLabel = description + ". \"(entryCount) \"(prefix + (entryCount ==
1 ? \"\" : \"s\"))\"

element.accessibilityFrame = chart.bounds

element.isHeader = true

accessibleChartElements.append(element)

for j in 0 ..< entryCount

{

    let sliceAngle = drawAngles[j]

    var innerRadius = userInnerRadius

    guard let e = dataSet.entryForIndex(j) else { continue }

    defer

    {

        // From here on, even when skipping (i.e for highlight),

        // increase the angle

```

```

        angle += sliceAngle * CGFloat(phaseX)

    }

    // draw only if the value is greater than zero

    if abs(e.y) < Double.ulpOfOne { continue }

    // Skip if highlighted

    if dataSet.isHighlightEnabled && chart.needsHighlight(index: j)

    {

        continue

    }

    let accountForSliceSpacing = sliceSpace > 0.0 && sliceAngle <= 180.0

    context.setFill_color(dataSet.color(atIndex: j).cgColor)

    let sliceSpaceAngleOuter = visibleAngleCount == 1 ?

```

0.0 :

sliceSpace / radius.DEG2RAD

let startAngleOuter = rotationAngle + (angle + sliceSpaceAngleOuter / 2.0) *
CGFloat(phaseY)

var sweepAngleOuter = (sliceAngle - sliceSpaceAngleOuter) * CGFloat(phaseY)

if sweepAngleOuter < 0.0

{

sweepAngleOuter = 0.0

}

let arcStartPointX = center.x + radius * cos(startAngleOuter.DEG2RAD)

let arcStartPointY = center.y + radius * sin(startAngleOuter.DEG2RAD)

let path = CGMutablePath()

path.move(to: CGPoint(x: arcStartPointX,

y: arcStartPointY))

```
        path.addRelativeArc(center:    center,    radius:    radius,    startAngle:
startAngleOuter.DEG2RAD, delta: sweepAngleOuter.DEG2RAD)
```

```
if drawInnerArc &&
```

```
    (innerRadius > 0.0 || accountForSliceSpacing)
```

```
{
```

```
    if accountForSliceSpacing
```

```
{
```

```
        var minSpacedRadius = calculateMinimumRadiusForSpacedSlice(
```

```
            center: center,
```

```
            radius: radius,
```

```
            angle: sliceAngle * CGFloat(phaseY),
```

```
            arcStartPointX: arcStartPointX,
```

```
            arcStartPointY: arcStartPointY,
```

```
            startAngle: startAngleOuter,
```

```
            sweepAngle: sweepAngleOuter)
```

```
        if minSpacedRadius < 0.0
```

```
{
```



```

        minSpacedRadius = -minSpacedRadius

    }

    innerRadius = min(max(innerRadius, minSpacedRadius), radius)

}

let sliceSpaceAngleInner = visibleAngleCount == 1 || innerRadius == 0.0 ?

    0.0 :

    sliceSpace / innerRadius.DEG2RAD

let startAngleInner = rotationAngle + (angle + sliceSpaceAngleInner / 2.0) *
CGFloat(phaseY)

var sweepAngleInner = (sliceAngle - sliceSpaceAngleInner) * CGFloat(phaseY)

if sweepAngleInner < 0.0

{

    sweepAngleInner = 0.0

}

let endAngleInner = startAngleInner + sweepAngleInner

path.addLine(

```

```

        to: CGPoint(

            x: center.x + innerRadius * cos(endAngleInner.DEG2RAD),

            y: center.y + innerRadius * sin(endAngleInner.DEG2RAD)))

        path.addRelativeArc(center: center, radius: innerRadius, startAngle:
endAngleInner.DEG2RAD, delta: -sweepAngleInner.DEG2RAD)

    }

    else

    {

        if accountForSliceSpacing

        {

            let angleMiddle = startAngleOuter + sweepAngleOuter / 2.0

            let sliceSpaceOffset =

            calculateMinimumRadiusForSpacedSlice(

                center: center,

                radius: radius,

                angle: sliceAngle * CGFloat(phaseY),

```

```
arcStartPointX: arcStartPointX,
```

```
arcStartPointY: arcStartPointY,
```

```
startAngle: startAngleOuter,
```

```
sweepAngle: sweepAngleOuter)
```

```
let arcEndPointX = center.x + sliceSpaceOffset *  
cos(angleMiddle.DEG2RAD)
```

```
let arcEndPointY = center.y + sliceSpaceOffset * sin(angleMiddle.DEG2RAD)
```

```
path.addLine(  
  

```

```
to: CGPoint(  
  

```

```
x: arcEndPointX,  
  

```

```
y: arcEndPointY))  
  

```

```
}  
  

```

```
else  
  

```

```
{  
  

```

```
path.addLine(to: center)  
  

```

```
}  
  

```

```
}
```

```
path.closeSubpath()
```

```
context.beginPath()
```

```
context.addPath(path)
```

```
context.fillPath(using: .evenOdd)
```

```
let axElement = createAccessibleElement(withIndex: j,
```

```
    container: chart,
```

```
    dataSet: dataSet)
```

```
{ (element) in
```

```
    element.accessibilityFrame = path.boundingBoxOfPath
```

```
}
```

```
accessibleChartElements.append(axElement)
```

```
}
```

```

// Post this notification to let VoiceOver account for the redrawn frames

accessibilityPostLayoutChangedNotification()

context.restoreGState()

}

open override func drawValues(context: CGContext)

{

    guard

        let chart = chart,

        let data = chart.data

    else { return }

    let center = chart.centerCircleBox

    // get whole the radius

```

```
let radius = chart.radius
```

```
let rotationAngle = chart.rotationAngle
```

```
let drawAngles = chart.drawAngles
```

```
let absoluteAngles = chart.absoluteAngles
```

```
let phaseX = animator.phaseX
```

```
let phaseY = animator.phaseY
```

```
var labelRadiusOffset = radius / 10.0 * 3.0
```

```
if chart.drawHoleEnabled
```

```
{
```

```
    labelRadiusOffset = (radius - (radius * chart.holeRadiusPercent)) / 2.0
```

```
}
```

```
let labelRadius = radius - labelRadiusOffset
```

```
let dataSets = data.dataSets
```

```
let yValueSum = (data as! PieChartData).yValueSum
```

```
let drawEntryLabels = chart.isDrawEntryLabelsEnabled
```

```
let usePercentValuesEnabled = chart.usePercentValuesEnabled
```

```
var angle: CGFloat = 0.0
```

```
var xIndex = 0
```

```
context.saveGState()
```

```
defer { context.restoreGState() }
```

```
for i in 0 ..< dataSets.count
```

```
{
```

```
    guard let dataSet = dataSets[i] as? IPieChartDataSet else { continue }
```

```
let drawValues = dataSet.isDrawValuesEnabled
```

```
if !drawValues && !drawEntryLabels && !dataSet.isDrawIconsEnabled
```

```
{
```

```
    continue
```

```
}
```

```
let iconsOffset = dataSet.iconsOffset
```

```
let xValuePosition = dataSet.xValuePosition
```

```
let yValuePosition = dataSet.yValuePosition
```

```
let valueFont = dataSet.valueFont
```

```
let entryLabelFont = dataSet.entryLabelFont ?? chart.entryLabelFont
```

```
let lineHeight = valueFont.lineHeight
```

```
guard let formatter = dataSet.valueFormatter else { continue }
```



```

for j in 0 ..< dataSet.entryCount

{

    guard let e = dataSet.entryForIndex(j) else { continue }

    let pe = e as? PieChartDataEntry

    if xIndex == 0

    {

        angle = 0.0

    }

    else

    {

        angle = absoluteAngles[xIndex - 1] * CGFloat(phaseX)

    }

    let sliceAngle = drawAngles[xIndex]

    let sliceSpace = getSliceSpace(dataSet: dataSet)

```

```
let sliceSpaceMiddleAngle = sliceSpace / labelRadius.DEG2RAD
```

```
// offset needed to center the drawn text in the slice
```

```
let angleOffset = (sliceAngle - sliceSpaceMiddleAngle / 2.0) / 2.0
```

```
angle = angle + angleOffset
```

```
let transformedAngle = rotationAngle + angle * CGFloat(phaseY)
```

```
let value = usePercentValuesEnabled ? e.y / yValueSum * 100.0 : e.y
```

```
let valueText = formatter.stringForValue(
```

```
    value,
```

```
    entry: e,
```

```
    dataSetIndex: i,
```

```
    viewPortHandler: viewPortHandler)
```

```
let sliceXBase = cos(transformedAngle.DEG2RAD)
```

```
let sliceYBase = sin(transformedAngle.DEG2RAD)
```

```
let drawXOutside = drawEntryLabels && xValuePosition == .outsideSlice
```

```
let drawYOutside = drawValues && yValuePosition == .outsideSlice
```

```
let drawXInside = drawEntryLabels && xValuePosition == .insideSlice
```

```
let drawYInside = drawValues && yValuePosition == .insideSlice
```

```
let valueTextColor = dataSet.valueTextColorAt(j)
```

```
let entryLabelColor = dataSet.entryLabelColor ?? chart.entryLabelColor
```

```
if drawXOutside || drawYOutside
```

```
{
```

```
    let valueLineLength1 = dataSet.valueLinePart1Length
```

```
    let valueLineLength2 = dataSet.valueLinePart2Length
```

```
    let valueLinePart1OffsetPercentage =  
dataSet.valueLinePart1OffsetPercentage
```

```
    var pt2: CGPoint
```

```
var labelPoint: CGPoint
```

```
var align: NSTextAlignment
```

```
var line1Radius: CGFloat
```

```
if chart.drawHoleEnabled
```

```
{
```

```
    line1Radius = (radius - (radius * chart.holeRadiusPercent)) *  
valueLinePart1OffsetPercentage + (radius * chart.holeRadiusPercent)
```

```
}
```

```
else
```

```
{
```

```
    line1Radius = radius * valueLinePart1OffsetPercentage
```

```
}
```

```
let polyline2Length = dataSet.valueLineVariableLength
```

```
    ? labelRadius * valueLineLength2 *  
abs(sin(transformedAngle.DEG2RAD))
```

```
: labelRadius * valueLineLength2
```

```
let pt0 = CGPoint(
```

```
  x: line1Radius * sliceXBase + center.x,
```

```
  y: line1Radius * sliceYBase + center.y)
```

```
let pt1 = CGPoint(
```

```
  x: labelRadius * (1 + valueLineLength1) * sliceXBase + center.x,
```

```
  y: labelRadius * (1 + valueLineLength1) * sliceYBase + center.y)
```

```
if transformedAngle.truncatingRemainder(dividingBy: 360.0) >= 90.0 &&  
transformedAngle.truncatingRemainder(dividingBy: 360.0) <= 270.0
```

```
{
```

```
  pt2 = CGPoint(x: pt1.x - polyline2Length, y: pt1.y)
```

```
  align = .right
```

```
  labelPoint = CGPoint(x: pt2.x - 5, y: pt2.y - lineHeight)
```

```
}
```

```
else
```

```

{

    pt2 = CGPoint(x: pt1.x + polyline2Length, y: pt1.y)

    align = .left

    labelPoint = CGPoint(x: pt2.x + 5, y: pt2.y - lineHeight)

}

```

DrawLine: do

```

{

    if dataSet.useValueColorForLine

    {

        context.setStrokeColor(dataSet.color(atIndex: j).CGColor)

    }

    else if let valueLineColor = dataSet.valueLineColor

    {

        context.setStrokeColor(valueLineColor.CGColor)

    }

    else

```

```

    {

        return

    }

    context.setLineWidth(dataSet.valueLineWidth)


    context.move(to: CGPoint(x: pt0.x, y: pt0.y))

    context.addLine(to: CGPoint(x: pt1.x, y: pt1.y))

    context.addLine(to: CGPoint(x: pt2.x, y: pt2.y))


    context.drawPath(using: CGPathDrawingMode.stroke)

}


if drawXOutside && drawYOutside

{

    ChartUtils.drawText(

        context: context,

        text: valueText,

```

```

        point: labelPoint,

        align: align,

        attributes: [NSAttributedString.Key.font: valueFont,
NSAttributedString.Key.foregroundColor: valueTextColor]

    )

```

```

if j < data.entryCount && pe?.label != nil

{

    ChartUtils.drawText(

        context: context,

        text: pe!.label!,

        point: CGPoint(x: labelPoint.x, y: labelPoint.y + lineHeight),

        align: align,

        attributes: [

            NSAttributedString.Key.font: entryLabelFont ?? valueFont,

            NSAttributedString.Key.foregroundColor: entryLabelColor ??
valueTextColor]

    )

```



```

    }

}

else if drawXOutside

{

    if j < data.entryCount && pe?.label != nil

    {

        ChartUtils.drawText(

            context: context,

            text: pe!.label!,

            point: CGPoint(x: labelPoint.x, y: labelPoint.y + lineHeight / 2.0),

            align: align,

            attributes: [

                NSAttributedString.Key.font: entryLabelFont ?? valueFont,

                NSAttributedString.Key.foregroundColor: entryLabelColor ??
valueTextColor]

        )

    }

}

```

```

else if drawYOutside

{

    ChartUtils.drawText(

        context: context,

        text: valueText,

        point: CGPoint(x: labelPoint.x, y: labelPoint.y + lineHeight / 2.0),

        align: align,

        attributes: [NSAttributedString.Key.font: valueFont,
NSAttributedString.Key.foregroundColor: valueTextColor]

    )

}

}

```

```

if drawXInside || drawYInside

```

```

{

    // calculate the text position

    let x = labelRadius * sliceXBase + center.x

    let y = labelRadius * sliceYBase + center.y - lineHeight

```

```

if drawXInside && drawYInside

{

    ChartUtils.drawText(

        context: context,

        text: valueText,

        point: CGPoint(x: x, y: y),

        align: .center,

        attributes: [NSAttributedString.Key.font: valueFont,
NSAttributedString.Key.foregroundColor: valueTextColor]

    )

```

```

if j < data.entryCount && pe?.label != nil

```

```

{

    ChartUtils.drawText(

        context: context,

        text: pe!.label!,

        point: CGPoint(x: x, y: y + lineHeight),

```

```

        align: .center,

        attributes: [

            NSAttributedString.Key.font: entryLabelFont ?? valueFont,

            NSAttributedString.Key.foregroundColor: entryLabelColor ??
valueTextColor]

        )

    }

}

else if drawXInside

{

    if j < data.entryCount && pe?.label != nil

    {

        ChartUtils.drawText(

            context: context,

            text: pe!.label!,

            point: CGPoint(x: x, y: y + lineHeight / 2.0),

            align: .center,

            attributes: [

```

```

        NSAttributedString.Key.font: entryLabelFont ?? valueFont,

        NSAttributedString.Key.foregroundColor: entryLabelColor ??
valueTextColor]

    )

}

}

else if drawYInside

{

    ChartUtils.drawText(

        context: context,

        text: valueText,

        point: CGPoint(x: x, y: y + lineHeight / 2.0),

        align: .center,

        attributes: [NSAttributedString.Key.font: valueFont,
NSAttributedString.Key.foregroundColor: valueTextColor]

    )

}

}

```

```

if let icon = e.icon, dataSet.isDrawIconsEnabled

{

    // calculate the icon's position

    let x = (labelRadius + iconsOffset.y) * sliceXBase + center.x

    var y = (labelRadius + iconsOffset.y) * sliceYBase + center.y

    y += iconsOffset.x

    ChartUtils.drawImage(context: context,

        image: icon,

        x: x,

        y: y,

        size: icon.size)

}

xIndex += 1

```

```
    }  
  }  
}
```

open override fun drawExtras(context: CGContext)

```
{  
  
    drawHole(context: context)  
  
    drawCenterText(context: context)  
}
```

/// draws the hole in the center of the chart and the transparent circle / hole

private fun drawHole(context: CGContext)

```
{  
  
    guard let chart = chart else { return }
```

if chart.drawHoleEnabled

```
{
```

```

context.saveGState()

let radius = chart.radius

let holeRadius = radius * chart.holeRadiusPercent

let center = chart.centerCircleBox

if let holeColor = chart.holeColor
{
    if holeColor != NSUIColor.clear
    {
        // draw the hole-circle

        context.setFillColor(chart.holeColor!.cgColor)

        context.fillEllipse(in: CGRect(x: center.x - holeRadius, y: center.y -
holeRadius, width: holeRadius * 2.0, height: holeRadius * 2.0))

    }
}

// only draw the circle if it can be seen (not covered by the hole)

```



```

if let transparentCircleColor = chart.transparentCircleColor

{

    if transparentCircleColor != NSUIColor.clear &&

        chart.transparentCircleRadiusPercent > chart.holeRadiusPercent

    {

        let alpha = animator.phaseX * animator.phaseY

        let secondHoleRadius = radius * chart.transparentCircleRadiusPercent


        // make transparent

        context.setAlpha(CGFloat(alpha))

        context.setFillColor(transparentCircleColor.cgColor)


        // draw the transparent-circle

        context.beginPath()

        context.addEllipse(in: CGRect(

            x: center.x - secondHoleRadius,

            y: center.y - secondHoleRadius,

```

```

        width: secondHoleRadius * 2.0,

        height: secondHoleRadius * 2.0))

context.addEllipse(in: CGRect(

    x: center.x - holeRadius,

    y: center.y - holeRadius,

    width: holeRadius * 2.0,

    height: holeRadius * 2.0))

context.fillPath(using: .evenOdd)

    }

}

```

```

context.restoreGState()

}

}

```

/// draws the description text in the center of the pie chart makes most sense when center-hole is enabled

```
private func drawCenterText(context: CGContext)
```

```

{

    guard

        let chart = chart,

        let centerAttributedText = chart.centerAttributedText

        else { return }

    if chart.drawCenterTextEnabled && centerAttributedText.length > 0

    {

        let center = chart.centerCircleBox

        let offset = chart.centerTextOffset

        let innerRadius = chart.drawHoleEnabled
        && !chart.drawSlicesUnderHoleEnabled ? chart.radius * chart.holeRadiusPercent :
        chart.radius

        let x = center.x + offset.x

        let y = center.y + offset.y

        let holeRect = CGRect(

```

```

        x: x - innerRadius,

        y: y - innerRadius,

        width: innerRadius * 2.0,

        height: innerRadius * 2.0)

var boundingRect = holeRect

if chart.centerTextRadiusPercent > 0.0

{

    boundingRect = boundingRect.insetBy(dx: (boundingRect.width -
boundingRect.width * chart.centerTextRadiusPercent) / 2.0, dy: (boundingRect.height -
boundingRect.height * chart.centerTextRadiusPercent) / 2.0)

}

let textBounds = centerAttributedText.boundingBox(with: boundingRect.size,
options: [.usesLineFragmentOrigin, .usesFontLeading, .truncatesLastVisibleLine], context:
nil)

var drawingRect = boundingRect

drawingRect.origin.x += (boundingRect.size.width - textBounds.size.width) / 2.0

```

```
drawingRect.origin.y += (boundingRect.size.height - textBounds.size.height) / 2.0
```

```
drawingRect.size = textBounds.size
```

```
context.saveGState()
```

```
let clippingPath = CGPath(ellipseIn: holeRect, transform: nil)
```

```
context.beginPath()
```

```
context.addPath(clippingPath)
```

```
context.clip()
```

```
centerAttributedText.draw(with: drawingRect, options:  
[.usesLineFragmentOrigin, .usesFontLeading, .truncatesLastVisibleLine], context: nil)
```

```
context.restoreGState()
```

```
}
```

```
}
```

```
open override func drawHighlighted(context: CGContext, indices: [Highlight])
```

```
{  
  
    guard  
  
        let chart = chart,  
  
        let data = chart.data  
  
        else { return }  
  
    context.saveGState()  
  
    let phaseX = animator.phaseX  
  
    let phaseY = animator.phaseY  
  
    var angle: CGFloat = 0.0  
  
    let rotationAngle = chart.rotationAngle  
  
    let drawAngles = chart.drawAngles  
  
    let absoluteAngles = chart.absoluteAngles  
  
    let center = chart.centerCircleBox
```

```

let radius = chart.radius

let drawInnerArc = chart.drawHoleEnabled && !chart.drawSlicesUnderHoleEnabled

let userInnerRadius = drawInnerArc ? radius * chart.holeRadiusPercent : 0.0


// Append highlighted accessibility slices into this array, so we can prioritize them
over unselected slices

var highlightedAccessibleElements: [NSUIAccessibilityElement] = []


for i in 0 ..< indices.count

{

    // get the index to highlight

    let index = Int(indices[i].x)

    if index >= drawAngles.count

    {

        continue

    }

```

```
guard let set = data.getDataSetByIndex(indices[i].dataSetIndex) as?
IPieChartDataSet else { continue }
```

```
if !set.isHighlightEnabled { continue }
```

```
let entryCount = set.entryCount
```

```
var visibleAngleCount = 0
```

```
for j in 0 ..< entryCount
```

```
{
```

```
guard let e = set.entryForIndex(j) else { continue }
```

```
if ((abs(e.y) > Double.ulpOfOne))
```

```
{
```

```
visibleAngleCount += 1
```

```
}
```

```
}
```

```
if index == 0
```

```
{
```



```

        angle = 0.0

    }

    else

    {

        angle = absoluteAngles[index - 1] * CGFloat(phaseX)

    }

let sliceSpace = visibleAngleCount <= 1 ? 0.0 : set.sliceSpace

let sliceAngle = drawAngles[index]

var innerRadius = userInnerRadius

let shift = set.selectionShift

let highlightedRadius = radius + shift

let accountForSliceSpacing = sliceSpace > 0.0 && sliceAngle <= 180.0

```

```
context.setFillColors(set.highlightColor?.cgColor ?? set.color(atIndex:
index).cgColor)
```

```
let sliceSpaceAngleOuter = visibleAngleCount == 1 ?
```

```
0.0 :
```

```
sliceSpace / radius.DEG2RAD
```

```
let sliceSpaceAngleShifted = visibleAngleCount == 1 ?
```

```
0.0 :
```

```
sliceSpace / highlightedRadius.DEG2RAD
```

```
let startAngleOuter = rotationAngle + (angle + sliceSpaceAngleOuter / 2.0) *
CGFloat(phaseY)
```

```
var sweepAngleOuter = (sliceAngle - sliceSpaceAngleOuter) * CGFloat(phaseY)
```

```
if sweepAngleOuter < 0.0
```

```
{
```

```
sweepAngleOuter = 0.0
```

```
}
```

```

        let startAngleShifted = rotationAngle + (angle + sliceSpaceAngleShifted / 2.0) *
CGFloat(phaseY)

        var sweepAngleShifted = (sliceAngle - sliceSpaceAngleShifted) *
CGFloat(phaseY)

        if sweepAngleShifted < 0.0

        {

            sweepAngleShifted = 0.0

        }

        let path = CGMutablePath()

        path.move(to: CGPoint(x: center.x + highlightedRadius *
cos(startAngleShifted.DEG2RAD),

                                y: center.y + highlightedRadius * sin(startAngleShifted.DEG2RAD)))

        path.addRelativeArc(center: center, radius: highlightedRadius, startAngle:
startAngleShifted.DEG2RAD,

                                delta: sweepAngleShifted.DEG2RAD)

```

```

var sliceSpaceRadius: CGFloat = 0.0

if accountForSliceSpacing
{

    sliceSpaceRadius = calculateMinimumRadiusForSpacedSlice(

        center: center,

        radius: radius,

        angle: sliceAngle * CGFloat(phaseY),

        arcStartPointX: center.x + radius * cos(startAngleOuter.DEG2RAD),

        arcStartPointY: center.y + radius * sin(startAngleOuter.DEG2RAD),

        startAngle: startAngleOuter,

        sweepAngle: sweepAngleOuter)

}

if drawInnerArc &&

    (innerRadius > 0.0 || accountForSliceSpacing)

{

```

```

if accountForSliceSpacing

{

    var minSpacedRadius = sliceSpaceRadius

    if minSpacedRadius < 0.0

    {

        minSpacedRadius = -minSpacedRadius

    }

    innerRadius = min(max(innerRadius, minSpacedRadius), radius)

}


let sliceSpaceAngleInner = visibleAngleCount == 1 || innerRadius == 0.0 ?

    0.0 :

    sliceSpace / innerRadius.DEG2RAD


let startAngleInner = rotationAngle + (angle + sliceSpaceAngleInner / 2.0) *
CGFloat(phaseY)

var sweepAngleInner = (sliceAngle - sliceSpaceAngleInner) * CGFloat(phaseY)

if sweepAngleInner < 0.0

{

```

```

        sweepAngleInner = 0.0

    }

    let endAngleInner = startAngleInner + sweepAngleInner

    path.addLine(

        to: CGPoint(

            x: center.x + innerRadius * cos(endAngleInner.DEG2RAD),

            y: center.y + innerRadius * sin(endAngleInner.DEG2RAD)))

    path.addRelativeArc(center: center, radius: innerRadius,

        startAngle: endAngleInner.DEG2RAD,

        delta: -sweepAngleInner.DEG2RAD)

    }

    else

    {

        if accountForSliceSpacing

        {

```

```
let angleMiddle = startAngleOuter + sweepAngleOuter / 2.0
```

```
let    arcEndPointX    =    center.x    +    sliceSpaceRadius    *  
cos(angleMiddle.DEG2RAD)
```

```
let    arcEndPointY    =    center.y    +    sliceSpaceRadius    *  
sin(angleMiddle.DEG2RAD)
```

```
path.addLine(  
  
    to: CGPoint(  
  
        x: arcEndPointX,  
  
        y: arcEndPointY))  
  
}  
  
else  
  
{  
  
    path.addLine(to: center)  
  
}  
  
}
```

```
path.closeSubpath()
```

```
context.beginPath()
```

```
context.addPath(path)
```

```
context.fillPath(using: .evenOdd)
```

```
let axElement = createAccessibleElement(withIndex: index,
```

```
        container: chart,
```

```
        dataSet: set)
```

```
{ (element) in
```

```
    element.accessibilityFrame = path.boundingBoxOfPath
```

```
    element.isSelected = true
```

```
}
```

```
highlightedAccessibleElements.append(axElement)
```

```
}
```



```
// Prepend selected slices before the already rendered unselected ones.
```

```
// NOTE: - This relies on drawDataSet() being called before drawHighlighted in  
PieChartView.
```

```
if !accessibleChartElements.isEmpty {
```

```
    accessibleChartElements.insert(contentsOf: highlightedAccessibleElements, at: 1)
```

```
}
```

```
context.restoreGState()
```

```
}
```

```
/// Creates an NSUIAccessibilityElement representing a slice of the PieChart.
```

```
/// The element only has it's container and label set based on the chart and dataSet. Use  
the modifier to alter traits and frame.
```

```
private func createAccessibleElement(withIndex idx: Int,
```

```
    container: PieChartView,
```

```
    dataSet: IPieChartDataSet,
```

```
    modifier: (NSUIAccessibilityElement) -> () ->
```

```
NSUIAccessibilityElement {
```

```
let element = NSUIAccessibilityElement(accessibilityContainer: container)
```

```
guard let e = dataSet.entryForIndex(idx) else { return element }
```

```
guard let formatter = dataSet.valueFormatter else { return element }
```

```
guard let data = container.data as? PieChartData else { return element }
```

```
var elementValueText = formatter.stringForValue(
```

```
    e.y,
```

```
    entry: e,
```

```
    dataSetIndex: idx,
```

```
    viewPortHandler: viewPortHandler)
```

```
if container.usePercentValuesEnabled {
```

```
    let value = e.y / data.yValueSum * 100.0
```

```
    let valueText = formatter.stringForValue(
```

```
        value,
```

```
        entry: e,
```

```

        dataSetIndex: idx,

        viewPortHandler: viewPortHandler)

    elementValueText = valueText

}

let pieChartDataEntry = (dataSet.entryForIndex(idx) as? PieChartDataEntry)

let isCount = data.accessibilityEntryLabelSuffixIsCount

let prefix = data.accessibilityEntryLabelPrefix?.appending("(idx + 1)") ??
pieChartDataEntry?.label ?? ""

let suffix = data.accessibilityEntryLabelSuffix ?? ""

element.accessibilityLabel = "(prefix) : \(elementValueText) \(suffix + (isCount ?
(e.y == 1.0 ? "" : "s") : "")) )"

// The modifier allows changing of traits and frame depending on highlight, rotation,
etc

modifier(element)

return element

```

```
}

}

//

// PieChartView.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

#if canImport(Uikit)
```

```

import UIKit

#endif

#if canImport(Cocoa)

import Cocoa

#endif

/// View that represents a pie chart. Draws cake like slices.

open class PieChartView: PieRadarChartViewBase

{

    /// rect object that represents the bounds of the piechart, needed for drawing the circle

    private var _circleBox = CGRect()


    /// flag indicating if entry labels should be drawn or not

    private var _drawEntryLabelsEnabled = true


    /// array that holds the width of each pie-slice in degrees

```

```

private var _drawAngles = [CGFloat]()

/// array that holds the absolute angle in degrees of each slice

private var _absoluteAngles = [CGFloat]()


/// if true, the hole inside the chart will be drawn

private var _drawHoleEnabled = true


private var _holeColor: NSUIColor? = NSUIColor.white


/// Sets the color the entry labels are drawn with.

private var _entryLabelColor: NSUIColor? = NSUIColor.white


/// Sets the font the entry labels are drawn with.

private var _entryLabelFont: NSUIFont? = NSUIFont(name: "HelveticaNeue", size:
13.0)


/// if true, the hole will see-through to the inner tips of the slices

```

```

private var _drawSlicesUnderHoleEnabled = false

/// if true, the values inside the piechart are drawn as percent values

private var _usePercentValuesEnabled = false

/// variable for the text that is drawn in the center of the pie-chart

private var _centerAttributedText: NSAttributedString?

/// the offset on the x- and y-axis the center text has in dp.

private var _centerTextOffset: CGPoint = CGPoint()

/// indicates the size of the hole in the center of the piechart

///

/// **default**: `0.5`

private var _holeRadiusPercent = CGFloat(0.5)

private var _transparentCircleColor: NSUIColor? = NSUIColor(white: 1.0, alpha:
105.0/255.0)

```

```
/// the radius of the transparent circle next to the chart-hole in the center
```

```
private var _transparentCircleRadiusPercent = CGFloat(0.55)
```

```
/// if enabled, centertext is drawn
```

```
private var _drawCenterTextEnabled = true
```

```
private var _centerTextRadiusPercent: CGFloat = 1.0
```

```
/// maximum angle for this pie
```

```
private var _maxAngle: CGFloat = 360.0
```

```
public override init(frame: CGRect)
```

```
{
```

```
    super.init(frame: frame)
```

```
}
```



```
public required init?(coder aDecoder: NSCoder)
```

```
{
```

```
    super.init(coder: aDecoder)
```

```
}
```

```
internal override func initialize()
```

```
{
```

```
    super.initialize()
```

```
    renderer = PieChartRenderer(chart: self, animator: _animator, viewPortHandler:  
_viewPortHandler)
```

```
    _xAxis = nil
```

```
    self.highlighter = PieHighlighter(chart: self)
```

```
}
```

```
open override func draw(_ rect: CGRect)
```

```
{
```

```
super.draw(rect)
```

```
if _data === nil
```

```
{
```

```
    return
```

```
}
```

```
let optionalContext = NSUIGraphicsGetCurrentContext()
```

```
guard let context = optionalContext, let renderer = renderer else
```

```
{
```

```
    return
```

```
}
```

```
renderer.drawData(context: context)
```

```
if (valuesToHighlight())
```

```
{
```

```

        renderer.drawHighlighted(context: context, indices: _indicesToHighlight)
    }

    renderer.drawExtras(context: context)

    renderer.drawValues(context: context)

    legendRenderer.renderLegend(context: context)

    drawDescription(context: context)

    drawMarkers(context: context)
}

/// if width is larger than height

private var widthLarger: Bool

{

```

```

        return _viewPortHandler.contentRect.orientation == .landscape
    }

    /// adjusted radius. Use diameter when it's half pie and width is larger

    private var adjustedRadius: CGFloat

    {

        return maxAngle <= 180 && widthLarger ? diameter : diameter / 2.0
    }

    /// true centerOffsets considering half pie & width is larger

    private func adjustedCenterOffsets() -> CGPoint

    {

        var c = self.centerOffsets

        c.y = maxAngle <= 180 && widthLarger ? c.y + adjustedRadius / 2 : c.y

        return c
    }

```

```

internal override func calculateOffsets()

{

    super.calculateOffsets()


    // prevent nullpointer when no data set

    if _data === nil

    {

        return

    }


    let radius = adjustedRadius


    let c = adjustedCenterOffsets()


    let shift = (data as? PieChartData)?.dataSet?.selectionShift ?? 0.0


    // create the circle box that will contain the pie-chart (the bounds of the pie-chart)

```

```
_circleBox.origin.x = (c.x - radius) + shift
```

```
_circleBox.origin.y = (c.y - radius) + shift
```

```
_circleBox.size.width = radius * 2 - shift * 2.0
```

```
_circleBox.size.height = radius * 2 - shift * 2.0
```

```
}
```

```
internal override func calcMinMax()
```

```
{
```

```
    calcAngles()
```

```
}
```

```
@objc open override func angleForPoint(x: CGFloat, y: CGFloat) -> CGFloat
```

```
{
```

```
    let c = adjustedCenterOffsets()
```

```
    let tx = Double(x - c.x)
```

```
let ty = Double(y - c.y)
```

```
let length = sqrt(tx * tx + ty * ty)
```

```
let r = acos(ty / length)
```

```
var angle = r.RAD2DEG
```

```
if x > c.x
```

```
{
```

```
    angle = 360.0 - angle
```

```
}
```

```
// add 90° because chart starts EAST
```

```
angle = angle + 90.0
```

```
// neutralize overflow
```

```
if angle > 360.0
```

```
{
```

```
angle = angle - 360.0
```

```
}
```

```
return CGFloat(angle)
```

```
}
```

/// - Returns: The distance of a certain point on the chart to the center of the chart.

@objc open override func distanceToCenter(x: CGFloat, y: CGFloat) -> CGFloat

```
{
```

```
let c = adjustedCenterOffsets()
```

```
var dist = CGFloat(0.0)
```

```
var xDist = CGFloat(0.0)
```

```
var yDist = CGFloat(0.0)
```

```
if x > c.x
```



```
{  
  
    xDist = x - c.x  
  
}  
  
else  
  
{  
  
    xDist = c.x - x  
  
}  
  
if y > c.y  
  
{  
  
    yDist = y - c.y  
  
}  
  
else  
  
{  
  
    yDist = c.y - y  
  
}
```

```

    // pythagoras

    dist = sqrt(pow(xDist, 2.0) + pow(yDist, 2.0))

    return dist

}

open override func getMarkerPosition(highlight: Highlight) -> CGPoint
{

    let center = self.centerCircleBox

    var r = self.radius

    var off = r / 10.0 * 3.6

    if self.isDrawHoleEnabled
    {

        off = (r - (r * self.holeRadiusPercent)) / 2.0

    }

```

```

    r -= off // offset to keep things inside the chart

    let rotationAngle = self.rotationAngle

    let entryIndex = Int(highlight.x)

    // offset needed to center the drawn text in the slice

    let offset = drawAngles[entryIndex] / 2.0

    // calculate the text position

    let x: CGFloat = (r * cos(((rotationAngle + absoluteAngles[entryIndex] - offset) *
CGFloat(_animator.phaseY)).DEG2RAD) + center.x)

    let y: CGFloat = (r * sin(((rotationAngle + absoluteAngles[entryIndex] - offset) *
CGFloat(_animator.phaseY)).DEG2RAD) + center.y)

    return CGPoint(x: x, y: y)

}

```

```

/// calculates the needed angles for the chart slices

private func calcAngles()

{

    _drawAngles = [CGFloat]()

    _absoluteAngles = [CGFloat]()

    guard let data = _data else { return }

    let entryCount = data.entryCount

    _drawAngles.reserveCapacity(entryCount)

    _absoluteAngles.reserveCapacity(entryCount)

    let yValueSum = (_data as! PieChartData).yValueSum

    var cnt = 0

```

```

for set in data.dataSets

{

    for j in 0 ..< set.entryCount

    {

        guard let e = set.entryForIndex(j) else { continue }

        _drawAngles.append(calcAngle(value: abs(e.y), yValueSum: yValueSum))

        if cnt == 0

        {

            _absoluteAngles.append(_drawAngles[cnt])

        }

        else

        {

            _absoluteAngles.append(_absoluteAngles[cnt - 1] + _drawAngles[cnt])

        }
    }
}

```

```

        cnt += 1

    }

}

}

```

/// Checks if the given index is set to be highlighted.

```
@objc open func needsHighlight(index: Int) -> Bool
```

```

{

    return _indicesToHighlight.contains { Int($0.x) == index }

}

```

/// calculates the needed angle for a given value

```
private func calcAngle(_ value: Double) -> CGFloat
```

```

{

    return calcAngle(value: value, yValueSum: (_data as! PieChartData).yValueSum)

}

```

```

/// calculates the needed angle for a given value

private fun calcAngle(value: Double, yValueSum: Double) -> CGFloat

{

    return CGFloat(value) / CGFloat(yValueSum) * _maxAngle

}


/// This will throw an exception, PieChart has no XAxis object.

open override var xAxis: XAxis

{

    fatalError("PieChart has no XAxis")

}


open override fun indexForAngle(_ angle: CGFloat) -> Int

{

    // TODO: Return nil instead of -1

    // take the current angle of the chart into consideration

```

```

    let a = (angle - self.rotationAngle).normalizedAngle

    return _absoluteAngles.firstIndex { $0 > a } ?? -1

}

/// - Returns: The index of the DataSet this x-index belongs to.

@objc open func dataSetIndexForIndex(_ xValue: Double) -> Int

{

    // TODO: Return nil instead of -1

    return _data?.dataSets.firstIndex {

        $0.entryForXValue(xValue, closestToY: .nan) != nil

    } ?? -1

}

/// - Returns: An integer array of all the different angles the chart slices

/// have the angles in the returned array determine how much space (of 360°)

/// each slice takes

@objc open var drawAngles: [CGFloat]

```



```
{

    return _drawAngles

}
```

```
/// - Returns: The absolute angles of the different chart slices (where the

/// slices end)
```

```
@objc open var absoluteAngles: [CGFloat]

{

    return _absoluteAngles

}
```

```
/// The color for the hole that is drawn in the center of the PieChart (if enabled).
```

```
///
```

```
/// - Note: Use holeTransparent with holeColor = nil to make the hole transparent.*
```

```
@objc open var holeColor: NSUIColor?

{

    get
```

```

    {

        return _holeColor

    }

    set

    {

        _holeColor = newValue

        setNeedsDisplay()

    }

}

/// if true, the hole will see-through to the inner tips of the slices

///

/// **default**: `false`

@objc open var drawSlicesUnderHoleEnabled: Bool

{

    get

    {

```

```

        return _drawSlicesUnderHoleEnabled

    }

    set

    {

        _drawSlicesUnderHoleEnabled = newValue

        setNeedsDisplay()

    }

}

```

/// `true` if the inner tips of the slices are visible behind the hole, `false` if not.

```

@objc open var isDrawSlicesUnderHoleEnabled: Bool

{

    return drawSlicesUnderHoleEnabled

}

```

/// `true` if the hole in the center of the pie-chart is set to be visible, `false` if not

```

@objc open var drawHoleEnabled: Bool

```

```

{

    get

    {

        return _drawHoleEnabled

    }

    set

    {

        _drawHoleEnabled = newValue

        setNeedsDisplay()

    }

}

```

/// `true` if the hole in the center of the pie-chart is set to be visible, `false` if not

@objc open var isDrawHoleEnabled: Bool

```

{

    get

    {

```

```

        return drawHoleEnabled

    }

}

/// the text that is displayed in the center of the pie-chart

@objc open var centerText: String?

{

    get

    {

        return self.centerAttributedText?.string

    }

    set

    {

        var attrString: NSMutableAttributedString?

        if newValue == nil

        {

            attrString = nil

```

```

    }

    else

    {

        let    paragraphStyle    =    NSMutableParagraphStyle.default.mutableCopy()    as!
NSMutableParagraphStyle

        paragraphStyle.lineBreakMode = .byTruncatingTail

        paragraphStyle.alignment = .center


        attrString = NSMutableAttributedString(string: newValue!)

        attrString?.setAttributes([

            .foregroundColor: UIColor.labelOrBlack,

            .font: UIFont.systemFont(ofSize: 12.0),

            .paragraphStyle: paragraphStyle

        ], range: NSRange(0, attrString!.length))

    }

    self.centerAttributedText = attrString

}

}

```

/// the text that is displayed in the center of the pie-chart

@objc open var centerAttributedText: NSAttributedString?

```
{  
  
    get  
  
    {  
  
        return _centerAttributedText  
  
    }  
  
    set  
  
    {  
  
        _centerAttributedText = newValue  
  
        setNeedsDisplay()  
  
    }  
  
}
```

/// Sets the offset the center text should have from it's original position in dp. Default x = 0, y = 0

@objc open var centerTextOffset: CGPoint

```

{

    get

    {

        return _centerTextOffset

    }

    set

    {

        _centerTextOffset = newValue

        setNeedsDisplay()

    }

}

/// `true` if drawing the center text is enabled

@objc open var drawCenterTextEnabled: Bool

{

    get

    {

```



```

        return _drawCenterTextEnabled

    }

    set

    {

        _drawCenterTextEnabled = newValue

        setNeedsDisplay()

    }

}

/// `true` if drawing the center text is enabled

@objc open var isDrawCenterTextEnabled: Bool

{

    get

    {

        return drawCenterTextEnabled

    }

}

```

```
internal override var requiredLegendOffset: CGFloat
```

```
{  
  
    return _legend.font.pointSize * 2.0  
  
}
```

```
internal override var requiredBaseOffset: CGFloat
```

```
{  
  
    return 0.0  
  
}
```

```
open override var radius: CGFloat
```

```
{  
  
    return _circleBox.width / 2.0  
  
}
```

```
/// The circlebox, the boundingbox of the pie-chart slices
```

```
@objc open var circleBox: CGRect
```

```
{
```

```
    return _circleBox
```

```
}
```

```
/// The center of the circlebox
```

```
@objc open var centerCircleBox: CGPoint
```

```
{
```

```
    return CGPoint(x: _circleBox.midX, y: _circleBox.midY)
```

```
}
```

```
/// the radius of the hole in the center of the piechart in percent of the maximum radius  
(max = the radius of the whole chart)
```

```
///
```

```
/// default: 0.5 (50%) (half the pie)
```

```
@objc open var holeRadiusPercent: CGFloat
```

```
{
```

```
    get
```

```

    {

        return _holeRadiusPercent

    }

    set

    {

        _holeRadiusPercent = newValue

        setNeedsDisplay()

    }

}

/// The color that the transparent-circle should have.

///

/// **default**: `nil`

@objc open var transparentCircleColor: NSUIColor?

{

    get

    {

```

```

        return _transparentCircleColor
    }

    set

    {

        _transparentCircleColor = newValue

        setNeedsDisplay()

    }

}

```

/// the radius of the transparent circle that is drawn next to the hole in the piechart in percent of the maximum radius (max = the radius of the whole chart)

///

/// **default**: 0.55 (55%) -> means 5% larger than the center-hole by default

@objc open var transparentCircleRadiusPercent: CGFloat

```

{

    get

    {

        return _transparentCircleRadiusPercent
    }
}

```

```

    }

    set

    {

        _transparentCircleRadiusPercent = newValue

        setNeedsDisplay()

    }

}

```

/// The color the entry labels are drawn with.

```
@objc open var entryLabelColor: NSUIColor?
```

```

{

    get { return _entryLabelColor }

    set

    {

        _entryLabelColor = newValue

        setNeedsDisplay()

    }

}

```

```
}
```

```
/// The font the entry labels are drawn with.
```

```
@objc open var entryLabelFont: NSUIFont?
```

```
{
```

```
    get { return _entryLabelFont }
```

```
    set
```

```
{
```

```
        _entryLabelFont = newValue
```

```
        setNeedsDisplay()
```

```
}
```

```
}
```

```
/// Set this to true to draw the entry labels into the pie slices
```

```
@objc open var drawEntryLabelsEnabled: Bool
```

```
{
```

```
    get
```

```

    {

        return _drawEntryLabelsEnabled

    }

    set

    {

        _drawEntryLabelsEnabled = newValue

        setNeedsDisplay()

    }

}

/// `true` if drawing entry labels is enabled, `false` ifnot

@objc open var isDrawEntryLabelsEnabled: Bool

{

    get

    {

        return drawEntryLabelsEnabled

    }

}

```



```
}
```

/// If this is enabled, values inside the PieChart are drawn in percent and not with their original value. Values provided for the ValueFormatter to format are then provided in percent.

```
@objc open var usePercentValuesEnabled: Bool
```

```
{
```

```
    get
```

```
    {
```

```
        return _usePercentValuesEnabled
```

```
    }
```

```
    set
```

```
    {
```

```
        _usePercentValuesEnabled = newValue
```

```
        setNeedsDisplay()
```

```
    }
```

```
}
```

```
/// `true` if drawing x-values is enabled, `false` ifnot
```

```
@objc open var isUsePercentValuesEnabled: Bool
```

```
{
```

```
    get
```

```
    {
```

```
        return usePercentValuesEnabled
```

```
    }
```

```
}
```

```
/// the rectangular radius of the bounding box for the center text, as a percentage of the  
pie hole
```

```
@objc open var centerTextRadiusPercent: CGFloat
```

```
{
```

```
    get
```

```
    {
```

```
        return _centerTextRadiusPercent
```

```
    }
```

```
    set
```

```

{

    _centerTextRadiusPercent = newValue

    setNeedsDisplay()

}

}

```

/// The max angle that is used for calculating the pie-circle.

/// 360 means it's a full pie-chart, 180 results in a half-pie-chart.

/// **default**: 360.0

@objc open var maxAngle: CGFloat

```

{

    get

    {

        return _maxAngle

    }

    set

    {

```

```
        _maxAngle = newValue

        if _maxAngle > 360.0

        {

            _maxAngle = 360.0

        }

        if _maxAngle < 90.0

        {

            _maxAngle = 90.0

        }

    }

}

//

// PieHighlighter.swift

// Charts
```

```

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc(PieChartHighlighter)

open class PieHighlighter: PieRadarHighlighter

{

    open override func closestHighlight(index: Int, x: CGFloat, y: CGFloat) -> Highlight?

    {

        guard

```

```

        let set = chart?.data?.dataSets[0],

        let entry = set.entryForIndex(index)

        else { return nil }

        return Highlight(x: Double(index), y: entry.y, xPx: x, yPx: y, dataSetIndex: 0, axis:
set.axisDependency)

    }

}

//

// PieRadarChartViewBase.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

```

```
import Foundation
```

```
import CoreGraphics
```

```
import QuartzCore
```

```
#if canImport(AppKit)
```

```
import AppKit
```

```
#endif
```

```
/// Base class of PieChartView and RadarChartView.
```

```
open class PieRadarChartViewBase: ChartViewBase
```

```
{
```

```
    /// holds the normalized version of the current rotation angle of the chart
```

```
    private var _rotationAngle = CGFloat(270.0)
```

```
    /// holds the raw version of the current rotation angle of the chart
```

```

private var _rawRotationAngle = CGFloat(270.0)

/// flag that indicates if rotation is enabled or not

@objc open var rotationEnabled = true

/// Sets the minimum offset (padding) around the chart, defaults to 0.0

@objc open var minOffset = CGFloat(0.0)

/// iOS && OSX only: Enabled multi-touch rotation using two fingers.

private var _rotationWithTwoFingers = false

private var _tapGestureRecognizer: NSUITapGestureRecognizer!

#if !os(tvOS)

private var _rotationGestureRecognizer: NSUIRotationGestureRecognizer!

#endif

public override init(frame: CGRect)

```



```
{  
  
    super.init(frame: frame)  
  
}
```

```
public required init?(coder aDecoder: NSCoder)
```

```
{  
  
    super.init(coder: aDecoder)  
  
}
```

```
deinit
```

```
{  
  
    stopDeceleration()  
  
}
```

```
internal override func initialize()
```

```
{  
  
    super.initialize()  
  
}
```

```
        _tapGestureRecognizer = NSUITapGestureRecognizer(target: self, action:
#selector(tapGestureRecognized(_:)))
```

```
self.addGestureRecognizer(_tapGestureRecognizer)
```

```
#if !os(tvOS)
```

```
        _rotationGestureRecognizer = NSUIRotationGestureRecognizer(target: self, action:
#selector(rotationGestureRecognized(_:)))
```

```
self.addGestureRecognizer(_rotationGestureRecognizer)
```

```
_rotationGestureRecognizer.isEnabled = rotationWithTwoFingers
```

```
#endif
```

```
}
```

```
internal override func calcMinMax()
```

```
{
```

```
    /*_xAxis.axisRange = Double((_data?.xVals.count ?? 0) - 1)*/
```

```
}
```

```
open override var maxVisibleCount: Int
```

```
{
```

```
    get
```

```
    {
```

```
        return data?.entryCount ?? 0
```

```
    }
```

```
}
```

```
open override fun notifyDataSetChanged()
```

```
{
```

```
    calcMinMax()
```

```
    if let data = _data , _legend !== nil
```

```
    {
```

```
        legendRenderer.computeLegend(data: data)
```

```
    }
```

```
calculateOffsets()
```

```
setNeedsDisplay()
```

```
}
```

```
internal override func calculateOffsets()
```

```
{
```

```
    var legendLeft = CGFloat(0.0)
```

```
    var legendRight = CGFloat(0.0)
```

```
    var legendBottom = CGFloat(0.0)
```

```
    var legendTop = CGFloat(0.0)
```

```
    if _legend != nil && _legend.enabled && !_legend.drawInside
```

```
    {
```

```
        let fullLegendWidth = min(_legend.neededWidth, _viewPortHandler.chartWidth *  
        _legend.maxSizePercent)
```

```

switch _legend.orientation

{

case .vertical:

    var xLegendOffset: CGFloat = 0.0


    if _legend.horizontalAlignment == .left

        || _legend.horizontalAlignment == .right

    {

        if _legend.verticalAlignment == .center

        {

            // this is the space between the legend and the chart

            let spacing = CGFloat(13.0)


            xLegendOffset = fullLegendWidth + spacing

        }

        else

```

```

{

    // this is the space between the legend and the chart

    let spacing = CGFloat(8.0)

    let legendWidth = fullLegendWidth + spacing

    let legendHeight = _legend.neededHeight + _legend.textHeightMax


    let c = self.midPoint


    let bottomX = _legend.horizontalAlignment == .right

        ? self.bounds.width - legendWidth + 15.0

        : legendWidth - 15.0

    let bottomY = legendHeight + 15

    let distLegend = distanceToCenter(x: bottomX, y: bottomY)


    let reference = getPosition(center: c, dist: self.radius,

                                angle: angleForPoint(x: bottomX, y: bottomY))

```

```

let distReference = distanceToCenter(x: reference.x, y: reference.y)

let minOffset = CGFloat(5.0)

if bottomY >= c.y

    && self.bounds.height - legendWidth > self.bounds.width

{

    xLegendOffset = legendWidth

}

else if distLegend < distReference

{

    let diff = distReference - distLegend

    xLegendOffset = minOffset + diff

}

}

```

```

switch _legend.horizontalAlignment
{

case .left:

    legendLeft = xLegendOffset


case .right:

    legendRight = xLegendOffset


case .center:


switch _legend.verticalAlignment
{

case .top:

    legendTop = min(_legend.neededHeight, _viewPortHandler.chartHeight *
_legend.maxSizePercent)


case .bottom:

```



```
        legendBottom = min(_legend.neededHeight,  
_viewPortHandler.chartHeight * _legend.maxSizePercent)
```

```
    default:
```

```
        break
```

```
    }
```

```
}
```

```
case .horizontal:
```

```
    var yLegendOffset: CGFloat = 0.0
```

```
    if _legend.verticalAlignment == .top
```

```
        || _legend.verticalAlignment == .bottom
```

```
{
```

```
    // It's possible that we do not need this offset anymore as it
```

```
    // is available through the extraOffsets, but changing it can mean
```

```
    // changing default visibility for existing apps.
```

```

let yOffset = self.requiredLegendOffset

yLegendOffset = min(

    _legend.neededHeight + yOffset,

    _viewPortHandler.chartHeight * _legend.maxSizePercent)

}

switch _legend.verticalAlignment

{

case .top:

    legendTop = yLegendOffset

case .bottom:

    legendBottom = yLegendOffset

```

```
default:
```

```
break
```

```
}
```

```
}
```

```
legendLeft += self.requiredBaseOffset
```

```
legendRight += self.requiredBaseOffset
```

```
legendTop += self.requiredBaseOffset
```

```
legendBottom += self.requiredBaseOffset
```

```
}
```

```
legendTop += self.extraTopOffset
```

```
legendRight += self.extraRightOffset
```

```
legendBottom += self.extraBottomOffset
```

```
legendLeft += self.extraLeftOffset
```

```
var minOffset = self.minOffset
```

```

if self is RadarChartView

{

    let x = self.xAxis


    if x.isEnabled && x.drawLabelsEnabled

    {

        minOffset = max(minOffset, x.labelRotatedWidth)

    }

}


let offsetLeft = max(minOffset, legendLeft)

let offsetTop = max(minOffset, legendTop)

let offsetRight = max(minOffset, legendRight)

let offsetBottom = max(minOffset, max(self.requiredBaseOffset, legendBottom))


    _viewPortHandler.restrainViewPort(offsetLeft: offsetLeft, offsetTop: offsetTop,
offsetRight: offsetRight, offsetBottom: offsetBottom)

```

```
}
```

/// - Returns: The angle relative to the chart center for the given point on the chart in degrees.

/// The angle is always between 0 and 360°, 0° is NORTH, 90° is EAST, ...

```
@objc open func angleForPoint(x: CGFloat, y: CGFloat) -> CGFloat
```

```
{
```

```
    let c = centerOffsets
```

```
    let tx = Double(x - c.x)
```

```
    let ty = Double(y - c.y)
```

```
    let length = sqrt(tx * tx + ty * ty)
```

```
    let r = acos(ty / length)
```

```
    var angle = r.RAD2DEG
```

```
    if x > c.x
```

```
{
```

```

        angle = 360.0 - angle

    }

    // add 90° because chart starts EAST

    angle = angle + 90.0

    // neutralize overflow

    if angle > 360.0

    {

        angle = angle - 360.0

    }

    return CGFloat(angle)

}

/// Calculates the position around a center point, depending on the distance

/// from the center, and the angle of the position around the center.

```

```
@objc open func getPosition(center: CGPoint, dist: CGFloat, angle: CGFloat) ->
CGPoint
```

```
{

    return CGPoint(x: center.x + dist * cos(angle.DEG2RAD),

        y: center.y + dist * sin(angle.DEG2RAD))

}
```

/// - Returns: The distance of a certain point on the chart to the center of the chart.

```
@objc open func distanceToCenter(x: CGFloat, y: CGFloat) -> CGFloat
```

```
{

    let c = self.centerOffsets
```

```
    var dist = CGFloat(0.0)
```

```
    var xDist = CGFloat(0.0)
```

```
    var yDist = CGFloat(0.0)
```

```
    if x > c.x
```

```
{  
  
    xDist = x - c.x  
  
}  
  
else  
  
    {  
  
        xDist = c.x - x  
  
    }  
  
    if y > c.y  
  
        {  
  
            yDist = y - c.y  
  
        }  
  
    else  
  
        {  
  
            yDist = c.y - y  
  
        }
```



```

// pythagoras

dist = sqrt(pow(xDist, 2.0) + pow(yDist, 2.0))

return dist

}

/// - Returns: The xIndex for the given angle around the center of the chart.

/// -1 if not found / outofbounds.

@objc open func indexForAngle(_ angle: CGFloat) -> Int

{

    fatalError("indexForAngle() cannot be called on PieRadarChartViewBase")

}

/// current rotation angle of the pie chart

///

/// **default**: 270 --> top (NORTH)

/// Will always return a normalized value, which will be between 0.0 < 360.0

```

```

@objc open var rotationAngle: CGFloat

{

    get

    {

        return _rotationAngle

    }

    set

    {

        _rawRotationAngle = newValue

        _rotationAngle = newValue.normalizedAngle

        setNeedsDisplay()

    }

}

```

/// gets the raw version of the current rotation angle of the pie chart the returned value could be any value, negative or positive, outside of the 360 degrees.

/// this is used when working with rotation direction, mainly by gestures and animations.

```

@objc open var rawRotationAngle: CGFloat

```

```

{

    return _rawRotationAngle

}

```

/// The diameter of the pie- or radar-chart

```
@objc open var diameter: CGFloat
```

```

{

    var content = _viewPortHandler.contentRect

    content.origin.x += extraLeftOffset

    content.origin.y += extraTopOffset

    content.size.width -= extraLeftOffset + extraRightOffset

    content.size.height -= extraTopOffset + extraBottomOffset

    return min(content.width, content.height)

}

```

/// The radius of the chart in pixels.

```
@objc open var radius: CGFloat
```

```

{

    fatalError("radius cannot be called on PieRadarChartViewBase")

}


/// The required offset for the chart legend.

internal var requiredLegendOffset: CGFloat

{

    fatalError("requiredLegendOffset cannot be called on PieRadarChartViewBase")

}


/// - Returns: The base offset needed for the chart without calculating the

/// legend size.

internal var requiredBaseOffset: CGFloat

{

    fatalError("requiredBaseOffset cannot be called on PieRadarChartViewBase")

}

```

```
open override var chartYMax: Double
```

```
{
```

```
    return 0.0
```

```
}
```

```
open override var chartYMin: Double
```

```
{
```

```
    return 0.0
```

```
}
```

```
@objc open var isRotationEnabled: Bool { return rotationEnabled }
```

```
/// flag that indicates if rotation is done with two fingers or one.
```

```
/// when the chart is inside a scrollview, you need a two-finger rotation because a one-  
finger rotation eats up all touch events.
```

```
///
```

```
/// On iOS this will disable one-finger rotation.
```

```
/// On OSX this will keep two-finger multitouch rotation, and one-pointer mouse rotation.
```

```

///

/// **default**: false

@objc open var rotationWithTwoFingers: Bool

{

    get

    {

        return _rotationWithTwoFingers

    }

    set

    {

        _rotationWithTwoFingers = newValue

        #if !os(tvOS)

            _rotationGestureRecognizer.isEnabled = _rotationWithTwoFingers

        #endif

    }

}

```

```
/// flag that indicates if rotation is done with two fingers or one.
```

```
/// when the chart is inside a scrollview, you need a two-finger rotation because a one-  
finger rotation eats up all touch events.
```

```
///
```

```
/// On iOS this will disable one-finger rotation.
```

```
/// On OSX this will keep two-finger multitouch rotation, and one-pointer mouse rotation.
```

```
///
```

```
/// default: false
```

```
@objc open var isRotationWithTwoFingers: Bool
```

```
{
```

```
    return _rotationWithTwoFingers
```

```
}
```

```
// MARK: - Animation
```

```
private var _spinAnimator: Animator!
```

```
/// Applies a spin animation to the Chart.
```

```
@objc open func spin(duration: TimeInterval, fromAngle: CGFloat, toAngle: CGFloat,  
easing: ChartEasingFunctionBlock?)
```

```
{  
  
    if _spinAnimator != nil  
  
    {  
  
        _spinAnimator.stop()  
  
    }  
  
    _spinAnimator = Animator()  
  
    _spinAnimator.updateBlock = {  
  
        self.rotationAngle = (toAngle - fromAngle) * CGFloat(self._spinAnimator.phaseX)  
+ fromAngle  
  
    }  
  
    _spinAnimator.stopBlock = { self._spinAnimator = nil }  
  
    _spinAnimator.animate(xAxisDuration: duration, easing: easing)  
  
}
```



```
@objc open func spin(duration: TimeInterval, fromAngle: CGFloat, toAngle: CGFloat,
easingOption: ChartEasingOption)
```

```
{
```

```
    spin(duration: duration, fromAngle: fromAngle, toAngle: toAngle, easing:
easingFunctionFromOption(easingOption))
```

```
}
```

```
@objc open func spin(duration: TimeInterval, fromAngle: CGFloat, toAngle: CGFloat)
```

```
{
```

```
    spin(duration: duration, fromAngle: fromAngle, toAngle: toAngle, easing: nil)
```

```
}
```

```
@objc open func stopSpinAnimation()
```

```
{
```

```
    if _spinAnimator != nil
```

```
    {
```

```
        _spinAnimator.stop()
```

```
    }
```

```
}
```

```
// MARK: - Gestures
```

```
private var _rotationGestureStartPoint: CGPoint!
```

```
private var _isRotating = false
```

```
private var _startAngle = CGFloat(0.0)
```

```
private struct AngularVelocitySample
```

```
{
```

```
    var time: TimeInterval
```

```
    var angle: CGFloat
```

```
}
```

```
private var velocitySamples = [AngularVelocitySample]()
```

```
private var _decelerationLastTime: TimeInterval = 0.0
```

```

private var _decelerationDisplayLink: NSUIDisplayLink!

private var _decelerationAngularVelocity: CGFloat = 0.0


internal final func processRotationGestureBegan(location: CGPoint)

{

    self.resetVelocity()


    if rotationEnabled

    {

        self.sampleVelocity(touchLocation: location)

    }


    self.setGestureStartAngle(x: location.x, y: location.y)


    _rotationGestureStartPoint = location

}

```

```

internal final func processRotationGestureMoved(location: CGPoint)

{

    if isDragDecelerationEnabled

    {

        sampleVelocity(touchLocation: location)

    }


    if !_isRotating &&

        distance(

            eventX: location.x,

            startX: _rotationGestureStartPoint.x,

            eventY: location.y,

            startY: _rotationGestureStartPoint.y) > CGFloat(8.0)

    {

        _isRotating = true

    }

    else

```

```

{

    self.updateGestureRotation(x: location.x, y: location.y)

    setNeedsDisplay()

}

}

internal final func processRotationGestureEnded(location: CGPoint)

{

    if isDragDecelerationEnabled

    {

        stopDeceleration()

        sampleVelocity(touchLocation: location)

        _decelerationAngularVelocity = calculateVelocity()

        if _decelerationAngularVelocity != 0.0

```

```

    {

        _decelerationLastTime = CACurrentMediaTime()

        _decelerationDisplayLink = NSUIDisplayLink(target: self, selector:
#selector(PieRadarChartViewBase.decelerationLoop))

        _decelerationDisplayLink.add(to: RunLoop.main, forMode:
RunLoop.Mode.common)

    }

}

}

```

```

internal final func processRotationGestureCancelled()

```

```

{

    if _isRotating

    {

        _isRotating = false

    }

}

```

```

#if !os(OSX)

    open override func nsuiTouchesBegan(_ touches: Set<NSUITouch>, withEvent event:
    NSUIEvent?)

    {

        // if rotation by touch is enabled

        if rotationEnabled

        {

            stopDeceleration()

            if !rotationWithTwoFingers, let touchLocation = touches.first?.location(in: self)

            {

                processRotationGestureBegan(location: touchLocation)

            }

        }

        if !_isRotating

        {

            super.nsuiTouchesBegan(touches, withEvent: event)

```

```
}
```

```
}
```

```
open override func nsuiTouchesMoved(_ touches: Set<NSUITouch>, withEvent event:  
NSUIEvent?)
```

```
{
```

```
    if rotationEnabled && !rotationWithTwoFingers, let touch = touches.first
```

```
    {
```

```
        let touchLocation = touch.location(in: self)
```

```
        processRotationGestureMoved(location: touchLocation)
```

```
    }
```

```
    if !_isRotating
```

```
    {
```

```
        super.nsuiTouchesMoved(touches, withEvent: event)
```

```
    }
```

```
}
```



```
open override func nsuiTouchesEnded(_ touches: Set<NSUITouch>, withEvent event:
NSUIEvent?)
```

```
{
```

```
    if !_isRotating
```

```
    {
```

```
        super.nsuiTouchesEnded(touches, withEvent: event)
```

```
    }
```

```
    if rotationEnabled && !rotationWithTwoFingers, let touch = touches.first
```

```
    {
```

```
        let touchLocation = touch.location(in: self)
```

```
        processRotationGestureEnded(location: touchLocation)
```

```
    }
```

```
    if _isRotating
```

```
    {
```

```
        _isRotating = false
```

```
    }
```

```
}
```

```
open override func nsuiTouchesCancelled(_ touches: Set<NSUITouch>?, withEvent  
event: NSUIEvent?)
```

```
{
```

```
    super.nsuiTouchesCancelled(touches, withEvent: event)
```

```
    processRotationGestureCancelled()
```

```
}
```

```
#endif
```

```
#if os(OSX)
```

```
open override func mouseDown(with theEvent: NSEvent)
```

```
{
```

```
    // if rotation by touch is enabled
```

```
    if rotationEnabled
```

```
    {
```

```
        stopDeceleration()
```

```
let location = self.convert(theEvent.locationInWindow, from: nil)
```

```
processRotationGestureBegan(location: location)
```

```
}
```

```
if !_isRotating
```

```
{
```

```
super.mouseDown(with: theEvent)
```

```
}
```

```
}
```

```
open override func mouseDragged(with theEvent: NSEvent)
```

```
{
```

```
if rotationEnabled
```

```
{
```

```
let location = self.convert(theEvent.locationInWindow, from: nil)
```

```

        processRotationGestureMoved(location: location)

    }

    if !_isRotating

    {

        super.mouseDragged(with: theEvent)

    }

}

```

```

open override func mouseUp(with theEvent: NSEvent)

{

    if !_isRotating

    {

        super.mouseUp(with: theEvent)

    }

}

```

```

if rotationEnabled

{

    let location = self.convert(theEvent.locationInWindow, from: nil)


    processRotationGestureEnded(location: location)

}


if _isRotating

{

    _isRotating = false

}

}

#endif


private func resetVelocity()

{

    velocitySamples.removeAll(keepingCapacity: false)

```

```
}
```

```
private func sampleVelocity(touchLocation: CGPoint)
```

```
{
```

```
    let currentSample: AngularVelocitySample = {
```

```
        let time = CACurrentMediaTime()
```

```
        let angle = angleForPoint(x: touchLocation.x, y: touchLocation.y)
```

```
        return AngularVelocitySample(time: time, angle: angle)
```

```
    }()
```

```
// Remove samples older than our sample time - 1 seconds
```

```
// while keeping at least one sample
```

```
var i = 0, count = velocitySamples.count
```

```
while (i < count - 2)
```

```
{
```

```
    if currentSample.time - velocitySamples[i].time > 1.0
```

```

    {

        velocitySamples.remove(at: 0)

        i -= 1

        count -= 1

    }

    else

    {

        break

    }

    i += 1

}

velocitySamples.append(currentSample)

}

```

```
private func calculateVelocity() -> CGFloat
```

```

{

guard var firstSample = velocitySamples.first,

var lastSample = velocitySamples.last

else { return 0 }


// Look for a sample that's closest to the latest sample, but not the same, so we can
deduce the direction

let beforeLastSample = velocitySamples.last { $0.angle != lastSample.angle }

?? firstSample


// Calculate the sampling time

let timeDelta: CGFloat = {

let delta = CGFloat(lastSample.time - firstSample.time)

return delta == 0 ? 0.1 : delta

}()


// Calculate clockwise/ccw by choosing two values that should be closest to each other,

// so if the angles are two far from each other we know they are inverted "for sure"

```



```

let isClockwise: Bool = {

    let isClockwise = lastSample.angle >= beforeLastSample.angle

    let isInverted = abs(lastSample.angle - beforeLastSample.angle) > 270.0

    return isInverted ? !isClockwise : isClockwise

}()

```

// Now if the "gesture" is over a too big of an angle - then we know the angles are inverted, and we need to move them closer to each other from both sides of the 360.0 wrapping point

```

if lastSample.angle - firstSample.angle > 180.0

{

    firstSample.angle += 360.0

}

else if firstSample.angle - lastSample.angle > 180.0

{

    lastSample.angle += 360.0

}

```

```

// The velocity

let velocity = abs((lastSample.angle - firstSample.angle) / timeDelta)

return isClockwise ? velocity : -velocity

}

/// sets the starting angle of the rotation, this is only used by the touch listener, x and y
is the touch position

private func setGestureStartAngle(x: CGFloat, y: CGFloat)

{

    _startAngle = angleForPoint(x: x, y: y)

    // take the current angle into consideration when starting a new drag

    _startAngle -= _rotationAngle

}

/// updates the view rotation depending on the given touch position, also takes the starting
angle into consideration

private func updateGestureRotation(x: CGFloat, y: CGFloat)

```

```

{

    self.rotationAngle = angleForPoint(x: x, y: y) - _startAngle

}


@objc open func stopDeceleration()

{

    if _decelerationDisplayLink !== nil

    {

        _decelerationDisplayLink.remove(from: RunLoop.main, forMode:
RunLoop.Mode.common)

        _decelerationDisplayLink = nil

    }

}


@objc private func decelerationLoop()

{

    let currentTime = CACurrentMediaTime()

```

```
_decelerationAngularVelocity *= self.dragDecelerationFrictionCoef
```

```
let timeInterval = CGFloat(currentTime - _decelerationLastTime)
```

```
self.rotationAngle += _decelerationAngularVelocity * timeInterval
```

```
_decelerationLastTime = currentTime
```

```
if(abs(_decelerationAngularVelocity) < 0.001)
```

```
{
```

```
    stopDeceleration()
```

```
}
```

```
}
```

```
/// - Returns: The distance between two points
```

```
private func distance(eventX: CGFloat, startX: CGFloat, eventY: CGFloat, startY:  
CGFloat) -> CGFloat
```

```
{
```

```

    let dx = eventX - startX

    let dy = eventY - startY

    return sqrt(dx * dx + dy * dy)

}

```

/// - Returns: The distance between two points

```

private func distance(from: CGPoint, to: CGPoint) -> CGFloat

{

    let dx = from.x - to.x

    let dy = from.y - to.y

    return sqrt(dx * dx + dy * dy)

}

```

/// reference to the last highlighted object

```

private var _lastHighlight: Highlight!

```

```

@objc private func tapGestureRecognized(_ recognizer: NSUITapGestureRecognizer)

```

```

{

    if recognizer.state == NSUIGestureRecognizerState.ended

    {

        if !self.isHighlightPerTapEnabled { return }

        let location = recognizer.location(in: self)

        let high = self.getHighlightByTouchPoint(location)

        self.highlightValue(high, callDelegate: true)

    }

}

#if !os(tvOS)

@objc private func rotationGestureRecognized(_ recognizer:
NSUIRotationGestureRecognizer)

{

    if recognizer.state == NSUIGestureRecognizerState.began

    {

```

```
stopDeceleration()
```

```
    _startAngle = self.rawRotationAngle
```

```
}
```

```
    if recognizer.state == NSUIGestureRecognizerState.began || recognizer.state ==  
    NSUIGestureRecognizerState.changed
```

```
{
```

```
    let angle = recognizer.nsuiRotation.RAD2DEG
```

```
    self.rotationAngle = _startAngle + angle
```

```
    setNeedsDisplay()
```

```
}
```

```
else if recognizer.state == NSUIGestureRecognizerState.ended
```

```
{
```

```
    let angle = recognizer.nsuiRotation.RAD2DEG
```

```
    self.rotationAngle = _startAngle + angle
```

```
setNeedsDisplay()
```

```
if isDragDecelerationEnabled
```

```
{
```

```
    stopDeceleration()
```

```
    _decelerationAngularVelocity = recognizer.velocity.RAD2DEG
```

```
    if _decelerationAngularVelocity != 0.0
```

```
    {
```

```
        _decelerationLastTime = CACurrentMediaTime()
```

```
        _decelerationDisplayLink = NSUIDisplayLink(target: self, selector:  
#selector(PieRadarChartViewBase.decelerationLoop))
```

```
        _decelerationDisplayLink.add(to: RunLoop.main, forMode:  
RunLoop.Mode.common)
```

```
    }
```

```
}
```

```
}
```



```
}

#endif

}

//

// PieRadarHighlighter.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics
```

```
@objc(PieRadarChartHighlighter)
```

```
open class PieRadarHighlighter: ChartHighlighter
```

```
{
```

```
    open override func getHighlight(x: CGFloat, y: CGFloat) -> Highlight?
```

```
    {
```

```
        guard let chart = self.chart as? PieRadarChartViewBase else { return nil }
```

```
        let touchDistanceToCenter = chart.distanceToCenter(x: x, y: y)
```

```
        // check if a slice was touched
```

```
        guard touchDistanceToCenter <= chart.radius else
```

```
        {
```

```
            // if no slice was touched, highlight nothing
```

```
            return nil
```

```
        }
```

```
        var angle = chart.angleForPoint(x: x, y: y)
```

```

if chart is PieChartView

{

    angle /= CGFloat(chart.chartAnimator.phaseY)

}

let index = chart.indexForAngle(angle)

// check if the index could be found

if index < 0 || index >= chart.data?.maxEntryCountSet?.entryCount ?? 0

{

    return nil

}

else

{

    return closestHighlight(index: index, x: x, y: y)

}

```

```

    }

    /// - Parameters:

    /// - index:

    /// - x:

    /// - y:

    /// - Returns: The closest Highlight object of the given objects based on the touch position
    inside the chart.

    @objc open func closestHighlight(index: Int, x: CGFloat, y: CGFloat) -> Highlight?

    {

        fatalError("closestHighlight(index, x, y) cannot be called on
        PieRadarChartHighlighter")

    }

}

//

// Platform+Accessibility.swift

// Charts

```

```
//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

#if os(iOS) || os(tvOS)

#if canImport(UiKit)

    import UIKit

#endif

#endif

internal func accessibilityPostLayoutChangedNotification(withElement element: Any? =
nil)

{
```

```

        UIAccessibility.post(notification: UIAccessibility.Notification.layoutChanged,
argument: element)

    }

```

```

internal fun accessibilityPostScreenChangedNotification(withElement element: Any? =
nil)

```

```

{

    UIAccessibility.post(notification: UIAccessibility.Notification.screenChanged,
argument: element)

}

```

```

/// A simple abstraction over UIAccessibilityElement and NSAccessibilityElement.

```

```

open class NSUIAccessibilityElement: UIAccessibilityElement

```

```

{

    private weak var containerView: UIView?

```

```

    final var isHeader: Bool = false

```

```

    {

        didSet

```

```

    {
        accessibilityTraits = isHeader ? UIAccessibilityTraits.header :
UIAccessibilityTraits.none
    }
}

```

```

final var isSelected: Bool = false

```

```

    {
        didSet
        {
            accessibilityTraits = isSelected ? UIAccessibilityTraits.selected :
UIAccessibilityTraits.none
        }
    }
}

```

```

override public init(accessibilityContainer container: Any)

```

```

{
    // We can force unwrap since all chart views are subclasses of UIView

```

```

        containerView = (container as! UIView)

        super.init(accessibilityContainer: container)

    }

    override open var accessibilityFrame: CGRect

    {

        get

        {

            return super.accessibilityFrame

        }

        set

        {

            guard let containerView = containerView else { return }

            super.accessibilityFrame = containerView.convert(newValue, to:
UIScreen.main.coordinateSpace)

        }

    }

```



```
}
```

```
extension NSUIView
```

```
{
```

```
    /// An array of accessibilityElements that is used to implement UIAccessibilityContainer internally.
```

```
    /// Subclasses MUST override this with an array of such elements.
```

```
    @objc open func accessibilityChildren() -> [Any]?
```

```
{
```

```
    return nil
```

```
}
```

```
public final override var isAccessibilityElement: Bool
```

```
{
```

```
    get { return false } // Return false here, so we can make individual elements accessible
```

```
    set { }
```

```
}
```

```
open override fun accessibilityElementCount() -> Int
```

```
{  
  
    return accessibilityChildren()?.count ?? 0  
  
}
```

```
open override fun accessibilityElement(at index: Int) -> Any?
```

```
{  
  
    return accessibilityChildren()?.[index]  
  
}
```

```
open override fun index(ofAccessibilityElement element: Any) -> Int
```

```
{  
  
    guard let axElement = element as? NSUIAccessibilityElement else { return  
    NSNotFound }  
  
    return (accessibilityChildren() as? [NSUIAccessibilityElement])?  
  
        .firstIndex(of: axElement) ?? NSNotFound  
  
}
```

```
}
```

```
#endif
```

```
#if os(OSX)
```

```
#if canImport(AppKit)
```

```
import AppKit
```

```
#endif
```

```
internal func accessibilityPostLayoutChangedNotification(withElement element: Any? =  
nil)
```

```
{
```

```
    guard let validElement = element else { return }
```

```
    NSAccessibility.post(element: validElement, notification: .layoutChanged)
```

```
}
```

```
internal func accessibilityPostScreenChangedNotification(withElement element: Any? =  
nil)
```

```

{

    // Placeholder

}

/// A simple abstraction over UIAccessibilityElement and NSAccessibilityElement.

open class NSUIAccessibilityElement: NSAccessibilityElement

{

    private weak var containerView: NSView?

    final var isHeader: Bool = false

    {

        didSet

        {

            setAccessibilityRole(isHeader ? .staticText : .none)

        }

    }
}

```

```
final var isSelected: Bool = false

{

    didSet

    {

        setAccessibilitySelected(isSelected)

    }

}
```

```
open var accessibilityLabel: String

{

    get

    {

        return accessibilityLabel() ?? ""

    }

}
```

```
set

{
```

```

        setAccessibilityLabel(newValue)

    }

}

open var accessibilityFrame: NSRect

{

    get

    {

        return accessibilityFrame()

    }

    set

    {

        guard let containerView = containerView else { return }

        let bounds = NSAccessibility.screenRect(fromView: containerView, rect:
newValue)

```

// This works, but won't auto update if the window is resized or moved.

// setAccessibilityFrame(bounds)

// using FrameInParentSpace allows for automatic updating of frame when windows are moved and resized.

// However, there seems to be a bug right now where using it causes an offset in the frame.

// This is a slightly hacky workaround that calculates the offset and removes it from frame calculation.

setAccessibilityFrameInParentSpace(bounds)

let axFrame = accessibilityFrame()

let widthOffset = abs(axFrame.origin.x - bounds.origin.x)

let heightOffset = abs(axFrame.origin.y - bounds.origin.y)

let rect = NSRect(x: bounds.origin.x - widthOffset,

y: bounds.origin.y - heightOffset,

width: bounds.width,

height: bounds.height)

setAccessibilityFrameInParentSpace(rect)

}

```
}
```

```
public init(accessibilityContainer container: Any)
```

```
{
```

```
    // We can force unwrap since all chart views are subclasses of NSView
```

```
    containerView = (container as! NSView)
```

```
    super.init()
```

```
    setAccessibilityParent(containerView)
```

```
    setAccessibilityRole(.row)
```

```
}
```

```
}
```

```
/// - Note: setAccessibilityRole(.list) is called at init. See Platform.swift.
```

```
extension NSUIView: NSAccessibilityGroup
```

```
{
```



```
open override func accessibilityLabel() -> String?

{

    return "Chart View"

}


open override func accessibilityRows() -> [Any]?

{

    return accessibilityChildren()

}

}


#endif


//

// Platform+Color.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

#if canImport(Uikit)

import UIKit

public typealias NSUIColor = UIColor

private func fetchLabelColor() -> UIColor

{

    if #available(iOS 13, tvOS 13, *)

    {

        return .label

    }

    else

```

```

    {

        return .black

    }

}

private let labelColor: UIColor = fetchLabelColor()

extension UIColor

{

    static var labelOrBlack: UIColor { labelColor }

}

#endif

#if canImport(AppKit) && !targetEnvironment(macCatalyst)

import AppKit

public typealias NSUIColor = NSColor

```

```

private func fetchLabelColor() -> NSColor

{

    if #available(macOS 10.14, *)

    {

        return .labelColor

    }

    else

    {

        return .black

    }

}

private let labelColor: NSColor = fetchLabelColor()


extension NSColor

{

    static var labelOrBlack: NSColor { labelColor }

}

```

```
#endif

//

// Platform+Gestures.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

// MARK: - UIKit

#if canImport(Uikit)

import UIKit

public typealias NSUIGestureRecognizer = UIGestureRecognizer
```

```
public typealias NSUIGestureRecognizerState = UIGestureRecognizer.State

public typealias NSUIGestureRecognizerDelegate = UIGestureRecognizerDelegate

public typealias NSUITapGestureRecognizer = UITapGestureRecognizer

public typealias NSUIPanGestureRecognizer = UIPanGestureRecognizer
```

```
extension NSUITapGestureRecognizer

{

    @objc final fun nsuiNumberOfTouches() -> Int

    {

        return numberOfTouches

    }

}
```

```
@objc final var nsuiNumberOfTapsRequired: Int

{

    get

    {

        return self.numberOfTapsRequired

    }

}
```

```

    }

    set

    {

        self.numberOfTapsRequired = newValue

    }

}

}

```

```

extension NSUIPanGestureRecognizer

```

```

{

    @objc final func nsuiNumberOfTouches() -> Int

    {

        return numberOfTouches

    }

}

```

```

    @objc final func nsuiLocationOfTouch(_ touch: Int, inView: UIView?) -> CGPoint

```

```

{

```

```

        return super.location(ofTouch: touch, in: inView)

    }

}

#if !os(tvOS)

public typealias NSUIPinchGestureRecognizer = UIPinchGestureRecognizer

public typealias NSUIRotationGestureRecognizer = UIRotationGestureRecognizer

extension NSUIRotationGestureRecognizer

{

    @objc final var nsuiRotation: CGFloat

    {

        get { return rotation }

        set { rotation = newValue }

    }

}

```



```
extension NSUIPinchGestureRecognizer
```

```
{
```

```
    @objc final var nsuiScale: CGFloat
```

```
    {
```

```
        get
```

```
        {
```

```
            return scale
```

```
        }
```

```
        set
```

```
        {
```

```
            scale = newValue
```

```
        }
```

```
    }
```

```
    @objc final func nsuiLocationOfTouch(_ touch: Int, inView: UIView?) -> CGPoint
```

```
    {
```

```
        return super.location(ofTouch: touch, in: inView)
```

```

    }

}

#endif

#endif

// MARK: - AppKit

#if canImport(AppKit) && !targetEnvironment(macCatalyst)

import AppKit

public typealias NSUIGestureRecognizer = NSGestureRecognizer

public typealias NSUIGestureRecognizerState = NSGestureRecognizer.State

public typealias NSUIGestureRecognizerDelegate = NSGestureRecognizerDelegate

public typealias NSUITapGestureRecognizer = NSClickGestureRecognizer

public typealias NSUIPanGestureRecognizer = NSPanGestureRecognizer

public typealias NSUIPinchGestureRecognizer = NSMagnificationGestureRecognizer

public typealias NSUIRotationGestureRecognizer = NSRotationGestureRecognizer

```

```

/** The 'tap' gesture is mapped to clicks. */

extension NSUITapGestureRecognizer

{

    final func nsuiNumberOfTouches() -> Int

    {

        return 1

    }

    final var nsuiNumberOfTapsRequired: Int

    {

        get

        {

            return self.numberOfClicksRequired

        }

        set

        {

            self.numberOfClicksRequired = newValue

        }

    }

}

```

```

    }

}

}

```

extension NSUIPanGestureRecognizer

```

{

    final func nsuiNumberOfTouches() -> Int

    {

        return 1

    }

}

```

/// FIXME: Currently there are no more than 1 touch in OSX gestures, and not way to create custom touch gestures.

```

    final func nsuiLocationOfTouch(_ touch: Int, inView: NSView?) -> NSPoint

    {

        return super.location(in: inView)

    }

}

```

```
extension NSUIRotationGestureRecognizer
```

```
{
```

```
    /// FIXME: Currently there are no velocities in OSX gestures, and not way to create  
    custom touch gestures.
```

```
    final var velocity: CGFloat
```

```
{
```

```
    return 0.1
```

```
}
```

```
    final var nsuiRotation: CGFloat
```

```
{
```

```
    get { return -rotation }
```

```
    set { rotation = -newValue }
```

```
}
```

```
}
```

```
extension NSUIPinchGestureRecognizer
```

```

{

    final var nsuiScale: CGFloat

    {

        get

        {

            return magnification + 1.0

        }

        set

        {

            magnification = newValue - 1.0

        }

    }

}

```

/// FIXME: Currently there are no more than 1 touch in OSX gestures, and not way to create custom touch gestures.

```

final func nsuiLocationOfTouch(_ touch: Int, inView view: NSView?) -> NSPoint

{

    return super.location(in: view)

}

```

```
    }

}

#endif

//

// Platform+Graphics.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

enum Orientation

{

    case portrait, landscape
```

```
}
```

```
extension CGSize
```

```
{
```

```
    var orientation: Orientation { return width > height ? .landscape : .portrait }
```

```
}
```

```
extension CGRect
```

```
{
```

```
    var orientation: Orientation { size.orientation }
```

```
}
```

```
// MARK: - UIKit
```

```
#if canImport(Uikit)
```

```
import UIKit
```

```
func NSUIGraphicsGetCurrentContext() -> CGContext?
```



```
{  
  
    return UIGraphicsGetCurrentContext()  
  
}
```

```
func NSUIGraphicsGetImageFromCurrentImageContext() -> NSUIImage!
```

```
{  
  
    return UIGraphicsGetImageFromCurrentImageContext()  
  
}
```

```
func NSUIGraphicsPushContext(_ context: CGContext)
```

```
{  
  
    UIGraphicsPushContext(context)  
  
}
```

```
func NSUIGraphicsPopContext()
```

```
{  
  
    UIGraphicsPopContext()  
  
}
```

```
}
```

```
func NSUIGraphicsEndImageContext()
```

```
{
```

```
    UIGraphicsEndImageContext()
```

```
}
```

```
func NSUIImagePNGRepresentation(_ image: NSUIImage) -> Data?
```

```
{
```

```
    return image.pngData()
```

```
}
```

```
func NSUIImageJPEGRepresentation(_ image: NSUIImage, _ quality: CGFloat = 0.8) ->  
Data?
```

```
{
```

```
    return image.jpegData(compressionQuality: quality)
```

```
}
```

```
func NSUIGraphicsBeginImageContextWithOptions(_ size: CGSize, _ opaque: Bool, _
scale: CGFloat)
```

```
{
```

```
    UIGraphicsBeginImageContextWithOptions(size, opaque, scale)
```

```
}
```

```
#endif
```

```
// MARK: - AppKit
```

```
#if canImport(AppKit) && !targetEnvironment(macCatalyst)
```

```
import AppKit
```

```
func NSUIGraphicsGetCurrentContext() -> CGContext?
```

```
{
```

```
    return NSGraphicsContext.current?.cgContext
```

```
}
```

```
func NSUIGraphicsPushContext(_ context: CGContext)
```

```
{
```

```

let cx = NSGraphicsContext(CGContext: context, flipped: true)

NSGraphicsContext.saveGraphicsState()

NSGraphicsContext.current = cx

}

func NSUIGraphicsPopContext()

{

    NSGraphicsContext.restoreGraphicsState()

}

func NSUIImagePNGRepresentation(_ image: NSUIImage) -> Data?

{

    image.lockFocus()

    let rep = NSBitmapImageRep(focusedViewRect: NSMakeRect(0, 0, image.size.width,
image.size.height))

    image.unlockFocus()

    return rep?.representation(using: .png, properties: [:])

}

```

```
func NSUIImageJPEGRepresentation(_ image: NSUIImage, _ quality: CGFloat = 0.9) ->
Data?
```

```
{

    image.lockFocus()

    let rep = NSBitmapImageRep(focusedViewRect: NSMakeRect(0, 0, image.size.width,
image.size.height))

    image.unlockFocus()

    return      rep?.representation(using:      .jpeg,      properties:
[NSBitmapImageRep.PropertyKey.compressionFactor: quality])

}
```

```
private var imageContextStack: [CGFloat] = []
```

```
func NSUIGraphicsBeginImageContextWithOptions(_ size: CGSize, _ opaque: Bool, _
scale: CGFloat)
```

```
{

    var scale = scale

    if scale == 0.0
```

```

{

    scale = NSScreen.main?.backingScaleFactor ?? 1.0

}

let width = Int(size.width * scale)

let height = Int(size.height * scale)

if width > 0 && height > 0

{

    imageContextStack.append(scale)


    let colorSpace = CGColorSpaceCreateDeviceRGB()


    guard let ctx = CGContext(data: nil, width: width, height: height, bitsPerComponent:
8, bytesPerRow: 4*width, space: colorSpace, bitmapInfo: (opaque ?
CGImageAlphaInfo.noneSkipFirst.rawValue :
CGImageAlphaInfo.premultipliedFirst.rawValue))

    else { return }

```

```

        ctx.concatenate(CGAffineTransform(a: 1, b: 0, c: 0, d: -1, tx: 0, ty: CGFloat(height)))

        ctx.scaleBy(x: scale, y: scale)

        NSUIGraphicsPushContext(ctx)

    }

}

```

```

func NSUIGraphicsGetImageFromCurrentImageContext() -> NSUIImage?

{

    if !imageContextStack.isEmpty

    {

        guard let ctx = NSUIGraphicsGetCurrentContext()

        else { return nil }

        let scale = imageContextStack.last!

        if let theCGImage = ctx.makeImage()

        {

            let size = CGSize(width: CGFloat(ctx.width) / scale, height: CGFloat(ctx.height) /
scale)

```

```

        let image = UIImage(cgImage: theCGImage, size: size)

        return image

    }

}

return nil

}

func NSUIGraphicsEndImageContext()

{

    if imageContextStack.last != nil

    {

        imageContextStack.removeLast()

        NSUIGraphicsPopContext()

    }

}

#endif

//

```



```
// Platform+Touch Handling.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

#if canImport(Uikit)

import UIKit

public typealias NSUIEvent = UIEvent

public typealias NSUITouch = UITouch

@objc
```

```

extension NSUIView {

    public final override fun touchesBegan(_ touches: Set<NSUITouch>, with event:
    NSUIEvent?)

    {

        self.nsuiTouchesBegan(touches, withEvent: event)

    }


    public final override fun touchesMoved(_ touches: Set<NSUITouch>, with event:
    NSUIEvent?)

    {

        self.nsuiTouchesMoved(touches, withEvent: event)

    }


    public final override fun touchesEnded(_ touches: Set<NSUITouch>, with event:
    NSUIEvent?)

    {

        self.nsuiTouchesEnded(touches, withEvent: event)

    }
}

```

```
public final override fun touchesCancelled(_ touches: Set<NSUITouch>, with event:
NSUIEvent?)
```

```
{

    self.nsuiTouchesCancelled(touches, withEvent: event)

}
```

```
open fun nsuiTouchesBegan(_ touches: Set<NSUITouch>, withEvent event:
NSUIEvent?)
```

```
{

    super.touchesBegan(touches, with: event!)

}
```

```
open fun nsuiTouchesMoved(_ touches: Set<NSUITouch>, withEvent event:
NSUIEvent?)
```

```
{

    super.touchesMoved(touches, with: event!)

}
```

```
open func nsuiTouchesEnded(_ touches: Set<NSUITouch>, withEvent event:
NSUIEvent?)
```

```
{

    super.touchesEnded(touches, with: event!)

}
```

```
open func nsuiTouchesCancelled(_ touches: Set<NSUITouch>?, withEvent event:
NSUIEvent?)
```

```
{

    super.touchesCancelled(touches!, with: event!)

}

}
```

```
extension UIView
```

```
{

    @objc final var nsuiGestureRecognizers: [NSUIGestureRecognizer]?

    {

        return self.gestureRecognizers

    }

}
```

```

    }

}

#endif

#if canImport(AppKit) && !targetEnvironment(macCatalyst)

import AppKit

public typealias NSUIEvent = NSEvent

public typealias NSUITouch = NSTouch

@objc

extension NSUIView

{

    public final override func touchesBegan(with event: NSEvent)

    {

        self.nsuiTouchesBegan(event.touches(matching: .any, in: self), withEvent: event)
    }
}

```

```
}
```

```
public final override fun touchesEnded(with event: NSEvent)
```

```
{
```

```
    self.nsuiTouchesEnded(event.touches(matching: .any, in: self), withEvent: event)
```

```
}
```

```
public final override fun touchesMoved(with event: NSEvent)
```

```
{
```

```
    self.nsuiTouchesMoved(event.touches(matching: .any, in: self), withEvent: event)
```

```
}
```

```
open override fun touchesCancelled(with event: NSEvent)
```

```
{
```

```
    self.nsuiTouchesCancelled(event.touches(matching: .any, in: self), withEvent: event)
```

```
}
```

```
open func nsuiTouchesBegan(_ touches: Set<NSUITouch>, withEvent event:
NSUIEvent?)
```

```
{

    super.touchesBegan(with: event!)

}
```

```
open func nsuiTouchesMoved(_ touches: Set<NSUITouch>, withEvent event:
NSUIEvent?)
```

```
{

    super.touchesMoved(with: event!)

}
```

```
open func nsuiTouchesEnded(_ touches: Set<NSUITouch>, withEvent event:
NSUIEvent?)
```

```
{

    super.touchesEnded(with: event!)

}
```

```
open func nsuiTouchesCancelled(_ touches: Set<NSUITouch>?, withEvent event:
NSUIEvent?)
```

```
{

    super.touchesCancelled(with: event!)

}

}
```

```
extension NSTouch
```

```
{

    /** Touch locations on OS X are relative to the trackpad, whereas on iOS they are
    actually *on* the view. */
```

```
func locationInView(view: NSView) -> NSPoint
```

```
{

    let n = self.normalizedPosition

    let b = view.bounds

    return NSPoint(

        x: b.origin.x + b.size.width * n.x,

        y: b.origin.y + b.size.height * n.y
```



```

    )

}

}

#endif

import Foundation

/** This file provides a thin abstraction layer atop of UIKit (iOS, tvOS) and Cocoa (OS X).
The two APIs are very much

alike, and for the chart library's usage of the APIs it is often sufficient to typealias one to
the other. The NSUI*

types are aliased to either their UI* implementation (on iOS) or their NS* implementation
(on OS X). */

#if os(iOS) || os(tvOS)

#if canImport(Uikit)

    import UIKit

#endif

public typealias NSUIFont = UIFont

public typealias NSUIImage = UIImage

```

```
public typealias NSUIScrollView = UIScrollView
```

```
public typealias NSUIScreen = UIScreen
```

```
public typealias NSUIDisplayLink = CADisplayLink
```

```
open class NSUIView: UIView
```

```
{  
  
    @objc var nsuiLayer: CALayer?  
  
    {  
  
        return self.layer  
  
    }  
  
}
```

```
extension UIScrollView
```

```
{  
  
    @objc var nsuiIsScrollEnabled: Bool  
  
    {  
  
        get { return isScrollEnabled }  
  
    }  
  
}
```

```
        set { isEnabled = newValue }  
    }  
}
```

```
extension UIScreen
```

```
{  
  
    @objc final var nsuiScale: CGFloat  
  
    {  
  
        return self.scale  
  
    }  
}
```

```
#endif
```

```
#if os(OSX)
```

```
import Cocoa
```

```
import Quartz
```

```
public typealias NSUIFont = NSFont
```

```
public typealias NSUIImage = NSImage
```

```
public typealias NSUIScrollView = NSScrollView
```

```
public typealias NSScreen = NSScreen
```

```
/** On OS X there is no CADisplayLink. Use a 60 fps timer to render the animations. */
```

```
public class NSUIDisplayLink
```

```
{
```

```
    private var timer: Timer?
```

```
    private var displayLink: CVDisplayLink?
```

```
    private var _timestamp: CFTimeInterval = 0.0
```

```
    private weak var _target: AnyObject?
```

```
    private var _selector: Selector
```

```
    public var timestamp: CFTimeInterval
```

```

{

    return _timestamp

}

init(target: Any, selector: Selector)

{

    _target = target as AnyObject

    _selector = selector

    if CVDisplayLinkCreateWithActiveCGDisplays(&displayLink) ==
    kCVReturnSuccess

    {

        CVDisplayLinkSetOutputCallback(displayLink!, { (displayLink, inNow,
inOutputTime, flagsIn, flagsOut, userData) -> CVReturn in

        let _self = unsafeBitCast(userData, to: NSUIDisplayLink.self)

```

```

        _self._timestamp = CFAbsoluteTimeGetCurrent()

        _self._target?.performSelector(onMainThread: _self._selector, with: _self,
waitUntilDone: false)

        return kCVReturnSuccess

    }, Unmanaged.passUnretained(self).toOpaque())

}

else

{

    timer = Timer(timeInterval: 1.0 / 60.0, target: target, selector: selector, userInfo: nil,
repeats: true)

}

}

deinit

{

    stop()

}

```

```
open func add(to runloop: RunLoop, forMode mode: RunLoop.Mode)
```

```
{
```

```
    if displayLink != nil
```

```
    {
```

```
        CVDisplayLinkStart(displayLink!)
```

```
    }
```

```
    else if timer != nil
```

```
    {
```

```
        runloop.add(timer!, forMode: mode)
```

```
    }
```

```
}
```

```
open func remove(from: RunLoop, forMode: RunLoop.Mode)
```

```
{
```

```
    stop()
```

```
}
```

```

private fun stop()

{

    if displayLink != nil

    {

        CVDisplayLinkStop(displayLink!)

    }

    if timer != nil

    {

        timer?.invalidate()

    }

}

extension UIView

{

    final var nsuiGestureRecognizers: [NSGestureRecognizer]?

```



```

    {

        return self.gestureRecognizers

    }

}

```

extension NSScrollView

```

{

    var nsuiIsScrollEnabled: Bool

    {

        get { return scrollEnabled }

        set { scrollEnabled = newValue }

    }

}

```

open class NSUIView: NSView

```

{

    /// A private constant to set the accessibility role during initialization.

```

/// It ensures parity with the iOS element ordering as well as numbered counts of chart components.

/// (See Platform+Accessibility for details)

private let role: NSAccessibility.Role = .list

public override init(frame frameRect: NSRect)

{

 super.init(frame: frameRect)

 setAccessibilityRole(role)

}

required public init?(coder decoder: NSCoder)

{

 super.init(coder: decoder)

 setAccessibilityRole(role)

}

public final override var isFlipped: Bool

```
{  
  
    return true  
  
}
```

```
func setNeedsDisplay()  
  
{  
  
    self.setNeedsDisplay(self.bounds)  
  
}
```

```
open var backgroundColor: NSUIColor?  
  
{  
  
    get  
  
    {  
  
        return self.layer?.backgroundColor == nil  
  
        ? nil  
  
        : NSColor(cgColor: self.layer!.backgroundColor!)
```

```

    }

    set

    {

        self.wantsLayer = true

        self.layer?.backgroundColor = newValue == nil ? nil : newValue!.cgColor

    }

}

```

```

final var nsuiLayer: CALayer?

{

    return self.layer

}

}

```

```

extension NSFont

{

    var lineHeight: CGFloat

```

```

{

    // Not sure if this is right, but it looks okay

    return self.boundingBoxForFont.size.height

}

}

```

```

extension NSScreen

```

```

{

    final var nsuiScale: CGFloat

    {

        return self.backingScaleFactor

    }

}

```

```

extension UIImage

```

```

{

    var cgImage: CGImage?

```

```

{

    return self.cgImage(forProposedRect: nil, context: nil, hints: nil)

}

}

```

extension NSScrollView

```

{

    /// NOTE: Unable to disable scrolling in macOS

    var scrollEnabled: Bool

    {

        get

        {

            return true

        }

        set

        {

        }

    }

}

```

```

    }

}

#endif

extension NSUIScreen

{

    class var nsuiMain: NSUIScreen? { .main }

}

//

// Protector.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

```

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.


```
//
```

```
import Foundation
```

```
// MARK: -
```

```
/// An `os_unfair_lock` wrapper.
```

```
final class UnfairLock {
```

```
    private let unfairLock: os_unfair_lock_t
```

```
    init() {
```

```
        unfairLock = .allocate(capacity: 1)
```

```
        unfairLock.initialize(to: os_unfair_lock())
```

```
    }
```

```
    deinit {
```

```
        unfairLock.deinitialize(count: 1)
```

```
        unfairLock.deallocate()

    }
```

```
private func lock() {

    os_unfair_lock_lock(unfairLock)

}
```

```
private func unlock() {

    os_unfair_lock_unlock(unfairLock)

}
```

```
/// Executes a closure returning a value while acquiring the lock.
```

```
///
```

```
/// - Parameter closure: The closure to run.
```

```
///
```

```
/// - Returns:         The value the closure generated.
```

```
func around<T>(_ closure: () -> T) -> T {
```

```

        lock(); defer { unlock() }

        return closure()

    }

    /// Execute a closure while acquiring the lock.

    ///

    /// - Parameter closure: The closure to run.

    func around(_ closure: () -> Void) {

        lock(); defer { unlock() }

        return closure()

    }

}

/// A thread-safe wrapper around a value.

final class Protector<T> {

    private let lock = UnfairLock()

    private var value: T

```

```
init(_ value: T) {
```

```
    self.value = value
```

```
}
```

```
/// The contained value. Unsafe for anything more than direct read or write.
```

```
var directValue: T {
```

```
    get { return lock.around { value } }
```

```
    set { lock.around { value = newValue } }
```

```
}
```

```
/// Synchronously read or transform the contained value.
```

```
///
```

```
/// - Parameter closure: The closure to execute.
```

```
///
```

```
/// - Returns:      The return value of the closure passed.
```

```
func read<U>(_ closure: (T) -> U) -> U {
```

```

        return lock.around { closure(self.value) }

    }

    /// Synchronously modify the protected value.

    ///

    /// - Parameter closure: The closure to execute.

    ///

    /// - Returns:      The modified value.

    @discardableResult

    func write<U>(_ closure: (inout T) -> U) -> U {

        return lock.around { closure(&self.value) }

    }

}

extension Protector where T: RangeReplaceableCollection {

    /// Adds a new element to the end of this protected collection.

    ///

```

/// - Parameter newElement: The `Element` to append.

```
func append(_ newElement: T.Element) {
```

```
    write { (ward: inout T) in
```

```
        ward.append(newElement)
```

```
    }
```

```
}
```

/// Adds the elements of a sequence to the end of this protected collection.

///

/// - Parameter newElements: The `Sequence` to append.

```
func append<S: Sequence>(contentsOf newElements: S) where S.Element ==  
T.Element {
```

```
    write { (ward: inout T) in
```

```
        ward.append(contentsOf: newElements)
```

```
    }
```

```
}
```

/// Add the elements of a collection to the end of the protected collection.

```

///

/// - Parameter newElements: The `Collection` to append.

func append<C: Collection>(contentsOf newElements: C) where C.Element ==
T.Element {

    write { (ward: inout T) in

        ward.append(contentsOf: newElements)

    }

}

```

```

extension Protector where T == Data? {

    /// Adds the contents of a `Data` value to the end of the protected `Data`.

    ///

    /// - Parameter data: The `Data` to be appended.

    func append(_ data: Data) {

        write { (ward: inout T) in

            ward?.append(data)

        }
    }
}

```

```
}  
  
}
```

```
extension Protector where T == Request.MutableState {
```

```
    /// Attempts to transition to the passed `State`.
```

```
    ///
```

```
    /// - Parameter state: The `State` to attempt transition to.
```

```
    ///
```

```
    /// - Returns:      Whether the transition occurred.
```

```
    func attemptToTransitionTo(_ state: Request.State) -> Bool {
```

```
        return lock.around {
```

```
            guard value.state.canTransitionTo(state) else { return false }
```

```
            value.state = state
```

```
            return true
```

```
        }
```



```
}
```

```
/// Perform a closure while locked with the provided `Request.State`.
```

```
///
```

```
/// - Parameter perform: The closure to perform while locked.
```

```
func withState(perform: (Request.State) -> Void) {
```

```
    lock.around { perform(value.state) }
```

```
}
```

```
}
```

```
<a href="https://www.ramotion.com/agency/app-development?utm_source=github&utm_medium=repo&utm_campaign=folding-cell"></a>
```

```
<a href="https://github.com/Ramotion/folding-cell">
```

```
</a>
```

```
<p><h1 align="left">FOLDING CELL</h1></p>
```

<h4>Expanding content cell with animation inspired by folding paper card material design.</h4>

<p><h6>We specialize in the designing and coding of custom UI for Mobile Apps and Websites.</h6>

</p>

<p><h6>Stay tuned for the latest updates:</h6>

</p>

</br>

[![CocoaPods](https://img.shields.io/cocoapods/p/FoldingCell.svg)](https://cocoapods.org/pods/FoldingCell)

[![CocoaPods](https://img.shields.io/cocoapods/v/FoldingCell.svg)](http://cocoapods.org/pods/FoldingCell)

[![Twitter](https://img.shields.io/badge/Twitter-@Ramotion-blue.svg?style=flat)](http://twitter.com/Ramotion)

[![Travis](https://img.shields.io/travis/Ramotion/folding-cell.svg)](https://travis-ci.org/Ramotion/folding-cell)

[![codebeat badge](https://codebeat.co/badges/6f67da5d-c416-4bac-9fb7-c2dc938feedc)](https://codebeat.co/projects/github-com-ramotion-folding-cell)

[![Carthage compatible](https://img.shields.io/badge/Carthage-compatible-4BC51D.svg?style=flat)](https://github.com/Carthage/Carthage)

[![Swift 4.0](https://img.shields.io/badge/Swift-5.0-green.svg?style=flat)](https://developer.apple.com/swift/)

[![Analytics](https://ga-beacon.appspot.com/UA-84973210-1/ramotion/folding-cell)](https://github.com/igrigorik/ga-beacon)

[![Donate](https://img.shields.io/badge/Donate-PayPal-blue.svg)](https://paypal.me/Ramotion)

Requirements

- iOS 8.0+

- Xcode 10.2

Installation

Just add the FoldingCell.swift file to your project.

or use [CocoaPods](<https://cocoapods.org>) with Podfile:

```
'''
```

```
pod 'FoldingCell'
```

```
'''
```

or [Carthage](<https://github.com/Carthage/Carthage>) users can simply add Mantle to their `Cartfile`:

```
'''
```

```
github "Ramotion/folding-cell"
```

```
'''
```

or [Swift Package Manager](<https://swift.org/package-manager/>) by adding:

```
'''
```

```
dependencies: [
```

```
.package(url: "https://github.com/Ramotion/folding-cell.git", from: "5.0.2")
```

```
]
```

```
'''
```

```
to `Package.swift`
```

or just drag and drop FoldingCell.swift file to your project

```
## Solution
```

```
![Solution](https://raw.githubusercontent.com/Ramotion/folding-cell/master/Tutorial-resources/Solution.png)
```

```
## Usage
```

1) Create a new cell inheriting from `FoldingCell`

2) Add a UIView to your cell in your storyboard or nib file, inheriting from `RotatedView`.

Connect the outlet from this view to the cell property `foregroundView`.

Add constraints from this view to the superview, as in this picture:

![[1.1]](<https://raw.githubusercontent.com/Ramotion/folding-cell/master/Tutorial-resources/1.1.png>)

(constants of constraints may be different). Connect the outlet from this top constraint to the cell property `foregroundViewTop`

. (This view will be shown when the cell is in its normal state).

3) Add other UIViews to your cell, connect the outlet from this view to the cell

property `containerView`. Add constraints from this view to the superview like in the picture:

![[1.2]](<https://raw.githubusercontent.com/Ramotion/folding-cell/master/Tutorial-resources/1.2.png>)

(constants of constraints may be different). Connect the outlet from this top constraint to the cell property `containerViewTop`.

(This view will be shown when the cell is opened)

Your result should be something like this picture:

![[1.3]](<https://raw.githubusercontent.com/Ramotion/folding-cell/master/Tutorial-resources/1.3.png>)

4) Set ``@IBInspectable var itemCount: NSInteger`` property is a count of folding (it IBInspectable you can set in storyboard). range 2 or greater. Default value is 2

Ok, we've finished configuring the cell.

5) Adding code to your UITableViewController

5.1) Add constants:

```
`` swift
```

```
fileprivate struct C {
```

```
    struct CellHeight {
```

```
        static let close: CGFloat = *** // equal or greater foregroundView height
```

```

static let open: CGFloat = *** // equal or greater containerView height

}

}

'''

```

5.2) Add property for calculate cells height

```

''' swift

var cellHeights = (0..

```

5.3) Override method:

```

''' swift

override func tableView(tableView: UITableView, heightForRowAtIndexPath
indexPath: NSIndexPath) -> CGFloat {

    return cellHeights[indexPath.row]

}

'''

```


5.4) Added code to method:

```
``` swift
```

```
 override func tableView(tableView: UITableView, didSelectRowAtIndexPath
indexPath: NSIndexPath) {
```

```
 guard case let cell as FoldingCell = tableView.cellForRowAtIndexPath(indexPath)
 else {
```

```
 return
```

```
 }
```

```
 var duration = 0.0
```

```
 if cellIsCollapsed {
```

```
 cellHeights[indexPath.row] = Const.openCellHeight
```

```
 cell.unfold(true, animated: true, completion: nil)
```

```
 duration = 0.5
```

```
 } else {
```

```
 cellHeights[indexPath.row] = Const.closeCellHeight
```

```
 cell.unfold(false, animated: true, completion: nil)
```

```
 duration = 0.8
```

```
}
```

```
 UIView.animateWithDuration(duration, delay: 0, options: .CurveEaseOut,
animations: { _ in
```

```
 tableView.beginUpdates()
```

```
 tableView.endUpdates()
```

```
 }, completion: nil)
```

```
}
```

```
'''
```

5.5) Control if the cell is open or closed

```
''' swift
```

```
 override func tableView(tableView: UITableView, willDisplayCell cell:
UITableViewCell, forRowAtIndexPath indexPath: NSIndexPath) {
```

```
 if case let cell as FoldingCell = cell {
```

```
 if cellHeights![indexPath.row] == C.cellHeights.close {
```

```
 foldingCell.selectedAnimation(false, animated: false, completion:nil)
```

```
 } else {
```

```

 foldingCell.selectedAnimation(true, animated: false, completion: nil)

 }

}

}

'''

```

6) Add this code to your new cell class

```

''' swift

 override func animationDuration(itemIndex:NSInteger, type:AnimationType)->
NSTimeInterval {

 // durations count equal it itemCount

 let durations = [0.33, 0.26, 0.26] // timing animation for each view

 return durations[itemIndex]

 }

'''

```

## if don't use storyboard and xib files

Create foregroundView and containerView from code (steps 2 - 3) look example:

[Folding-cell-programmatically](https://github.com/ober01/Folding-cell-programmatically)

## 📁 Check this library on other language:

<a href="https://github.com/Ramotion/folding-cell-android">

</a>

## 📄 License

Folding cell is released under the MIT license.

See [LICENSE](./LICENSE) for details.

This library is a part of a <a href="https://github.com/Ramotion/swift-ui-animation-components-and-libraries"><b>selection of our best UI open-source projects.</b></a>

If you use the open-source library in your project, please make sure to credit and backlink to [www.ramotion.com](http://www.ramotion.com)

## 📱 Get the Showroom App for iOS to give it a try

Try this UI component and more like this in our iOS app. Contact us if interested.

```

```

```

```

```

```

```

```

```
![Alamofire: Elegant Networking in Swift](https://raw.githubusercontent.com/Alamofire/Alamofire/master/alamofire.png)
```

```
[![Build Status](https://github.com/Alamofire/Alamofire/workflows/Alamofire%20CI/badge.svg?branch=master)](https://github.com/Alamofire/Alamofire/actions)
```

[![CocoaPods  
Compatible](https://img.shields.io/cocoapods/v/Alamofire.svg)](https://img.shields.io/coapods/v/Alamofire.svg)

[![Carthage  
Compatible](https://img.shields.io/badge/Carthage-compatible-4BC51D.svg?style=flat)](https://github.com/Carthage/Carthage)

[![Platform](https://img.shields.io/cocoapods/p/Alamofire.svg?style=flat)](https://alamofire.github.io/Alamofire)

[![Twitter](https://img.shields.io/badge/twitter-@AlamofireSF-blue.svg?style=flat)](https://twitter.com/AlamofireSF)

[![Gitter](https://badges.gitter.im/Alamofire/Alamofire.svg)](https://gitter.im/Alamofire/Alamofire?utm\_source=badge&utm\_medium=badge&utm\_campaign=pr-badge)

[![Open  
Helpers](https://www.codetriage.com/alamofire/alamofire/badges/users.svg)](https://www.codetriage.com/alamofire/alamofire) Source

Alamofire is an HTTP networking library written in Swift.

- [Features](#features)
- [Component Libraries](#component-libraries)
- [Requirements](#requirements)
- [Migration Guides](#migration-guides)

- [Communication](#communication)

- [Installation](#installation)

-

[Usage](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md#using-alamofire)

- **Introduction** -  
[Introduction](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md#introduction) [Making Requests](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md#making-requests), [Response Handling](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md#response-handling), [Response Validation](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md#response-validation), [Response Caching](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md#response-caching)

- **HTTP** - [HTTP Methods](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md#http-methods), [Parameters and Parameter Encoder](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md##request-parameters-and-parameter-encoders), [HTTP Headers](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md#http-headers), [Authentication](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md#authentication)

- **Large Data** - [Downloading Data to a File](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md#downloading-data-to-a-file), [Uploading Data to a

Server](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md#uploading-data-to-a-server)

- **Tools** **Statistical Metrics**](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md#statistical-metrics), [cURL Command Output](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md#curl-command-output)

- **Advanced Usage**](https://github.com/Alamofire/Alamofire/blob/master/Documentation/AdvancedUsage.md)

- **URL Session** **Session Manager**](https://github.com/Alamofire/Alamofire/blob/master/Documentation/AdvancedUsage.md#session), [Session Delegate](https://github.com/Alamofire/Alamofire/blob/master/Documentation/AdvancedUsage.md#sessiondelegate), [Request](https://github.com/Alamofire/Alamofire/blob/master/Documentation/AdvancedUsage.md#request)

- **Routing** **Routing Requests**](https://github.com/Alamofire/Alamofire/blob/master/Documentation/AdvancedUsage.md#routing-requests), [Adapting and Retrying Requests](https://github.com/Alamofire/Alamofire/blob/master/Documentation/AdvancedUsage.md#adapting-and-retrying-requests)

- **Model Objects** **Custom Response Serialization**](https://github.com/Alamofire/Alamofire/blob/master/Documentation/AdvancedUsage.md#custom-response-serialization)

- **Connection** **Security**](https://github.com/Alamofire/Alamofire/blob/master/Documentation/AdvancedUsage.md#connection-security)



dUsage.md#security), [Network  
Reachability](https://github.com/Alamofire/Alamofire/blob/master/Documentation/AdvancedUsage.md#network-reachability)

- [Open Radars](#open-radars)

- [FAQ](#faq)

- [Credits](#credits)

- [Donations](#donations)

- [License](#license)

## ## Features

- [x] Chainable Request / Response Methods

- [x] URL / JSON Parameter Encoding

- [x] Upload File / Data / Stream / MultipartFormData

- [x] Download File using Request or Resume Data

- [x] Authentication with URLLCredential

- [x] HTTP Response Validation

- [x] Upload and Download Progress Closures with Progress

- [x] cURL Command Output
- [x] Dynamically Adapt and Retry Requests
- [x] TLS Certificate and Public Key Pinning
- [x] Network Reachability
- [x] Comprehensive Unit and Integration Test Coverage
- [x] [Complete Documentation](https://alamofire.github.io/Alamofire)

## ## Component Libraries

In order to keep Alamofire focused specifically on core networking implementations, additional component libraries have been created by the [Alamofire Software Foundation](https://github.com/Alamofire/Foundation) to bring additional functionality to the Alamofire ecosystem.

- [AlamofireImage](https://github.com/Alamofire/AlamofireImage) - An image library including image response serializers, `UIImage` and `UIImageView` extensions, custom image filters, an auto-purging in-memory cache and a priority-based image downloading system.

-

- [AlamofireNetworkActivityIndicator](https://github.com/Alamofire/AlamofireNetworkActivityIndicator) - Controls the visibility of the network activity indicator on iOS using

Alamofire. It contains configurable delay timers to help mitigate flicker and can support `URLSession` instances not managed by Alamofire.

## ## Requirements

- iOS 10.0+ / macOS 10.12+ / tvOS 10.0+ / watchOS 3.0+

- Xcode 10.2+

- Swift 5+

## ## Migration Guides

- [Alamofire 5.0 Migration Guide](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Alamofire%205.0%20Migration%20Guide.md)

- [Alamofire 4.0 Migration Guide](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Alamofire%204.0%20Migration%20Guide.md)

- [Alamofire 3.0 Migration Guide](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Alamofire%203.0%20Migration%20Guide.md)

- [Alamofire 2.0 Migration Guide](https://github.com/Alamofire/Alamofire/blob/master/Documentation/Alamofire%202.0%20Migration%20Guide.md)

## ## Communication

- If you **need help with making network requests** using Alamofire, use [Stack Overflow](https://stackoverflow.com/questions/tagged/alamofire) and tag `alamofire`.
- If you need to **find or understand an API**, check [our documentation](http://alamofire.github.io/Alamofire/) or [Apple's documentation for `NSURLSession`](https://developer.apple.com/documentation/foundation/url\_loading\_system), on top of which Alamofire is built.
- If you need **help with an Alamofire feature**, use [our forum on swift.org](https://forums.swift.org/c/related-projects/alamofire).
- If you'd like to **discuss Alamofire best practices**, use [our forum on swift.org](https://forums.swift.org/c/related-projects/alamofire).
- If you'd like to **discuss a feature request**, use [our forum on swift.org](https://forums.swift.org/c/related-projects/alamofire).
- If you **found a bug**, open an issue here on GitHub and follow the guide. The more detail the better!
- If you **want to contribute**, submit a pull request!

## ## Installation

### ### CocoaPods

[CocoaPods](https://cocoapods.org) is a dependency manager for Cocoa projects. For usage and installation instructions, visit their website. To integrate Alamofire into your Xcode project using CocoaPods, specify it in your `Podfile`:

```
``ruby
```

```
pod 'Alamofire', '~> 5.0'
```

```
``
```

### ### Carthage

[Carthage](https://github.com/Carthage/Carthage) is a decentralized dependency manager that builds your dependencies and provides you with binary frameworks. To integrate Alamofire into your Xcode project using Carthage, specify it in your `Cartfile`:

```
``ogdl
```

```
github "Alamofire/Alamofire" ~> 5.0
```

```
'''
```

### ### Swift Package Manager

The [Swift Package Manager](https://swift.org/package-manager/) is a tool for automating the distribution of Swift code and is integrated into the `swift` compiler. It is in early development, but Alamofire does support its use on supported platforms.

Once you have your Swift package set up, adding Alamofire as a dependency is as easy as adding it to the `dependencies` value of your `Package.swift`.

```
```swift
```

```
dependencies: [
```

```
    .package(url: "https://github.com/Alamofire/Alamofire.git", .upToNextMajor(from: "5.0.0"))
```

```
]
```

```
'''
```

Manually

If you prefer not to use any of the aforementioned dependency managers, you can integrate Alamofire into your project manually.

Embedded Framework

- Open up Terminal, `cd` into your top-level project directory, and run the following command "if" your project is not initialized as a git repository:

```
```bash
```

```
$ git init
```

```
```
```

- Add Alamofire as a git [submodule](<https://git-scm.com/docs/git-submodule>) by running the following command:

```
```bash
```

```
$ git submodule add https://github.com/Alamofire/Alamofire.git
```

```
```
```

- Open the new `Alamofire` folder, and drag the `Alamofire.xcodeproj` into the Project Navigator of your application's Xcode project.

- > It should appear nested underneath your application's blue project icon. Whether it is above or below all the other Xcode groups does not matter.

- Select the `Alamofire.xcodeproj` in the Project Navigator and verify the deployment target matches that of your application target.

- Next, select your application project in the Project Navigator (blue project icon) to navigate to the target configuration window and select the application target under the "Targets" heading in the sidebar.

- In the tab bar at the top of that window, open the "General" panel.

- Click on the `+` button under the "Embedded Binaries" section.

- You will see two different `Alamofire.xcodeproj` folders each with two different versions of the `Alamofire.framework` nested inside a `Products` folder.

- > It does not matter which `Products` folder you choose from, but it does matter whether you choose the top or bottom `Alamofire.framework`.

- Select the top `Alamofire.framework` for iOS and the bottom one for macOS.

> You can verify which one you selected by inspecting the build log for your project. The build target for `Alamofire` will be listed as either `Alamofire iOS`, `Alamofire macOS`, `Alamofire tvOS` or `Alamofire watchOS`.

- And that's it!

> The `Alamofire.framework` is automatically added as a target dependency, linked framework and embedded framework in a copy files build phase which is all you need to build on the simulator and a device.

Open Radars

The following radars have some effect on the current implementation of Alamofire.

- [`rdar://21349340`](<http://www.openradar.me/radar?id=5517037090635776>) - Compiler throwing warning due to toll-free bridging issue in test case
- `rdar://26870455` - Background URL Session Configurations do not work in the simulator
- `rdar://26849668` - Some URLProtocol APIs do not properly handle `URLRequest`

- `FB7624529` - `urlSession(\_:task:didFinishCollecting:)` never called on watchOS

Resolved Radars

The following radars have been resolved over time after being filed against the Alamofire project.

- [`rdar://26761490`](http://www.openradar.me/radar?id=5010235949318144) - Swift string interpolation causing memory leak with common usage.
 - (Resolved): 9/1/17 in Xcode 9 beta 6.
- [`rdar://36082113`](http://openradar.appspot.com/radar?id=4942308441063424) - `URLSessionTaskMetrics` failing to link on watchOS 3.0+
 - (Resolved): Just add `CFNetwork` to your linked frameworks.

Workarounds

- Collection of `URLSessionTaskMetrics` is currently disabled on watchOS due to `FB7624529`.

FAQ

What's the origin of the name Alamofire?

Alamofire is named after the [Alamo Fire flower](<https://aggie-horticulture.tamu.edu/wildseed/alamofire.html>), a hybrid variant of the Bluebonnet, the official state flower of Texas.

Credits

Alamofire is owned and maintained by the [Alamofire Software Foundation](<http://alamofire.org>). You can follow them on Twitter at [@AlamofireSF](<https://twitter.com/AlamofireSF>) for project updates and releases.

Security Disclosure

If you believe you have identified a security vulnerability with Alamofire, you should report it as soon as possible via email to security@alamofire.org. Please do not post it to a public issue tracker.

Donations

The [ASF](<https://github.com/Alamofire/Foundation#members>) is looking to raise money to officially stay registered as a federal non-profit organization.

Registering will allow us members to gain some legal protections and also allow us to put donations to use, tax-free.

Donating to the ASF will enable us to:

- Pay our yearly legal fees to keep the non-profit in good status
- Pay for our mail servers to help us stay on top of all questions and security issues
- Potentially fund test servers to make it easier for us to test the edge cases
- Potentially fund developers to work on one of our projects full-time

The community adoption of the ASF libraries has been amazing.

We are greatly humbled by your enthusiasm around the projects and want to continue to do everything we can to move the needle forward.

With your continued support, the ASF will be able to improve its reach and also provide better legal safety for the core members.

If you use any of our libraries for work, see if your employers would be interested in donating.

Any amount you can donate today to help us reach our goal would be greatly appreciated.

[![paypal](https://www.paypalobjects.com/en_US/i/btn/btn_donateCC_LG.gif)](https://www.paypal.com/cgi-bin/webscr?cmd=_s-xclick&hosted_button_id=W34WPEE74APJQ)

License

Alamofire is released under the MIT license. [See LICENSE](https://github.com/Alamofire/Alamofire/blob/master/LICENSE) for details.

//

// RadarChartData.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// <https://github.com/danielgindi/Charts>

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
open class RadarChartData: ChartData
```

```
{
```

```
    @objc open var highlightColor = NSUIColor(red: 255.0/255.0, green: 187.0/255.0, blue:  
115.0/255.0, alpha: 1.0)
```

```
    @objc open var highlightLineWidth = CGFloat(1.0)
```

```
    @objc open var highlightLineDashPhase = CGFloat(0.0)
```

```
    @objc open var highlightLineDashLengths: [CGFloat]?
```

```
/// Sets labels that should be drawn around the RadarChart at the end of each web line.
```

```
    @objc open var labels = [String]()
```

/// Sets the labels that should be drawn around the RadarChart at the end of each web line.

```
open fun setLabels(_ labels: String...)
```

```
{
```

```
    self.labels = labels
```

```
}
```

```
public override init()
```

```
{
```

```
    super.init()
```

```
}
```

```
public override init(dataSets: [IChartDataSet]?)
```

```
{
```

```
    super.init(dataSets: dataSets)
```

```
}
```

```
open override fun entryForHighlight(_ highlight: Highlight) -> ChartDataEntry?
```

```

    {

        return getDataSetByIndex(highlight.dataSetIndex)?.entryForIndex(Int(highlight.x))

    }

}

//

// RadarChartDataEntry.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

```



```
open class RadarChartDataEntry: ChartDataEntry
```

```
{
```

```
    public required init()
```

```
    {
```

```
        super.init()
```

```
    }
```

```
/// - Parameters:
```

```
/// - value: The value on the y-axis.
```

```
@objc public init(value: Double)
```

```
{
```

```
    super.init(x: .nan, y: value)
```

```
}
```

```
/// - Parameters:
```

```
/// - value: The value on the y-axis.
```

```
/// - data: Spot for additional data this Entry represents.

@objc public convenience init(value: Double, data: Any?)

{

    self.init(value: value)

    self.data = data

}
```

```
// MARK: Data property accessors
```

```
@objc open var value: Double

{

    get { return y }

    set { y = newValue }

}
```

```
// MARK: NSCopying
```

```

open override func copy(with zone: NSZone? = nil) -> Any

{

    let copy = super.copy(with: zone) as! RadarChartDataEntry


    return copy

}

}

//

// RadarChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

```

```
import Foundation
```

```
import CoreGraphics
```

```
open class RadarChartDataSet: LineRadarChartDataSet, IRadarChartDataSet
```

```
{
```

```
    private func initialize()
```

```
    {
```

```
        self.valueFont = NSUIFont.systemFont(ofSize: 13.0)
```

```
    }
```

```
    public required init()
```

```
    {
```

```
        super.init()
```

```
        initialize()
```

```
    }
```

```

public required override init(entries: [ChartDataEntry]?, label: String?)

{

    super.init(entries: entries, label: label)

    initialize()

}


// MARK: - Data functions and accessors


// MARK: - Styling functions and accessors


// flag indicating whether highlight circle should be drawn or not


// **default**: false


open var drawHighlightCircleEnabled: Bool = false


// `true` if highlight circle should be drawn, `false` if not


open var isDrawHighlightCircleEnabled: Bool { return drawHighlightCircleEnabled }

```

```
open var highlightCircleFillColor: NSUIColor? = NSUIColor.white

/// The stroke color for highlight circle.

/// If `nil`, the color of the dataset is taken.

open var highlightCircleStrokeColor: NSUIColor?

open var highlightCircleStrokeAlpha: CGFloat = 0.3

open var highlightCircleInnerRadius: CGFloat = 3.0

open var highlightCircleOuterRadius: CGFloat = 4.0

open var highlightCircleStrokeWidth: CGFloat = 2.0

}

//

// RadarChartRenderer.swift
```

```
// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//
```

```
import Foundation
```

```
import CoreGraphics
```

```
open class RadarChartRenderer: LineRadarRenderer
```

```
{
```

```
    private lazy var accessibilityXLabels: [String] = {
```

```
        guard let chart = chart else { return [] }
```

```
        guard let formatter = chart.xAxis.valueFormatter else { return [] }
```

```

        let maxEntryCount = chart.data?.maxEntryCountSet?.entryCount ?? 0

        return stride(from: 0, to: maxEntryCount, by: 1).map {

            formatter.stringForValue(Double($0), axis: chart.xAxis)

        }

    }()
}

```

```

@objc open weak var chart: RadarChartView?

```

```

@objc public init(chart: RadarChartView, animator: Animator, viewPortHandler:
ViewPortHandler)

```

```

{

    super.init(animator: animator, viewPortHandler: viewPortHandler)

    self.chart = chart

}

```

```

open override func drawData(context: CGContext)

```



```

{

    guard let chart = chart else { return }

    let radarData = chart.data

    if radarData != nil

    {

        let mostEntries = radarData?.maxEntryCountSet?.entryCount ?? 0

        // If we redraw the data, remove and repopulate accessible elements to update label
        values and frames

        self.accessibleChartElements.removeAll()

        // Make the chart header the first element in the accessible elements array

        if let accessibilityHeaderData = radarData as? RadarChartData {

            let element = createAccessibleHeader(usingChart: chart,

                                                andData: accessibilityHeaderData,

                                                withDefaultDescription: "Radar Chart")

```

```

        self.accessibleChartElements.append(element)

    }

    for set in radarData!.dataSets as! [IRadarChartDataSet] where set.isVisible

    {

        drawDataSet(context: context, dataSet: set, mostEntries: mostEntries)

    }

    }

}

/// Draws the RadarDataSet

///

/// - Parameters:

/// - context:

/// - dataSet:

/// - mostEntries: the entry count of the dataset with the most entries

internal func drawDataSet(context: CGContext, dataSet: IRadarChartDataSet,
mostEntries: Int)

```

```
{  
  
  guard let chart = chart else { return }  
  
  context.saveGState()  
  
  let phaseX = animator.phaseX  
  
  let phaseY = animator.phaseY  
  
  let sliceangle = chart.sliceAngle  
  
  // calculate the factor that is needed for transforming the value to pixels  
  
  let factor = chart.factor  
  
  let center = chart.centerOffsets  
  
  let entryCount = dataSet.entryCount  
  
  let path = CGMutablePath()  
  
  var hasMovedToPoint = false
```

```

let prefix: String = chart.data?.accessibilityEntryLabelPrefix ?? "Item"

let description = dataSet.label ?? ""

// Make a tuple of (xLabels, value, originalIndex) then sort it

// This is done, so that the labels are narrated in decreasing order of their corresponding
value

// Otherwise, there is no non-visual logic to the data presented

let accessibilityEntryValues = Array(0 ..< entryCount).map
{ (dataSet.entryForIndex($0)?.y ?? 0, $0) }

let accessibilityAxisLabelValueTuples = zip(accessibilityXLabels,
accessibilityEntryValues).map { ($0, $1.0, $1.1) }.sorted { $0.1 > $1.1 }

let accessibilityDataSetDescription: String = description + ". \$(entryCount) \$(prefix +
(entryCount == 1 ? "" : "s")). "

let accessibilityFrameWidth: CGFloat = 22.0 // To allow a tap target of 44x44

var accessibilityEntryElements: [NSUIAccessibilityElement] = []

for j in 0 ..< entryCount

```

```

{

    guard let e = dataSet.entryForIndex(j) else { continue }

    let p = center.moving(distance: CGFloat((e.y - chart.chartYMin) * Double(factor)
* phaseY),

                                atAngle: sliceangle * CGFloat(j) * CGFloat(phaseX) +
chart.rotationAngle)

    if p.x.isNaN

    {

        continue

    }

    if !hasMovedToPoint

    {

        path.move(to: p)

        hasMovedToPoint = true

    }

```

```

else

{

    path.addLine(to: p)

}


let accessibilityLabel = accessibilityAxisLabelValueTuples[j].0

let accessibilityValue = accessibilityAxisLabelValueTuples[j].1

let accessibilityValueIndex = accessibilityAxisLabelValueTuples[j].2


let axp = center.moving(distance: CGFloat((accessibilityValue - chart.chartYMin)
* Double(factor) * phaseY),

                        atAngle: sliceangle * CGFloat(accessibilityValueIndex) *
CGFloat(phaseX) + chart.rotationAngle)


let axDescription = description + " - " + accessibilityLabel + ": \"(accessibilityValue)
\"(chart.data?.accessibilityEntryLabelSuffix ?? "")"

let axElement = createAccessibleElement(withDescription: axDescription,

                                        container: chart,

                                        dataSet: dataSet)

```

```

    { (element) in

        element.accessibilityFrame = CGRect(x: axp.x - accessibilityFrameWidth,

            y: axp.y - accessibilityFrameWidth,

            width: 2 * accessibilityFrameWidth,

            height: 2 * accessibilityFrameWidth)

    }

    accessibilityEntryElements.append(axElement)

}

// if this is the largest set, close it

if dataSet.entryCount < mostEntries

{

    // if this is not the largest set, draw a line to the center before closing

    path.addLine(to: center)

}

```

```

path.closeSubpath()

// draw filled

if dataSet.isDrawFilledEnabled

{

    if dataSet.fill != nil

    {

        drawFilledPath(context: context, path: path, fill: dataSet.fill!, fillAlpha:
dataSet.fillAlpha)

    }

    else

    {

        drawFilledPath(context: context, path: path, fillColor: dataSet.fillColor,
fillAlpha: dataSet.fillAlpha)

    }

}

// draw the line (only if filled is disabled or alpha is below 255)

```



```

if !dataSet.isDrawFilledEnabled || dataSet.fillAlpha < 1.0

{

    context.setStrokeColor(dataSet.color(atIndex: 0).cgColor)

    context.setLineWidth(dataSet.lineWidth)

    context.setAlpha(1.0)


    context.beginPath()

    context.addPath(path)

    context.strokePath()


    let      axElement      =      createAccessibleElement(withDescription:
accessibilityDataSetDescription,

                                container: chart,

                                dataSet: dataSet)

    { (element) in

        element.isHeader = true

        element.accessibilityFrame = path.boundingBoxOfPath

    }

```

```
accessibleChartElements.append(axElement)
```

```
accessibleChartElements.append(contentsOf: accessibilityEntryElements)
```

```
}
```

```
accessibilityPostLayoutChangedNotification()
```

```
context.restoreGState()
```

```
}
```

```
open override func drawValues(context: CGContext)
```

```
{
```

```
    guard
```

```
        let chart = chart,
```

```
        let data = chart.data
```

```
    else { return }
```

```
let phaseX = animator.phaseX
```

```
let phaseY = animator.phaseY
```

```
let sliceangle = chart.sliceAngle
```

```
// calculate the factor that is needed for transforming the value to pixels
```

```
let factor = chart.factor
```

```
let center = chart.centerOffsets
```

```
let yoffset = CGFloat(5.0)
```

```
for i in 0 ..< data.dataSetCount
```

```
{
```

```
    guard let
```

```
        dataSet = data.getDataSetByIndex(i) as? IRadarChartDataSet,
```

```
        shouldDrawValues(forDataSet: dataSet)
```

```
else { continue }
```

```
let entryCount = dataSet.entryCount
```

```
let iconsOffset = dataSet.iconsOffset
```

```
for j in 0 ..< entryCount
```

```
{
```

```
  guard let e = dataSet.entryForIndex(j) else { continue }
```

```
    let p = center.moving(distance: CGFloat(e.y - chart.chartYMin) * factor *  
CGFloat(phaseY),
```

```
                           atAngle: sliceangle * CGFloat(j) * CGFloat(phaseX) +  
chart.rotationAngle)
```

```
let valueFont = dataSet.valueFont
```

```
guard let formatter = dataSet.valueFormatter else { continue }
```

```

if dataSet.isDrawValuesEnabled

{

    ChartUtils.drawText(

        context: context,

        text: formatter.stringForValue(

            e.y,

            entry: e,

            dataSetIndex: i,

            viewPortHandler: viewPortHandler),

        point: CGPoint(x: p.x, y: p.y - yoffset - valueFont.lineHeight),

        align: .center,

        attributes: [NSAttributedString.Key.font: valueFont,

                    NSAttributedString.Key.foregroundColor: dataSet.valueTextColorAt(j)]

    )

}

```

```

        if let icon = e.icon, dataSet.isDrawIconsEnabled

        {

            var pIcon = center.moving(distance: CGFloat(e.y) * factor * CGFloat(phaseY)
+ iconsOffset.y,

                                atAngle: sliceangle * CGFloat(j) * CGFloat(phaseX) +
chart.rotationAngle)

            pIcon.y += iconsOffset.x

            ChartUtils.drawImage(context: context,

                                image: icon,

                                x: pIcon.x,

                                y: pIcon.y,

                                size: icon.size)

        }

    }

}

```

```
open override func drawExtras(context: CGContext)
```

```
{
```

```
    drawWeb(context: context)
```

```
}
```

```
private var _webLineSegmentsBuffer = [CGPoint](repeating: CGPoint(), count: 2)
```

```
@objc open func drawWeb(context: CGContext)
```

```
{
```

```
    guard
```

```
        let chart = chart,
```

```
        let data = chart.data
```

```
    else { return }
```

```
    let sliceangle = chart.sliceAngle
```

```
    context.saveGState()
```

```
// calculate the factor that is needed for transforming the value to

// pixels

let factor = chart.factor

let rotationangle = chart.rotationAngle


let center = chart.centerOffsets


// draw the web lines that come from the center

context.setLineWidth(chart.webLineWidth)

context.setStrokeColor(chart.webColor.cgColor)

context.setAlpha(chart.webAlpha)


let xIncrements = 1 + chart.skipWebLineCount

let maxEntryCount = chart.data?.maxEntryCountSet?.entryCount ?? 0


for i in stride(from: 0, to: maxEntryCount, by: xIncrements)
```



```

{

    let p = center.moving(distance: CGFloat(chart.yRange) * factor,

                           atAngle: sliceangle * CGFloat(i) + rotationangle)

    _webLineSegmentsBuffer[0].x = center.x

    _webLineSegmentsBuffer[0].y = center.y

    _webLineSegmentsBuffer[1].x = p.x

    _webLineSegmentsBuffer[1].y = p.y

    context.strokeLineSegments(between: _webLineSegmentsBuffer)

}

// draw the inner-web

context.setLineWidth(chart.innerWebLineWidth)

context.setStrokeColor(chart.innerWebColor.cgColor)

context.setAlpha(chart.webAlpha)

```

```

let labelCount = chart.yAxis.entryCount

for j in 0 ..< labelCount

{

    for i in 0 ..< data.entryCount

    {

        let r = CGFloat(chart.yAxis.entries[j] - chart.chartYMin) * factor


        let p1 = center.moving(distance: r, atAngle: sliceangle * CGFloat(i) +
rotationangle)

        let p2 = center.moving(distance: r, atAngle: sliceangle * CGFloat(i + 1) +
rotationangle)


        _webLineSegmentsBuffer[0].x = p1.x

        _webLineSegmentsBuffer[0].y = p1.y

        _webLineSegmentsBuffer[1].x = p2.x

        _webLineSegmentsBuffer[1].y = p2.y

```

```

        context.strokeLineSegments(between: _webLineSegmentsBuffer)

    }

}

context.restoreGState()

}

private var _highlightPointBuffer = CGPoint()

open override func drawHighlighted(context: CGContext, indices: [Highlight])

{

    guard

        let chart = chart,

        let radarData = chart.data as? RadarChartData

    else { return }

    context.saveGState()

```

```

let sliceangle = chart.sliceAngle

// calculate the factor that is needed for transforming the value pixels

let factor = chart.factor

let center = chart.centerOffsets

for high in indices
{

    guard

        let set = chart.data?.getDataSetByIndex(high.dataSetIndex) as?
IRadarChartDataSet,

        set.isHighlightEnabled

    else { continue }

    guard let e = set.entryForIndex(Int(high.x)) as? RadarChartDataEntry

    else { continue }

```

```

if !isInBoundsX(entry: e, dataSet: set)

{

    continue

}


context.setLineWidth(radarData.highlightLineWidth)

if radarData.highlightLineDashLengths != nil

{

    context.setLineDash(phase:    radarData.highlightLineDashPhase,    lengths:
radarData.highlightLineDashLengths!)

}

else

{

    context.setLineDash(phase: 0.0, lengths: [])

}


context.setStrokeColor(set.highlightColor.cgColor)

```

```
let y = e.y - chart.chartYMin
```

```
    _highlightPointBuffer = center.moving(distance: CGFloat(y) * factor *  
CGFloat(animotor.phaseY),  
  
        atAngle: sliceangle * CGFloat(high.x) *  
CGFloat(animotor.phaseX) + chart.rotationAngle)
```

```
high.setDraw(pt: _highlightPointBuffer)
```

```
// draw the lines
```

```
drawHighlightLines(context: context, point: _highlightPointBuffer, set: set)
```

```
if set.isDrawHighlightCircleEnabled
```

```
{
```

```
    if !_highlightPointBuffer.x.isNaN && !_highlightPointBuffer.y.isNaN
```

```
{
```

```
    var strokeColor = set.highlightCircleStrokeColor
```

```

if strokeColor == nil

{

    strokeColor = set.color(atIndex: 0)

}

if set.highlightCircleStrokeAlpha < 1.0

{

    strokeColor
strokeColor?.withAlphaComponent(set.highlightCircleStrokeAlpha)

}

drawHighlightCircle(

    context: context,

    atPoint: _highlightPointBuffer,

    innerRadius: set.highlightCircleInnerRadius,

    outerRadius: set.highlightCircleOuterRadius,

    fillColor: set.highlightCircleFillColor,

    strokeColor: strokeColor,

    strokeWidth: set.highlightCircleStrokeWidth)

```

```
}
```

```
}
```

```
}
```

```
context.restoreGState()
```

```
}
```

```
internal func drawHighlightCircle(
```

```
    context: CGContext,
```

```
    atPoint point: CGPoint,
```

```
    innerRadius: CGFloat,
```

```
    outerRadius: CGFloat,
```

```
    fillColor: NSUIColor?,
```

```
    strokeColor: NSUIColor?,
```

```
    strokeWidth: CGFloat)
```

```
{
```

```
    context.saveGState()
```



```

if let fillColor = fillColor

{

    context.beginPath()

    context.addEllipse(in: CGRect(x: point.x - outerRadius, y: point.y - outerRadius,
width: outerRadius * 2.0, height: outerRadius * 2.0))

    if innerRadius > 0.0

    {

        context.addEllipse(in: CGRect(x: point.x - innerRadius, y: point.y - innerRadius,
width: innerRadius * 2.0, height: innerRadius * 2.0))

    }

    context.setFillColor(fillColor.cgColor)

    context.fillPath(using: .evenOdd)

}

if let strokeColor = strokeColor

{

```

```

        context.beginPath()

        context.addEllipse(in: CGRect(x: point.x - outerRadius, y: point.y - outerRadius,
width: outerRadius * 2.0, height: outerRadius * 2.0))

        context.setStrokeColor(strokeColor.cgColor)

        context.setLineWidth(strokeWidth)

        context.strokePath()

    }

    context.restoreGState()

}

```

```

private func createAccessibleElement(withDescription description: String,

                                     container: RadarChartView,

                                     dataSet: IRadarChartDataSet,

                                     modifier: (NSUIAccessibilityElement) -> () ->
NSUIAccessibilityElement {

```

```

    let element = NSUIAccessibilityElement(accessibilityContainer: container)

```

```
        element.accessibilityLabel = description

    // The modifier allows changing of traits and frame depending on highlight, rotation,
    etc

    modifier(element)

    return element

}

}

//

// RadarChartView.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts
```

```
//

import Foundation

import CoreGraphics

/// Implementation of the RadarChart, a "spidernet"-like chart. It works best
/// when displaying 5-10 entries per DataSet.

open class RadarChartView: PieRadarChartViewBase

{

    /// width of the web lines that come from the center.

    @objc open var webLineWidth = CGFloat(1.5)

    /// width of the web lines that are in between the lines coming from the center

    @objc open var innerWebLineWidth = CGFloat(0.75)

    /// color for the web lines that come from the center
```

```
@objc open var webColor = NSUIColor(red: 122/255.0, green: 122/255.0, blue:  
122.0/255.0, alpha: 1.0)
```

```
/// color for the web lines in between the lines that come from the center.
```

```
@objc open var innerWebColor = NSUIColor(red: 122/255.0, green: 122/255.0, blue:  
122.0/255.0, alpha: 1.0)
```

```
/// transparency the grid is drawn with (0.0 - 1.0)
```

```
@objc open var webAlpha: CGFloat = 150.0 / 255.0
```

```
/// flag indicating if the web lines should be drawn or not
```

```
@objc open var drawWeb = true
```

```
/// modulus that determines how many labels and web-lines are skipped before the next  
is drawn
```

```
private var _skipWebLineCount = 0
```

```
/// the object representing the y-axis labels
```

```
private var _yAxis: YAxis!
```

```
internal var _yAxisRenderer: YAxisRendererRadarChart!
```

```
internal var _xAxisRenderer: XAxisRendererRadarChart!
```

```
public override init(frame: CGRect)
```

```
{
```

```
    super.init(frame: frame)
```

```
}
```

```
public required init?(coder aDecoder: NSCoder)
```

```
{
```

```
    super.init(coder: aDecoder)
```

```
}
```

```
internal override func initialize()
```

```
{
```

```
    super.initialize()
```

```
_yAxis = YAxis(position: .left)
```

```
_yAxis.labelXOffset = 10.0
```

```
renderer = RadarChartRenderer(chart: self, animator: _animator, viewPortHandler:  
_viewPortHandler)
```

```
_yAxisRenderer = YAxisRendererRadarChart(viewPortHandler: _viewPortHandler,  
yAxis: _yAxis, chart: self)
```

```
_xAxisRenderer = XAxisRendererRadarChart(viewPortHandler: _viewPortHandler,  
xAxis: _xAxis, chart: self)
```

```
self.highlighter = RadarHighlighter(chart: self)
```

```
}
```

```
internal override func calcMinMax()
```

```
{
```

```
super.calcMinMax()
```

```
guard let data = _data else { return }
```

```
    _yAxis.calculate(min: data.getYMin(axis: .left), max: data.getYMax(axis: .left))
```

```
    _xAxis.calculate(min: 0.0, max: Double(data.maxEntryCountSet?.entryCount ?? 0))
```

```
}
```

```
open override func notifyDataSetChanged()
```

```
{
```

```
    calcMinMax()
```

```
        _yAxisRenderer?.computeAxis(min:          _yAxis._axisMinimum,          max:  
        _yAxis._axisMaximum, inverted: _yAxis.isInverted)
```

```
        _xAxisRenderer?.computeAxis(min:          _xAxis._axisMinimum,          max:  
        _xAxis._axisMaximum, inverted: false)
```

```
if let data = _data,
```

```
    let legend = _legend,
```

```
    !legend.isLegendCustom
```



```
{  
  
    legendRenderer?.computeLegend(data: data)  
  
}
```

```
calculateOffsets()
```

```
setNeedsDisplay()
```

```
}
```

```
open override func draw(_ rect: CGRect)
```

```
{
```

```
    super.draw(rect)
```

```
    guard data != nil, let renderer = renderer else { return }
```

```
    let optionalContext = NSUIGraphicsGetCurrentContext()
```

```
    guard let context = optionalContext else { return }
```

```

        if _xAxis.isEnabled

        {

            _xAxisRenderer.computeAxis(min:          _xAxis._axisMinimum,          max:
            _xAxis._axisMaximum, inverted: false)

        }

        _xAxisRenderer?.renderAxisLabels(context: context)

    if drawWeb

    {

        renderer.drawExtras(context: context)

    }

    if _yAxis.isEnabled && _yAxis.isDrawLimitLinesBehindDataEnabled

    {

        _yAxisRenderer.renderLimitLines(context: context)

    }

```

```
renderer.drawData(context: context)
```

```
if valuesToHighlight()
```

```
{
```

```
    renderer.drawHighlighted(context: context, indices: _indicesToHighlight)
```

```
}
```

```
if _yAxis.isEnabled && !_yAxis.isDrawLimitLinesBehindDataEnabled
```

```
{
```

```
    _yAxisRenderer.renderLimitLines(context: context)
```

```
}
```

```
_yAxisRenderer.renderAxisLabels(context: context)
```

```
renderer.drawValues(context: context)
```

```

        legendRenderer.renderLegend(context: context)

        drawDescription(context: context)

        drawMarkers(context: context)
    }

    /// The factor that is needed to transform values into pixels.
    @objc open var factor: CGFloat

    {

        let content = _viewPortHandler.contentRect

        return min(content.width / 2.0, content.height / 2.0)

        / CGFloat(_yAxis.axisRange)
    }

    /// The angle that each slice in the radar chart occupies.
    @objc open var sliceAngle: CGFloat

```

```

{

    return 360.0 / CGFloat(_data?.maxEntryCountSet?.entryCount ?? 0)

}

```

open override func indexForAngle(_ angle: CGFloat) -> Int

```

{

    // take the current angle of the chart into consideration

    let a = (angle - self.rotationAngle).normalizedAngle


    let sliceAngle = self.sliceAngle


    let max = _data?.maxEntryCountSet?.entryCount ?? 0

    return (0..

```

/// The object that represents all y-labels of the RadarChart.

@objc open var yAxis: YAxis

{

return _yAxis

}

/// Sets the number of web-lines that should be skipped on chart web before the next one is drawn. This targets the lines that come from the center of the RadarChart.

/// if count = 1 -> 1 line is skipped in between

@objc open var skipWebLineCount: Int

{

get

{

return _skipWebLineCount

}

set

{

_skipWebLineCount = max(0, newValue)

```
}
```

```
}
```

internal override var requiredLegendOffset: CGFloat

```
{
```

```
    return _legend.font.pointSize * 4.0
```

```
}
```

internal override var requiredBaseOffset: CGFloat

```
{
```

```
    return      _xAxis.isEnabled      &&      _xAxis.isDrawLabelsEnabled      ?  
    _xAxis.labelRotatedWidth : 10.0
```

```
}
```

open override var radius: CGFloat

```
{
```

```
    let content = _viewPortHandler.contentRect
```

```
    return min(content.width / 2.0, content.height / 2.0)
```

```

    }

    /// The maximum value this chart can display on it's y-axis.

    open override var maxY: Double { return _yAxis._axisMaximum }

    /// The minimum value this chart can display on it's y-axis.

    open override var minY: Double { return _yAxis._axisMinimum }

    /// The range of y-values this chart can display.

    @objc open var yRange: Double { return _yAxis.axisRange }

}

//

// RadarHighlighter.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

```



```

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc(RadarChartHighlighter)

open class RadarHighlighter: PieRadarHighlighter

{

    open override func closestHighlight(index: Int, x: CGFloat, y: CGFloat) -> Highlight?

    {

        guard let chart = self.chart as? RadarChartView else { return nil }

        let highlights = getHighlights(forIndex: index)

```

```
let distanceToCenter = Double(chart.distanceToCenter(x: x, y: y) / chart.factor)
```

```
var closest: Highlight?
```

```
var distance = Double.greatestFiniteMagnitude
```

```
for high in highlights
```

```
{
```

```
    let cdistance = abs(high.y - distanceToCenter)
```

```
    if cdistance < distance
```

```
    {
```

```
        closest = high
```

```
        distance = cdistance
```

```
    }
```

```
}
```

```
return closest
```

```
}
```

/// - Parameters:

/// - index:

/// - Returns: An array of Highlight objects for the given index.

/// The Highlight objects give information about the value at the selected index and DataSet it belongs to.

```
internal fun getHighlights(forIndex index: Int) -> [Highlight]
```

```
{
```

```
    var vals = [Highlight]()
```

```
    guard
```

```
        let chart = self.chart as? RadarChartView,
```

```
        let chartData = chart.data
```

```
        else { return vals }
```

```
        let phaseX = chart.chartAnimator.phaseX
```

```
        let phaseY = chart.chartAnimator.phaseY
```

```
        let sliceangle = chart.sliceAngle
```

```

let factor = chart.factor

for i in chartData.dataSets.indices

{

    guard

        let dataSet = chartData.getDataSetByIndex(i),

        let entry = dataSet.entryForIndex(index)

        else { continue }

    let y = (entry.y - chart.chartYMin)

    let p = chart.centerOffsets.moving(distance: CGFloat(y) * factor *
CGFloat(phaseY),

                                     atAngle: sliceangle * CGFloat(index) * CGFloat(phaseX) +
chart.rotationAngle)

    let highlight = Highlight(x: Double(index), y: entry.y, xPx: p.x, yPx: p.y,
dataSetIndex: i, axis: dataSet.axisDependency)

    vals.append(highlight)

```

```
    }

    return vals

}

}

//

// Range.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation
```

```
@objc(ChartRange)
```

```
open class Range: NSObject
```

```
{
```

```
    @objc open var from: Double
```

```
    @objc open var to: Double
```

```
    @objc public init(from: Double, to: Double)
```

```
{
```

```
    self.from = from
```

```
    self.to = to
```

```
    super.init()
```

```
}
```

```
/// - Parameters:
```

```
///   - value:
```

/// - Returns: `true` if this range contains (if the value is in between) the given value, `false` if not.

```
@objc open func contains(_ value: Double) -> Bool
```

```
{  
  
    if value > from && value <= to  
  
    {  
  
        return true  
  
    }  
  
    else  
  
    {  
  
        return false  
  
    }  
  
}
```

```
@objc open func isLarger(_ value: Double) -> Bool
```

```
{  
  
    return value > to  
  
}
```

```

@objc open func isSmaller(_ value: Double) -> Bool

{

    return value < from

}

}

//

// RedirectHandler.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

```


//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

import Foundation

/// A type that handles how an HTTP redirect response from a remote server should be redirected to the new request.

public protocol RedirectHandler {

/// Determines how the HTTP redirect response should be redirected to the new request.

///

/// The `completion` closure should be passed one of three possible options:

///

/// 1. The new request specified by the redirect (this is the most common use case).

/// 2. A modified version of the new request (you may want to route it somewhere else).

/// 3. A `nil` value to deny the redirect request and return the body of the redirect response.

///

/// - Parameters:

/// - task: The `URLSessionTask` whose request resulted in a redirect.

/// - request: The `URLRequest` to the new location specified by the redirect response.

/// - response: The `HTTPURLResponse` containing the server's response to the original request.

/// - completion: The closure to execute containing the new `URLRequest`, a modified `URLRequest`, or `nil`.

```
func task(_ task: URLSessionTask,  
  
         willBeRedirectedTo request: URLRequest,  
  
         for response: HTTPURLResponse,  
  
         completion: @escaping (URLRequest?) -> Void)  
  
}
```

// MARK: -

/// `Redirector` is a convenience `RedirectHandler` making it easy to follow, not follow, or modify a redirect.

```
public struct Redirector {
```

```
    /// Defines the behavior of the `Redirector` type.
```

```
    public enum Behavior {
```

```
        /// Follow the redirect as defined in the response.
```

```
        case follow
```

```
        /// Do not follow the redirect defined in the response.
```

```

    case doNotFollow

    /// Modify the redirect request defined in the response.

    case modify((URLSessionTask,   URLRequest,   HTTPURLResponse)   ->
URLRequest?)

    }

    /// Returns a `Redirector` with a `.follow` `Behavior`.

    public static let follow = Redirector(behavior: .follow)

    /// Returns a `Redirector` with a `.doNotFollow` `Behavior`.

    public static let doNotFollow = Redirector(behavior: .doNotFollow)

    /// The `Behavior` of the `Redirector`.

    public let behavior: Behavior

    /// Creates a `Redirector` instance from the `Behavior`.

    ///

    /// - Parameter behavior: The `Behavior`.

    public init(behavior: Behavior) {

```

```

        self.behavior = behavior
    }
}

```

// MARK: -

```

extension Redirector: RedirectHandler {

    public func task(_ task: URLSessionTask,

                    willBeRedirectedTo request: URLRequest,

                    for response: HTTPURLResponse,

                    completion: @escaping (URLRequest?) -> Void) {

        switch behavior {

        case .follow:

            completion(request)

        case .doNotFollow:

            completion(nil)

        case let .modify(closure):

```

```
        let request = closure(task, request, response)

        completion(request)

    }

}

}

//

// Renderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation
```

```

import CoreGraphics

@objc(ChartRenderer)

open class Renderer: NSObject

{

    /// the component that handles the drawing area of the chart and it's offsets

    @objc public let viewPortHandler: ViewPortHandler

    @objc public init(viewPortHandler: ViewPortHandler)

    {

        self.viewPortHandler = viewPortHandler

        super.init()

    }

}

//

// Request.swift

//

```

// Copyright (c) 2014-2018 Alamofire Software Foundation (<http://alamofire.org/>)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE


```
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,  
DAMAGES OR OTHER
```

```
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,  
ARISING FROM,
```

```
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER  
DEALINGS IN
```

```
// THE SOFTWARE.
```

```
//
```

```
import Foundation
```

```
/// `Request` is the common superclass of all Alamofire request types and provides common  
state, delegate, and callback
```

```
/// handling.
```

```
public class Request {
```

```
    /// State of the `Request`, with managed transitions between states set when calling  
    `resume()`, `suspend()`, or
```

```
    /// `cancel()` on the `Request`.
```

```
    public enum State {
```

```
        /// Initial state of the `Request`.
```

case initialized

/// `State` set when `resume()` is called. Any tasks created for the `Request` will have
`resume()` called on

/// them in this state.

case resumed

/// `State` set when `suspend()` is called. Any tasks created for the `Request` will have
`suspend()` called on

/// them in this state.

case suspended

/// `State` set when `cancel()` is called. Any tasks created for the `Request` will have
`cancel()` called on

/// them. Unlike `resumed` or `suspended`, once in the `cancelled` state, the `Request`
can no longer transition

/// to any other state.

case cancelled

/// `State` set when all response serialization completion closures have been cleared
on the `Request` and

/// enqueued on their respective queues.

case finished

```

/// Determines whether `self` can be transitioned to the provided `State`.

func canTransitionTo(_ state: State) -> Bool {

    switch (self, state) {

    case (.initialized, _):

        return true

    case (_, .initialized), (.cancelled, _), (.finished, _):

        return false

    case (.resumed, .cancelled), (.suspended, .cancelled), (.resumed, .suspended),
(.suspended, .resumed):

        return true

    case (.suspended, .suspended), (.resumed, .resumed):

        return false

    case (_, .finished):

        return true

    }

}

}

```

```
// MARK: - Initial State
```

```
/// `UUID` providing a unique identifier for the `Request`, used in the `Hashable` and  
`Equatable` conformances.
```

```
public let id: UUID
```

```
/// The serial queue for all internal async actions.
```

```
public let underlyingQueue: DispatchQueue
```

```
/// The queue used for all serialization actions. By default it's a serial queue that targets  
`underlyingQueue`.
```

```
public let serializationQueue: DispatchQueue
```

```
/// `EventManager` used for event callbacks.
```

```
public let eventMonitor: EventMonitor?
```

```
/// The `Request`'s interceptor.
```

```
public let interceptor: RequestInterceptor?
```

```
/// The `Request`'s delegate.
```

```
public private(set) weak var delegate: RequestDelegate?
```

```
// MARK: - Mutable State
```

/// Type encapsulating all mutable state that may need to be accessed from anything other than the `underlyingQueue`.

```
struct MutableState {
```

```
    /// State of the `Request`.
```

```
    var state: State = .initialized
```

```
    /// `ProgressHandler` and `DispatchQueue` provided for upload progress callbacks.
```

```
    var uploadProgressHandler: (handler: ProgressHandler, queue: DispatchQueue)?
```

```
    /// `ProgressHandler` and `DispatchQueue` provided for download progress callbacks.
```

```
    var downloadProgressHandler: (handler: ProgressHandler, queue: DispatchQueue)?
```

```
    /// `RedirectHandler` provided for to handle request redirection.
```

```
    var redirectHandler: RedirectHandler?
```

```
    /// `CachedResponseHandler` provided to handle response caching.
```

```
    var cachedResponseHandler: CachedResponseHandler?
```

```
    /// Closure called when the `Request` is able to create a cURL description of itself.
```

```
    var cURLHandler: ((String) -> Void)?
```

```
    /// Response serialization closures that handle response parsing.
```

```
    var responseSerializers: [(() -> Void)] = []
```

/// Response serialization completion closures executed once all response serializers are complete.

var responseSerializerCompletions: [() -> Void] = []

/// Whether response serializer processing is finished.

var responseSerializerProcessingFinished = false

/// `URLCredential` used for authentication challenges.

var credential: URLCredential?

/// All `URLRequest`'s created by Alamofire on behalf of the `Request`.

var requests: [URLRequest] = []

/// All `URLSessionTask`'s created by Alamofire on behalf of the `Request`.

var tasks: [URLSessionTask] = []

/// All `URLSessionTaskMetrics` values gathered by Alamofire on behalf of the `Request`. Should correspond

/// exactly the the `tasks` created.

var metrics: [URLSessionTaskMetrics] = []

/// Number of times any retriers provided retried the `Request`.

var retryCount = 0

/// Final `AFError` for the `Request`, whether from various internal Alamofire calls or as a result of a `task`.

```

var error: AFError?

    /// Whether the instance has had `finish()` called and is running the serializers. Should
    be replaced with a

    /// representation in the state machine in the future.

    var isFinishing = false

}

    /// Protected `MutableState` value that provides thread-safe access to state values.

    fileprivate let protectedMutableState: Protector<MutableState> =
    Protector(MutableState())

    /// `State` of the `Request`.

    public var state: State { return protectedMutableState.directValue.state }

    /// Returns whether `state` is `.initialized`.

    public var isInitialized: Bool { return state == .initialized }

    /// Returns whether `state` is `.resumed`.

    public var isResumed: Bool { return state == .resumed }

    /// Returns whether `state` is `.suspended`.

```

```

public var isSuspended: Bool { return state == .suspended }

/// Returns whether `state` is `.cancelled`.

public var isCancelled: Bool { return state == .cancelled }

/// Returns whether `state` is `.finished`.

public var isFinished: Bool { return state == .finished }


// MARK: Progress

/// Closure type executed when monitoring the upload or download progress of a request.

public typealias ProgressHandler = (Progress) -> Void


/// `Progress` of the upload of the body of the executed `URLRequest`. Reset to `0` if
the `Request` is retried.

public let uploadProgress = Progress(totalUnitCount: 0)

/// `Progress` of the download of any response data. Reset to `0` if the `Request` is retried.

public let downloadProgress = Progress(totalUnitCount: 0)

/// `ProgressHandler` called when `uploadProgress` is updated, on the provided
`DispatchQueue`.

```



```
fileprivate var uploadProgressHandler: (handler: ProgressHandler, queue:
DispatchQueue)? {
```

```
    get { return protectedMutableState.directValue.uploadProgressHandler }
```

```
    set { protectedMutableState.write { $0.uploadProgressHandler = newValue } }
```

```
}
```

```
/// `ProgressHandler` called when `downloadProgress` is updated, on the provided
`DispatchQueue`.
```

```
fileprivate var downloadProgressHandler: (handler: ProgressHandler, queue:
DispatchQueue)? {
```

```
    get { return protectedMutableState.directValue.downloadProgressHandler }
```

```
    set { protectedMutableState.write { $0.downloadProgressHandler = newValue } }
```

```
}
```

```
// MARK: Redirect Handling
```

```
/// `RedirectHandler` set on the instance.
```

```
public private(set) var redirectHandler: RedirectHandler? {
```

```
    get { return protectedMutableState.directValue.redirectHandler }
```

```

        set { protectedMutableState.write { $0.redirectHandler = newValue } }

    }

    // MARK: Cached Response Handling

    /// `CachedResponseHandler` set on the instance.

    public private(set) var cachedResponseHandler: CachedResponseHandler? {

        get { return protectedMutableState.directValue.cachedResponseHandler }

        set { protectedMutableState.write { $0.cachedResponseHandler = newValue } }

    }

```

```

    // MARK: URLCredential

```

```

    /// `URLCredential` used for authentication challenges. Created by calling one of the
    `authenticate` methods.

```

```

    public private(set) var credential: URLCredential? {

        get { return protectedMutableState.directValue.credential }

        set { protectedMutableState.write { $0.credential = newValue } }

    }

```

```
}
```

```
// MARK: Validators
```

```
/// `Validator` callback closures that store the validation calls enqueued.
```

```
fileprivate var protectedValidators: Protector<[() -> Void]> = Protector([])
```

```
// MARK: URLRequest
```

```
/// All `URLRequest` created on behalf of the `Request`, including original and adapted requests.
```

```
public var requests: [URLRequest] { return protectedMutableState.directValue.requests }
```

```
/// First `URLRequest` created on behalf of the `Request`. May not be the first one actually executed.
```

```
public var firstRequest: URLRequest? { return requests.first }
```

```
/// Last `URLRequest` created on behalf of the `Request`.
```

```
public var lastRequest: URLRequest? { return requests.last }
```

```
/// Current `URLRequest` created on behalf of the `Request`.
```

```

public var request: URLRequest? { return lastRequest }

    /// `URLRequest`s from all of the `URLSessionTask`s executed on behalf of the
    `Request`. May be different from

    /// `requests` due to `URLSession` manipulation.

    public var performedRequests: [URLRequest] {

        return protectedMutableState.read { $0.tasks.compactMap { $0.currentRequest } }

    }

    // MARK: HTTPURLResponse

    /// `HTTPURLResponse` received from the server, if any. If the `Request` was retried,
    this is the response of the

    /// last `URLSessionTask`.

    public var response: HTTPURLResponse? { return lastTask?.response as?
    HTTPURLResponse }

    // MARK: Tasks

```

```
/// All `URLSessionTask`s created on behalf of the `Request`.
```

```
public var tasks: [URLSessionTask] { return protectedMutableState.directValue.tasks }
```

```
/// First `URLSessionTask` created on behalf of the `Request`.
```

```
public var firstTask: URLSessionTask? { return tasks.first }
```

```
/// Last `URLSessionTask` created on behalf of the `Request`.
```

```
public var lastTask: URLSessionTask? { return tasks.last }
```

```
/// Current `URLSessionTask` created on behalf of the `Request`.
```

```
public var task: URLSessionTask? { return lastTask }
```

```
// MARK: Metrics
```

```
/// All `URLSessionTaskMetrics` gathered on behalf of the `Request`. Should  
correspond to the `tasks` created.
```

```
public var allMetrics: [URLSessionTaskMetrics] { return  
protectedMutableState.directValue.metrics }
```

```
/// First `URLSessionTaskMetrics` gathered on behalf of the `Request`.
```

```
public var firstMetrics: URLSessionTaskMetrics? { return allMetrics.first }
```

```
/// Last `URLSessionTaskMetrics` gathered on behalf of the `Request`.
```

```

public var lastMetrics: URLSessionTaskMetrics? { return allMetrics.last }

/// Current `URLSessionTaskMetrics` gathered on behalf of the `Request`.

public var metrics: URLSessionTaskMetrics? { return lastMetrics }


// MARK: Retry Count


/// Number of times the `Request` has been retried.

public var retryCount: Int { return protectedMutableState.directValue.retryCount }


// MARK: Error


/// `Error` returned from Alamofire internally, from the network request directly, or any
validators executed.

public fileprivate(set) var error: AFError? {

    get { return protectedMutableState.directValue.error }

    set { protectedMutableState.write { $0.error = newValue } }

}

```

```

/// Default initializer for the `Request` superclass.

///

/// - Parameters:

///   - id:          `UUID` used for the `Hashable` and `Equatable` implementations.
///               `UUID()` by default.

///   - underlyingQueue:  `DispatchQueue` on which all internal `Request` work is
///               performed.

///   - serializationQueue: `DispatchQueue` on which all serialization work is performed.
///               By default targets
///               `underlyingQueue`, but can be passed another queue from a `Session`.

///   - eventMonitor:    `EventMonitor` called for event callbacks from internal `Request`
///               actions.

///   - interceptor:     `RequestInterceptor` used throughout the request lifecycle.

///   - delegate:        `RequestDelegate` that provides an interface to actions not performed
///               by the `Request`.

init(id: UUID = UUID(),

    underlyingQueue: DispatchQueue,

    serializationQueue: DispatchQueue,

    eventMonitor: EventMonitor?,

    interceptor: RequestInterceptor?,

```

```
delegate: RequestDelegate) {  
  
    self.id = id  
  
    self.underlyingQueue = underlyingQueue  
  
    self.serializationQueue = serializationQueue  
  
    self.eventMonitor = eventMonitor  
  
    self.interceptor = interceptor  
  
    self.delegate = delegate  
  
}
```

```
// MARK: - Internal Event API
```

```
// All API must be called from underlyingQueue.
```

```
/// Called when an initial `URLRequest` has been created on behalf of the instance. If a  
`RequestAdapter` is active,
```

```
/// the `URLRequest` will be adapted before being issued.
```

```
///
```

```
/// - Parameter request: The `URLRequest` created.
```



```

func didCreateInitialURLRequest(_ request: URLRequest) {

    dispatchPrecondition(condition: .onQueue(underlyingQueue))

    protectedMutableState.write { $0.requests.append(request) }

    eventMonitor?.request(self, didCreateInitialURLRequest: request)

}

```

/// Called when initial `URLRequest` creation has failed, typically through a
`URLRequestConvertible`.

///

/// - Note: Triggers retry.

///

/// - Parameter error: `AFError` thrown from the failed creation.

```

func didFailToCreateURLRequest(with error: AFError) {

    dispatchPrecondition(condition: .onQueue(underlyingQueue))

    self.error = error

```

```

eventMonitor?.request(self, didFailToCreateURLRequestWithError: error)

callCURLHandlerIfNecessary()

retryOrFinish(error: error)

}

/// Called when a `RequestAdapter` has successfully adapted a `URLRequest`.

///

/// - Parameters:

/// - initialRequest: The `URLRequest` that was adapted.

/// - adaptedRequest: The `URLRequest` returned by the `RequestAdapter`.

func didAdaptInitialRequest(_ initialRequest: URLRequest, to adaptedRequest:
URLRequest) {

    dispatchPrecondition(condition: .onQueue(underlyingQueue))

    protectedMutableState.write { $0.requests.append(adaptedRequest) }

```

```

        eventMonitor?.request(self, didAdaptInitialRequest: initialRequest, to:
adaptedRequest)

    }

```

```

/// Called when a `RequestAdapter` fails to adapt a `URLRequest`.

```

```

///

```

```

/// - Note: Triggers retry.

```

```

///

```

```

/// - Parameters:

```

```

/// - request: The `URLRequest` the adapter was called with.

```

```

/// - error: The `AFError` returned by the `RequestAdapter`.

```

```

func didFailToAdaptURLRequest(_ request: URLRequest, withError error: AFError) {

```

```

    dispatchPrecondition(condition: .onQueue(underlyingQueue))

```

```

    self.error = error

```

```

    eventMonitor?.request(self, didFailToAdaptURLRequest: request, withError: error)

```

```

        callCURLHandlerIfNecessary()

        retryOrFinish(error: error)
    }

    /// Final `URLRequest` has been created for the instance.

    ///

    /// - Parameter request: The `URLRequest` created.

    func didCreateURLRequest(_ request: URLRequest) {

        dispatchPrecondition(condition: .onQueue(underlyingQueue))

        eventMonitor?.request(self, didCreateURLRequest: request)

        callCURLHandlerIfNecessary()
    }

```

```
/// Asynchronously calls any stored `cURLHandler` and then removes it from  
`mutableState`.
```

```
private func callCURLHandlerIfNecessary() {  
  
    protectedMutableState.write { mutableState in  
  
        guard let cURLHandler = mutableState.cURLHandler else { return }  
  
        self.underlyingQueue.async { cURLHandler(self.cURLDescription()) }  
  
        mutableState.cURLHandler = nil  
  
    }  
  
}
```

```
/// Called when a `URLSessionTask` is created on behalf of the instance.
```

```
///
```

```
/// - Parameter task: The `URLSessionTask` created.
```

```
func didCreateTask(_ task: URLSessionTask) {  
  
    dispatchPrecondition(condition: .onQueue(underlyingQueue))  
  
  
    protectedMutableState.write { $0.tasks.append(task) }
```

```
eventMonitor?.request(self, didCreateTask: task)

}
```

```
/// Called when resumption is completed.
```

```
func didResume() {

    dispatchPrecondition(condition: .onQueue(underlyingQueue))

    eventMonitor?.requestDidResume(self)

}
```

```
/// Called when a `URLSessionTask` is resumed on behalf of the instance.
```

```
///
```

```
/// - Parameter task: The `URLSessionTask` resumed.
```

```
func didResumeTask(_ task: URLSessionTask) {

    dispatchPrecondition(condition: .onQueue(underlyingQueue))

}
```

```

        eventMonitor?.request(self, didResumeTask: task)

    }

    /// Called when suspension is completed.

    func didSuspend() {

        dispatchPrecondition(condition: .onQueue(underlyingQueue))

        eventMonitor?.requestDidSuspend(self)

    }

    /// Called when a `URLSessionTask` is suspended on behalf of the instance.

    ///

    /// - Parameter task: The `URLSessionTask` suspended.

    func didSuspendTask(_ task: URLSessionTask) {

        dispatchPrecondition(condition: .onQueue(underlyingQueue))

        eventMonitor?.request(self, didSuspendTask: task)

```

```
}
```

```
/// Called when cancellation is completed, sets `error` to `AFError.explicitlyCancelled`.
```

```
func didCancel() {
```

```
    dispatchPrecondition(condition: .onQueue(underlyingQueue))
```

```
    error = AFError.explicitlyCancelled
```

```
    eventMonitor?.requestDidCancel(self)
```

```
}
```

```
/// Called when a `URLSessionTask` is cancelled on behalf of the instance.
```

```
///
```

```
/// - Parameter task: The `URLSessionTask` cancelled.
```

```
func didCancelTask(_ task: URLSessionTask) {
```

```
    dispatchPrecondition(condition: .onQueue(underlyingQueue))
```



```

        eventMonitor?.request(self, didCancelTask: task)

    }

    /// Called when a `URLSessionTaskMetrics` value is gathered on behalf of the instance.

    ///

    /// - Parameter metrics: The `URLSessionTaskMetrics` gathered.

    func didGatherMetrics(_ metrics: URLSessionTaskMetrics) {

        dispatchPrecondition(condition: .onQueue(underlyingQueue))

        protectedMutableState.write { $0.metrics.append(metrics) }

        eventMonitor?.request(self, didGatherMetrics: metrics)

    }

    /// Called when a `URLSessionTask` fails before it is finished, typically during
    certificate pinning.

    ///

    /// - Parameters:

```

```
/// - task: The `URLSessionTask` which failed.
```

```
/// - error: The early failure `AFError`.
```

```
func didFailTask(_ task: URLSessionTask, earlyWithError error: AFError) {
```

```
    dispatchPrecondition(condition: .onQueue(underlyingQueue))
```

```
    self.error = error
```

```
    // Task will still complete, so didCompleteTask(_:with:) will handle retry.
```

```
    eventMonitor?.request(self, didFailTask: task, earlyWithError: error)
```

```
}
```

```
/// Called when a `URLSessionTask` completes. All tasks will eventually call this method.
```

```
///
```

```
/// - Note: Response validation is synchronously triggered in this step.
```

```
///
```

```
/// - Parameters:
```

```
/// - task: The `URLSessionTask` which completed.
```

/// - error: The `AFError` `task` may have completed with. If `error` has already been set on the instance, this

/// value is ignored.

```
func didCompleteTask(_ task: URLSessionTask, with error: AFError?) {
```

```
    dispatchPrecondition(condition: .onQueue(underlyingQueue))
```

```
    self.error = self.error ?? error
```

```
    protectedValidators.directValue.forEach { $0() }
```

```
    eventMonitor?.request(self, didCompleteTask: task, with: error)
```

```
    retryOrFinish(error: self.error)
```

```
}
```

/// Called when the `RequestDelegate` is going to retry this `Request`. Calls `reset()`.

```
func prepareForRetry() {
```

```
    dispatchPrecondition(condition: .onQueue(underlyingQueue))
```

```
protectedMutableState.write { $0.retryCount += 1 }
```

```
reset()
```

```
eventMonitor?.requestIsRetrying(self)
```

```
}
```

```
/// Called to determine whether retry will be triggered for the particular error, or whether  
the instance should
```

```
/// call `finish()`.
```

```
///
```

```
/// - Parameter error: The possible `AFError` which may trigger retry.
```

```
func retryOrFinish(error: AFError?) {
```

```
    dispatchPrecondition(condition: .onQueue(underlyingQueue))
```

```
    guard let error = error, let delegate = delegate else { finish(); return }
```

```

delegate.retryResult(for: self, dueTo: error) { retryResult in

    switch retryResult {

    case .doNotRetry:

        self.finish()

    case let .doNotRetryWithError(retryError):

        self.finish(error: retryError.asAFError(orFailWith: "Received retryError was not
already AFError"))

    case .retry, .retryWithDelay:

        delegate.retryRequest(self, withDelay: retryResult.delay)

    }

}

}

}

```

/// Finishes this `Request` and starts the response serializers.

///

/// - Parameter error: The possible `Error` with which the instance will finish.

```

func finish(error: AFError? = nil) {

    dispatchPrecondition(condition: .onQueue(underlyingQueue))

```

```
guard !protectedMutableState.directValue.isFinishing else { return }
```

```
protectedMutableState.directValue.isFinishing = true
```

```
if let error = error { self.error = error }
```

```
// Start response handlers
```

```
processNextResponseSerializer()
```

```
eventMonitor?.requestDidFinish(self)
```

```
}
```

```
/// Appends the response serialization closure to the instance.
```

```
///
```

```
/// - Note: This method will also `resume` the instance if `delegate.startImmediately`  
returns `true`.
```

```
///
```

/// - Parameter closure: The closure containing the response serialization call.

```
func appendResponseSerializer(_ closure: @escaping () -> Void) {

    protectedMutableState.write { mutableState in

        mutableState.responseSerializers.append(closure)

    }

    if mutableState.state == .finished {

        mutableState.state = .resumed

    }

    if mutableState.responseSerializerProcessingFinished {

        underlyingQueue.async { self.processNextResponseSerializer() }

    }

    if mutableState.state.canTransitionTo(.resumed) {

        underlyingQueue.async { if self.delegate?.startImmediately == true
        { self.resume() } }

    }

}
```

```
}
```

```
/// Returns the next response serializer closure to execute if there's one left.
```

```
///
```

```
/// - Returns: The next response serialization closure, if there is one.
```

```
func nextResponseSerializer() -> (() -> Void)? {
```

```
    var responseSerializer: (() -> Void)?
```

```
    protectedMutableState.write { mutableState in
```

```
        let responseSerializerIndex = mutableState.responseSerializerCompletions.count
```

```
        if responseSerializerIndex < mutableState.responseSerializers.count {
```

```
            responseSerializer = mutableState.responseSerializers[responseSerializerIndex]
```

```
        }
```

```
    }
```

```
    return responseSerializer
```



```
}
```

```
/// Processes the next response serializer and calls all completions if response  
serialization is complete.
```

```
func processNextResponseSerializer() {
```

```
    guard let responseSerializer = nextResponseSerializer() else {
```

```
        // Execute all response serializer completions and clear them
```

```
        var completions: [() -> Void] = []
```

```
        protectedMutableState.write { mutableState in
```

```
            completions = mutableState.responseSerializerCompletions
```

```
            // Clear out all response serializers and response serializer completions in  
mutable state since the
```

```
            // request is complete. It's important to do this prior to calling the completion  
closures in case
```

```
            // the completions call back into the request triggering a re-processing of the  
response serializers.
```

```
            // An example of how this can happen is by calling cancel inside a response  
completion closure.
```

```
mutableState.responseSerializers.removeAll()
```

```
mutableState.responseSerializerCompletions.removeAll()
```

```
if mutableState.state.canTransitionTo(.finished) {
```

```
    mutableState.state = .finished
```

```
}
```

```
mutableState.responseSerializerProcessingFinished = true
```

```
mutableState.isFinishing = false
```

```
}
```

```
completions.forEach { $0() }
```

```
// Cleanup the request
```

```
cleanup()
```

```
return
```

```
}
```

```
serializationQueue.async { responseSerializer() }
```

```
}
```

```
/// Notifies the `Request` that the response serializer is complete.
```

```
///
```

```
/// - Parameter completion: The completion handler provided with the response serializer,  
called when all serializers
```

```
/// are complete.
```

```
func responseSerializerDidComplete(completion: @escaping () -> Void) {
```

```
    protectedMutableState.write
```

```
{ $0.responseSerializerCompletions.append(completion) }
```

```
    processNextResponseSerializer()
```

```
}
```

```
/// Resets all task and response serializer related state for retry.
```

```
func reset() {
```

```
error = nil
```

```
uploadProgress.totalUnitCount = 0
```

```
uploadProgress.completedUnitCount = 0
```

```
downloadProgress.totalUnitCount = 0
```

```
downloadProgress.completedUnitCount = 0
```

```
protectedMutableState.write { state in
```

```
    state.isFinishing = false
```

```
    state.responseSerializerCompletions = []
```

```
}
```

```
}
```

```
/// Called when updating the upload progress.
```

```
///
```

```
/// - Parameters:
```

```
/// - totalBytesSent: Total bytes sent so far.
```

```

/// - totalBytesExpectedToSend: Total bytes expected to send.

func updateUploadProgress(totalBytesSent: Int64, totalBytesExpectedToSend: Int64) {

    uploadProgress.totalUnitCount = totalBytesExpectedToSend

    uploadProgress.completedUnitCount = totalBytesSent


    uploadProgressHandler?.queue.async
{ self.uploadProgressHandler?.handler(self.uploadProgress) }

}


/// Perform a closure on the current `state` while locked.

///

/// - Parameter perform: The closure to perform.

func withState(perform: (State) -> Void) {

    protectedMutableState.withState(perform: perform)

}


// MARK: Task Creation

```

/// Called when creating a `URLSessionTask` for this `Request`. Subclasses must override.

///

/// - Parameters:

/// - request: `URLRequest` to use to create the `URLSessionTask`.

/// - session: `URLSession` which creates the `URLSessionTask`.

///

/// - Returns: The `URLSessionTask` created.

```
func task(for request: URLRequest, using session: URLSession) -> URLSessionTask {  
  
    fatalError("Subclasses must override.")  
  
}
```

// MARK: - Public API

// These APIs are callable from any queue.

// MARK: State

/// Cancels the instance. Once cancelled, a `Request` can no longer be resumed or suspended.

///

/// - Returns: The instance.

@discardableResult

public func cancel() -> Self {

protectedMutableState.write { mutableState in

guard mutableState.state.canTransitionTo(.cancelled) else { return }

mutableState.state = .cancelled

underlyingQueue.async { self.didCancel() }

guard let task = mutableState.tasks.last, task.state != .completed else {

underlyingQueue.async { self.finish() }

return

}

```

        // Resume to ensure metrics are gathered.

        task.resume()

        task.cancel()

        underlyingQueue.async { self.didCancelTask(task) }

    }

    return self
}

/// Suspends the instance.

///

/// - Returns: The instance.

@discardableResult

public func suspend() -> Self {

    protectedMutableState.write { mutableState in

        guard mutableState.state.canTransitionTo(.suspended) else { return }

```



```
mutableState.state = .suspended
```

```
underlyingQueue.async { self.didSuspend() }
```

```
guard let task = mutableState.tasks.last, task.state != .completed else { return }
```

```
task.suspend()
```

```
underlyingQueue.async { self.didSuspendTask(task) }
```

```
}
```

```
return self
```

```
}
```

```
/// Resumes the instance.
```

```
///
```

```
/// - Returns: The instance.
```

```
@discardableResult
```

```

public func resume() -> Self {

    protectedMutableState.write { mutableState in

        guard mutableState.state.canTransitionTo(.resumed) else { return }

        mutableState.state = .resumed

        underlyingQueue.async { self.didResume() }

        guard let task = mutableState.tasks.last, task.state != .completed else { return }

        task.resume()

        underlyingQueue.async { self.didResumeTask(task) }

    }

    return self
}

```

```

// MARK: - Closure API

/// Associates a credential using the provided values with the instance.

///

/// - Parameters:

/// - username: The username.

/// - password: The password.

/// - persistence: The `URLCredential.Persistence` for the created `URLCredential`.
`.forSession` by default.

///

/// - Returns: The instance.

@discardableResult

public func authenticate(username: String, password: String, persistence:
URLCredential.Persistence = .forSession) -> Self {

    let credential = URLCredential(user: username, password: password, persistence:
persistence)

    return authenticate(with: credential)

}

```

```
/// Associates the provided credential with the instance.
```

```
///
```

```
/// - Parameter credential: The `URLCredential`.
```

```
///
```

```
/// - Returns: The instance.
```

```
@discardableResult
```

```
public func authenticate(with credential: URLCredential) -> Self {
```

```
    protectedMutableState.write { $0.credential = credential }
```

```
    return self
```

```
}
```

```
/// Sets a closure to be called periodically during the lifecycle of the instance as data is  
read from the server.
```

```
///
```

```
/// - Note: Only the last closure provided is used.
```

```
///
```

/// - Parameters:

/// - queue: The `DispatchQueue` to execute the closure on. `.main` by default.

/// - closure: The closure to be executed periodically as data is read from the server.

///

/// - Returns: The instance.

@discardableResult

```
public func downloadProgress(queue: DispatchQueue = .main, closure: @escaping  
ProgressHandler) -> Self {
```

```
    protectedMutableState.write { $0.downloadProgressHandler = (handler: closure,  
queue: queue) }
```

```
    return self
```

```
}
```

/// Sets a closure to be called periodically during the lifecycle of the instance as data is sent to the server.

///

/// - Note: Only the last closure provided is used.

///

/// - Parameters:

/// - queue: The `DispatchQueue` to execute the closure on. `.main` by default.

/// - closure: The closure to be executed periodically as data is sent to the server.

///

/// - Returns: The instance.

@discardableResult

```
public func uploadProgress(queue: DispatchQueue = .main, closure: @escaping
ProgressHandler) -> Self {
```

```
    protectedMutableState.write { $0.uploadProgressHandler = (handler: closure, queue:
queue) }
```

```
    return self
```

```
}
```

// MARK: Redirects

/// Sets the redirect handler for the instance which will be used if a redirect response is encountered.

///

/// - Note: Attempting to set the redirect handler more than once is a logic error and will crash.

///

/// - Parameter handler: The `RedirectHandler`.

///

/// - Returns: The instance.

@discardableResult

```
public func redirect(using handler: RedirectHandler) -> Self {  
  
    protectedMutableState.write { mutableState in  
  
        precondition(mutableState.redirectHandler == nil, "Redirect handler has already  
been set.")  
  
        mutableState.redirectHandler = handler  
  
    }  
  
    return self  
  
}
```

// MARK: Cached Responses

/// Sets the cached response handler for the `Request` which will be used when attempting to cache a response.

///

/// - Note: Attempting to set the cache handler more than once is a logic error and will crash.

///

/// - Parameter handler: The `CachedResponseHandler`.

///

/// - Returns: The instance.

@discardableResult

public func cacheResponse(using handler: CachedResponseHandler) -> Self {

protectedMutableState.write { mutableState in

precondition(mutableState.cachedResponseHandler == nil, "Cached response handler has already been set.")

mutableState.cachedResponseHandler = handler

}

return self


```
}
```

```
/// Sets a handler to be called when the cURL description of the request is available.
```

```
///
```

```
/// - Note: When waiting for a `Request`'s `URLRequest` to be created, only the last  
`handler` will be called.
```

```
///
```

```
/// - Parameter handler: Closure to be called when the cURL description is available.
```

```
///
```

```
/// - Returns: The instance.
```

```
@discardableResult
```

```
public func cURLDescription(calling handler: @escaping (String) -> Void) -> Self {
```

```
    protectedMutableState.write { mutableState in
```

```
        if mutableState.requests.last != nil {
```

```
            underlyingQueue.async { handler(self.cURLDescription()) }
```

```
        } else {
```

```
            mutableState.cURLHandler = handler
```

```
        }
```

```
}
```

```
return self
```

```
}
```

```
// MARK: Cleanup
```

```
/// Final cleanup step executed when the instance finishes response serialization.
```

```
func cleanup() {
```

```
    delegate?.cleanup(after: self)
```

```
    // No-op: override in subclass
```

```
}
```

```
}
```

```
// MARK: - Protocol Conformances
```

```
extension Request: Equatable {
```

```

public static func ==(lhs: Request, rhs: Request) -> Bool {

    return lhs.id == rhs.id

}

}

```

```

extension Request: Hashable {

    public func hash(into hasher: inout Hasher) {

        hasher.combine(id)

    }

}

```

```

extension Request: CustomStringConvertible {

    /// A textual representation of this instance, including the `HTTPMethod` and `URL` if
    the `URLRequest` has been

    /// created, as well as the response status code, if a response has been received.

    public var description: String {

        guard let request = performedRequests.last ?? lastRequest,

            let url = request.url,

```

```
let method = request.httpMethod else { return "No request created yet." }
```

```
let requestDescription = "\(method) \(url.absoluteString)"
```

```
return response.map { "\(requestDescription) \($0.statusCode)" } ??  
requestDescription  
  
}  
  
}
```

```
extension Request {
```

```
    /// cURL representation of the instance.
```

```
    ///
```

```
    /// - Returns: The cURL equivalent of the instance.
```

```
    public func cURLDescription() -> String {
```

```
        guard
```

```
            let request = lastRequest,
```

```
            let url = request.url,
```

```
            let host = url.host,
```

```
        let method = request.httpMethod else { return "$ curl command could not be
created" }
```

```
var components = ["$ curl -v"]
```

```
components.append("-X \$(method)")
```

```
if let credentialStorage = delegate?.sessionConfiguration.urlCredentialStorage {
```

```
    let protectionSpace = URLProtectionSpace(host: host,
```

```
        port: url.port ?? 0,
```

```
        protocol: url.scheme,
```

```
        realm: host,
```

```
        authenticationMethod:
```

```
NSURLAuthenticationMethodHTTPBasic)
```

```
    if let credentials = credentialStorage.credentials(for: protectionSpace)?.values {
```

```
        for credential in credentials {
```

```
            guard let user = credential.user, let password = credential.password else
{ continue }
```

```

        components.append("-u \$(user):\$(password)")

    }

} else {

    if let credential = credential, let user = credential.user, let password =
credential.password {

        components.append("-u \$(user):\$(password)")

    }

}

}

if let configuration = delegate?.sessionConfiguration,
configuration.httpShouldSetCookies {

    if

        let cookieStorage = configuration.httpCookieStorage,

        let cookies = cookieStorage.cookies(for: url), !cookies.isEmpty {

            let allCookies = cookies.map { "\$(0.name)=\$(0.value)" }.joined(separator: ";")

            components.append("-b \"\$(allCookies)\")")

```

```

    }

}

var headers = HTTPHeaders()

if let sessionHeaders = delegate?.sessionConfiguration.headers {

    for header in sessionHeaders where header.name != "Cookie" {

        headers[header.name] = header.value

    }

}

for header in request.headers where header.name != "Cookie" {

    headers[header.name] = header.value

}

for header in headers {

    let escapedValue = header.value.replacingOccurrences(of: "\"", with: "\\")

```

```

        components.append("-H \"(header.name): \"(escapedValue)\"")
    }

    if let httpBodyData = request.httpBody {

        let httpBody = String(decoding: httpBodyData, as: UTF8.self)

        var escapedBody = httpBody.replacingOccurrences(of: "\"", with: "\\\"")

        escapedBody = escapedBody.replacingOccurrences(of: "\\", with: "\\")

        components.append("-d \"(escapedBody)\"")
    }

    components.append("\"(url.absoluteString)\"")

    return components.joined(separator: "\n\t")
}
}

```



```

/// Protocol abstraction for `Request`'s communication back to the `SessionDelegate`.

public protocol RequestDelegate: AnyObject {

    /// `URLSessionConfiguration` used to create the underlying `URLSessionTask`'s.

    var sessionConfiguration: URLSessionConfiguration { get }

    /// Determines whether the `Request` should automatically call `resume()` when adding
    the first response handler.

    var startImmediately: Bool { get }

    /// Notifies the delegate the `Request` has reached a point where it needs cleanup.

    ///

    /// - Parameter request: The `Request` to cleanup after.

    func cleanup(after request: Request)

    /// Asynchronously ask the delegate whether a `Request` will be retried.

    ///

    /// - Parameters:

    /// - request: `Request` which failed.

```

```

    /// - error:    `Error` which produced the failure.

    /// - completion: Closure taking the `RetryResult` for evaluation.

    func retryResult(for request: Request, dueTo error: NSError, completion: @escaping
    (RetryResult) -> Void)

    /// Asynchronously retry the `Request`.

    ///

    /// - Parameters:

    /// - request:  `Request` which will be retried.

    /// - timeDelay: `TimeInterval` after which the retry will be triggered.

    func retryRequest(_ request: Request, withDelay timeDelay: TimeInterval?)

}

// MARK: - Subclasses

// MARK: - DataRequest

```

```
/// `Request` subclass which handles in-memory `Data` download using  
`URLSessionDataTask`.
```

```
public class DataRequest: Request {
```

```
    /// `URLRequestConvertible` value used to create `URLRequest`'s for this instance.
```

```
    public let convertible: URLRequestConvertible
```

```
    /// `Data` read from the server so far.
```

```
    public var data: Data? { return protectedData.directValue }
```

```
    /// Protected storage for the `Data` read by the instance.
```

```
    private var protectedData: Protector<Data?> = Protector(nil)
```

```
    /// Creates a `DataRequest` using the provided parameters.
```

```
    ///
```

```
    /// - Parameters:
```

```
    /// - id:          `UUID` used for the `Hashable` and `Equatable` implementations.  
    ///               `UUID()` by default.
```

```
    /// - convertible: `URLRequestConvertible` value used to create `URLRequest`'s  
    ///               for this instance.
```

/// - underlyingQueue: `DispatchQueue` on which all internal `Request` work is performed.

/// - serializationQueue: `DispatchQueue` on which all serialization work is performed.
By default targets

/// `underlyingQueue`, but can be passed another queue from a `Session`.

/// - eventMonitor: `EventMonitor` called for event callbacks from internal `Request` actions.

/// - interceptor: `RequestInterceptor` used throughout the request lifecycle.

/// - delegate: `RequestDelegate` that provides an interface to actions not performed by the `Request`.

init(id: UUID = UUID(),

convertible: URLRequestConvertible,

underlyingQueue: DispatchQueue,

serializationQueue: DispatchQueue,

eventMonitor: EventMonitor?,

interceptor: RequestInterceptor?,

delegate: RequestDelegate) {

self.convertible = convertible

```

super.init(id: id,

        underlyingQueue: underlyingQueue,

        serializationQueue: serializationQueue,

        eventMonitor: eventMonitor,

        interceptor: interceptor,

        delegate: delegate)

}

override func reset() {

    super.reset()

    protectedData.directValue = nil

}

/// Called when `Data` is received by this instance.

///

/// - Note: Also calls `updateDownloadProgress`.

```

```
///
```

```
/// - Parameter data: The `Data` received.
```

```
func didReceive(data: Data) {
```

```
    if self.data == nil {
```

```
        protectedData.directValue = data
```

```
    } else {
```

```
        protectedData.append(data)
```

```
    }
```

```
    updateDownloadProgress()
```

```
}
```

```
override func task(for request: URLRequest, using session: URLSession) ->  
URLSessionTask {
```

```
    let copiedRequest = request
```

```
    return session.dataTask(with: copiedRequest)
```

```
}
```

```

/// Called to updated the `downloadProgress` of the instance.

func updateDownloadProgress() {

    let totalBytesReceived = Int64(data?.count ?? 0)

    let    totalBytesExpected    =    task?.response?.expectedContentLength    ??
NSURLSessionTransferSizeUnknown

    downloadProgress.totalUnitCount = totalBytesExpected

    downloadProgress.completedUnitCount = totalBytesReceived

    downloadProgressHandler?.queue.async
{ self.downloadProgressHandler?.handler(self.downloadProgress) }

}

/// Validates the request, using the specified closure.

///

/// - Note: If validation fails, subsequent calls to response handlers will have an
associated error.

///

/// - Parameter validation: `Validation` closure used to validate the response.

```

```
///
```

```
/// - Returns:          The instance.
```

```
@discardableResult
```

```
public func validate(_ validation: @escaping Validation) -> Self {
```

```
    let validator: () -> Void = { [unowned self] in
```

```
        guard self.error == nil, let response = self.response else { return }
```

```
        let result = validation(self.request, response, self.data)
```

```
        if case let .failure(error) = result { self.error =  
error.asAFError(or: .responseValidationFailed(reason: .customValidationFailed(error:  
error)))) }
```

```
        self.eventMonitor?.request(self,
```

```
            didValidateRequest: self.request,
```

```
            response: response,
```

```
            data: self.data,
```

```
            withResult: result)
```



```
}
```

```
protectedValidators.append(validator)
```

```
return self
```

```
}
```

```
}
```

```
// MARK: - DownloadRequest
```

```
/// `Request` subclass which downloads `Data` to a file on disk using  
`URLSessionDownloadTask`.
```

```
public class DownloadRequest: Request {
```

```
    /// A set of options to be executed prior to moving a downloaded file from the temporary  
    `URL` to the destination
```

```
    /// `URL`.
```

```
    public struct Options: OptionSet {
```

```
        /// Specifies that intermediate directories for the destination URL should be created.
```

```
public static let createIntermediateDirectories = Options(rawValue: 1 << 0)
```

```
/// Specifies that any previous file at the destination `URL` should be removed.
```

```
public static let removePreviousFile = Options(rawValue: 1 << 1)
```

```
public let rawValue: Int
```

```
public init(rawValue: Int) {
```

```
    self.rawValue = rawValue
```

```
}
```

```
}
```

```
// MARK: Destination
```

```
/// A closure executed once a `DownloadRequest` has successfully completed in order  
to determine where to move the
```

```
/// temporary file written to during the download process. The closure takes two  
arguments: the temporary file URL
```

```
/// and the URL response, and returns a two arguments: the file URL where the temporary  
file should be moved and
```

```

    /// the options defining how the file should be moved.

    public typealias Destination = (_ temporaryURL: URL,
                                   _ response: HTTPURLResponse) -> (destinationURL: URL,
options: Options)

    /// Creates a download file destination closure which uses the default file manager to
move the temporary file to a

    /// file URL in the first available directory with the specified search path directory and
search path domain mask.

    ///

    /// - Parameters:

    /// - directory: The search path directory. `.documentDirectory` by default.

    /// - domain: The search path domain mask. `.userDomainMask` by default.

    /// - options: `DownloadRequest.Options` used when moving the downloaded file to
its destination. None by

    /// default.

    /// - Returns: The `Destination` closure.

    public class func suggestedDownloadDestination(for directory:
FileManager.SearchPathDirectory = .documentDirectory,

```

```

                                in      domain:      FileManager.SearchPathDomainMask
= .userDomainMask,

                                options: Options = []) -> Destination {

return { temporaryURL, response in

                                let directoryURLs = FileManager.default.urls(for: directory, in: domain)

                                let                                url                                =
directoryURLs.first?.appendingPathComponent(response.suggestedFilename!)                                ??
temporaryURL

                                return (url, options)

                                }

                                }

```

/// Default `Destination` used by Alamofire to ensure all downloads persist. This
`Destination` prepends

/// `Alamofire\_` to the automatically generated download name and moves it within the
temporary directory. Files

/// with this destination must be additionally moved if they should survive the system
reclamation of temporary

/// space.

```

static let defaultDestination: Destination = { url, _ in

    let filename = "Alamofire_\(url.lastPathComponent)"

    let destination =
        url.deletingLastPathComponent().appendingPathComponent(filename)

    return (destination, [])
}

// MARK: Downloadable

/// Type describing the source used to create the underlying
/// `URLSessionDownloadTask`.

public enum Downloadable {

    /// Download should be started from the `URLRequest` produced by the associated
    /// `URLRequestConvertible` value.

    case request(URLRequestConvertible)

    /// Download should be started from the associated resume `Data` value.

    case resumeData(Data)
}

```

```

// MARK: Mutable State

/// Type containing all mutable state for `DownloadRequest` instances.

private struct DownloadRequestMutableState {

    /// Possible resume `Data` produced when cancelling the instance.

    var resumeData: Data?

    /// `URL` to which `Data` is being downloaded.

    var fileURL: URL?

}

/// Protected mutable state specific to `DownloadRequest`.

private let protectedDownloadMutableState: Protector<DownloadRequestMutableState>
= Protector(DownloadRequestMutableState())

/// If the download is resumable and eventually cancelled, this value may be used to
resume the download using the

/// `download(resumingWith data:)` API.

```

```

///

/// - Note: For more information about `resumeData`, see [Apple's
documentation](https://developer.apple.com/documentation/foundation/urlsessiondownlo
adtask/1411634-cancel).

public var resumeData: Data? { return
protectedDownloadMutableState.directValue.resumeData }

/// If the download is successful, the `URL` where the file was downloaded.

public var fileURL: URL? { return
protectedDownloadMutableState.directValue.fileURL }

// MARK: Initial State

/// `Downloadable` value used for this instance.

public let downloadable: Downloadable

/// The `Destination` to which the downloaded file is moved.

let destination: Destination

/// Creates a `DownloadRequest` using the provided parameters.

///

```

/// - Parameters:

/// - id: `UUID` used for the `Hashable` and `Equatable` implementations.
`UUID()` by default.

/// - downloadable: `Downloadable` value used to create
`URLSessionDownloadTasks` for the instance.

/// - underlyingQueue: `DispatchQueue` on which all internal `Request` work is
performed.

/// - serializationQueue: `DispatchQueue` on which all serialization work is performed.
By default targets

/// `underlyingQueue`, but can be passed another queue from a `Session`.

/// - eventMonitor: `EventMonitor` called for event callbacks from internal `Request`
actions.

/// - interceptor: `RequestInterceptor` used throughout the request lifecycle.

/// - delegate: `RequestDelegate` that provides an interface to actions not performed
by the `Request`

/// - destination: `Destination` closure used to move the downloaded file to its final
location.

init(id: UUID = UUID(),

downloadable: Downloadable,

underlyingQueue: DispatchQueue,

serializationQueue: DispatchQueue,


```

    eventMonitor: EventMonitor?,

    interceptor: RequestInterceptor?,

    delegate: RequestDelegate,

    destination: (@escaping Destination) {

        self.downloadable = downloadable

        self.destination = destination

    }

    super.init(id: id,

        underlyingQueue: underlyingQueue,

        serializationQueue: serializationQueue,

        eventMonitor: eventMonitor,

        interceptor: interceptor,

        delegate: delegate)

    }

    override func reset() {

        super.reset()
    }

```

```

protectedDownloadMutableState.write {

    $0.resumeData = nil

    $0.fileURL = nil

}

}

/// Called when a download has finished.

///

/// - Parameters:

/// - task: `URLSessionTask` that finished the download.

/// - result: `Result` of the automatic move to `destination`.

func didFinishDownloading(using task: URLSessionTask, with result: Result<URL,
AFError>) {

    eventMonitor?.request(self, didFinishDownloadingUsing: task, with: result)

    switch result {

    case let .success(url): protectedDownloadMutableState.write { $0.fileURL = url }

```

```

        case let .failure(error): self.error = error

    }

}

/// Updates the `downloadProgress` using the provided values.

///

/// - Parameters:

///   - bytesWritten:      Total bytes written so far.

///   - totalBytesExpectedToWrite: Total bytes expected to write.

func updateDownloadProgress(bytesWritten: Int64, totalBytesExpectedToWrite: Int64)
{
    downloadProgress.totalUnitCount = totalBytesExpectedToWrite

    downloadProgress.completedUnitCount += bytesWritten

    downloadProgressHandler?.queue.async
    { self.downloadProgressHandler?.handler(self.downloadProgress) }

}

```

```
override func task(for request: URLRequest, using session: URLSession) ->
URLSessionTask {
```

```
    return session.downloadTask(with: request)
```

```
}
```

```
/// Creates a `URLSessionTask` from the provided resume data.
```

```
///
```

```
/// - Parameters:
```

```
/// - data: `Data` used to resume the download.
```

```
/// - session: `URLSession` used to create the `URLSessionTask`.
```

```
///
```

```
/// - Returns: The `URLSessionTask` created.
```

```
public func task(forResumeData data: Data, using session: URLSession) ->
URLSessionTask {
```

```
    return session.downloadTask(withResumeData: data)
```

```
}
```

```
/// Cancels the instance. Once cancelled, a `DownloadRequest` can no longer be resumed
or suspended.
```

///

/// - Note: This method will NOT produce resume data. If you wish to cancel and produce resume data, use

/// `cancel(producingResumeData:)` or `cancel(byProducingResumeData:)`.

///

/// - Returns: The instance.

@discardableResult

public override func cancel() -> Self {

 return cancel(producingResumeData: false)

}

/// Cancels the instance, optionally producing resume data. Once cancelled, a `DownloadRequest` can no longer be

/// resumed or suspended.

///

/// - Note: If `producingResumeData` is `true`, the `resumeData` property will be populated with any resume data, if

/// available.

///

/// - Returns: The instance.

@discardableResult

```
public func cancel(producingResumeData shouldProduceResumeData: Bool) -> Self {  
  
    return cancel(optionallyProducingResumeData: shouldProduceResumeData ? { _  
in } : nil)  
  
}
```

/// Cancels the instance while producing resume data. Once cancelled, a
`DownloadRequest` can no longer be resumed

/// or suspended.

///

/// - Note: The resume data passed to the completion handler will also be available on
the instance's `resumeData`

/// property.

///

/// - Parameter completionHandler: The completion handler that is called when the
download has been successfully

/// cancelled. It is not guaranteed to be called on a particular queue,
so you may

/// want use an appropriate queue to perform your work.

```

///

/// - Returns:           The instance.

@discardableResult

public func cancel(byProducingResumeData completionHandler: @escaping (_ data:
Data?) -> Void) -> Self {

    return cancel(optionallyProducingResumeData: completionHandler)

}

/// Internal implementation of cancellation that optionally takes a resume data handler.
If no handler is passed,

/// cancellation is performed without producing resume data.

///

/// - Parameter completionHandler: Optional resume data handler.

///

/// - Returns:           The instance.

private func cancel(optionallyProducingResumeData completionHandler: ((_
resumeData: Data?) -> Void)?) -> Self {

    protectedMutableState.write { mutableState in

        guard mutableState.state.canTransitionTo(.cancelled) else { return }

```

```
mutableState.state = .cancelled
```

```
underlyingQueue.async { self.didCancel() }
```

```
guard let task = mutableState.tasks.last as? URLSessionDownloadTask,  
task.state != .completed else {
```

```
    underlyingQueue.async { self.finish() }
```

```
    return
```

```
}
```

```
if let completionHandler = completionHandler {
```

```
    // Resume to ensure metrics are gathered.
```

```
    task.resume()
```

```
    task.cancel { resumeData in
```

```
        self.protectedDownloadMutableState.write { $0.resumeData = resumeData }
```

```
        self.underlyingQueue.async { self.didCancelTask(task) }
```

```
        completionHandler(resumeData)
```



```

    }

    } else {

        // Resume to ensure metrics are gathered.

        task.resume()

        task.cancel(byProducingResumeData: { _ in })

        self.underlyingQueue.async { self.didCancelTask(task) }

    }

}

return self

}

```

```

/// Validates the request, using the specified closure.

```

```

///

```

```

/// - Note: If validation fails, subsequent calls to response handlers will have an
associated error.

```

```

///

```

```

/// - Parameter validation: `Validation` closure to validate the response.

```

```
///
```

```
/// - Returns:      The instance.
```

```
@discardableResult
```

```
public func validate(_ validation: @escaping Validation) -> Self {
```

```
    let validator: () -> Void = { [unowned self] in
```

```
        guard self.error == nil, let response = self.response else { return }
```

```
        let result = validation(self.request, response, self.fileURL)
```

```
        if case let .failure(error) = result { self.error =  
error.asAFError(or: .responseValidationFailed(reason: .customValidationFailed(error:  
error))) }
```

```
        self.eventMonitor?.request(self,
```

```
            didValidateRequest: self.request,
```

```
            response: response,
```

```
            fileURL: self.fileURL,
```

```
            withResult: result)
```

```
}
```

```
protectedValidators.append validator)
```

```
return self
```

```
}
```

```
}
```

```
// MARK: - UploadRequest
```

```
/// `DataRequest` subclass which handles `Data` upload from memory, file, or stream using  
`URLSessionUploadTask`.
```

```
public class UploadRequest: DataRequest {
```

```
    /// Type describing the origin of the upload, whether `Data`, file, or stream.
```

```
    public enum Uploadable {
```

```
        /// Upload from the provided `Data` value.
```

```
        case data(Data)
```

```
    /// Upload from the provided file `URL`, as well as a `Bool` determining whether the
    source file should be
```

```
    /// automatically removed once uploaded.
```

```
    case file(URL, shouldRemove: Bool)
```

```
    /// Upload from the provided `InputStream`.
```

```
    case stream(InputStream)
```

```
}
```

```
// MARK: Initial State
```

```
    /// The `UploadableConvertible` value used to produce the `Uploadable` value for this
    instance.
```

```
    public let upload: UploadableConvertible
```

```
    /// `FileManager` used to perform cleanup tasks, including the removal of multipart form
    encoded payloads written
```

```
    /// to disk.
```

```
    public let fileManager: FileManager
```

```

// MARK: Mutable State

/// `Uploadable` value used by the instance.

public var uploadable: Uploadable?

/// Creates an `UploadRequest` using the provided parameters.

///

/// - Parameters:

///   - id:          `UUID` used for the `Hashable` and `Equatable` implementations.
///                `UUID()` by default.

///   - convertible: `UploadConvertible` value used to determine the type of upload
///                to be performed.

///   - underlyingQueue: `DispatchQueue` on which all internal `Request` work is
///                performed.

///   - serializationQueue: `DispatchQueue` on which all serialization work is performed.
///                By default targets

///                `underlyingQueue`, but can be passed another queue from a `Session`.

///   - eventMonitor:  `EventMonitor` called for event callbacks from internal `Request`
///                actions.

///   - interceptor:    `RequestInterceptor` used throughout the request lifecycle.

```

/// - delegate: 'RequestDelegate' that provides an interface to actions not performed by the 'Request'.

init(id: UUID = UUID(),

convertible: UploadConvertible,

underlyingQueue: DispatchQueue,

serializationQueue: DispatchQueue,

eventMonitor: EventMonitor?,

interceptor: RequestInterceptor?,

fileManager: FileManager,

delegate: RequestDelegate) {

upload = convertible

self.fileManager = fileManager

super.init(id: id,

convertible: convertible,

underlyingQueue: underlyingQueue,

serializationQueue: serializationQueue,

eventMonitor: eventMonitor,

```

        interceptor: interceptor,

        delegate: delegate)

    }

    /// Called when the `Uploadable` value has been created from the `UploadConvertible`.

    ///

    /// - Parameter uploadable: The `Uploadable` that was created.

    func didCreateUploadable(_ uploadable: Uploadable) {

        self.uploadable = uploadable

        eventMonitor?.request(self, didCreateUploadable: uploadable)

    }

    /// Called when the `Uploadable` value could not be created.

    ///

    /// - Parameter error: `AFError` produced by the failure.

    func didFailToCreateUploadable(with error: AFError) {

```

```
self.error = error
```

```
eventMonitor?.request(self, didFailToCreateUploadableWithError: error)
```

```
retryOrFinish(error: error)
```

```
}
```

```
override func task(for request: URLRequest, using session: URLSession) ->  
URLSessionTask {
```

```
    guard let uploadable = uploadable else {
```

```
        fatalError("Attempting to create a URLSessionUploadTask when Uploadable value  
doesn't exist.")
```

```
    }
```

```
    switch uploadable {
```

```
    case let .data(data): return session.uploadTask(with: request, from: data)
```

```
    case let .file(url, _): return session.uploadTask(with: request, fromFile: url)
```

```
    case .stream: return session.uploadTask(withStreamedRequest: request)
```



```
}
```

```
}
```

```
override fun reset() {
```

```
    // Uploadable must be recreated on every retry.
```

```
    uploadable = nil
```

```
    super.reset()
```

```
}
```

```
/// Produces the `InputStream` from `uploadable`, if it can.
```

```
///
```

```
/// - Note: Calling this method with a non-`.stream` `Uploadable` is a logic error and will  
crash.
```

```
///
```

```
/// - Returns: The `InputStream`.
```

```
fun inputStream() -> InputStream {
```

```
    guard let uploadable = uploadable else {
```

```

        fatalError("Attempting to access the input stream but the uploadable doesn't exist.")
    }

    guard case let .stream(stream) = uploadable else {

        fatalError("Attempted to access the stream of an UploadRequest that wasn't created
with one.")
    }

    eventMonitor?.request(self, didProvideInputStream: stream)

    return stream
}

public override func cleanup() {

    defer { super.cleanup() }

    guard

        let uploadable = self.uploadable,

```

```

        case let .file(url, shouldRemove) = uploadable,

        shouldRemove

    else { return }

    try? fileManager.removeItem(at: url)

}

}

/// A type that can produce an `UploadRequest.Uploadable` value.

public protocol UploadableConvertible {

    /// Produces an `UploadRequest.Uploadable` value from the instance.

    ///

    /// - Returns: The `UploadRequest.Uploadable`.

    /// - Throws: Any `Error` produced during creation.

    func createUploadable() throws -> UploadRequest.Uploadable

}

```

```

extension UploadRequest.Uploadable: UploadableConvertible {

    public func createUploadable() throws -> UploadRequest.Uploadable {

        return self

    }

}

/// A type that can be converted to an upload, whether from an `UploadRequest.Uploadable`
or `URLRequestConvertible`.

public protocol UploadConvertible: UploadableConvertible & URLRequestConvertible {}

//

// RequestInterceptor.swift

//

// Copyright (c) 2019 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

```

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

```
import Foundation
```

```
/// A type that can inspect and optionally adapt a `URLRequest` in some manner if  
necessary.
```

```
public protocol RequestAdapter {
```

```
    /// Inspects and adapts the specified `URLRequest` in some manner and calls the  
    completion handler with the Result.
```

```
    ///
```

```
    /// - Parameters:
```

```
    /// - URLRequest: The `URLRequest` to adapt.
```

```
    /// - session: The `Session` that will execute the `URLRequest`.
```

```
    /// - completion: The completion handler that must be called when adaptation is  
    complete.
```

```
    func adapt(_ URLRequest: URLRequest, for session: Session, completion: @escaping  
    (Result<URLRequest, Error>) -> Void)
```

```
}
```

```
// MARK: -
```

```
/// Outcome of determination whether retry is necessary.
```

```
public enum RetryResult {
```

```
    /// Retry should be attempted immediately.
```

```
    case retry
```

```
    /// Retry should be attempted after the associated `TimeInterval`.
```

```
    case retryWithDelay(TimeInterval)
```

```
    /// Do not retry.
```

```
    case doNotRetry
```

```
    /// Do not retry due to the associated `Error`.
```

```
    case doNotRetryWithError(Error)
```

```
}
```

```
extension RetryResult {
```

```
    var retryRequired: Bool {
```

```
        switch self {
```

```
            case .retry, .retryWithDelay: return true
```

```
        default: return false

    }

}
```

```
var delay: TimeInterval? {

    switch self {

        case let .retryWithDelay(delay): return delay

        default: return nil

    }

}
```

```
var error: Error? {

    guard case let .doNotRetryWithError(error) = self else { return nil }

    return error

}

}
```


/// A type that determines whether a request should be retried after being executed by the specified session manager

/// and encountering an error.

public protocol RequestRetrier {

/// Determines whether the `Request` should be retried by calling the `completion` closure.

///

/// This operation is fully asynchronous. Any amount of time can be taken to determine whether the request needs

/// to be retried. The one requirement is that the completion closure is called to ensure the request is properly

/// cleaned up after.

///

/// - Parameters:

/// - request: `Request` that failed due to the provided `Error`.

/// - session: `Session` that produced the `Request`.

/// - error: `Error` encountered while executing the `Request`.

/// - completion: Completion closure to be executed when a retry decision has been determined.

```

    func retry(_ request: Request, for session: Session, dueTo error: Error, completion:
@escaping (RetryResult) -> Void)

}

```

// MARK: -

/// Type that provides both `RequestAdapter` and `RequestRetrier` functionality.

```

public protocol RequestInterceptor: RequestAdapter, RequestRetrier {}

```

```

extension RequestInterceptor {

```

```

    public func adapt(_ urlRequest: URLRequest, for session: Session, completion:
@escaping (Result<URLRequest, Error>) -> Void) {

        completion(.success(urlRequest))

    }

```

```

    public func retry(_ request: Request,

        for session: Session,

        dueTo error: Error,

```

```

        completion: @escaping (RetryResult) -> Void) {

        completion(.doNotRetry)

    }

}

/// `RequestAdapter` closure definition.

public typealias AdaptHandler = (URLRequest, Session, _ completion: @escaping
(Result<URLRequest, Error>) -> Void) -> Void

/// `RequestRetrier` closure definition.

public typealias RetryHandler = (Request, Session, Error, _ completion: @escaping
(RetryResult) -> Void) -> Void

// MARK: -

/// Closure-based `RequestAdapter`.

open class Adapter: RequestInterceptor {

    private let adaptHandler: AdaptHandler

```

```

/// Creates an instance using the provided closure.

///

/// - Parameter adaptHandler: `AdaptHandler` closure to be executed when handling
request adaptation.

public init(_ adaptHandler: @escaping AdaptHandler) {

    self.adaptHandler = adaptHandler

}

open func adapt(_ urlRequest: URLRequest, for session: Session, completion:
@escaping (Result<URLRequest, Error>) -> Void) {

    adaptHandler(urlRequest, session, completion)

}

}

// MARK: -

/// Closure-based `RequestRetrier`.

open class Retrier: RequestInterceptor {

```

```

private let retryHandler: RetryHandler

/// Creates an instance using the provided closure.

///

/// - Parameter retryHandler: `RetryHandler` closure to be executed when handling
request retry.

public init(_ retryHandler: @escaping RetryHandler) {

    self.retryHandler = retryHandler

}

open func retry(_ request: Request,

                for session: Session,

                dueTo error: Error,

                completion: @escaping (RetryResult) -> Void) {

    retryHandler(request, session, error, completion)

}

}

```

// MARK: -

/// `RequestInterceptor` which can use multiple `RequestAdapter` and `RequestRetrier` values.

open class Interceptor: RequestInterceptor {

/// All `RequestAdapter`s associated with the instance. These adapters will be run until one fails.

public let adapters: [RequestAdapter]

/// All `RequestRetrier`s associated with the instance. These retriers will be run one at a time until one triggers retry.

public let retriers: [RequestRetrier]

/// Creates an instance from `AdaptHandler` and `RetryHandler` closures.

///

/// - Parameters:

/// - adaptHandler: `AdaptHandler` closure to be used.

/// - retryHandler: `RetryHandler` closure to be used.

public init(adaptHandler: @escaping AdaptHandler, retryHandler: @escaping RetryHandler) {

```

        adapters = [Adapter(adaptHandler)]

        retriers = [Retrier(retryHandler)]

    }

    /// Creates an instance from `RequestAdapter` and `RequestRetrier` values.

    ///

    /// - Parameters:

    /// - adapter: `RequestAdapter` value to be used.

    /// - retrier: `RequestRetrier` value to be used.

    public init(adapter: RequestAdapter, retrier: RequestRetrier) {

        adapters = [adapter]

        retriers = [retrier]

    }

    /// Creates an instance from the arrays of `RequestAdapter` and `RequestRetrier` values.

    ///

    /// - Parameters:

```

```
/// - adapters: `RequestAdapter` values to be used.
```

```
/// - retriers: `RequestRetrier` values to be used.
```

```
public init(adapters: [RequestAdapter] = [], retriers: [RequestRetrier] = []) {
```

```
    self.adapters = adapters
```

```
    self.retriers = retriers
```

```
}
```

```
open func adapt(_ urlRequest: URLRequest, for session: Session, completion:  
@escaping (Result<URLRequest, Error>) -> Void) {
```

```
    adapt(urlRequest, for: session, using: adapters, completion: completion)
```

```
}
```

```
private func adapt(_ urlRequest: URLRequest,
```

```
    for session: Session,
```

```
    using adapters: [RequestAdapter],
```

```
    completion: @escaping (Result<URLRequest, Error>) -> Void) {
```

```
    var pendingAdapters = adapters
```



```
guard !pendingAdapters.isEmpty else { completion(.success(urlRequest)); return }
```

```
let adapter = pendingAdapters.removeFirst()
```

```
adapter.adapt(urlRequest, for: session) { result in
```

```
    switch result {
```

```
        case let .success(urlRequest):
```

```
            self.adapt(urlRequest, for: session, using: pendingAdapters, completion:
completion)
```

```
        case .failure:
```

```
            completion(result)
```

```
    }
```

```
}
```

```
}
```

```
open func retry(_ request: Request,
```

```
               for session: Session,
```

```
               dueTo error: Error,
```

```

        completion: @escaping (RetryResult) -> Void) {

    retry(request, for: session, dueTo: error, using: retriers, completion: completion)

}

```

```

private func retry(_ request: Request,

    for session: Session,

    dueTo error: Error,

    using retriers: [RequestRetrier],

    completion: @escaping (RetryResult) -> Void) {

    var pendingRetriers = retriers

    guard !pendingRetriers.isEmpty else { completion(.doNotRetry); return }

    let retrier = pendingRetriers.removeFirst()

    retrier.retry(request, for: session, dueTo: error) { result in

        switch result {

```

```

        case .retry, .retryWithDelay, .doNotRetryWithError:

            completion(result)

        case .doNotRetry:

            // Only continue to the next retrier if retry was not triggered and no error was
            encountered

            self.retry(request, for: session, dueTo: error, using: pendingRetriers, completion:
            completion)

        }

    }

}

//

// RequestTaskMap.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

```

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

```
// THE SOFTWARE.
```

```
//
```

```
import Foundation
```

```
/// A type that maintains a two way, one to one map of `URLSessionTask`'s to `Request`'s.
```

```
struct RequestTaskMap {
```

```
    private var tasksToRequests: [URLSessionTask: Request]
```

```
    private var requestsToTasks: [Request: URLSessionTask]
```

```
    private var taskEvents: [URLSessionTask: (completed: Bool, metricsGathered: Bool)]
```

```
    var requests: [Request] {
```

```
        return Array(tasksToRequests.values)
```

```
    }
```

```
    init(tasksToRequests: [URLSessionTask: Request] = [:],
```

```
          requestsToTasks: [Request: URLSessionTask] = [:],
```

```

        taskEvents: [URLSessionTask: (completed: Bool, metricsGathered: Bool)] = [:]) {

self.tasksToRequests = tasksToRequests

self.requestsToTasks = requestsToTasks

self.taskEvents = taskEvents

}

subscript(_ request: Request) -> URLSessionTask? {

    get { return requestsToTasks[request] }

    set {

        guard let newValue = newValue else {

            guard let task = requestsToTasks[request] else {

                fatalError("RequestTaskMap consistency error: no task corresponding to
request found.")

            }

            requestsToTasks.removeValue(forKey: request)

            tasksToRequests.removeValue(forKey: task)

            taskEvents.removeValue(forKey: task)

```

```

        return

    }

    requestsToTasks[request] = newValue

    tasksToRequests[newValue] = request

    taskEvents[newValue] = (completed: false, metricsGathered: false)

}

}

subscript(_ task: URLSessionTask) -> Request? {

    get { return tasksToRequests[task] }

    set {

        guard let newValue = newValue else {

            guard let request = tasksToRequests[task] else {

                fatalError("RequestTaskMap consistency error: no request corresponding to
task found.")

            }

```

```

        tasksToRequests.removeValue(forKey: task)

        requestsToTasks.removeValue(forKey: request)

        taskEvents.removeValue(forKey: task)

        return
    }

    tasksToRequests[task] = newValue

    requestsToTasks[newValue] = task

    taskEvents[task] = (completed: false, metricsGathered: false)
}

}

var count: Int {

    precondition(tasksToRequests.count == requestsToTasks.count,

        "RequestTaskMap.count invalid, requests.count: \(tasksToRequests.count) !=
tasks.count: \(requestsToTasks.count)")

```



```

        return tasksToRequests.count
    }

    var eventCount: Int {

        precondition(taskEvents.count == count, "RequestTaskMap.eventCount invalid,
count: \$(count) != taskEvents.count: \$(taskEvents.count)")

        return taskEvents.count
    }

    var isEmpty: Bool {

        precondition(tasksToRequests.isEmpty == requestsToTasks.isEmpty,

            "RequestTaskMap.isEmpty          invalid,          requests.isEmpty:
\$(tasksToRequests.isEmpty) != tasks.isEmpty: \$(requestsToTasks.isEmpty)")

        return tasksToRequests.isEmpty
    }

```

```

var isEmpty: Bool {

    precondition(taskEvents.isEmpty == isEmpty, "RequestTaskMap.isEmpty
invalid, isEmpty: \((isEmpty) != taskEvents.isEmpty: \((taskEvents.isEmpty)")

    return taskEvents.isEmpty

}

mutating func disassociateIfNecessaryAfterGatheringMetricsForTask(_ task:
URLSessionTask) -> Bool {

    guard let events = taskEvents[task] else {

        fatalError("RequestTaskMap consistency error: no events corresponding to task
found.")

    }

    switch (events.completed, events.metricsGathered) {

        case (_, true): fatalError("RequestTaskMap consistency error: duplicate
metricsGatheredForTask call.")

        case (false, false): taskEvents[task] = (completed: false, metricsGathered: true); return
false

```

```

        case (true, false): self[task] = nil; return true

    }

}

mutating func disassociateIfNecessaryAfterCompletingTask(_ task: URLSessionTask)
-> Bool {

    guard let events = taskEvents[task] else {

        fatalError("RequestTaskMap consistency error: no events corresponding to task
found.")

    }

    switch (events.completed, events.metricsGathered) {

        case (true, _): fatalError("RequestTaskMap consistency error: duplicate
completionReceivedForTask call.")

        #if os(watchOS) // watchOS doesn't gather metrics, so unconditionally remove the
reference and return true.

        default: self[task] = nil; return true

        #else

        case (false, false): taskEvents[task] = (completed: true, metricsGathered: false); return
false

```

```
case (false, true): self[task] = nil; return true

#endif

}

}

}

//

// Response.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//
```

```
// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//
```

```
import Foundation
```

/// Default type of `DataResponse` returned by Alamofire, with an `AFError` `Failure` type.

```
public typealias AFDataResponse<Success> = DataResponse<Success, AFError>
```

/// Default type of `DownloadResponse` returned by Alamofire, with an `AFError` `Failure` type.

```
public typealias AFDownloadResponse<Success> = DownloadResponse<Success, AFError>
```

/// Type used to store all values associated with a serialized response of a `DataRequest` or `UploadRequest`.

```
public struct DataResponse<Success, Failure: Error> {
```

```
    /// The URL request sent to the server.
```

```
    public let request: URLRequest?
```

```
    /// The server's response to the URL request.
```

```
    public let response: HTTPURLResponse?
```

```
    /// The data returned by the server.
```

```
    public let data: Data?
```

```
/// The final metrics of the response.
```

```
///
```

```
/// - Note: Due to `FB7624529`, collection of `URLSessionTaskMetrics` on watchOS is  
currently disabled.`
```

```
///
```

```
public let metrics: URLSessionTaskMetrics?
```

```
/// The time taken to serialize the response.
```

```
public let serializationDuration: TimeInterval
```

```
/// The result of response serialization.
```

```
public let result: Result<Success, Failure>
```

```
/// Returns the associated value of the result if it is a success, `nil` otherwise.
```

```
public var value: Success? { return result.success }
```

```
/// Returns the associated error value if the result if it is a failure, `nil` otherwise.
```

```
public var error: Failure? { return result.failure }
```

/// Creates a `DataResponse` instance with the specified parameters derived from the response serialization.

///

/// - Parameters:

/// - request: The `URLRequest` sent to the server.

/// - response: The `HTTPURLResponse` from the server.

/// - data: The `Data` returned by the server.

/// - metrics: The `URLSessionTaskMetrics` of the `DataRequest` or `UploadRequest`.

/// - serializationDuration: The duration taken by serialization.

/// - result: The `Result` of response serialization.

```
public init(request: URLRequest?,
            response: HTTPURLResponse?,
            data: Data?,
            metrics: URLSessionTaskMetrics?,
            serializationDuration: TimeInterval,
            result: Result<Success, Failure>) {
```



```

        self.request = request

        self.response = response

        self.data = data

        self.metrics = metrics

        self.serializationDuration = serializationDuration

        self.result = result

    }

}

```

// MARK: -

```

extension DataResponse: CustomStringConvertible, CustomDebugStringConvertible {

    /// The textual representation used when written to an output stream, which includes
    whether the result was a

    /// success or failure.

    public var description: String {

        return "\(result)"

    }
}

```

```
/// The debug textual representation used when written to an output stream, which
includes the URL request, the URL
```

```
/// response, the server data, the duration of the network and serialization actions, and
the response serialization
```

```
/// result.
```

```
public var debugDescription: String {
```

```
    let requestDescription = request.map { "\($0.httpMethod!) \($0)" } ?? "nil"
```

```
    let requestBody = request?.httpBody.map { String(decoding: $0, as: UTF8.self) } ??
"None"
```

```
    let responseDescription = response.map { response in
```

```
        let sortedHeaders = response.headers.sorted()
```

```
        return ""
```

```
        [Status Code]: \(response.statusCode)
```

```
        [Headers]:
```

```
        \(sortedHeaders)
```

```
        ""
```

```
    } ?? "nil"
```

```
let responseBody = data.map { String(decoding: $0, as: UTF8.self) } ?? "None"
```

```
let metricsDescription = metrics.map { "\($0.taskInterval.duration)s" } ?? "None"
```

```
return ""
```

```
[Request]: \(requestDescription)
```

```
[Request Body]: \n(requestBody)
```

```
[Response]: \n(responseDescription)
```

```
[Response Body]: \n(responseBody)
```

```
[Data]: \ (data?.description ?? "None")
```

```
[Network Duration]: \ (metricsDescription)
```

```
[Serialization Duration]: \ (serializationDuration)s
```

```
[Result]: \ (result)
```

```
""
```

```
}
```

```
}
```

```
// MARK: -
```

```

extension DataResponse {

    /// Evaluates the specified closure when the result of this `DataResponse` is a success,
    passing the unwrapped

    /// result value as a parameter.

    ///

    /// Use the `map` method with a closure that does not throw. For example:

    ///

    /// let possibleData: DataResponse<Data> = ...

    /// let possibleInt = possibleData.map { $0.count }

    ///

    /// - parameter transform: A closure that takes the success value of the instance's result.

    ///

    /// - returns: A `DataResponse` whose result wraps the value returned by the given
    closure. If this instance's

    /// result is a failure, returns a response wrapping the same failure.

    public func map<NewSuccess>(_ transform: (Success) -> NewSuccess) ->
    DataResponse<NewSuccess, Failure> {

        return DataResponse<NewSuccess, Failure>(request: request,

```

```
        response: response,

        data: data,

        metrics: metrics,

        serializationDuration: serializationDuration,

        result: result.map(transform))

    }
```

/// Evaluates the given closure when the result of this `DataResponse` is a success,
passing the unwrapped result

```
/// value as a parameter.
```

```
///
```

/// Use the `tryMap` method with a closure that may throw an error. For example:

```
///
```

```
/// let possibleData: DataResponse<Data> = ...
```

```
/// let possibleObject = possibleData.tryMap {
```

```
///     try JSONSerialization.jsonObject(with: $0)
```

```
/// }
```

```
///
```

/// - parameter transform: A closure that takes the success value of the instance's result.

///

/// - returns: A success or failure `DataResponse` depending on the result of the given closure. If this instance's

/// result is a failure, returns the same failure.

```
public fun tryMap<NewSuccess>(_ transform: (Success) throws -> NewSuccess) ->
DataResponse<NewSuccess, Error> {
```

```
    return DataResponse<NewSuccess, Error>(request: request,
```

```
        response: response,
```

```
        data: data,
```

```
        metrics: metrics,
```

```
        serializationDuration: serializationDuration,
```

```
        result: result.tryMap(transform))
```

```
}
```

/// Evaluates the specified closure when the `DataResponse` is a failure, passing the unwrapped error as a parameter.

///

/// Use the `mapError` function with a closure that does not throw. For example:

```

///

/// let possibleData: DataResponse<Data> = ...

/// let withMyError = possibleData.mapError { MyError.error($0) }

///

/// - Parameter transform: A closure that takes the error of the instance.

///

/// - Returns: A `DataResponse` instance containing the result of the transform.

public func mapError<NewFailure: Error>(_ transform: (Failure) -> NewFailure) ->
DataResponse<Success, NewFailure> {

    return DataResponse<Success, NewFailure>(request: request,

                                             response: response,

                                             data: data,

                                             metrics: metrics,

                                             serializationDuration: serializationDuration,

                                             result: result.mapError(transform))

}

```

/// Evaluates the specified closure when the `DataResponse` is a failure, passing the unwrapped error as a parameter.

///

/// Use the `tryMapError` function with a closure that may throw an error. For example:

///

/// let possibleData: DataResponse<Data> = ...

/// let possibleObject = possibleData.tryMapError {

/// try someFailableFunction(taking: \$0)

/// }

///

/// - Parameter transform: A throwing closure that takes the error of the instance.

///

/// - Returns: A `DataResponse` instance containing the result of the transform.

```
public func tryMapError<NewFailure: Error>(_ transform: (Failure) throws ->
NewFailure) -> DataResponse<Success, Error> {
```

```
    return DataResponse<Success, Error>(request: request,
```

```
        response: response,
```

```
        data: data,
```



```

        metrics: metrics,

        serializationDuration: serializationDuration,

        result: result.tryMapError(transform))

    }

}

// MARK: -

/// Used to store all data associated with a serialized response of a download request.

public struct DownloadResponse<Success, Failure: Error> {

    /// The URL request sent to the server.

    public let request: URLRequest?

    /// The server's response to the URL request.

    public let response: HTTPURLResponse?

    /// The final destination URL of the data returned from the server after it is moved.

```

```
public let fileURL: URL?
```

```
/// The resume data generated if the request was cancelled.
```

```
public let resumeData: Data?
```

```
/// The final metrics of the response.
```

```
///
```

```
/// - Note: Due to `FB7624529`, collection of `URLSessionTaskMetrics` on watchOS is  
currently disabled.`
```

```
///
```

```
public let metrics: URLSessionTaskMetrics?
```

```
/// The time taken to serialize the response.
```

```
public let serializationDuration: TimeInterval
```

```
/// The result of response serialization.
```

```
public let result: Result<Success, Failure>
```

/// Returns the associated value of the result if it is a success, `nil` otherwise.

```
public var value: Success? { return result.success }
```

/// Returns the associated error value if the result if it is a failure, `nil` otherwise.

```
public var error: Failure? { return result.failure }
```

/// Creates a `DownloadResponse` instance with the specified parameters derived from response serialization.

///

/// - Parameters:

/// - request: The `URLRequest` sent to the server.

/// - response: The `HTTPURLResponse` from the server.

/// - temporaryURL: The temporary destination `URL` of the data returned from the server.

/// - destinationURL: The final destination `URL` of the data returned from the server, if it was moved.

/// - resumeData: The resume `Data` generated if the request was cancelled.

/// - metrics: The `URLSessionTaskMetrics` of the `DownloadRequest`.

/// - serializationDuration: The duration taken by serialization.

/// - result: The `Result` of response serialization.

```
public init(request: URLRequest?,  
  
            response: HTTPURLResponse?,  
  
            fileURL: URL?,  
  
            resumeData: Data?,  
  
            metrics: URLSessionTaskMetrics?,  
  
            serializationDuration: TimeInterval,  
  
            result: Result<Success, Failure>) {  
  
    self.request = request  
  
    self.response = response  
  
    self.fileURL = fileURL  
  
    self.resumeData = resumeData  
  
    self.metrics = metrics  
  
    self.serializationDuration = serializationDuration  
  
    self.result = result  
  
}  
  
}
```

```
// MARK: -
```

```
extension DownloadResponse: CustomStringConvertible,  
CustomDebugStringConvertible {
```

```
    /// The textual representation used when written to an output stream, which includes  
    whether the result was a
```

```
    /// success or failure.
```

```
    public var description: String {
```

```
        return "\(result)"
```

```
    }
```

```
    /// The debug textual representation used when written to an output stream, which  
    includes the URL request, the URL
```

```
    /// response, the temporary and destination URLs, the resume data, the durations of the  
    network and serialization
```

```
    /// actions, and the response serialization result.
```

```
    public var debugDescription: String {
```

```
        let requestDescription = request.map { "($0.httpMethod!) \($0)" } ?? "nil"
```

```
let requestBody = request?.httpBody.map { String(decoding: $0, as: UTF8.self) } ??  
"None"
```

```
let responseDescription = response.map { response in
```

```
    let sortedHeaders = response.headers.sorted()
```

```
    return ""
```

```
    [Status Code]: \(response.statusCode)
```

```
    [Headers]:
```

```
        \(sortedHeaders)
```

```
    ""
```

```
    } ?? "nil"
```

```
let metricsDescription = metrics.map { "\($0.taskInterval.duration)s" } ?? "None"
```

```
let resumeDataDescription = resumeData.map { "\($0)" } ?? "None"
```

```
return ""
```

```
[Request]: \(requestDescription)
```

```
[Request Body]: \n\(requestBody)
```

```
[Response]: \n\(responseDescription)
```

[File URL]: \ (fileURL?.path ?? "nil")

[ResumeData]: \ (resumeDataDescription)

[Network Duration]: \ (metricsDescription)

[Serialization Duration]: \ (serializationDuration)s

[Result]: \ (result)

""

}

}

// MARK: -

extension DownloadResponse {

/// Evaluates the given closure when the result of this `DownloadResponse` is a success,
passing the unwrapped

/// result value as a parameter.

///

/// Use the `map` method with a closure that does not throw. For example:

///

```

/// let possibleData: DownloadResponse<Data> = ...

/// let possibleInt = possibleData.map { $0.count }

///

/// - parameter transform: A closure that takes the success value of the instance's result.

///

/// - returns: A `DownloadResponse` whose result wraps the value returned by the given
closure. If this instance's

///      result is a failure, returns a response wrapping the same failure.

public func map<NewSuccess>(<_ transform: (Success) -> NewSuccess) ->
DownloadResponse<NewSuccess, Failure> {

    return DownloadResponse<NewSuccess, Failure>(request: request,

        response: response,

        fileURL: fileURL,

        resumeData: resumeData,

        metrics: metrics,

        serializationDuration: serializationDuration,

        result: result.map(transform))

}

```


/// Evaluates the given closure when the result of this `DownloadResponse` is a success,
passing the unwrapped

/// result value as a parameter.

///

/// Use the `tryMap` method with a closure that may throw an error. For example:

///

/// let possibleData: DownloadResponse<Data> = ...

/// let possibleObject = possibleData.tryMap {

/// try JSONSerialization.jsonObject(with: \$0)

/// }

///

/// - parameter transform: A closure that takes the success value of the instance's result.

///

/// - returns: A success or failure `DownloadResponse` depending on the result of the
given closure. If this

/// instance's result is a failure, returns the same failure.

```
public func tryMap<NewSuccess>(_ transform: (Success) throws -> NewSuccess) ->
DownloadResponse<NewSuccess, Error> {
```

```

return DownloadResponse<NewSuccess, Error>(request: request,

    response: response,

    fileURL: fileURL,

    resumeData: resumeData,

    metrics: metrics,

    serializationDuration: serializationDuration,

    result: result.tryMap(transform))

}

```

/// Evaluates the specified closure when the `DownloadResponse` is a failure, passing the unwrapped error as a parameter.

///

/// Use the `mapError` function with a closure that does not throw. For example:

///

/// let possibleData: DownloadResponse<Data> = ...

/// let withMyError = possibleData.mapError { MyError.error(\$0) }

///

/// - Parameter transform: A closure that takes the error of the instance.

///

/// - Returns: A `DownloadResponse` instance containing the result of the transform.

```
public fun mapError<NewFailure: Error>(_ transform: (Failure) -> NewFailure) ->
DownloadResponse<Success, NewFailure> {

    return DownloadResponse<Success, NewFailure>(request: request,

        response: response,

        fileURL: fileURL,

        resumeData: resumeData,

        metrics: metrics,

        serializationDuration: serializationDuration,

        result: result.mapError(transform))

}
```

/// Evaluates the specified closure when the `DownloadResponse` is a failure, passing the unwrapped error as a parameter.

///

/// Use the `tryMapError` function with a closure that may throw an error. For example:

///

```

/// let possibleData: DownloadResponse<Data> = ...

/// let possibleObject = possibleData.tryMapError {

///     try someFailableFunction(taking: $0)

/// }

///

/// - Parameter transform: A throwing closure that takes the error of the instance.

///

/// - Returns: A `DownloadResponse` instance containing the result of the transform.

public func tryMapError<NewFailure: Error>(_ transform: (Failure) throws ->
NewFailure) -> DownloadResponse<Success, Error> {

    return DownloadResponse<Success, Error>(request: request,

        response: response,

        fileURL: fileURL,

        resumeData: resumeData,

        metrics: metrics,

        serializationDuration: serializationDuration,

        result: result.tryMapError(transform))

}

```

```
}

//

// ResponseSerialization.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,

// EXPRESS OR
```

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

import Foundation

// MARK: Protocols

/// The type to which all data response serializers must conform in order to serialize a
response.

public protocol DataResponseSerializerProtocol {

```

    /// The type of serialized object to be created.

    associatedtype SerializedObject

    /// Serialize the response `Data` into the provided type..

    ///

    /// - Parameters:

    ///   - request: `URLRequest` which was used to perform the request, if any.

    ///   - response: `HTTPURLResponse` received from the server, if any.

    ///   - data: `Data` returned from the server, if any.

    ///   - error: `Error` produced by Alamofire or the underlying `URLSession` during the
request.

    ///

    /// - Returns: The `SerializedObject`.

    /// - Throws: Any `Error` produced during serialization.

    func serialize(request: URLRequest?, response: HTTPURLResponse?, data: Data?,
error: Error?) throws -> SerializedObject
}

```

/// The type to which all download response serializers must conform in order to serialize a response.

```
public protocol DownloadResponseSerializerProtocol {
```

/// The type of serialized object to be created.

associatedtype SerializedObject

/// Serialize the downloaded response `Data` from disk into the provided type..

///

/// - Parameters:

/// - request: `URLRequest` which was used to perform the request, if any.

/// - response: `HTTPURLResponse` received from the server, if any.

/// - fileURL: File `URL` to which the response data was downloaded.

/// - error: `Error` produced by Alamofire or the underlying `URLSession` during the request.

///

/// - Returns: The `SerializedObject`.

/// - Throws: Any `Error` produced during serialization.

```
func serializeDownload(request: URLRequest?, response: HTTPURLResponse?,  
fileURL: URL?, error: Error?) throws -> SerializedObject
```



```
}
```

```
/// A serializer that can handle both data and download responses.
```

```
public protocol ResponseSerializer: DataResponseSerializerProtocol &  
DownloadResponseSerializerProtocol {
```

```
    /// `DataPreprocessor` used to prepare incoming `Data` for serialization.
```

```
    var dataPreprocessor: DataPreprocessor { get }
```

```
    /// `HTTPMethod`'s for which empty response bodies are considered appropriate.
```

```
    var emptyRequestMethods: Set<HTTPMethod> { get }
```

```
    /// HTTP response codes for which empty response bodies are considered appropriate.
```

```
    var emptyResponseCodes: Set<Int> { get }
```

```
}
```

```
/// Type used to preprocess `Data` before it handled by a serializer.
```

```
public protocol DataPreprocessor {
```

```
    /// Process `Data` before it's handled by a serializer.
```

```
    /// - Parameter data: The raw `Data` to process.
```

```
    func preprocess(_ data: Data) throws -> Data
```

```
}
```

```
/// `DataPreprocessor` that returns passed `Data` without any transform.
```

```
public struct PassthroughPreprocessor: DataPreprocessor {
```

```
    public init() {}
```

```
    public func preprocess(_ data: Data) throws -> Data { return data }
```

```
}
```

```
/// `DataPreprocessor` that trims Google's typical `]]}',\n` XSSI JSON header.
```

```
public struct GoogleXSSIPreprocessor: DataPreprocessor {
```

```
    public init() {}
```

```
    public func preprocess(_ data: Data) throws -> Data {
```

```
        return (data.prefix(6) == Data("]]}',\n".utf8)) ? data.dropFirst(6) : data
```

```
    }
```

```
}
```

```

extension ResponseSerializer {

    /// Default `DataPreprocessor`. `PassthroughPreprocessor` by default.

    public static var defaultDataPreprocessor: DataPreprocessor { return
PassthroughPreprocessor() }

    /// Default `HTTPMethod`'s for which empty response bodies are considered appropriate.
    `[.head]` by default.

    public static var defaultEmptyRequestMethods: Set<HTTPMethod> { return [.head] }

    /// HTTP response codes for which empty response bodies are considered appropriate.
    `[204, 205]` by default.

    public static var defaultEmptyResponseCodes: Set<Int> { return [204, 205] }


    public var dataPreprocessor: DataPreprocessor { return Self.defaultDataPreprocessor }

    public var emptyRequestMethods: Set<HTTPMethod> { return
Self.defaultEmptyRequestMethods }

    public var emptyResponseCodes: Set<Int> { return Self.defaultEmptyResponseCodes }


    /// Determines whether the `request` allows empty response bodies, if `request` exists.

    ///

```

```

/// - Parameter request: `URLRequest` to evaluate.

///

/// - Returns:      `Bool` representing the outcome of the evaluation, or `nil` if `request`
was `nil`.

public func requestAllowsEmptyResponseData(_ request: URLRequest?) -> Bool? {

    return request.flatMap { $0.httpMethod }

        .flatMap(HTTPMethod.init)

        .map { emptyRequestMethods.contains($0) }

    }

/// Determines whether the `response` allows empty response bodies, if `response` exists`.

///

/// - Parameter response: `HTTPURLResponse` to evaluate.

///

/// - Returns:      `Bool` representing the outcome of the evaluation, or `nil` if
`response` was `nil`.

public func responseAllowsEmptyResponseData(_ response: HTTPURLResponse?) ->
Bool? {

    return response.flatMap { $0.statusCode }

```

```

        .map { emptyResponseCodes.contains($0) }

    }

    /// Determines whether `request` and `response` allow empty response bodies.

    ///

    /// - Parameters:

    ///   - request: `URLRequest` to evaluate.

    ///   - response: `HTTPURLResponse` to evaluate.

    ///

    /// - Returns: `true` if `request` or `response` allow empty bodies, `false` otherwise.

    public func emptyResponseAllowed(forRequest request: URLRequest?, response:
    HTTPURLResponse?) -> Bool {

        return      (requestAllowsEmptyResponseData(request)      ==      true)      ||
        (responseAllowsEmptyResponseData(response) == true)

    }

}

```

/// By default, any serializer declared to conform to both types will get file serialization for free, as it just feeds

/// the data read from disk into the data response serializer.

```
public extension DownloadResponseSerializerProtocol where Self:
DataResponseSerializerProtocol {
```

```
    func serializeDownload(request: URLRequest?, response: HTTPURLResponse?,
fileURL: URL?, error: Error?) throws -> Self.SerializedObject {
```

```
        guard error == nil else { throw error! }
```

```
        guard let fileURL = fileURL else {
```

```
            throw AFError.responseSerializationFailed(reason: .inputFileNil)
```

```
        }
```

```
        let data: Data
```

```
        do {
```

```
            data = try Data(contentsOf: fileURL)
```

```
        } catch {
```

```
            throw AFError.responseSerializationFailed(reason: .inputFileReadFailed(at:
fileURL))
```

```
        }
```

```

do {

    return try serialize(request: request, response: response, data: data, error: error)

} catch {

    throw error

}

}

}

```

// MARK: - Default

```

extension DataRequest {

    /// Adds a handler to be called once the request has finished.

    ///

    /// - Parameters:

    /// - queue:          The queue on which the completion handler is dispatched. `.main`
    by default.

    /// - completionHandler: The code to be executed once the request has finished.

    ///

```

/// - Returns: The request.

@discardableResult

```
public func response(queue: DispatchQueue = .main, completionHandler: @escaping
(AFDataResponse<Data?>) -> Void) -> Self {
```

```
    appendResponseSerializer {
```

```
        // Start work that should be on the serialization queue.
```

```
        let result = AFResult<Data?>(value: self.data, error: self.error)
```

```
        // End work that should be on the serialization queue.
```

```
    self.underlyingQueue.async {
```

```
        let response = DataResponse(request: self.request,
```

```
                                   response: self.response,
```

```
                                   data: self.data,
```

```
                                   metrics: self.metrics,
```

```
                                   serializationDuration: 0,
```

```
                                   result: result)
```

```
        self.eventMonitor?.request(self, didParseResponse: response)
```



```

        self.responseSerializerDidComplete { queue.async
{ completionHandler(response) } }

    }

}

```

```

    return self

}

```

/// Adds a handler to be called once the request has finished.

///

/// - Parameters:

/// - queue: The queue on which the completion handler is dispatched. ``.main`` by default

/// - responseSerializer: The response serializer responsible for serializing the request, response, and data.

/// - completionHandler: The code to be executed once the request has finished.

///

/// - Returns: The request.

```

@discardableResult

public func response<Serializer: DataResponseSerializerProtocol>(queue:
DispatchQueue = .main,

                        responseSerializer: Serializer,

                        completionHandler: @escaping
(AFDataResponse<Serializer.SerializedObject>) -> Void)

-> Self {

    appendResponseSerializer {

        // Start work that should be on the serialization queue.

        let start = CFAbsoluteTimeGetCurrent()

        let result: AFResult<Serializer.SerializedObject> = Result {

            try responseSerializer.serialize(request: self.request,

                                           response: self.response,

                                           data: self.data,

                                           error: self.error)

        }.mapError { error in

            error.asAFError(or: .responseSerializationFailed(reason: .customSerializationFailed(error:
error)))

```

```
}
```

```
let end = CFAbsoluteTimeGetCurrent()
```

```
// End work that should be on the serialization queue.
```

```
self.underlyingQueue.async {
```

```
    let response = DataResponse(request: self.request,
```

```
                                response: self.response,
```

```
                                data: self.data,
```

```
                                metrics: self.metrics,
```

```
                                serializationDuration: end - start,
```

```
                                result: result)
```

```
self.eventMonitor?.request(self, didParseResponse: response)
```

```
guard let serializerError = result.failure, let delegate = self.delegate else {
```

```
    self.responseSerializerDidComplete { queue.async  
{ completionHandler(response) } }
```

```

        return

    }

    delegate.retryResult(for: self, dueTo: serializerError) { retryResult in

        var didComplete: (() -> Void)?

        defer {

            if let didComplete = didComplete {

                self.responseSerializerDidComplete { queue.async { didComplete() } }

            }

        }

        switch retryResult {

        case .doNotRetry:

            didComplete = { completionHandler(response) }

        case let .doNotRetryWithError(retryError):

```

```
        let result: AFResult<Serializer.SerializedObject>
= .failure(retryError.asAFError(orFailWith: "Received retryError was not already
AFError"))
```

```
        let response = DataResponse(request: self.request,

                                     response: self.response,

                                     data: self.data,

                                     metrics: self.metrics,

                                     serializationDuration: end - start,

                                     result: result)
```

```
        didComplete = { completionHandler(response) }
```

```
        case .retry, .retryWithDelay:
```

```
            delegate.retryRequest(self, withDelay: retryResult.delay)
```

```
        }
```

```
    }
```

```
}
```

```

    }

    return self

}

}

extension DownloadRequest {

    /// Adds a handler to be called once the request has finished.

    ///

    /// - Parameters:

    /// - queue:          The queue on which the completion handler is dispatched. `.main`
    by default.

    /// - completionHandler: The code to be executed once the request has finished.

    ///

    /// - Returns:          The request.

    @discardableResult

    public func response(queue: DispatchQueue = .main,

                        completionHandler: @escaping (AFDownloadResponse<URL?>) -> Void)

```

```

-> Self {

appendResponseSerializer {

    // Start work that should be on the serialization queue.

    let result = AFResult<URL?>(value: self.fileURL, error: self.error)

    // End work that should be on the serialization queue.


self.underlyingQueue.async {

    let response = DownloadResponse(request: self.request,

                                    response: self.response,

                                    fileURL: self.fileURL,

                                    resumeData: self.resumeData,

                                    metrics: self.metrics,

                                    serializationDuration: 0,

                                    result: result)


    self.eventMonitor?.request(self, didParseResponse: response)

```

```

        self.responseSerializerDidComplete { queue.async
{ completionHandler(response) } }

    }

}

return self

}

```

/// Adds a handler to be called once the request has finished.

///

/// - Parameters:

/// - queue: The queue on which the completion handler is dispatched. `.main` by default.

/// - responseSerializer: The response serializer responsible for serializing the request, response, and data

/// contained in the destination `'URL'`.

/// - completionHandler: The code to be executed once the request has finished.

///

/// - Returns: The request.


```

@discardableResult

public func response<Serializer: DownloadResponseSerializerProtocol>(queue:
DispatchQueue = .main,

                                responseSerializer: Serializer,

                                completionHandler: @escaping
(AFDownloadResponse<Serializer.SerializedObject>) -> Void)

-> Self {

    appendResponseSerializer {

        // Start work that should be on the serialization queue.

        let start = CFAbsoluteTimeGetCurrent()

        let result: AFResult<Serializer.SerializedObject> = Result {

            try responseSerializer.serializeDownload(request: self.request,

                                                    response: self.response,

                                                    fileURL: self.fileURL,

                                                    error: self.error)

        }.mapError { error in

            error.asAFError(or: .responseSerializationFailed(reason: .customSerializationFailed(error:
error)))

```

```

    }

    let end = CFAbsoluteTimeGetCurrent()

    // End work that should be on the serialization queue.

    self.underlyingQueue.async {

        let response = DownloadResponse(request: self.request,

                                         response: self.response,

                                         fileURL: self.fileURL,

                                         resumeData: self.resumeData,

                                         metrics: self.metrics,

                                         serializationDuration: end - start,

                                         result: result)

        self.eventMonitor?.request(self, didParseResponse: response)

        guard let serializerError = result.failure, let delegate = self.delegate else {

            self.responseSerializerDidComplete { queue.async {
                { completionHandler(response) } }
            }
        }
    }

```

```

        return

    }

    delegate.retryResult(for: self, dueTo: serializerError) { retryResult in

        var didComplete: (() -> Void)?

        defer {

            if let didComplete = didComplete {

                self.responseSerializerDidComplete { queue.async { didComplete() } }

            }

        }

        switch retryResult {

        case .doNotRetry:

            didComplete = { completionHandler(response) }

        case let .doNotRetryWithError(retryError):

```

```
        let result: AFResult<Serializer.SerializedObject>
= .failure(retryError.asAFError(orFailWith: "Received retryError was not already
AFError"))
```

```
        let response = DownloadResponse(request: self.request,

                                         response: self.response,

                                         fileURL: self.fileURL,

                                         resumeData: self.resumeData,

                                         metrics: self.metrics,

                                         serializationDuration: end - start,

                                         result: result)
```

```
        didComplete = { completionHandler(response) }
```

```
        case .retry, .retryWithDelay:
```

```
            delegate.retryRequest(self, withDelay: retryResult.delay)
```

```
        }
```

```
    }
```

```

    }

}

return self

}

}

// MARK: - Data

extension DataRequest {

    /// Adds a handler to be called once the request has finished.

    ///

    /// - Parameters:

    ///   - queue:          The queue on which the completion handler is dispatched. `.main`
    by default.

    ///   - completionHandler: The code to be executed once the request has finished.

    ///

    /// - Returns:          The request.

```

@discardableResult

```
public func responseData(queue: DispatchQueue = .main,  
  
    completionHandler: @escaping (AFDataResponse<Data>) -> Void)  
  
-> Self {  
  
    return response(queue: queue,  
  
        responseSerializer: DataResponseSerializer(),  
  
        completionHandler: completionHandler)  
  
    }  
  
}
```

/// A `ResponseSerializer` that performs minimal response checking and returns any response data as-is. By default, a

/// request returning `nil` or no data is considered an error. However, if the response is has a status code valid for

/// empty responses (`204`, `205`), then an empty `Data` value is returned.

```
public final class DataResponseSerializer: ResponseSerializer {
```

```
    public let dataPreprocessor: DataPreprocessor
```

```
    public let emptyResponseCodes: Set<Int>
```

```

public let emptyRequestMethods: Set<HTTPMethod>

/// Creates an instance using the provided values.

///

/// - Parameters:

/// - dataPreprocessor:  `DataPreprocessor` used to prepare the received `Data` for
serialization.

/// - emptyResponseCodes: The HTTP response codes for which empty responses are
allowed. `[204, 205]` by default.

/// - emptyRequestMethods: The HTTP request methods for which empty responses are
allowed. `[.head]` by default.

public init(dataPreprocessor: DataPreprocessor =
DataResponseSerializer.defaultDataPreprocessor,

            emptyResponseCodes: Set<Int> =
DataResponseSerializer.defaultEmptyResponseCodes,

            emptyRequestMethods: Set<HTTPMethod> =
DataResponseSerializer.defaultEmptyRequestMethods) {

    self.dataPreprocessor = dataPreprocessor

    self.emptyResponseCodes = emptyResponseCodes

    self.emptyRequestMethods = emptyRequestMethods

```

```
}
```

```
public func serialize(request: URLRequest?, response: HTTPURLResponse?, data:  
Data?, error: Error?) throws -> Data {
```

```
    guard error == nil else { throw error! }
```

```
    guard var data = data, !data.isEmpty else {
```

```
        guard emptyResponseAllowed(forRequest: request, response: response) else {
```

```
            throw AFError.responseSerializationFailed(reason: .inputDataNilOrZeroLength)
```

```
        }
```

```
        return Data()
```

```
    }
```

```
    data = try dataPreprocessor.preprocess(data)
```

```
    return data
```

```
}
```



```
}
```

```
extension DownloadRequest {
```

```
    /// Adds a handler to be called once the request has finished.
```

```
    ///
```

```
    /// - Parameters:
```

```
    /// - queue:          The queue on which the completion handler is dispatched. `.main`  
by default.
```

```
    /// - completionHandler: The code to be executed once the request has finished.
```

```
    ///
```

```
    /// - Returns:        The request.
```

```
@discardableResult
```

```
public func responseData(queue: DispatchQueue = .main,
```

```
                           completionHandler: @escaping (AFDownloadResponse<Data>) ->  
Void)
```

```
    -> Self {
```

```
        return response(queue: queue,
```

```
                        responseSerializer: DataResponseSerializer(),
```

```

        completionHandler: completionHandler)

    }

}

// MARK: - String

/// A `ResponseSerializer` that decodes the response data as a `String`. By default, a request
returning `nil` or no

/// data is considered an error. However, if the response is has a status code valid for empty
responses (`204`, `205`),

/// then an empty `String` is returned.

public final class StringResponseSerializer: ResponseSerializer {

    public let dataPreprocessor: DataPreprocessor

    /// Optional string encoding used to validate the response.

    public let encoding: String.Encoding?

    public let emptyResponseCodes: Set<Int>

    public let emptyRequestMethods: Set<HTTPMethod>

```

```

/// Creates an instance with the provided values.

///

/// - Parameters:

/// - dataPreprocessor: `DataPreprocessor` used to prepare the received `Data` for
serialization.

/// - encoding: A string encoding. Defaults to `nil`, in which case the encoding
will be determined

/// from the server response, falling back to the default HTTP character
set, `ISO-8859-1`.

/// - emptyResponseCodes: The HTTP response codes for which empty responses are
allowed. `[204, 205]` by default.

/// - emptyRequestMethods: The HTTP request methods for which empty responses are
allowed. `[.head]` by default.

public init(dataPreprocessor: DataPreprocessor =
StringResponseSerializer.defaultDataPreprocessor,

encoding: String.Encoding? = nil,

emptyResponseCodes: Set<Int> =
StringResponseSerializer.defaultEmptyResponseCodes,

emptyRequestMethods: Set<HTTPMethod> =
StringResponseSerializer.defaultEmptyRequestMethods) {

self.dataPreprocessor = dataPreprocessor

```

```

self.encoding = encoding

self.emptyResponseCodes = emptyResponseCodes

self.emptyRequestMethods = emptyRequestMethods

}

```

```

public func serialize(request: URLRequest?, response: HTTPURLResponse?, data:
Data?, error: Error?) throws -> String {

    guard error == nil else { throw error! }

    guard var data = data, !data.isEmpty else {

        guard emptyResponseAllowed(forRequest: request, response: response) else {

            throw AFError.responseSerializationFailed(reason: .inputDataNilOrZeroLength)

        }

        return ""

    }

```

```

data = try dataPreprocessor.preprocess(data)

```

```

var convertedEncoding = encoding

if let encodingName = response?.textEncodingName as CFString?,
convertedEncoding == nil {

    let ianaCharSet =
CFStringConvertIANACharSetNameToEncoding(encodingName)

    let nsStringEncoding =
CFStringConvertEncodingToNSStringEncoding(ianaCharSet)

    convertedEncoding = String.Encoding(rawValue: nsStringEncoding)

}

let actualEncoding = convertedEncoding ?? .isoLatin1

guard let string = String(data: data, encoding: actualEncoding) else {

    throw
NSError.responseSerializationFailed(reason: .stringSerializationFailed(encoding:
actualEncoding))

}

```

```

        return string
    }
}

extension DataRequest {

    /// Adds a handler to be called once the request has finished.

    ///

    /// - Parameters:

    /// - queue:          The queue on which the completion handler is dispatched. `.main`
    by default.

    /// - encoding:       The string encoding. Defaults to `nil`, in which case the encoding
    will be determined from

    ///                  the server response, falling back to the default HTTP character set,
    `ISO-8859-1`.

    /// - completionHandler: A closure to be executed once the request has finished.

    ///

    /// - Returns:         The request.

    @discardableResult

    public func responseString(queue: DispatchQueue = .main,

```

```

        encoding: String.Encoding? = nil,

        completionHandler: @escaping (AFDataResponse<String>) -> Void)
-> Self {

    return response(queue: queue,

        responseSerializer: StringResponseSerializer(encoding: encoding),

        completionHandler: completionHandler)

    }

}

```

```

extension DownloadRequest {

```

```

    /// Adds a handler to be called once the request has finished.

```

```

    ///

```

```

    /// - Parameters:

```

```

    /// - queue:      The queue on which the completion handler is dispatched. `.main`
by default.

```

```

    /// - encoding:   The string encoding. Defaults to `nil`, in which case the encoding
will be determined from

```

```

    ///              the server response, falling back to the default HTTP character set,
`ISO-8859-1`.

```

```

/// - completionHandler: A closure to be executed once the request has finished.

///

/// - Returns:      The request.

@discardableResult

public func responseString(queue: DispatchQueue = .main,

                           encoding: String.Encoding? = nil,

                           completionHandler: @escaping (AFDownloadResponse<String>) ->
Void)

-> Self {

    return response(queue: queue,

                    responseSerializer: StringResponseSerializer(encoding: encoding),

                    completionHandler: completionHandler)

    }

}

// MARK: - JSON

```


/// A `ResponseSerializer` that decodes the response data using `JSONSerialization`. By default, a request returning

/// `nil` or no data is considered an error. However, if the response is has a status code valid for empty responses

/// (`204`, `205`), then an `NSNull` value is returned.

```
public final class JSONResponseSerializer: ResponseSerializer {
```

```
    public let dataPreprocessor: DataPreprocessor
```

```
    public let emptyResponseCodes: Set<Int>
```

```
    public let emptyRequestMethod: Set<HTTPMethod>
```

```
    /// `JSONSerialization.ReadingOptions` used when serializing a response.
```

```
    public let options: JSONSerialization.ReadingOptions
```

```
    /// Creates an instance with the provided values.
```

```
    ///
```

```
    /// - Parameters:
```

```
    /// - dataPreprocessor: `DataPreprocessor` used to prepare the received `Data` for serialization.
```

```
    /// - emptyResponseCodes: The HTTP response codes for which empty responses are allowed. `[204, 205]` by default.
```

/// - emptyRequestMethods: The HTTP request methods for which empty responses are allowed. `[.head]` by default.

/// - options: The options to use. `.allowFragments` by default.

```
public init(dataPreprocessor: DataPreprocessor =
JSONResponseSerializer.defaultDataPreprocessor,

           emptyResponseCodes: Set<Int> =
JSONResponseSerializer.defaultEmptyResponseCodes,

           emptyRequestMethods: Set<HTTPMethod> =
JSONResponseSerializer.defaultEmptyRequestMethods,

           options: JSONSerialization.ReadingOptions = .allowFragments) {

    self.dataPreprocessor = dataPreprocessor

    self.emptyResponseCodes = emptyResponseCodes

    self.emptyRequestMethods = emptyRequestMethods

    self.options = options

}
```

```
public func serialize(request: URLRequest?, response: HTTPURLResponse?, data:
Data?, error: Error?) throws -> Any {

    guard error == nil else { throw error! }
```

```

guard var data = data, !data.isEmpty else {

    guard emptyResponseAllowed(forRequest: request, response: response) else {

        throw AFError.responseSerializationFailed(reason: .inputDataNilOrZeroLength)

    }

    return NSNull()

}

data = try dataPreprocessor.preprocess(data)

do {

    return try JSONSerialization.jsonObject(with: data, options: options)

} catch {

    throw AFError.responseSerializationFailed(reason: .jsonSerializationFailed(error:
error))

}

}

}

```

```

extension DataRequest {

    /// Adds a handler to be called once the request has finished.

    ///

    /// - Parameters:

    /// - queue:          The queue on which the completion handler is dispatched. `.main`
    by default.

    /// - options:        The JSON serialization reading options. `.allowFragments` by
    default.

    /// - completionHandler: A closure to be executed once the request has finished.

    ///

    /// - Returns:         The request.

    @discardableResult

    public func responseJSON(queue: DispatchQueue = .main,

                             options: JSONSerialization.ReadingOptions = .allowFragments,

                             completionHandler: @escaping (AFDataResponse<Any>) -> Void) ->

    Self {

        return response(queue: queue,

                        responseSerializer: JSONResponseSerializer(options: options),

```

```

        completionHandler: completionHandler)

    }

}

extension DownloadRequest {

    /// Adds a handler to be called once the request has finished.

    ///

    /// - Parameters:

    /// - queue:          The queue on which the completion handler is dispatched. `.main`
    by default.

    /// - options:        The JSON serialization reading options. `.allowFragments` by
    default.

    /// - completionHandler: A closure to be executed once the request has finished.

    ///

    /// - Returns:         The request.

    @discardableResult

    public func responseJSON(queue: DispatchQueue = .main,

        options: JSONSerialization.ReadingOptions = .allowFragments,

```

```

        completionHandler: @escaping (AFDownloadResponse<Any>) ->
Void)

-> Self {

    return response(queue: queue,

        responseSerializer: JSONResponseSerializer(options: options),

        completionHandler: completionHandler)

    }

}

```

```

// MARK: - Empty

```

```

/// Protocol representing an empty response. Use `T.emptyValue()` to get an instance.

```

```

public protocol EmptyResponse {

    /// Empty value for the conforming type.

    ///

    /// - Returns: Value of `Self` to use for empty values.

    static func emptyValue() -> Self

}

```

/// Type representing an empty response. Use `Empty.value` to get the static instance.

```
public struct Empty: Decodable {
```

```
    /// Static `Empty` instance used for all `Empty` responses.
```

```
    public static let value = Empty()
```

```
}
```

```
extension Empty: EmptyResponse {
```

```
    public static func emptyValue() -> Empty {
```

```
        return value
```

```
    }
```

```
}
```

```
// MARK: - DataDecoder Protocol
```

/// Any type which can decode `Data` into a `Decodable` type.

```
public protocol DataDecoder {
```

```

/// Decode `Data` into the provided type.

///

/// - Parameters:

/// - type: The `Type` to be decoded.

/// - data: The `Data` to be decoded.

///

/// - Returns: The decoded value of type `D`.

/// - Throws: Any error that occurs during decode.

func decode<D: Decodable>(_ type: D.Type, from data: Data) throws -> D

}

/// `JSONDecoder` automatically conforms to `DataDecoder`.

extension JSONDecoder: DataDecoder {}

// MARK: - Decodable

/// A `ResponseSerializer` that decodes the response data as a generic value using any type
that conforms to

```


/// `DataDecoder`. By default, this is an instance of `JSONDecoder`. Additionally, a request returning `nil` or no data

/// is considered an error. However, if the response is has a status code valid for empty responses (`204`, `205`), then

/// the `Empty.value` value is returned.

```
public final class DecodableResponseSerializer<T: Decodable>: ResponseSerializer {
```

```
    public let dataPreprocessor: DataPreprocessor
```

```
    /// The `DataDecoder` instance used to decode responses.
```

```
    public let decoder: DataDecoder
```

```
    public let emptyResponseCodes: Set<Int>
```

```
    public let emptyRequestMethods: Set<HTTPMethod>
```

```
    /// Creates an instance using the values provided.
```

```
    ///
```

```
    /// - Parameters:
```

```
    /// - dataPreprocessor:  `DataPreprocessor` used to prepare the received `Data` for  
    serialization.
```

```
    /// - decoder:          The `DataDecoder`. `JSONDecoder()` by default.
```

/// - emptyResponseCodes: The HTTP response codes for which empty responses are allowed. `[204, 205]` by default.

/// - emptyRequestMethods: The HTTP request methods for which empty responses are allowed. `[.head]` by default.

```
public init(dataPreprocessor: DataPreprocessor =
DecodableResponseSerializer.defaultDataPreprocessor,

        decoder: DataDecoder = JSONDecoder(),

        emptyResponseCodes: Set<Int> =
DecodableResponseSerializer.defaultEmptyResponseCodes,

        emptyRequestMethods: Set<HTTPMethod> =
DecodableResponseSerializer.defaultEmptyRequestMethods) {

    self.dataPreprocessor = dataPreprocessor

    self.decoder = decoder

    self.emptyResponseCodes = emptyResponseCodes

    self.emptyRequestMethods = emptyRequestMethods

}
```

```
public func serialize(request: URLRequest?, response: HTTPURLResponse?, data:
Data?, error: Error?) throws -> T {

    guard error == nil else { throw error! }
```

```

guard var data = data, !data.isEmpty else {

    guard emptyResponseAllowed(forRequest: request, response: response) else {

        throw AFError.responseSerializationFailed(reason: .inputDataNilOrZeroLength)

    }

    guard let emptyResponseType = T.self as? EmptyResponse.Type, let emptyValue
= emptyResponseType.emptyValue() as? T else {

        throw
AFError.responseSerializationFailed(reason: .invalidEmptyResponse(type: "\(T.self)"))

    }

    return emptyValue

}

data = try dataPreprocessor.preprocess(data)

do {

```

```

        return try decoder.decode(T.self, from: data)

    } catch {

        throw AFError.responseSerializationFailed(reason: .decodingFailed(error: error))

    }

}

}

```

```

extension DataRequest {

    /// Adds a handler to be called once the request has finished.

    ///

    /// - Parameters:

    ///   - type:          `Decodable` type to decode from response data.

    ///   - queue:         The queue on which the completion handler is dispatched. `.main`
    by default.

    ///   - decoder:       `DataDecoder` to use to decode the response. `JSONDecoder()` by
    default.

    ///   - completionHandler: A closure to be executed once the request has finished.

    ///

```

/// - Returns: The request.

@discardableResult

```
public func responseDecodable<T: Decodable>(of type: T.Type = T.self,

                                             queue: DispatchQueue = .main,

                                             decoder: DataDecoder = JSONDecoder(),

                                             completionHandler: @escaping (AFDataResponse<T>) ->
Void) -> Self {

    return response(queue: queue,

                    responseSerializer: DecodableResponseSerializer(decoder: decoder),

                    completionHandler: completionHandler)

}
```

```
extension DownloadRequest {
```

```
    /// Adds a handler to be called once the request has finished.
```

```
    ///
```

```
    /// - Parameters:
```

```
    /// - type:           `Decodable` type to decode from response data.
```

/// - queue: The queue on which the completion handler is dispatched. `.main`
by default.

/// - decoder: `DataDecoder` to use to decode the response. `JSONDecoder()` by
default.

/// - completionHandler: A closure to be executed once the request has finished.

///

/// - Returns: The request.

@discardableResult

public func responseDecodable<T: Decodable>(of type: T.Type = T.self,

 queue: DispatchQueue = .main,

 decoder: DataDecoder = JSONDecoder(),

 completionHandler: @escaping
(AFDownloadResponse<T>) -> Void) -> Self {

 return response(queue: queue,

 responseSerializer: DecodableResponseSerializer(decoder: decoder),

 completionHandler: completionHandler)

 }

}

//

```
// Result+Alamofire.swift

//

// Copyright (c) 2019 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,

// EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF

// MERCHANTABILITY,
```

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

import Foundation

/// Default type of `Result` returned by Alamofire, with an `AFError` `Failure` type.

public typealias AFResult<Success> = Result<Success, AFError>

// MARK: - Internal APIs

extension Result {


```
/// Returns the associated value if the result is a success, `nil` otherwise.
```

```
var success: Success? {  
  
    guard case let .success(value) = self else { return nil }  
  
    return value  
  
}
```

```
/// Returns the associated error value if the result is a failure, `nil` otherwise.
```

```
var failure: Failure? {  
  
    guard case let .failure(error) = self else { return nil }  
  
    return error  
  
}
```

```
/// Initializes a `Result` from value or error. Returns `.failure` if the error is non-nil,  
`.success` otherwise.
```

```
///
```

```
/// - Parameters:
```

```
/// - value: A value.
```

```
/// - error: An `Error`.
```

```

init(value: Success, error: Failure?) {

    if let error = error {

        self = .failure(error)

    } else {

        self = .success(value)

    }

}

```

/// Evaluates the specified closure when the `Result` is a success, passing the unwrapped value as a parameter.

```
///
```

/// Use the `tryMap` method with a closure that may throw an error. For example:

```
///
```

```
/// let possibleData: Result<Data, Error> = .success(Data(...))
```

```
/// let possibleObject = possibleData.tryMap {
```

```
///     try JSONSerialization.jsonObject(with: $0)
```

```
/// }
```

```
///
```

/// - parameter transform: A closure that takes the success value of the instance.

///

/// - returns: A `Result` containing the result of the given closure. If this instance is a failure, returns the

/// same failure.

```
func tryMap<NewSuccess>(_ transform: (Success) throws -> NewSuccess) ->
Result<NewSuccess, Error> {
```

```
    switch self {
```

```
        case let .success(value):
```

```
            do {
```

```
                return try .success(transform(value))
```

```
            } catch {
```

```
                return .failure(error)
```

```
            }
```

```
        case let .failure(error):
```

```
            return .failure(error)
```

```
        }
```

```
    }
```

/// Evaluates the specified closure when the `Result` is a failure, passing the unwrapped error as a parameter.

///

/// Use the `tryMapError` function with a closure that may throw an error. For example:

///

/// let possibleData: Result<Data, Error> = .success(Data(...))

/// let possibleObject = possibleData.tryMapError {

/// try someFailableFunction(taking: \$0)

/// }

///

/// - Parameter transform: A throwing closure that takes the error of the instance.

///

/// - Returns: A `Result` instance containing the result of the transform. If this instance is a success, returns

/// the same success.

func tryMapError<NewFailure: Error>(_ transform: (Failure) throws -> NewFailure) -> Result<Success, Error> {

switch self {

```

    case let .failure(error):

        do {

            return try .failure(transform(error))

        } catch {

            return .failure(error)

        }

    case let .success(value):

        return .success(value)

    }

}

//

// RetryPolicy.swift

//

// Copyright (c) 2019 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

```

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

```
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER  
DEALINGS IN
```

```
// THE SOFTWARE.
```

```
//
```

```
import Foundation
```

```
/// A retry policy that retries requests using an exponential backoff for allowed HTTP  
methods and HTTP status codes
```

```
/// as well as certain types of networking errors.
```

```
open class RetryPolicy: RequestInterceptor {
```

```
    /// The default retry limit for retry policies.
```

```
    public static let defaultRetryLimit: UInt = 2
```

```
    /// The default exponential backoff base for retry policies (must be a minimum of 2).
```

```
    public static let defaultExponentialBackoffBase: UInt = 2
```

```
    /// The default exponential backoff scale for retry policies.
```

```

public static let defaultExponentialBackoffScale: Double = 0.5

/// The default HTTP methods to retry.

/// See [RFC 2616 - Section 9.1.2](https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html) for more information.

public static let defaultRetryableHTTPMethods: Set<HTTPMethod> = [.delete, //
[Delete](https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.7) - not always
idempotent

                                .get, //
[GET](https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.3) - generally
idempotent

                                .head, //
[HEAD](https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.4) - generally
idempotent

                                .options, //
[OPTIONS](https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.2) -
inherently idempotent

                                .put, //
[PUT](https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.6) - not always
idempotent

                                .trace //
[TRACE](https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.8) - inherently
idempotent

]

```



```

    /// The default HTTP status codes to retry.

    /// See [RFC 2616 - Section 10](https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10) for more information.

    public static let defaultRetryableHTTPStatusCodes: Set<Int> = [408, // [Request
Timeout](https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.9)

                                500, // [Internal Server
Error](https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.5.1)

                                502, // [Bad
Gateway](https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.5.3)

                                503, // [Service
Unavailable](https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.5.4)

                                504 // [Gateway
Timeout](https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.5.5)

    ]

```

```

    /// The default URL error codes to retry.

    public static let defaultRetryableURLErrorCodes: Set<URLError.Code> = [// [Security]
App Transport Security disallowed a connection because there is no secure network
connection.

    // - [Disabled] ATS settings do not change at runtime.

```

// .appTransportSecurityRequiresSecureConnection,

// [System] An app or app extension attempted to connect to a background session that is already connected to a

// process.

// - [Enabled] The other process could release the background session.

.backgroundSessionInUseByAnotherProcess,

// [System] The shared container identifier of the URL session configuration is needed but has not been set.

// - [Disabled] Cannot change at runtime.

// .backgroundSessionRequiresSharedContainer,

// [System] The app is suspended or exits while a background data task is processing.

// - [Enabled] App can be foregrounded or launched to recover.

.backgroundSessionWasDisconnected,

// [Network] The URL Loading system received bad data from the server.

// - [Enabled] Server could return valid data when retrying.

.badServerResponse,

// [Resource] A malformed URL prevented a URL request from being initiated.

// - [Disabled] URL was most likely constructed incorrectly.

// .badURL,

// [System] A connection was attempted while a phone call is active on a network that does not support

// simultaneous phone and data communication (EDGE or GPRS).

// - [Enabled] Phone call could be ended to allow request to recover.

.callIsActive,

// [Client] An asynchronous load has been canceled.

// - [Disabled] Request was cancelled by the client.

// .cancelled,

// [File System] A download task couldn't close the downloaded file on disk.

// - [Disabled] File system error is unlikely to recover with retry.

// .cannotCloseFile,

// [Network] An attempt to connect to a host failed.

// - [Enabled] Server or DNS lookup could recover during retry.

.cannotConnectToHost,

// [File System] A download task couldn't create the downloaded file on disk because of an I/O failure.

// - [Disabled] File system error is unlikely to recover with retry.

// .cannotCreateFile,

// [Data] Content data received during a connection request had an unknown content encoding.

// - [Disabled] Server is unlikely to modify the content encoding during a retry.

// .cannotDecodeContentData,

// [Data] Content data received during a connection request could not be decoded for a known content encoding.

// - [Disabled] Server is unlikely to modify the content encoding during a retry.

// .cannotDecodeRawData,

// [Network] The host name for a URL could not be resolved.

// - [Enabled] Server or DNS lookup could recover during retry.

.cannotFindHost,

// [Network] A request to load an item only from the cache could not be satisfied.

// - [Enabled] Cache could be populated during a retry.

.cannotLoadFromNetwork,

// [File System] A download task was unable to move a downloaded file on disk.

// - [Disabled] File system error is unlikely to recover with retry.

// .cannotMoveFile,

// [File System] A download task was unable to open the downloaded file on disk.

// - [Disabled] File system error is unlikely to recover with retry.

// .cannotOpenFile,

// [Data] A task could not parse a response.

// - [Disabled] Invalid response is unlikely to recover with retry.

// .cannotParseResponse,

// [File System] A download task was unable to remove a downloaded file from disk.

// - [Disabled] File system error is unlikely to recover with retry.

// .cannotRemoveFile,

// [File System] A download task was unable to write to the downloaded file on disk.

// - [Disabled] File system error is unlikely to recover with retry.

// .cannotWriteToFile,

// [Security] A client certificate was rejected.

// - [Disabled] Client certificate is unlikely to change with retry.

// .clientCertificateRejected,

// [Security] A client certificate was required to authenticate an SSL connection during a request.

// - [Disabled] Client certificate is unlikely to be provided with retry.

// .clientCertificateRequired,

// [Data] The length of the resource data exceeds the maximum allowed.

// - [Disabled] Resource will likely still exceed the length maximum on retry.

// .dataLengthExceedsMaximum,

// [System] The cellular network disallowed a connection.

// - [Enabled] WiFi connection could be established during retry.

.dataNotAllowed,

// [Network] The host address could not be found via DNS lookup.

// - [Enabled] DNS lookup could succeed during retry.

.dnsLookupFailed,

// [Data] A download task failed to decode an encoded file during the download.

// - [Enabled] Server could correct the decoding issue with retry.

.downloadDecodingFailedMidStream,

// [Data] A download task failed to decode an encoded file after downloading.

// - [Enabled] Server could correct the decoding issue with retry.

.downloadDecodingFailedToComplete,

// [File System] A file does not exist.

// - [Disabled] File system error is unlikely to recover with retry.

// .fileDoesNotExist,

// [File System] A request for an FTP file resulted in the server responding that the
file is not a plain file,

// but a directory.

// - [Disabled] FTP directory is not likely to change to a file during a retry.

// .fileIsDirectory,

// [Network] A redirect loop has been detected or the threshold for number of allowable redirects has been

// exceeded (currently 16).

// - [Disabled] The redirect loop is unlikely to be resolved within the retry window.

// .httpTooManyRedirects,

// [System] The attempted connection required activating a data context while roaming, but international roaming

// is disabled.

// - [Enabled] WiFi connection could be established during retry.

.internationalRoamingOff,

// [Connectivity] A client or server connection was severed in the middle of an in-progress load.

// - [Enabled] A network connection could be established during retry.

.networkConnectionLost,

// [File System] A resource couldn't be read because of insufficient permissions.

// - [Disabled] Permissions are unlikely to be granted during retry.

// .noPermissionsToReadFile,

// [Connectivity] A network resource was requested, but an internet connection has not been established and

// cannot be established automatically.

// - [Enabled] A network connection could be established during retry.

.notConnectedToInternet,

// [Resource] A redirect was specified by way of server response code, but the server did not accompany this

// code with a redirect URL.

// - [Disabled] The redirect URL is unlikely to be supplied during a retry.

// .redirectToNonExistentLocation,

// [Client] A body stream is needed but the client did not provide one.

// - [Disabled] The client will be unlikely to supply a body stream during retry.

// .requestBodyStreamExhausted,

// [Resource] A requested resource couldn't be retrieved.

// - [Disabled] The resource is unlikely to become available during the retry window.

// .resourceUnavailable,

// [Security] An attempt to establish a secure connection failed for reasons that can't be expressed more

// specifically.

// - [Enabled] The secure connection could be established during a retry given the lack of specificity

// provided by the error.

.secureConnectionFailed,

// [Security] A server certificate had a date which indicates it has expired, or is not yet valid.

// - [Enabled] The server certificate could become valid within the retry window.

.serverCertificateHasBadDate,

// [Security] A server certificate was not signed by any root server.

// - [Disabled] The server certificate is unlikely to change during the retry window.

// .serverCertificateHasUnknownRoot,

// [Security] A server certificate is not yet valid.

// - [Enabled] The server certificate could become valid within the retry window.

.serverCertificateNotYetValid,

// [Security] A server certificate was signed by a root server that isn't trusted.

// - [Disabled] The server certificate is unlikely to become trusted within the retry window.

// .serverCertificateUntrusted,

// [Network] An asynchronous operation timed out.

// - [Enabled] The request timed out for an unknown reason and should be retried.

.timedOut

// [System] The URL Loading System encountered an error that it can't interpret.

// - [Disabled] The error could not be interpreted and is unlikely to be recovered from during a retry.

// .unknown,

// [Resource] A properly formed URL couldn't be handled by the framework.

// - [Disabled] The URL is unlikely to change during a retry.

// .unsupportedURL,

// [Client] Authentication is required to access a resource.

// - [Disabled] The user authentication is unlikely to be provided by retrying.

// .userAuthenticationRequired,

// [Client] An asynchronous request for authentication has been canceled by the user.

// - [Disabled] The user cancelled authentication and explicitly took action to not
retry.

// .userCancelledAuthentication,

// [Resource] A server reported that a URL has a non-zero content length, but
terminated the network connection

// gracefully without sending any data.

```

    // - [Disabled] The server is unlikely to provide data during the retry window.

    // .zeroByteResource,

]

/// The total number of times the request is allowed to be retried.

public let retryLimit: UInt

/// The base of the exponential backoff policy (should always be greater than or equal to
2).

public let exponentialBackoffBase: UInt

/// The scale of the exponential backoff.

public let exponentialBackoffScale: Double

/// The HTTP methods that are allowed to be retried.

public let retryableHTTPMethods: Set<HTTPMethod>

/// The HTTP status codes that are automatically retried by the policy.

```

```

public let retryableHTTPStatusCodes: Set<Int>

/// The URL error codes that are automatically retried by the policy.

public let retryableURLErrorCodes: Set<URLError.Code>

/// Creates an `ExponentialBackoffRetryPolicy` from the specified parameters.

///

/// - Parameters:

/// - retryLimit:          The total number of times the request is allowed to be retried.
`2` by default.

/// - exponentialBackoffBase: The base of the exponential backoff policy. `2` by
default.

/// - exponentialBackoffScale: The scale of the exponential backoff. `0.5` by default.

/// - retryableHTTPMethods: The HTTP methods that are allowed to be retried.

///                          `RetryPolicy.defaultRetryableHTTPMethods` by default.

/// - retryableHTTPStatusCodes: The HTTP status codes that are automatically retried
by the policy.

///                          `RetryPolicy.defaultRetryableHTTPStatusCodes` by default.

```

/// - retryableURLErrorCodes: The URL error codes that are automatically retried by the policy.

/// `RetryPolicy.defaultRetryableURLErrorCodes` by default.

```
public init(retryLimit: UInt = RetryPolicy.defaultRetryLimit,

           exponentialBackoffBase: UInt = RetryPolicy.defaultExponentialBackoffBase,

           exponentialBackoffScale: Double =
RetryPolicy.defaultExponentialBackoffScale,

           retryableHTTPMethods: Set<HTTPMethod> =
RetryPolicy.defaultRetryableHTTPMethods,

           retryableHTTPStatusCodes: Set<Int> =
RetryPolicy.defaultRetryableHTTPStatusCodes,

           retryableURLErrorCodes: Set<URLError.Code> =
RetryPolicy.defaultRetryableURLErrorCodes) {

    precondition(exponentialBackoffBase >= 2, "The `exponentialBackoffBase` must be
a minimum of 2.")

    self.retryLimit = retryLimit

    self.exponentialBackoffBase = exponentialBackoffBase

    self.exponentialBackoffScale = exponentialBackoffScale

    self.retryableHTTPMethods = retryableHTTPMethods
```



```

self.retryableHTTPStatusCodes = retryableHTTPStatusCodes

self.retryableURLErrorCodes = retryableURLErrorCodes

}

```

```

open func retry(_ request: Request,

               for session: Session,

               dueTo error: Error,

               completion: @escaping (RetryResult) -> Void) {

    if

        request.retryCount < retryLimit,

        let httpMethod = request.request?.method,

        retryableHTTPMethods.contains(httpMethod),

        shouldRetry(response: request.response, error: error) {

            let          timeDelay          =          pow(Double(exponentialBackoffBase),
Double(request.retryCount)) * exponentialBackoffScale

            completion(.retryWithDelay(timeDelay))

        } else {

            completion(.doNotRetry)

```

```

    }

}

private func shouldRetry(response: HTTPURLResponse?, error: Error) -> Bool {

    if let statusCode = response?.statusCode,
retryableHTTPStatusCodes.contains(statusCode) {

        return true

    } else {

        let errorCode = (error as? URLError)?.code

        let afErrorCode = (error.asAFError?.underlyingError as? URLError)?.code

        if let code = errorCode ?? afErrorCode, retryableURLErrorCodes.contains(code) {

            return true

        } else {

            return false

        }

    }

}

```

// MARK: -

/// A retry policy that automatically retries idempotent requests for network connection lost errors. For more

/// information about retrying network connection lost errors, please refer to Apple's

/// [technical document](https://developer.apple.com/library/content/qa/qa1941/_index.html).

open class ConnectionLostRetryPolicy: RetryPolicy {

/// Creates a `ConnectionLostRetryPolicy` instance from the specified parameters.

///

/// - Parameters:

/// - retryLimit: The total number of times the request is allowed to be retried.

/// `RetryPolicy.defaultRetryLimit` by default.

/// - exponentialBackoffBase: The base of the exponential backoff policy.

/// `RetryPolicy.defaultExponentialBackoffBase` by default.

/// - exponentialBackoffScale: The scale of the exponential backoff.

/// `RetryPolicy.defaultExponentialBackoffScale` by default.

```

    /// - retryableHTTPMethods: The idempotent http methods to retry.

    /// `RetryPolicy.defaultRetryableHTTPMethods` by default.

    public init(retryLimit: UInt = RetryPolicy.defaultRetryLimit,

                exponentialBackoffBase: UInt = RetryPolicy.defaultExponentialBackoffBase,

                exponentialBackoffScale: Double =
RetryPolicy.defaultExponentialBackoffScale,

                retryableHTTPMethods: Set<HTTPMethod> =
RetryPolicy.defaultRetryableHTTPMethods) {

        super.init(retryLimit: retryLimit,

                    exponentialBackoffBase: exponentialBackoffBase,

                    exponentialBackoffScale: exponentialBackoffScale,

                    retryableHTTPMethods: retryableHTTPMethods,

                    retryableHTTPStatusCodes: [],

                    retryableURLErrorCodes: [.networkConnectionLost])

    }

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

A view that lets the user pick a room for an accessory.

*/

import UIKit

import HomeKit

class RoomPicker: UITableViewController {

/// The rooms to choose from.

var rooms = [HMRoom]()

/// The home holding the rooms.

var home: HMHome? {

didSet {

rooms = []

```

        guard let home = home else { rooms = []; return }

        rooms = [home.roomForEntireHome()] + home.rooms

        selectedRow = roomRow

    }

}

/// The service which lives in one of the rooms.

var service: HMService? {

    didSet {

        selectedRow = roomRow

    }

}

/// The row of the selected room before any changes.

var roomRow: Int? {

    guard let room = service?.accessory?.room else { return nil }

    return rooms.firstIndex(of: room)

```

```
}
```

```
/// The row the user has selected, starting as the initial room row.
```

```
var selectedRow: Int?
```

```
/// Registers interest in home delegate callbacks.
```

```
override fun viewDidLoad() {
```

```
    super.viewDidLoad()
```

```
    HomeStore.shared.addHomeDelegate(self)
```

```
}
```

```
/// Deregisters interest in home delegate callbacks.
```

```
deinit {
```

```
    HomeStore.shared.removeHomeDelegate(self)
```

```
}
```

```
/// Calls on the home store to save changes, if any.
```

```

override func viewWillDisappear(_ animated: Bool) {

    super.viewWillDisappear(animated)

    if let row = selectedRow,

        let accessory = service?.accessory,

        let home = home,

        selectedRow != roomRow {

        HomeStore.shared.move(accessory, in: home, to: rooms[row])

    }

}

/// Asks the user to supply a name for a new room, and then tries to create the room.

@IBAction func addRoom(_ sender: UIBarButtonItem) {

    let alert = UIAlertController(title: "Add a Room",

                                message: "Add a new room to the home.",

                                preferredStyle: .alert)

```



```

alert.addTextField { $0.placeholder = "Name" }

alert.addAction(UIAlertAction(title: "Create", style: .default) { _ in

    if let name = alert.textFields?[0].text {

        self.home?.addRoom(withName: name) { room, error in

            if let error = error {

                print("Error adding room: \(error)")

            } else if let home = self.home, let room = room {

                HomeStore.shared.home(home, didAdd: room)

            }

        }

    }

})

alert.addAction(UIAlertAction(title: "Cancel", style: .cancel))

present(alert, animated: true)

}

```

// MARK: - Table view data source

```

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {

        return rooms.count

    }

```

```

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

        let cell = tableView.dequeueReusableCell(withIdentifier: "RoomCell", for: indexPath)

        // Fill in the name and check the selected row.

        cell.textLabel?.text = rooms[indexPath.row].name

        cell.accessoryType = indexPath.row == selectedRow ? .checkmark : .none

        return cell

    }

```

```

    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {

```

```

// Turn all the check marks off.

tableView.visibleCells.forEach { $0.accessoryType = .none }


// Turn exactly one back on.

let cell = tableView.cellForRow(at: indexPath)

cell?.accessoryType = .checkmark

tableView.deselectRow(at: indexPath, animated: true)


// Remember which one was selected.

selectedRow = indexPath.row

}


// Enables the deletion of any room except the Default Room, which always appears first.

override func tableView(_ tableView: UITableView, canEditRowAt indexPath:
IndexPath) -> Bool {

    return indexPath != IndexPath(row: 0, section: 0)

}

```


Abstract:

The home delegate for the room picker.

*/

```
import HomeKit
```

```
/// Handle the home delegate callbacks.
```

```
extension RoomPicker: HMHomeDelegate {
```

```
    func home(_ home: HMHome, didUpdate room: HMRoom, for accessory:
HMAccessory) {
```

```
        guard home == self.home, accessory == service?.accessory else { return }
```

```
        // Record the row of the newly selected room.
```

```
        selectedRow = roomRow
```

```
        // Reset the checkmarks
```

```
        tableView.indexPathsForVisibleRows?.forEach { indexPath in
```

```
            let cell = tableView.cellForRow(at: indexPath)
```

```

        cell?.accessoryType = indexPath.row == selectedRow ? .checkmark : .none
    }
}

```

```

func home(_ home: HMHome, didAdd room: HMRoom) {

    guard home == self.home else { return }

    // Add a row at the end.

    let indexPaths = [IndexPath(row: rooms.count, section: 0)]

    rooms.append(room)

    tableView.insertRows(at: indexPaths, with: .fade)

}

```

```

func home(_ home: HMHome, didUpdateNameFor room: HMRoom) {

    guard home == self.home else { return }

    // Rewrite all the visible room names.

```

```

tableView.indexPathsForVisibleRows?.forEach { indexPath in

    let cell = tableView.cellForRow(at: indexPath)

    cell?.textLabel?.text = rooms[indexPath.row].name

}

}

```

```

func home(_ home: HMHome, didRemove room: HMRoom) {

    guard home == self.home else { return }

    // Delete the appropriate row.

    if let row = rooms.firstIndex(of: room) {

        let indexPaths = [IndexPath(row: row, section: 0)]

        rooms.remove(at: row)

        tableView.deleteRows(at: indexPaths, with: .fade)

    }

}

}

```

/*

See LICENSE folder for this sample's licensing information.

Abstract:

The home delegate for the room picker.

*/

```
import HomeKit
```

```
/// Handle the home delegate callbacks.
```

```
extension RoomPicker: HMHomeDelegate {
```

```
    func home(_ home: HMHome, didUpdate room: HMRoom, for accessory: HMAccessory) {
```

```
        guard home == self.home, accessory == service?.accessory else { return }
```

```
        // Record the row of the newly selected room.
```

```
        selectedRow = roomRow
```



```

// Reset the checkmarks

tableView.indexPathsForVisibleRows?.forEach { indexPath in

    let cell = tableView.cellForRow(at: indexPath)

    cell?.accessoryType = indexPath.row == selectedRow ? .checkmark : .none

}

}

func home(_ home: HMHome, didAdd room: HMRoom) {

    guard home == self.home else { return }

    // Add a row at the end.

    let indexPaths = [IndexPath(row: rooms.count, section: 0)]

    rooms.append(room)

    tableView.insertRows(at: indexPaths, with: .fade)

}

func home(_ home: HMHome, didUpdateNameFor room: HMRoom) {

```

```

guard home == self.home else { return }

// Rewrite all the visible room names.

tableView.indexPathsForVisibleRows?.forEach { indexPath in

    let cell = tableView.cellForRow(at: indexPath)

    cell?.textLabel?.text = rooms[indexPath.row].name

}

}

func home(_ home: HMHome, didRemove room: HMRoom) {

    guard home == self.home else { return }

    // Delete the appropriate row.

    if let row = rooms.firstIndex(of: room) {

        let indexPaths = [IndexPath(row: row, section: 0)]

        rooms.remove(at: row)

        tableView.deleteRows(at: indexPaths, with: .fade)
    }
}

```

```
    }  
  
    }  
  
}  
  
/*
```

See LICENSE folder for this sample's licensing information.

Abstract:

A view that lets the user pick a room for an accessory.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
class RoomPicker: UITableViewController {
```

```
    /// The rooms to choose from.
```

```
    var rooms = [HMRoom]()
```

```
/// The home holding the rooms.
```

```
var home: HMHome? {  
  
    didSet {  
  
        rooms = []  
  
        guard let home = home else { rooms = []; return }  
  
        rooms = [home.roomForEntireHome()] + home.rooms  
  
        selectedRow = roomRow  
  
    }  
  
}
```

```
/// The service which lives in one of the rooms.
```

```
var service: HMService? {  
  
    didSet {  
  
        selectedRow = roomRow  
  
    }  
  
}
```

```
/// The row of the selected room before any changes.
```

```
var roomRow: Int? {
```

```
    guard let room = service?.accessory?.room else { return nil }
```

```
    return rooms.firstIndex(of: room)
```

```
}
```

```
/// The row the user has selected, starting as the initial room row.
```

```
var selectedRow: Int?
```

```
/// Registers interest in home delegate callbacks.
```

```
override func viewDidLoad() {
```

```
    super.viewDidLoad()
```

```
    HomeStore.shared.addHomeDelegate(self)
```

```
}
```

```
/// Deregisters interest in home delegate callbacks.
```

```

deinit {

    HomeStore.shared.removeHomeDelegate(self)

}

/// Calls on the home store to save changes, if any.

override func viewWillDisappear(_ animated: Bool) {

    super.viewWillDisappear(animated)

    if let row = selectedRow,

        let accessory = service?.accessory,

        let home = home,

        selectedRow != roomRow {

        HomeStore.shared.move(accessory, in: home, to: rooms[row])

    }

}

```

/// Asks the user to supply a name for a new room, and then tries to create the room.

```
@IBAction func addRoom(_ sender: UIBarButtonItem) {

    let alert = UIAlertController(title: "Add a Room",

                                message: "Add a new room to the home.",

                                preferredStyle: .alert)

    alert.addTextField { $0.placeholder = "Name" }

    alert.addAction(UIAlertAction(title: "Create", style: .default) { _ in

        if let name = alert.textFields?[0].text {

            self.home?.addRoom(withName: name) { room, error in

                if let error = error {

                    print("Error adding room: \(error)")

                } else if let home = self.home, let room = room {

                    HomeStore.shared.home(home, didAdd: room)

                }

            }

        }

    })

}
```

```

        alert.addAction(UIAlertAction(title: "Cancel", style: .cancel))

        present(alert, animated: true)

    }

```

// MARK: - Table view data source

```

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {

        return rooms.count

    }

```

```

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

        let cell = tableView.dequeueReusableCell(withIdentifier: "RoomCell", for: indexPath)

        // Fill in the name and check the selected row.

        cell.textLabel?.text = rooms[indexPath.row].name

        cell.accessoryType = indexPath.row == selectedRow ? .checkmark : .none
    }

```



```
    return cell  
  
}
```

```
    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:  
IndexPath) {  
  
        // Turn all the check marks off.  
  
        tableView.visibleCells.forEach { $0.accessoryType = .none }  
  
        // Turn exactly one back on.  
  
        let cell = tableView.cellForRow(at: indexPath)  
  
        cell?.accessoryType = .checkmark  
  
        tableView.deselectRow(at: indexPath, animated: true)  
  
        // Remember which one was selected.  
  
        selectedRow = indexPath.row  
  
    }
```

```

// Enables the deletion of any room except the Default Room, which always appears first.

override func tableView(_ tableView: UITableView, canEditRowAt indexPath:
IndexPath) -> Bool {

    return indexPath != IndexPath(row: 0, section: 0)

}

// Deletes a room.

override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {

    if editingStyle == .delete, let home = home {

        let room = rooms[indexPath.row]

        home.removeRoom(room) { error in

            if let error = error {

                print(error.localizedDescription)

            } else {

                HomeStore.shared.home(home, didRemove: room)

            }

        }

    }

}

```

```
    }  
  
    }  
  
}  
  
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `RoomViewController` lists the accessory within a room.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
/// A view controller that lists the accessories within a room.
```

```
class RoomViewController: HMCatalogViewController, HMAccessoryDelegate {
```

```
// MARK: Types
```

```
struct Identifiers {
```

```
    static let accessoryCell = "AccessoryCell"
```

```
    static let unreachableAccessoryCell = "UnreachableAccessoryCell"
```

```
    static let modifyAccessorySegue = "Modify Accessory"
```

```
}
```

```
// MARK: Properties
```

```
var room: HMRoom! {
```

```
    didSet {
```

```
        navigationItem.title = room.name
```

```
    }
```

```
}
```

```
var accessories = [HMAccessory]()
```

```
// MARK: View Methods
```

```
override func viewWillAppear(_ animated: Bool) {  
  
    super.viewWillAppear(animated)  
  
    reloadData()  
  
}
```

```
// MARK: Table View Methods
```

```
/// - returns: The number of accessories within this room.
```

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section:  
Int) -> Int {  
  
    let rows = accessories.count  
  
    if rows == 0 {  
  
        let message = NSLocalizedString("No Accessories", comment: "No Accessories")  
  
        setBackgroundMessage(message)  
  
    }
```

```

else {

    setBackgroundMessage(nil)

}

return rows

}

/// - returns: `true` if the current room is not the home's roomForEntireHome; `false`
otherwise.

override func tableView(_ tableView: UITableView, canEditRowAt indexPath:
IndexPath) -> Bool {

    return room != home.roomForEntireHome()

}

/// - returns: Localized "Unassign".

override func tableView(_ tableView: UITableView,
titleForDeleteConfirmationButtonForRowAt indexPath: IndexPath) -> String? {

    return NSLocalizedString("Unassign", comment: "Unassign")

}

```

```
/// Assigns the 'deleted' room to the home's roomForEntireHome.
```

```
override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {
```

```
    if editingStyle == .delete {
```

```
        unassignAccessory(accessories[indexPath.row])
```

```
    }
```

```
}
```

```
/// - returns: A cell representing an accessory.
```

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
```

```
    let accessory = accessories[indexPath.row]
```

```
    var reuseIdentifier = Identifiers.accessoryCell
```

```
    if !accessory.isReachable {
```

```
        reuseIdentifier = Identifiers.unreachableAccessoryCell
```

```
}
```

```
let cell = tableView.dequeueReusableCell(withIdentifier: reuseIdentifier, for: indexPath)
```

```
cell.textLabel?.text = accessory.name
```

```
return cell
```

```
}
```

```
/// - returns: A localized description, "Accessories" if there are accessories to list.
```

```
override func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
```

```
    if accessories.isEmpty {
```

```
        return nil
```

```
    }
```

```
    return NSLocalizedString("Accessories", comment: "Accessories")
```



```
}
```

```
// MARK: Helper Methods
```

```
/// Updates the internal array of accessories and reloads the table view.
```

```
private func reloadData() {
```

```
    accessories = room.accessories.sortByLocalizedName()
```

```
    tableView.reloadData()
```

```
}
```

```
/// Sorts the internal list of accessories by localized name.
```

```
private func sortAccessories() {
```

```
    accessories = accessories.sortByLocalizedName()
```

```
}
```

```
/**
```

```
Registers as the delegate for the current home and
```

all accessories in our room.

*/

```
override func registerAsDelegate() {
```

```
    super.registerAsDelegate()
```

```
    for accessory in room.accessories {
```

```
        accessory.delegate = self
```

```
    }
```

```
}
```

/// Sets the accessory and home of the modifyAccessoryViewController that will be presented.

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
```

```
    super.prepare(for: segue, sender: sender)
```

```
    let indexPath = tableView.indexPath(for: sender as! UITableViewCell)!
```

```
    if segue.identifier == Identifiers.modifyAccessorySegue {
```

```
        let modifyViewController = segue.intendedDestinationViewController as!
ModifyAccessoryViewController
```

```
        modifyViewController.accessory = room.accessories[indexPath.row]
```

```

    }

}

/**

Adds an accessory into the internal list of accessories

and inserts the row into the table view.

- parameter accessory: The `HMAccessory` to add.

*/

private func didAssignAccessory(_ accessory: HMAccessory) {

    accessories.append(accessory)

    sortAccessories()

    if let newAccessoryIndex = accessories.firstIndex(of: accessory) {

        let newAccessoryIndexPath = IndexPath(row: newAccessoryIndex, section: 0)

        tableView.insertRows(at: [newAccessoryIndexPath], with: .automatic)

    }

}

```

/**

Removes an accessory from the internal list of accessory (if it exists) and deletes the row from the table view.

- parameter accessory: The `HMAccessory` to remove.

*/

```
private func didUnassignAccessory(_ accessory: HMAccessory) {  
  
    if let accessoryIndex = accessories.firstIndex(of: accessory) {  
  
        accessories.remove(at: accessoryIndex)  
  
        let accessoryIndexPath = IndexPath(row: accessoryIndex, section: 0)  
  
        tableView.deleteRows(at: [accessoryIndexPath], with: .automatic)  
  
    }  
  
}
```

/**

Assigns an accessory to the current room.

- parameter accessory: The `HMAccessory` to assign to the room.

*/

```
private func assignAccessory(_ accessory: HMAccessory) {
```

```
    didAssignAccessory(accessory)
```

```
    home.assignAccessory(accessory, to: room) { error in
```

```
        if let error = error {
```

```
            self.displayError(error)
```

```
            self.didUnassignAccessory(accessory)
```

```
        }
```

```
    }
```

```
}
```

/**

Assigns the current room back into `roomForEntireHome`.

- parameter accessory: The `HMAccessory` to reassign.

```
*/
```

```
private func unassignAccessory(_ accessory: HMAccessory) {  
  
    didUnassignAccessory(accessory)  
  
    home.assignAccessory(accessory, to: home.roomForEntireHome()) { error in  
  
        if let error = error {  
  
            self.displayError(error)  
  
            self.didAssignAccessory(accessory)  
  
        }  
  
    }  
  
}
```

```
/**
```

Finds an accessory in the internal array of accessories

and updates its row in the table view.

- parameter accessory: The `HMAccessory` to reload.

```
*/
```

```

func didModifyAccessory(_ accessory: HMAccessory){

    if let index = accessories.firstIndex(of: accessory) {

        let indexPaths = [

            IndexPath(row: index, section: 0)

        ]

        tableView.reloadRows(at: indexPaths, with: .automatic)

    }

}

```

// MARK: HMHomeDelegate Methods

/// If the accessory was added to this room, insert it.

```

func home(_ home: HMHome, didAddAccessory accessory: HMAccessory) {

    if accessory.room == room {

        accessory.delegate = self

        didAssignAccessory(accessory)
    }
}

```

```
}
```

```
}
```

```
/// Remove the accessory from our room, if required.
```

```
func home(_ home: HMHome, didRemoveAccessory accessory: HMAccessory) {
```

```
    didUnassignAccessory(accessory)
```

```
}
```

```
/**
```

```
    Handles the update.
```

```
    We act based on one of three options:
```

1. A new accessory is being added to this room.
2. An accessory is being assigned from this room to another room.
3. We can ignore this message.

```
*/
```



```
func home(_ home: HMHome, didUpdateRoom room: HMRoom, forAccessory  
accessory: HMAccessory) {
```

```
    if room == self.room {
```

```
        didAssignAccessory(accessory)
```

```
    }
```

```
    else if accessories.contains(accessory) {
```

```
        didUnassignAccessory(accessory)
```

```
    }
```

```
}
```

```
/// If our room was removed, pop back.
```

```
func home(_ home: HMHome, didRemoveRoom room: HMRoom) {
```

```
    if room == self.room {
```

```
        navigationController!.popViewController(animated: true)
```

```
    }
```

```
}
```

```
/// If our room was renamed, reload our title.
```

```
func home(_ home: HMHome, didUpdateNameFor room: HMRoom) {  
  
    if room == self.room {  
  
        navigationItem.title = room.name  
  
    }  
  
}
```

```
// MARK: HMAccessoryDelegate Methods
```

```
// Accessory updates will reload the cell for the accessory.
```

```
func accessoryDidUpdateReachability(_ accessory: HMAccessory) {  
  
    didModifyAccessory(accessory)  
  
}
```

```
func accessoryDidUpdateName(_ accessory: HMAccessory) {  
  
    didModifyAccessory(accessory)  
  
}
```

```

}

//

// ScatterChartData.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class ScatterChartData: BarLineScatterCandleBubbleChartData

{

```

```
public override init()
```

```
{
```

```
    super.init()
```

```
}
```

```
public override init(dataSets: [IChartDataSet]?)
```

```
{
```

```
    super.init(dataSets: dataSets)
```

```
}
```

```
/// - Returns: The maximum shape-size across all DataSets.
```

```
@objc open func getGreatestShapeSize() -> CGFloat
```

```
{
```

```
    return (_dataSets as? [IScatterChartDataSet])?
```

```
        .max { $0.scatterShapeSize < $1.scatterShapeSize }?
```

```
        .scatterShapeSize ?? 0
```

```
}
```

```

    }

    //

    // ScatterChartDataProvider.swift

    // Charts

    //

    // Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

    // A port of MPAndroidChart for iOS

    // Licensed under Apache License 2.0

    //

    // https://github.com/danielgindi/Charts

    //

import Foundation

import CoreGraphics

@objc

public protocol ScatterChartDataProvider:
    BarLineScatterCandleBubbleChartDataProvider

```

```

{

    var scatterData: ScatterChartData? { get }

}

// ScatterChartDataSet.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class ScatterChartDataSet: LineScatterCandleRadarChartDataSet,
IScatterChartDataSet

```

```

{

@objc(ScatterShape)

public enum Shape: Int

{

    case square

    case circle

    case triangle

    case cross

    case x

    case chevronUp

    case chevronDown

}

/// The size the scatter shape will have

open var scatterShapeSize = CGFloat(10.0)

```

```

/// The radius of the hole in the shape (applies to Square, Circle and Triangle)

/// **default**: 0.0

open var scatterShapeHoleRadius: CGFloat = 0.0


/// Color for the hole in the shape. Setting to `nil` will behave as transparent.

/// **default**: nil

open var scatterShapeHoleColor: NSUIColor? = nil


/// Sets the ScatterShape this DataSet should be drawn with.

/// This will search for an available IShapeRenderer and set this renderer for the DataSet

@objc open func setScatterShape(_ shape: Shape)

{

    self.shapeRenderer = ScatterChartDataSet.renderer(forShape: shape)

}


/// The IShapeRenderer responsible for rendering this DataSet.

/// This can also be used to set a custom IShapeRenderer aside from the default ones.

```



```

/// **default**: `SquareShapeRenderer`

open var shapeRenderer: IShapeRenderer? = SquareShapeRenderer()

@objc open class func renderer(forShape shape: Shape) -> IShapeRenderer
{
    switch shape
    {
        case .square: return SquareShapeRenderer()

        case .circle: return CircleShapeRenderer()

        case .triangle: return TriangleShapeRenderer()

        case .cross: return CrossShapeRenderer()

        case .x: return XShapeRenderer()

        case .chevronUp: return ChevronUpShapeRenderer()

        case .chevronDown: return ChevronDownShapeRenderer()

    }
}

```

```

// MARK: NSCopying

open override func copy(with zone: NSZone? = nil) -> Any
{
    let copy = super.copy(with: zone) as! ScatterChartDataSet

    copy.scatterShapeSize = scatterShapeSize

    copy.scatterShapeHoleRadius = scatterShapeHoleRadius

    copy.scatterShapeHoleColor = scatterShapeHoleColor

    copy.shapeRenderer = shapeRenderer

    return copy
}
}

//

// ScatterChartRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

```

```

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class ScatterChartRenderer: LineScatterCandleRadarRenderer

{

    @objc open weak var dataProvider: ScatterChartDataProvider?

    @objc public init(dataProvider: ScatterChartDataProvider, animator: Animator,
viewPortHandler: ViewPortHandler)

    {

        super.init(animator: animator, viewPortHandler: viewPortHandler)

```

```

        self.dataProvider = dataProvider
    }

    open override fun drawData(context: CGContext)

    {

        guard let scatterData = dataProvider?.scatterData else { return }

        // If we redraw the data, remove and repopulate accessible elements to update label
        values and frames

        accessibleChartElements.removeAll()

        if let chart = dataProvider as? ScatterChartView {

            // Make the chart header the first element in the accessible elements array

            let element = createAccessibleHeader(usingChart: chart,

                                                andData: scatterData,

                                                withDefaultDescription: "Scatter Chart")

            accessibleChartElements.append(element)

        }

```

// TODO: Due to the potential complexity of data presented in Scatter charts, a more usable way

// for VO accessibility would be to use axis based traversal rather than by dataset.

// Hence, accessibleChartElements is not populated below. (Individual renderers guard against dataSource being their respective views)

for i in 0 ..< scatterData.dataSetCount

{

guard let set = scatterData.getDataSetByIndex(i) else { continue }

if set.isVisible

{

if !(set is IScatterChartDataSet)

{

fatalError("Datasets for ScatterChartRenderer must conform to
IScatterChartDataSet")

}

drawDataSet(context: context, dataSet: set as! IScatterChartDataSet)

```
    }  
  }  
}
```

```
private var _lineSegments = [CGPoint](repeating: CGPoint(), count: 2)
```

```
@objc open func drawDataSet(context: CGContext, dataSet: IScatterChartDataSet)
```

```
{
```

```
    guard let dataProvider = dataProvider else { return }
```

```
    let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)
```

```
    let phaseY = animator.phaseY
```

```
    let entryCount = dataSet.entryCount
```

```
    var point = CGPoint()
```

```

let valueToPixelMatrix = trans.valueToPixelMatrix

if let renderer = dataSet.shapeRenderer

{

    context.saveGState()

    for j in 0 ..< Int(min(ceil(Double(entryCount) * animator.phaseX),
Double(entryCount)))

    {

        guard let e = dataSet.entryForIndex(j) else { continue }

        point.x = CGFloat(e.x)

        point.y = CGFloat(e.y * phaseY)

        point = point.applying(valueToPixelMatrix)

        if !viewPortHandler.isInBoundsRight(point.x)

        {

```

```

        break

    }

    if !viewPortHandler.isInBoundsLeft(point.x) ||

        !viewPortHandler.isInBoundsY(point.y)

    {

        continue

    }

    renderer.renderShape(context: context, dataSet: dataSet, viewPortHandler:
viewPortHandler, point: point, color: dataSet.color(atIndex: j))

}

    context.restoreGState()

}

else

{

    print("There's no IShapeRenderer specified for ScatterDataSet", terminator: "\n")

```



```
}  
  
}
```

```
open override func drawValues(context: CGContext)
```

```
{
```

```
    guard
```

```
        let dataProvider = dataProvider,
```

```
        let scatterData = dataProvider.scatterData
```

```
    else { return }
```

```
    // if values are drawn
```

```
    if isDrawingValuesAllowed(dataProvider: dataProvider)
```

```
    {
```

```
        guard let dataSets = scatterData.dataSets as? [IScatterChartDataSet] else { return }
```

```
        let phaseY = animator.phaseY
```

```

var pt = CGPoint()

for i in 0 ..< scatterData.dataSetCount
{
    let dataSet = dataSets[i]

    guard let
        formatter = dataSet.valueFormatter,
        shouldDrawValues(forDataSet: dataSet)
    else { continue }

    let valueFont = dataSet.valueFont

    let trans = dataProvider.getTransformer(forAxis: dataSet.axisDependency)

    let valueToPixelMatrix = trans.valueToPixelMatrix

    let iconsOffset = dataSet.iconsOffset

```

```
let shapeSize = dataSet.scatterShapeSize
```

```
let lineHeight = valueFont.lineHeight
```

```
_xBounds.set(chart: dataProvider, dataSet: dataSet, animator: animator)
```

```
for j in _xBounds
```

```
{
```

```
    guard let e = dataSet.entryForIndex(j) else { break }
```

```
    pt.x = CGFloat(e.x)
```

```
    pt.y = CGFloat(e.y * phaseY)
```

```
    pt = pt.applying(valueToPixelMatrix)
```

```
    if (!viewPortHandler.isInBoundsRight(pt.x))
```

```
    {
```

```
        break
```

```
    }
```

```
// make sure the lines don't do shitty things outside bounds
```

```
if (!viewPortHandler.isInBoundsLeft(pt.x)
```

```
    || !viewPortHandler.isInBoundsY(pt.y))
```

```
{
```

```
    continue
```

```
}
```

```
let text = formatter.stringForValue(
```

```
    e.y,
```

```
    entry: e,
```

```
    dataSetIndex: i,
```

```
    viewPortHandler: viewPortHandler)
```

```
if dataSet.isDrawValuesEnabled
```

```
{
```

```
    ChartUtils.drawText(
```

```

        context: context,

        text: text,

        point: CGPoint(

            x: pt.x,

            y: pt.y - shapeSize - lineHeight),

        align: .center,

        attributes: [NSAttributedString.Key.font: valueFont,
NSAttributedString.Key.foregroundColor: dataSet.valueTextColorAt(j)]

    )

}

```

```

if let icon = e.icon, dataSet.isDrawIconsEnabled

```

```

{

```

```

    ChartUtils.drawImage(context: context,

```

```

        image: icon,

```

```

        x: pt.x + iconsOffset.x,

```

```

        y: pt.y + iconsOffset.y,

```

```

        size: icon.size)

```

```
        }  
    }  
}  
}  
}
```

```
open override fun drawExtras(context: CGContext)
```

```
{  
  
}
```

```
open override fun drawHighlighted(context: CGContext, indices: [Highlight])
```

```
{
```

```
    guard
```

```
        let dataProvider = dataProvider,
```

```
        let scatterData = dataProvider.scatterData
```

```
    else { return }
```

```
context.saveGState()
```

```
for high in indices
```

```
{
```

```
    guard
```

```
        let set = scatterData.getDataSetByIndex(high.dataSetIndex) as?  
        IScatterChartDataSet,
```

```
        set.isHighlightEnabled
```

```
    else { continue }
```

```
    guard let entry = set.entryForXValue(high.x, closestToY: high.y) else { continue }
```

```
    if !isInBoundsX(entry: entry, dataSet: set) { continue }
```

```
    context.setStrokeColor(set.highlightColor.cgColor)
```

```
    context.setLineWidth(set.highlightLineWidth)
```

```
    if set.highlightLineDashLengths != nil
```

```

    {

        context.setLineDash(phase: set.highlightLineDashPhase, lengths:
set.highlightLineDashLengths!)

    }

    else

    {

        context.setLineDash(phase: 0.0, lengths: [])

    }

    let x = entry.x // get the x-position

    let y = entry.y * Double(animotor.phaseY)

    let trans = dataProvider.getTransformer(forAxis: set.axisDependency)

    let pt = trans.pixelForValues(x: x, y: y)

    high.setDraw(pt: pt)

```



```
// draw the lines

drawHighlightLines(context: context, point: pt, set: set)

}

context.restoreGState()

}

}

//

// ScatterChartView.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//
```

```
import Foundation
```

```
import CoreGraphics
```

```
/// The ScatterChart. Draws dots, triangles, squares and custom shapes into the chartview.
```

```
open class ScatterChartView: BarLineChartViewBase, ScatterChartDataProvider
```

```
{
```

```
    open override func initialize()
```

```
    {
```

```
        super.initialize()
```

```
        renderer = ScatterChartRenderer(dataProvider: self, animator: _animator,  
viewPortHandler: _viewPortHandler)
```

```
        xAxis.spaceMin = 0.5
```

```
        xAxis.spaceMax = 0.5
```

```
    }
```

```
// MARK: - ScatterChartDataProvider
```

```
open var scatterData: ScatterChartData? { return _data as? ScatterChartData }  
  
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `SegmentedControlCharacteristicCell` displays characteristics with associated values.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
/**
```

A `CharacteristicCell` subclass that contains a `UISegmentedControl`.

Used for `HMCharacteristic`s which have associated, non-numeric values, like Lock Management State.

```
*/
```

```
class SegmentedControlCharacteristicCell: CharacteristicCell {
```

```
    // MARK: Properties
```

```
    @IBOutlet weak var segmentedControl: UISegmentedControl!
```

```
    /**
```

```
        Calls the super class's didSet, and also computes a list
```

```
        of possible values.
```

```
    */
```

```
    override var characteristic: HMCharacteristic! {
```

```
        didSet {
```

```
            segmentedControl.alpha = enabled ? 1.0 : CharacteristicCell.DisabledAlpha
```

```
            segmentedControl.isUserInteractionEnabled = enabled
```

```

        if let values = characteristic.allPossibleValues as? [Int] {

            possibleValues = values

        }

    }

}

/**

The possible values for this characteristic.

When this is set, adds localized descriptions to the segmented control.

*/

var possibleValues = [Int]() {

    didSet {

        segmentedControl.removeAllSegments()

        for index in 0..

```

```
}
```

```
resetSelectedIndex()
```

```
}
```

```
}
```

```
/**
```

Responds to the segmented control's segment changing.

Sets the value and notifies its delegate.

- parameter sender: The segmented control that changed.

```
*/
```

```
@IBAction func segmentedControlDidChange(_ sender: UISegmentedControl) {
```

```
    let value = possibleValues[sender.selectedSegmentIndex]
```

```
    setValue(value as NSNumber?, notify: true)
```

```
}
```

```
/**
```

If notify is false, then this is an external change,

so update the selected segment on the segmented control.

- parameter newValue: The new value.

- parameter notify: Whether or not to notify the delegate.

```
*/
```

```
override func setValue(_ newValue: CellValueType?, notify: Bool) {
```

```
    super.setValue(newValue, notify: notify)
```

```
    if !notify {
```

```
        resetSelectedIndex()
```

```
    }
```

```
}
```

```
/// Sets the segmented control based on the set value.
```

```
func resetSelectedIndex() {
```

```
    if let intValue = value as? Int, let index = possibleValues.firstIndex(of: intValue) {
```

```

        segmentedControl.selectedSegmentIndex = index
    }

}

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `SegmentedTimeCell` has a segmented control, used for selecting the time type.

```
*/
```

```
import UIKit
```

```
/// A `UITableViewCell` subclass with a `UISegmentedControl`, used for selecting the
time type.
```

```
class SegmentedTimeCell: UITableViewCell {
```

```
    // MARK: Properties
```



```
@IBOutlet weak var segmentedControl: UISegmentedControl!

}

// ServerTrustPolicy.swift

//

// Copyright (c) 2014-2016 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//
```

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

import Foundation

/// Responsible for managing the mapping of `ServerTrustEvaluating` values to given hosts.

open class ServerTrustManager {

/// Determines whether all hosts for this `ServerTrustManager` must be evaluated. `true`
 by default.

```

public let allHostsMustBeEvaluated: Bool

/// The dictionary of policies mapped to a particular host.

public let evaluators: [String: ServerTrustEvaluating]

/// Initializes the `ServerTrustManager` instance with the given evaluators.

///

/// Since different servers and web services can have different leaf certificates,
intermediate and even root

/// certificates, it is important to have the flexibility to specify evaluation policies on a
per host basis. This

/// allows for scenarios such as using default evaluation for host1, certificate pinning for
host2, public key

/// pinning for host3 and disabling evaluation for host4.

///

/// - Parameters:

///   - allHostsMustBeEvaluated: The value determining whether all hosts for this
instance must be evaluated. `true`

///   by default.

```

```

/// - evaluators:          A dictionary of evaluators mapped to hosts.

public init(allHostsMustBeEvaluated: Bool = true, evaluators: [String:
ServerTrustEvaluating]) {

    self.allHostsMustBeEvaluated = allHostsMustBeEvaluated

    self.evaluators = evaluators

}

/// Returns the `ServerTrustEvaluating` value for the given host, if one is set.

///

/// By default, this method will return the policy that perfectly matches the given host.
Subclasses could override

/// this method and implement more complex mapping implementations such as
wildcards.

///

/// - Parameter host: The host to use when searching for a matching policy.

///

/// - Returns:          The `ServerTrustEvaluating` value for the given host if found, `nil`
otherwise.

/// - Throws:          `AFError.serverTrustEvaluationFailed` if `allHostsMustBeEvaluated`
is `true` and no matching

```

```

///          evaluators are found.

open func serverTrustEvaluator(forHost host: String) throws -> ServerTrustEvaluating?
{

    guard let evaluator = evaluators[host] else {

        if allHostsMustBeEvaluated {

            throw AFError.serverTrustEvaluationFailed(reason: .noRequiredEvaluator(host:
host))

        }

        return nil

    }

    return evaluator

}

```

/// A protocol describing the API used to evaluate server trusts.

```
public protocol ServerTrustEvaluating {
```

```

#if os(Linux)

// Implement this once Linux has API for evaluating server trusts.

#else

/// Evaluates the given `SecTrust` value for the given `host`.

///

/// - Parameters:

///   - trust: The `SecTrust` value to evaluate.

///   - host: The host for which to evaluate the `SecTrust` value.

///

/// - Returns: A `Bool` indicating whether the evaluator considers the `SecTrust` value
valid for `host`.

func evaluate(_ trust: SecTrust, forHost host: String) throws

#endif

}

// MARK: - Server Trust Evaluators

```

/// An evaluator which uses the default server trust evaluation while allowing you to control whether to validate the

/// host provided by the challenge. Applications are encouraged to always validate the host in production environments

/// to guarantee the validity of the server's certificate chain.

```
public final class DefaultTrustEvaluator: ServerTrustEvaluating {
```

```
    private let validateHost: Bool
```

```
    /// Creates a `DefaultTrustEvaluator`.
```

```
    ///
```

```
    /// - Parameter validateHost: Determines whether or not the evaluator should validate the host. `true` by default.
```

```
    public init(validateHost: Bool = true) {
```

```
        self.validateHost = validateHost
```

```
    }
```

```
    public func evaluate(_ trust: SecTrust, forHost host: String) throws {
```

```
        if validateHost {
```

```
            try trust.af.performValidation(forHost: host)
```

```

    }

    try trust.af.performDefaultValidation(forHost: host)

    }

}

/// An evaluator which Uses the default and revoked server trust evaluations allowing you
to control whether to validate

/// the host provided by the challenge as well as specify the revocation flags for testing for
revoked certificates.

/// Apple platforms did not start testing for revoked certificates automatically until iOS 10.1,
macOS 10.12 and tvOS

/// 10.1 which is demonstrated in our TLS tests. Applications are encouraged to always
validate the host in production

/// environments to guarantee the validity of the server's certificate chain.

public final class RevocationTrustEvaluator: ServerTrustEvaluating {

    /// Represents the options to be use when evaluating the status of a certificate.

    /// Only Revocation Policy Constants are valid, and can be found in [Apple's
documentation](https://developer.apple.com/documentation/security/certificate_key_and
_trust_services/policies/1563600-revocation_policy_constants).

```



```

public struct Options: OptionSet {

    /// Perform revocation checking using the CRL (Certification Revocation List) method.

    public static let crl = Options(rawValue: kSecRevocationCRLMethod)

    /// Consult only locally cached replies; do not use network access.

    public static let networkAccessDisabled = Options(rawValue:
kSecRevocationNetworkAccessDisabled)

    /// Perform revocation checking using OCSP (Online Certificate Status Protocol).

    public static let ocsp = Options(rawValue: kSecRevocationOCSPMethod)

    /// Prefer CRL revocation checking over OCSP; by default, OCSP is preferred.

    public static let preferCRL = Options(rawValue: kSecRevocationPreferCRL)

    /// Require a positive response to pass the policy. If the flag is not set, revocation
checking is done on a

    /// "best attempt" basis, where failure to reach the server is not considered fatal.

    public static let requirePositiveResponse = Options(rawValue:
kSecRevocationRequirePositiveResponse)

    /// Perform either OCSP or CRL checking. The checking is performed according to
the method(s) specified in the

    /// certificate and the value of `preferCRL`.

    public static let any = Options(rawValue: kSecRevocationUseAnyAvailableMethod)

```

```

    /// The raw value of the option.

    public let rawValue: COptionFlags

    /// Creates an `Options` value with the given `COptionFlags`.

    ///
    /// - Parameter rawValue: The `COptionFlags` value to initialize with.

    public init(rawValue: COptionFlags) {

        self.rawValue = rawValue

    }

}

private let performDefaultValidation: Bool

private let validateHost: Bool

private let options: Options

    /// Creates a `RevocationTrustEvaluator`.

```

```

///

/// - Note: Default and host validation will fail when using this evaluator with self-signed
certificates. Use

///     `PinnedCertificatesTrustEvaluator` if you need to use self-signed certificates.

///

/// - Parameters:

///   - performDefaultValidation:    Determines whether default validation should be
performed in addition to

///                                   evaluating the pinned certificates. `true` by default.

///   - validateHost:                Determines whether or not the evaluator should validate
the host, in addition

///                                   to performing the default evaluation, even if
`performDefaultValidation` is

///                                   `false`. `true` by default.

///   - options:                     The `Options` to use to check the revocation status of the
certificate. `.any`

///                                   by default.

public init(performDefaultValidation: Bool = true, validateHost: Bool = true, options:
Options = .any) {

    self.performDefaultValidation = performDefaultValidation

```

```

        self.validateHost = validateHost

        self.options = options
    }

    public func evaluate(_ trust: SecTrust, forHost host: String) throws {

        if performDefaultValidation {

            try trust.af.performDefaultValidation(forHost: host)

        }

        if validateHost {

            try trust.af.performValidation(forHost: host)

        }

        if #available(iOS 12, macOS 10.14, tvOS 12, watchOS 5, *) {

            try trust.af.evaluate(afterApplying: SecPolicy.af.revocation(options: options))

        } else {

            try trust.af.validate(policy: SecPolicy.af.revocation(options: options)) { status,
result in

```

```
AFError.serverTrustEvaluationFailed(reason: .revocationCheckFailed(output: .init(host,
trust, status, result), options: options))
```

```
}
```

```
}
```

```
}
```

```
}
```

```
/// Uses the pinned certificates to validate the server trust. The server trust is considered
valid if one of the pinned
```

```
/// certificates match one of the server certificates. By validating both the certificate chain
and host, certificate
```

```
/// pinning provides a very secure form of server trust validation mitigating most, if not all,
MITM attacks.
```

```
/// Applications are encouraged to always validate the host and require a valid certificate
chain in production
```

```
/// environments.
```

```
public final class PinnedCertificatesTrustEvaluator: ServerTrustEvaluating {
```

```
    private let certificates: [SecCertificate]
```

```
    private let acceptSelfSignedCertificates: Bool
```

```

private let performDefaultValidation: Bool

private let validateHost: Bool

/// Creates a `PinnedCertificatesTrustEvaluator`.

///

/// - Parameters:

///   - certificates:          The certificates to use to evaluate the trust. All `cer`, `crt`,
and `der`

///                           certificates in `Bundle.main` by default.

///   - acceptSelfSignedCertificates: Adds the provided certificates as anchors for the trust
evaluation, allowing

///                           self-signed certificates to pass. `false` by default. THIS SETTING
SHOULD BE

///                           FALSE IN PRODUCTION!

///   - performDefaultValidation: Determines whether default validation should be
performed in addition to

///                           evaluating the pinned certificates. `true` by default.

///   - validateHost:          Determines whether or not the evaluator should validate
the host, in addition

```

```
///                                     to performing the default evaluation, even if
`performDefaultValidation` is
```

```
///                                     `false`. `true` by default.
```

```
public init(certificates: [SecCertificate] = Bundle.main.af.certificates,
```

```
        acceptSelfSignedCertificates: Bool = false,
```

```
        performDefaultValidation: Bool = true,
```

```
        validateHost: Bool = true) {
```

```
    self.certificates = certificates
```

```
    self.acceptSelfSignedCertificates = acceptSelfSignedCertificates
```

```
    self.performDefaultValidation = performDefaultValidation
```

```
    self.validateHost = validateHost
```

```
}
```

```
public func evaluate(_ trust: SecTrust, forHost host: String) throws {
```

```
    guard !certificates.isEmpty else {
```

```
        throw AFError.serverTrustEvaluationFailed(reason: .noCertificatesFound)
```

```
    }
```

```
if acceptSelfSignedCertificates {  
  
    try trust.af.setAnchorCertificates(certificates)  
  
}
```

```
if performDefaultValidation {  
  
    try trust.af.performDefaultValidation(forHost: host)  
  
}
```

```
if validateHost {  
  
    try trust.af.performValidation(forHost: host)  
  
}
```

```
let serverCertificatesData = Set(trust.af.certificateData)
```

```
let pinnedCertificatesData = Set(certificates.af.data)
```

```
let pinnedCertificatesInServerData = !serverCertificatesData.isDisjoint(with:  
pinnedCertificatesData)
```

```
if !pinnedCertificatesInServerData {
```



```

        throw AFError.serverTrustEvaluationFailed(reason: .certificatePinningFailed(host:
host,

                                trust: trust,

                                pinnedCertificates: certificates,

                                serverCertificates:

trust.af.certificates))

    }

}

}

```

/// Uses the pinned public keys to validate the server trust. The server trust is considered valid if one of the pinned

/// public keys match one of the server certificate public keys. By validating both the certificate chain and host,

/// public key pinning provides a very secure form of server trust validation mitigating most, if not all, MITM attacks.

/// Applications are encouraged to always validate the host and require a valid certificate chain in production

/// environments.

```

public final class PublicKeysTrustEvaluator: ServerTrustEvaluating {

```

```

private let keys: [SecKey]

private let performDefaultValidation: Bool

private let validateHost: Bool


/// Creates a `PublicKeysTrustEvaluator`.

///

/// - Note: Default and host validation will fail when using this evaluator with self-signed
certificates. Use

///     `PinnedCertificatesTrustEvaluator` if you need to use self-signed certificates.

///

/// - Parameters:

///   - keys:                The `SecKey`'s to use to validate public keys. Defaults to the
public keys of all

///                         certificates included in the main bundle.

///   - performDefaultValidation: Determines whether default validation should be
performed in addition to

///                         evaluating the pinned certificates. `true` by default.

///   - validateHost:        Determines whether or not the evaluator should validate the
host, in addition to

```

```
/// performing the default evaluation, even if `performDefaultValidation`  
is `false`.
```

```
/// `true` by default.
```

```
public init(keys: [SecKey] = Bundle.main.af.publicKeys,
```

```
        performDefaultValidation: Bool = true,
```

```
        validateHost: Bool = true) {
```

```
    self.keys = keys
```

```
    self.performDefaultValidation = performDefaultValidation
```

```
    self.validateHost = validateHost
```

```
}
```

```
public func evaluate(_ trust: SecTrust, forHost host: String) throws {
```

```
    guard !keys.isEmpty else {
```

```
        throw AFError.serverTrustEvaluationFailed(reason: .noPublicKeysFound)
```

```
    }
```

```
    if performDefaultValidation {
```

```
        try trust.af.performDefaultValidation(forHost: host)
```

```
}
```

```
if validateHost {
```

```
    try trust.af.performValidation(forHost: host)
```

```
}
```

```
let pinnedKeysInServerKeys: Bool = {
```

```
    for serverPublicKey in trust.af.publicKeys {
```

```
        for pinnedPublicKey in keys {
```

```
            if serverPublicKey == pinnedPublicKey {
```

```
                return true
```

```
            }
```

```
        }
```

```
    }
```

```
    return false
```

```
}()
```

```

        if !pinnedKeysInServerKeys {

            throw AFError.serverTrustEvaluationFailed(reason: .publicKeyPinningFailed(host:
host,

                                                                    trust: trust,

                                                                    pinnedKeys: keys,

                                                                    serverKeys: trust.af.publicKeys))

        }

    }

}

```

/// Uses the provided evaluators to validate the server trust. The trust is only considered valid if all of the

/// evaluators consider it valid.

```

public final class CompositeTrustEvaluator: ServerTrustEvaluating {

```

```

    private let evaluators: [ServerTrustEvaluating]

```

/// Creates a `CompositeTrustEvaluator`.

```

    ///

```

/// - Parameter evaluators: The `ServerTrustEvaluating` values used to evaluate the server trust.

```
public init(evaluators: [ServerTrustEvaluating]) {  
  
    self.evaluators = evaluators  
  
}  
  
public func evaluate(_ trust: SecTrust, forHost host: String) throws {  
  
    try evaluators.evaluate(trust, forHost: host)  
  
}  
  
}
```

/// Disables all evaluation which in turn will always consider any server trust as valid.

///

/// ****THIS EVALUATOR SHOULD NEVER BE USED IN PRODUCTION!****

```
public final class DisabledEvaluator: ServerTrustEvaluating {  
  
    /// Creates an instance.  
  
    public init() {}  
  
}
```

```

public func evaluate(_ trust: SecTrust, forHost host: String) throws {}

}

```

// MARK: - Extensions

```

public extension Array where Element == ServerTrustEvaluating {

    #if os(Linux)

        // Add this same convenience method for Linux.

    #else

        /// Evaluates the given `SecTrust` value for the given `host`.

        ///

        /// - Parameters:

        ///   - trust: The `SecTrust` value to evaluate.

        ///   - host: The host for which to evaluate the `SecTrust` value.

        ///

        /// - Returns: Whether or not the evaluator considers the `SecTrust` value valid for `host`.

    func evaluate(_ trust: SecTrust, forHost host: String) throws {

```

```

    for evaluator in self {

        try evaluator.evaluate(trust, forHost: host)

    }

}

#endif

}

extension Bundle: AlamofireExtended {}

public extension AlamofireExtension where ExtendedType: Bundle {

    /// Returns all valid `cer`, `crt`, and `der` certificates in the bundle.

    var certificates: [SecCertificate] {

        return paths(forResourcesOfTypes: [".cer", ".CER", ".crt", ".CRT", ".der",
        ".DER"]).compactMap { path in

            guard

                let certificateData = try? Data(contentsOf: URL(fileURLWithPath: path)) as
                CFData,

                let certificate = SecCertificateCreateWithData(nil, certificateData) else { return
                nil }

```



```

        return certificate
    }
}

/// Returns all public keys for the valid certificates in the bundle.

var publicKeys: [SecKey] {

    return certificates.af.publicKeys

}

/// Returns all pathnames for the resources identified by the provided file extensions.

///

/// - Parameter types: The filename extensions locate.

///

/// - Returns:      All pathnames for the given filename extensions.

func paths(forResourcesOfTypes types: [String]) -> [String] {

    return Array(Set(types.flatMap { type.paths(forResourcesOfType: $0, inDirectory:
nil) })))

}

```

```
}
```

```
extension SecTrust: AlamofireExtended {}
```

```
public extension AlamofireExtension where ExtendedType == SecTrust {
```

```
    /// Evaluates `self` after applying the `SecPolicy` value provided.
```

```
    ///
```

```
    /// - Parameter policy: The `SecPolicy` to apply to `self` before evaluation.
```

```
    ///
```

```
    /// - Throws: Any `Error` from applying the `SecPolicy` or from evaluation.
```

```
    @available(iOS 12, macOS 10.14, tvOS 12, watchOS 5, *)
```

```
    func evaluate(afterApplying policy: SecPolicy) throws {
```

```
        try apply(policy: policy).af.evaluate()
```

```
    }
```

```
    /// Attempts to validate `self` using the `SecPolicy` provided and transforming any error  
    produced using the closure passed.
```

```
    ///
```

```
    /// - Parameters:
```

```

/// - policy:      The `SecPolicy` used to evaluate `self`.

///      - errorProducer: The closure used transform the failed `OSStatus` and
`SecTrustResultType`.

/// - Throws:      Any `Error` from applying the `policy`, or the result of `errorProducer`
if validation fails.

@available(iOS, introduced: 10, deprecated: 12, renamed: "evaluate(afterApplying:)")

@available(macOS,   introduced:   10.12,   deprecated:   10.14,   renamed:
"evaluate(afterApplying:)")

@available(tvOS, introduced: 10, deprecated: 12, renamed: "evaluate(afterApplying:)")

@available(watchOS, introduced: 3, deprecated: 5, renamed: "evaluate(afterApplying:)")

func validate(policy: SecPolicy, errorProducer: ( _ status: OSStatus, _ result:
SecTrustResultType) -> Error) throws {

    try apply(policy: policy).af.validate(errorProducer: errorProducer)

}

/// Applies a `SecPolicy` to `self`, throwing if it fails.

///

/// - Parameter policy: The `SecPolicy`.

///

```

```

/// - Returns: `self`, with the policy applied.

/// - Throws: An `AFError.serverTrustEvaluationFailed` instance with a
`.policyApplicationFailed` reason.

func apply(policy: SecPolicy) throws -> SecTrust {

    let status = SecTrustSetPolicies(type, policy)

    guard status.af.isSuccess else {

        throw
        AFError.serverTrustEvaluationFailed(reason: .policyApplicationFailed(trust: type,

                                                                           policy: policy,

                                                                           status: status))

    }

    return type

}

/// Evaluate `self`, throwing an `Error` if evaluation fails.

///

```

```

    /// - Throws: `AFError.serverTrustEvaluationFailed` with reason
    `.trustValidationFailed` and associated error from

    /// the underlying evaluation.

    @available(iOS 12, macOS 10.14, tvOS 12, watchOS 5, *)

    func evaluate() throws {

        var error: CFError?

        let evaluationSucceeded = SecTrustEvaluateWithError(type, &error)

        if !evaluationSucceeded {

            throw AFError.serverTrustEvaluationFailed(reason: .trustEvaluationFailed(error:
            error))

        }

    }

    /// Validate `self`, passing any failure values through `errorProducer`.

    ///

    /// - Parameter errorProducer: The closure used to transform the failed `OSStatus` and
    `SecTrustResultType` into an

    /// `Error`.

```

/// - Throws: The `Error` produced by the `errorProducer` closure.

@available(iOS, introduced: 10, deprecated: 12, renamed: "evaluate()")

@available(macOS, introduced: 10.12, deprecated: 10.14, renamed: "evaluate()")

@available(tvOS, introduced: 10, deprecated: 12, renamed: "evaluate()")

@available(watchOS, introduced: 3, deprecated: 5, renamed: "evaluate()")

```
func validate(errorProducer: (_ status: OSStatus, _ result: SecTrustResultType) -> Error)
throws {

    var result = SecTrustResultType.invalid

    let status = SecTrustEvaluate(type, &result)

    guard status.af.isSuccess && result.af.isSuccess else {

        throw errorProducer(status, result)

    }

}
```

/// Sets a custom certificate chain on `self`, allowing full validation of a self-signed certificate and its chain.

///

```

/// - Parameter certificates: The `SecCertificate`s to add to the chain.

/// - Throws: Any error produced when applying the new certificate chain.

func setAnchorCertificates(_ certificates: [SecCertificate]) throws {

    // Add additional anchor certificates.

    let status = SecTrustSetAnchorCertificates(type, certificates as CFArray)

    guard status.af.isSuccess else {

        throw
        NSError.serverTrustEvaluationFailed(reason: .settingAnchorCertificatesFailed(status:
        status,

                                                certificates: certificates))

    }

    // Reenable system anchor certificates.

    let systemStatus = SecTrustSetAnchorCertificatesOnly(type, true)

    guard systemStatus.af.isSuccess else {

        throw
        NSError.serverTrustEvaluationFailed(reason: .settingAnchorCertificatesFailed(status:
        systemStatus,

                                                certificates: certificates))

```

```

    }

}

/// The public keys contained in `self`.

var publicKeys: [SecKey] {

    return certificates.af.publicKeys

}

/// The `SecCertificate`s contained i `self`.

var certificates: [SecCertificate] {

    return (0..

```



```

        return certificates.af.data
    }

    /// Validates `self` after applying `SecPolicy.af.default`. This evaluation does not
    validate the hostname.

    ///

    /// - Parameter host: The hostname, used only in the error output if validation fails.

    /// - Throws: An `AFError.serverTrustEvaluationFailed` instance with a
    `.defaultEvaluationFailed` reason.

    func performDefaultValidation(forHost host: String) throws {

        if #available(iOS 12, macOS 10.14, tvOS 12, watchOS 5, *) {

            try evaluate(afterApplying: SecPolicy.af.default)

        } else {

            try validate(policy: SecPolicy.af.default) { status, result in

                AFError.serverTrustEvaluationFailed(reason: .defaultEvaluationFailed(output: .init(host,
                type, status, result)))

            }

        }
    }

```

```

    }

    /// Validates `self` after applying `SecPolicy.af.hostname(host)`, which performs the
    default validation as well as

    /// hostname validation.

    ///

    /// - Parameter host: The hostname to use in the validation.

    /// - Throws:          An `AFError.serverTrustEvaluationFailed` instance with a
    `.defaultEvaluationFailed` reason.

    func performValidation(forHost host: String) throws {

        if #available(iOS 12, macOS 10.14, tvOS 12, watchOS 5, *) {

            try evaluate(afterApplying: SecPolicy.af.hostname(host))

        } else {

            try validate(policy: SecPolicy.af.hostname(host)) { status, result in

                AFError.serverTrustEvaluationFailed(reason: .hostValidationFailed(output: .init(host,
                type, status, result)))

            }

        }

    }

```

```

    }

}

extension SecPolicy: AlamofireExtended {}

public extension AlamofireExtension where ExtendedType == SecPolicy {

    /// Creates a `SecPolicy` instance which will validate server certificates but not require
    a host name match.

    static let `default` = SecPolicyCreateSSL(true, nil)

    /// Creates a `SecPolicy` instance which will validate server certificates and much match
    the provided hostname.

    ///

    /// - Parameter hostname: The hostname to validate against.

    ///

    /// - Returns:      The `SecPolicy`.

    static func hostname(_ hostname: String) -> SecPolicy {

        return SecPolicyCreateSSL(true, hostname as CFString)

    }
}

```

```

/// Creates a `SecPolicy` which checks the revocation of certificates.

///

/// - Parameter options: The `RevocationTrustEvaluator.Options` for evaluation.

///

/// - Returns:      The `SecPolicy`.

/// - Throws:      An `AFError.serverTrustEvaluationFailed` error with reason
`.revocationPolicyCreationFailed`

///              if the policy cannot be created.

static func revocation(options: RevocationTrustEvaluator.Options) throws -> SecPolicy
{

    guard let policy = SecPolicyCreateRevocation(options.rawValue) else {

        throw
        AFError.serverTrustEvaluationFailed(reason: .revocationPolicyCreationFailed)

    }

    return policy

}

}

```

```

extension Array: AlamofireExtended {}

public extension AlamofireExtension where ExtendedType == [SecCertificate] {

    /// All `Data` values for the contained `SecCertificate`s.

    var data: [Data] {

        return type.map { SecCertificateCopyData($0) as Data }

    }

    /// All public `SecKey` values for the contained `SecCertificate`s.

    var publicKeys: [SecKey] {

        return type.compactMap { $0.af.publicKey }

    }

}

extension SecCertificate: AlamofireExtended {}

public extension AlamofireExtension where ExtendedType == SecCertificate {

    /// The public key for `self`, if it can be extracted.

```

```

var publicKey: SecKey? {

    let policy = SecPolicyCreateBasicX509()

    var trust: SecTrust?

    let trustCreationStatus = SecTrustCreateWithCertificates(type, policy, &trust)

    guard let createdTrust = trust, trustCreationStatus == errSecSuccess else { return nil }

    return SecTrustCopyPublicKey(createdTrust)

}

}

extension OSStatus: AlamofireExtended {}

public extension AlamofireExtension where ExtendedType == OSStatus {

    /// Returns whether `self` is `errSecSuccess`.

    var isSuccess: Bool { return type == errSecSuccess }

}

```

```

extension SecTrustResultType: AlamofireExtended {}

public extension AlamofireExtension where ExtendedType == SecTrustResultType {

    /// Returns whether `self` is `.unspecified` or `.proceed`.

    var isSuccess: Bool {

        return (type == .unspecified || type == .proceed)

    }

}

/*

```

See LICENSE folder for this sample's licensing information.

Abstract:

Computed service properties.

*/

```
import HomeKit
```

```
extension HMService {
```

```
/// The service name with "Service" at the end.
```

```
var displayName: String {
```

```
    var name = localizedDescription
```

```
    if !name.hasSuffix("Service") {
```

```
        name += " Service"
```

```
    }
```

```
    return name
```

```
}
```

```
/// Enables or disables notifications of all the service's characteristics that can be enabled.
```

```
/// - Tag: enableNotifications
```

```
func enableNotifications(_ enabled: Bool) {
```

```
    for characteristic in characteristics
```

```
        where
```

```
        characteristic.properties.contains(HMCharacteristicPropertySupportsEventNotification) {
```

```
            characteristic.enableNotification(enabled) { error in
```



```

        if let error = error {

            print(error.localizedDescription)

        }

    }

}

}

}

}

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `ServiceCell` displays a service and information about it.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

/// A `UITableViewCell` subclass for displaying a service and the room and accessory where it resides.

```
class ServiceCell: UITableViewCell {
```

```
    // MARK: Properties
```

```
    var includeAccessoryText = true
```

```
    /// Required init.
```

```
    required init?(coder aDecoder: NSCoder) {
```

```
        super.init(coder: aDecoder)
```

```
    }
```

```
    /**
```

```
        The cell's service.
```

When the service is set, the cell's `textLabel` will contain the service's name

or the accessory's name if the service has no name.

The detail text will contain information about where this service lives.

```
*/

var service: HMSERVICE? {

    didSet {

        if let service = service,

            let accessory = service.accessory {

                titleLabel?.text = service.name

                let accessoryName = accessory.name

                let roomName = accessory.room!.name

                if includeAccessoryText {

                    let inIdentifier = NSLocalizedString("%@ in %@", comment: "Accessory in
Room")

                    detailTextLabel?.text = String(format: inIdentifier, accessoryName,
roomName)

                }

                else {

                    detailTextLabel?.text = ""

                }

            }

        }

    }

}
```

```
    }  
  
    }  
  
    }  
  
    }  
  
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The 'ServiceGroupViewController' allows users to modify service groups.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
/// A view controller that allows the user to add services to a service group.
```

```
class ServiceGroupViewController: HMCatalogViewController, HMAccessoryDelegate {
```

```
// MARK: Types
```

```
struct Identifiers {
```

```
    static let serviceCell = "ServiceCell"
```

```
    static let addServicesSegue = "Add Services Plus"
```

```
}
```

```
// MARK: Properties
```

```
@IBOutlet weak var plusButton: UIBarButtonItem!
```

```
var serviceGroup: HMServiceGroup!
```

```
lazy private var accessories = [HMAccessory]()
```

```
lazy private var servicesForAccessory = [HMAccessory: [HMService]]()
```

```
// MARK: View Methods
```

```

/// Reloads the view.

override func viewWillAppear(_ animated: Bool) {

    super.viewWillAppear(animated)

    title = serviceGroup.name

    reloadData()

}

/// Pops the view controller if our data is invalid.

override func viewDidAppear(_ animated: Bool) {

    super.viewDidAppear(animated)

    if shouldPopViewController() {

        _ = navigationController?.popViewController(animated: true)

    }

}

// MARK: Table View Methods

```

```
/**
```

Generates the number of sections and adds a table view

back ground message, if required.

- returns: The number of accessories in the service group.

```
*/
```

```
override func numberOfSections(in tableView: UITableView) -> Int {
```

```
    let sections = accessories.count
```

```
    if sections == 0 {
```

```
        setBackgroundMessage(NSLocalizedString("No Services", comment: "No Services"))
```

```
    }
```

```
    else {
```

```
        setBackgroundMessage(nil)
```

```
    }
```

```
    return sections
```

```
}
```

/// - returns: The number of services for the accessory at the specified section.

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {

    let accessory = accessories[section]

    let services = servicesForAccessory[accessory]

    return services?.count ?? 0

}
```

/// - returns: The name of the accessory at the specified section.

```
override func tableView(_ tableView: UITableView, titleForHeaderInSection section:
Int) -> String? {

    return accessories[section].name

}
```

/// All cells in the table view represent services and can be deleted.

```
override func tableView(_ tableView: UITableView, canEditRowAt indexPath:
IndexPath) -> Bool {
```



```
return true
```

```
}
```

```
/// - returns: A `ServiceCell` with the service at the given index path.
```

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
```

```
    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.serviceCell, for: indexPath) as! ServiceCell
```

```
    let service = serviceAtPath(indexPath)
```

```
    cell.includeAccessoryText = false
```

```
    cell.service = service
```

```
    return cell
```

```
}
```

```
/**
```

```
- returns: `true` if there are any services not already in the service group;
```

```
    `false` otherwise.
```

```
*/
```

```
private func shouldEnableAdd() -> Bool {
```

```
    let unAddedServices = home.servicesNotAlreadyInServiceGroup(serviceGroup)
```

```
    return unAddedServices.count != 0
```

```
}
```

```
/// Deleting a cell removes the corresponding service from the service group.
```

```
override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {
```

```
    if editingStyle == .delete {
```

```
        removeServiceAtIndex(indexPath)
```

```
    }
```

```
}
```

```
/**
```

```
    Removes the service associated with the cell at a given index path.
```

```
    - parameters indexPath: The `NSIndexPath` to remove.
```

```
*/
```

```

private func removeServiceAtIndexPath(_ indexPath: IndexPath) {

    let service = serviceAtIndexPath(indexPath)

    serviceGroup.removeService(service) { error in

        if let error = error {

            self.displayError(error)

        }

        self.reloadData()

    }

}

```

/**

Finds the service at a given index path.

- parameter indexPath: An 'NSIndexPath'.

- returns: The service at the given index path

```
*/
```

```
private func serviceAtIndexPath(_ indexPath: IndexPath) -> HMService {
```

```
    let accessory = accessories[indexPath.section]
```

```
    let services = servicesForAccessory[accessory]!
```

```
    return services[indexPath.row]
```

```
}
```

```
/// Passes the service group into the `AddServicesViewController`
```

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
```

```
    super.prepare(for: segue, sender: sender)
```

```
    if segue.identifier == Identifiers.addServicesSegue {
```

```
        let addServicesVC = segue.intendedDestinationViewController as!  
AddServicesViewController
```

```
        addServicesVC.serviceGroup = serviceGroup
```

```
    }
```

```
}
```

```
// MARK: Helper Methods
```

```
/**
```

```
    Resets accessory and service lists, resets the plus
```

```
    button's enabled status and reloads the table view.
```

```
*/
```

```
private func reloadData() {
```

```
    resetLists()
```

```
    plusButton.isEnabled = shouldEnableAdd()
```

```
    tableView.reloadData()
```

```
}
```

```
/**
```

```
    Resets the accessories array and the service-accessory mapping
```

```
    using the original HomeKit objects.
```

```
*/
```

```
private func resetLists() {
```

```
    accessories = []
```

```

servicesForAccessory = [:]

for service in serviceGroup.services {

    if let accessory = service.accessory {

        if servicesForAccessory[accessory] == nil {

            accessories.append(accessory)

            servicesForAccessory[accessory] = [service]

        }

        else {

            servicesForAccessory[accessory]?.append(service)

        }

    }

}

// Sort all service lists.

for accessory in accessories {

    servicesForAccessory[accessory]
servicesForAccessory[accessory]?.sortByLocalizedName()

```

```

    }

    // Sort accessory list.

    accessories = accessories.sortByLocalizedNames()

}

/**
 - returns: `true` if our service group is not
        in the home any more; `false` otherwise.
 */

private func shouldPopViewController() -> Bool {

    guard let home = homeStore.home else { return true }

    return !home.serviceGroups.contains { group in

        return group == serviceGroup

    }

}

```

```
/**
```

```
Registers as the delegate for the home and
```

```
all accessories which are related to our service group.
```

```
*/
```

```
override func registerAsDelegate() {
```

```
    super.registerAsDelegate()
```

```
    for service in serviceGroup.services {
```

```
        service.accessory?.delegate = self
```

```
    }
```

```
}
```

```
// MARK: HMHomeDelegate Methods
```

```
/// Pops the view controller if our service group has been deleted.
```

```
func home(_ home: HMHome, didRemoveServiceGroup group: HMServiceGroup) {
```



```

    if group == serviceGroup {

        _ = navigationController?.popViewController(animated: true)

    }

}

```

// Home and accessory changes result in a full data reload.

```

func home(_ home: HMHome, didAddService service: HMService, toServiceGroup
group: HMSERVICEGROUP) {

    if serviceGroup == group {

        reloadData()

    }

}

```

```

func home(_ home: HMHome, didRemoveService service: HMService,
fromServiceGroup group: HMSERVICEGROUP) {

    if serviceGroup == group {

        reloadData()

    }

}

```

```

    }

}

func home(_ home: HMHome, didRemoveAccessory accessory: HMAccessory) {

    reloadData()

}

func accessoryDidUpdateServices(_ accessory: HMAccessory) {

    reloadData()

}

func accessory(_ accessory: HMAccessory, didUpdateNameFor service: HMService) {

    reloadData()

}

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The ``ServicesViewController`` displays an accessory's services.

`*/`

`import UIKit`

`import HomeKit`

`/// Represents the sections in the `ServicesViewController`.`

`enum AccessoryTableViewSection: Int {`

`case services, bridgedAccessories`

`}`

`/**`

A view controller which displays all the services of a provided accessory, and

passes its cell delegate onto a ``CharacteristicsViewController``.

```
*/
```

```
class ServicesViewController: HMCatalogViewController, HMAccessoryDelegate {
```

```
    // MARK: Types
```

```
    struct Identifiers {
```

```
        static let accessoryCell = "AccessoryCell"
```

```
        static let serviceCell = "ServiceCell"
```

```
        static let showServiceSegue = "Show Service"
```

```
    }
```

```
    // MARK: Properties
```

```
    var accessory: HMAccessory!
```

```
    lazy var cellDelegate: CharacteristicCellDelegate = AccessoryUpdateController()
```

```
    var showsFavorites = false
```

```
    var allowsAllWrites = false
```

```
    var onlyShowsControlServices = false
```

```
var displayedServices = [HMService]()
```

```
var bridgedAccessories = [HMAccessory]()
```

```
// MARK: View Methods
```

```
/// Configures table view.
```

```
override func viewDidLoad() {
```

```
    super.viewDidLoad()
```

```
    tableView.estimatedRowHeight = 44.0
```

```
    tableView.rowHeight = UITableView.automaticDimension
```

```
}
```

```
/// Reloads the view.
```

```
override func viewWillAppear(_ animated: Bool) {
```

```
    super.viewWillAppear(animated)
```

```
    updateTitle()
```

```
    reloadData()
```

```
}
```

```
/// Pops the view controller, if required.
```

```
override func viewWillAppear(_ animated: Bool) {
```

```
    super.viewWillAppear(animated)
```

```
    if shouldPopViewController() {
```

```
        _ = navigationController?.popToRootViewController(animated: true)
```

```
    }
```

```
}
```

```
/**
```

```
    Passes the `CharacteristicsViewController` the service from the cell and
```

```
    configures the view controller.
```

```
*/
```

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
```

```
    super.prepare(for: segue, sender: sender)
```

```
guard segue.identifier == Identifiers.showServiceSegue else { return }
```

```
if let indexPath = tableView.indexPath(for: sender as! UITableViewCell) {
```

```
    let selectedService = displayedServices[indexPath.row]
```

```
    let characteristicsViewController = segue.intendedDestinationViewController as!  
    CharacteristicsViewController
```

```
    characteristicsViewController.showsFavorites = showsFavorites
```

```
    characteristicsViewController.allowsAllWrites = allowsAllWrites
```

```
    characteristicsViewController.service = selectedService
```

```
    characteristicsViewController.cellDelegate = cellDelegate
```

```
}
```

```
}
```

```
/**
```

```
- returns: `true` if our accessory is no longer in the
```

```
    current home's list of accessories.
```

```
*/
```

```
private func shouldPopViewController() -> Bool {
```

```

    for accessory in homeStore.home!.accessories {

        if accessory == accessory {

            return false

        }

    }

    return true

}

```

// MARK: Delegate Registration

```
/**
```

Registers as the delegate for the current home
and for the current accessory.

```
*/
```

```

override func registerAsDelegate() {

    super.registerAsDelegate()

    accessory.delegate = self

```



```
}
```

```
// MARK: Table View Methods
```

```
/// Two sections if we're showing bridged accessories.
```

```
override func numberOfSections(in tableView: UITableView) -> Int {
```

```
    if accessory.uniqueIdentifiersForBridgedAccessories != nil {
```

```
        return 2
```

```
    }
```

```
    return 1
```

```
}
```

```
/**
```

```
    Section 1 contains the services within the accessory.
```

```
    Section 2 contains the bridged accessories.
```

```
*/
```

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section:  
Int) -> Int {
```

```

switch AccessoryTableViewSection(rawValue: section) {

    case .services?:

        return displayedServices.count


    case .bridgedAccessories?:

        return bridgedAccessories.count


    case nil:

        fatalError("Unexpected `AccessoryTableViewSection` raw value.")

}

}

/**

- returns: A Service or Bridged Accessory Cell based

on the section.

*/

override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

```

```

switch AccessoryTableViewSection(rawValue: indexPath.section) {

    case .services?:

        return self.tableView(tableView, serviceCellForRowAtIndexPath: indexPath)


    case .bridgedAccessories?:

        return self.tableView(tableView, bridgedAccessoryCellForRowAtIndexPath:
indexPath)


    case nil:

        fatalError("Unexpected `AccessoryTableViewSection` raw value.")

}

}

/**

- returns: A cell containing the name of a bridged

        accessory at a given index path.

*/

```

```
func tableView(_ tableView: UITableView, bridgedAccessoryCellForRowAtIndexPath  
indexPath: IndexPath) -> UITableViewCell {
```

```
    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.accessoryCell,  
for: indexPath)
```

```
    let accessory = bridgedAccessories[indexPath.row]
```

```
    cell.textLabel?.text = accessory.name
```

```
    return cell
```

```
}
```

```
/**
```

```
    - returns: A cell containing the name of a service at
```

```
        a given index path, as well as a localized
```

```
        description of its service type.
```

```
*/
```

```
func tableView(_ tableView: UITableView, serviceCellForRowAtIndexPath indexPath:  
IndexPath) -> UITableViewCell {
```

```
    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.serviceCell, for:  
indexPath)
```

```

let service = displayedServices[indexPath.row]

// Inherit the name from the accessory if the Service doesn't have one.

cell.textLabel?.text = service.name

cell.detailTextLabel?.text = service.localizedDescription

return cell

}

/// - returns: A title string for the section.

override func tableView(_ tableView: UITableView, titleForHeaderInSection section:
Int) -> String? {

    switch AccessoryTableViewSection(rawValue: section) {

        case .services?:

            return NSLocalizedString("Services", comment: "Services")

        case .bridgedAccessories?:

            return NSLocalizedString("Bridged Accessories", comment: "Bridged
Accessories")

```

```

        case nil:

            fatalError("Unexpected `AccessoryTableViewSection` raw value.")

        }

    }

    /// - returns: A localized description of the accessories bridged status.

    override func tableView(_ tableView: UITableView, titleForFooterInSection section:
Int) -> String? {

        if accessory.isBridged && AccessoryTableViewSection(rawValue: section)!
== .services {

            let formatString = NSLocalizedString("This accessory is being bridged into
HomeKit by %@.", comment: "Bridge Description")

            if let bridge = home.bridgeForAccessory(accessory) {

                return String(format: formatString, bridge.name)

            }

            else {

                return NSLocalizedString("This accessory is being bridged into HomeKit.",
comment: "Bridge Description Without Bridge")

```

```

    }

}

return nil

}

// MARK: Helper Methods

/// Updates the navigation bar's title.

func updateTitle() {

    navigationItem.title = accessory.name

}

/**

Updates the title, resets the displayed services based on

view controller configurations, reloads the bridge accessory

array and reloads the table view.

*/

```

```

private func reloadData() {

    displayedServices = accessory.services.sortByLocalizedNames()

    if onlyShowsControlServices {

        // We are configured to only show control services, filter the array.

        displayedServices = displayedServices.filter { service -> Bool in

            return service.isControlType

        }

    }

    if let identifiers = accessory.uniqueIdentifiersForBridgedAccessories {

        bridgedAccessories =
home.accessoriesWithIdentifiers(identifiers).sortByLocalizedNames()

    }

    tableView.reloadData()

}

// MARK: HMAccessoryDelegate Methods

```



```
/// Reloads the title based on the accessories new name.
```

```
func accessoryDidUpdateName(_ accessory: HMAccessory) {  
  
    updateTitle()  
  
}
```

```
/// Reloads the cell for the specified service.
```

```
func accessory(_ accessory: HMAccessory, didUpdateNameFor service: HMService) {  
  
    if let index = displayedServices.firstIndex(of: service) {  
  
        let path = IndexPath(row: index, section:  
AccessoryTableViewSection.services.rawValue)  
  
        tableView.reloadRows(at: [path], with: .automatic)  
  
    }  
  
}
```

```
/// Reloads the view.
```

```
func accessoryDidUpdateServices(_ accessory: HMAccessory) {  
  
    reloadData()  
  
}
```

```

    /// If our accessory has become unreachable, go back the previous view.

    func accessoryDidUpdateReachability(_ accessory: HMAccessory) {

        if self.accessory == accessory {

            _ = navigationController?.popViewController(animated: true)

        }

    }

}

//

// Session.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

```

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

```
import Foundation
```

```
/// `Session` creates and manages Alamofire's `Request` types during their lifetimes. It also provides common
```

```
/// functionality for all `Request`s, including queuing, interception, trust management, redirect handling, and response
```

```
/// cache handling.
```

```
open class Session {
```

```
    /// Shared singleton instance used by all `AF.request` APIs. Cannot be modified.
```

```
    public static let `default` = Session()
```

```
    /// Underlying `URLSession` used to create `URLSessionTasks` for this instance, and for which this instance's
```

```
    /// `delegate` handles `URLSessionDelegate` callbacks.
```

```
    public let session: URLSession
```

```
    /// Instance's `SessionDelegate`, which handles the `URLSessionDelegate` methods and `Request` interaction.
```

```
    public let delegate: SessionDelegate
```

/// Root `DispatchQueue` for all internal callbacks and state update. ****MUST**** be a serial queue.

```
public let rootQueue: DispatchQueue
```

/// Value determining whether this instance automatically calls `resume()` on all created `Request`s.

```
public let startRequestsImmediately: Bool
```

/// `DispatchQueue` on which `URLRequest`s are created asynchronously. By default this queue uses `rootQueue` as its

/// `target`, but a separate queue can be used if request creation is determined to be a bottleneck. Always profile

/// and test before introducing an additional queue.

```
public let requestQueue: DispatchQueue
```

/// `DispatchQueue` passed to all `Request`s on which they perform their response serialization. By default this

/// queue uses `rootQueue` as its `target` but a separate queue can be used if response serialization is determined

/// to be a bottleneck. Always profile and test before introducing an additional queue.

```
public let serializationQueue: DispatchQueue
```

/// `RequestInterceptor` used for all `Request` created by the instance. `RequestInterceptor`s can also be set on a

/// per-`Request` basis, in which case the `Request`'s interceptor takes precedence over this value.

```
public let interceptor: RequestInterceptor?
```

/// `ServerTrustManager` instance used to evaluate all trust challenges and provide certificate and key pinning.

```
public let serverTrustManager: ServerTrustManager?
```

/// `RedirectHandler` instance used to provide customization for request redirection.

```
public let redirectHandler: RedirectHandler?
```

/// `CachedResponseHandler` instance used to provide customization of cached response handling.

```
public let cachedResponseHandler: CachedResponseHandler?
```

/// `CompositeEventMonitor` used to compose Alamofire's `defaultEventMonitors` and any passed `EventMonitor`s.

```
public let eventMonitor: CompositeEventMonitor
```

/// `EventMonitor`s included in all instances. `[AlamofireNotifications()]` by default.

```
public let defaultEventMonitors: [EventMonitor] = [AlamofireNotifications()]
```

/// Internal map between `Request`s and any `URLSessionTasks` that may be in flight for them.

```
var requestTaskMap = RequestTaskMap()
```

```

/// Set of currently active `Request`s.

var activeRequests: Set<Request> = []

/// Completion events awaiting `URLSessionTaskMetrics`.

var waitingCompletions: [URLSessionTask: () -> Void] = [:]


///

/// - Note: When passing a `URLSession`, you must create the `URLSession` with a
specific `delegateQueue` value and

///     pass the `delegateQueue`'s `underlyingQueue` as the `rootQueue` parameter of
this initializer.

///

/// - Parameters:

/// - session:           Underlying `URLSession` for this instance.

/// - delegate:         `SessionDelegate` that handles `session`'s delegate callbacks as
well as `Request`

///                     interaction.

/// - rootQueue:         Root `DispatchQueue` for all internal callbacks and state
updates. **MUST** be a

```

/// serial queue.

/// - startRequestsImmediately: Determines whether this instance will automatically start all `Request`s. `true`

/// by default. If set to `false`, all `Request`s created must have
`.resume()` called.

/// on them for them to start.

/// - requestQueue: `DispatchQueue` on which to perform `URLRequest` creation.
By default this queue

/// will use the `rootQueue` as its `target`. A separate queue can be
used if it's

/// determined request creation is a bottleneck, but that should only be
done after

/// careful testing and profiling. `nil` by default.

/// - serializationQueue: `DispatchQueue` on which to perform all response
serialization. By default this

/// queue will use the `rootQueue` as its `target`. A separate queue can
be used if

/// it's determined response serialization is a bottleneck, but that should
only be

/// done after careful testing and profiling. `nil` by default.

/// - interceptor: `RequestInterceptor` to be used for all `Request`s created by
this instance. `nil`


```

///                                     by default.

/// - serverTrustManager:    `ServerTrustManager` to be used for all trust evaluations
by this instance. `nil`

///                                     by default.

/// - redirectHandler:      `RedirectHandler` to be used by all `Request`s created by
this instance. `nil` by

///                                     default.

/// - cachedResponseHandler: `CachedResponseHandler` to be used by all `Request`s
created by this instance.

///                                     `nil` by default.

/// - eventMonitors:        Additional `EventMonitor`s used by the instance. Alamofire
always adds a

///                                     `AlamofireNotifications` `EventMonitor` to the array passed here.
`[]` by default.

public init(session: URLSession,

           delegate: SessionDelegate,

           rootQueue: DispatchQueue,

           startRequestsImmediately: Bool = true,

           requestQueue: DispatchQueue? = nil,

           serializationQueue: DispatchQueue? = nil,

```

```

        interceptor: RequestInterceptor? = nil,

        serverTrustManager: ServerTrustManager? = nil,

        redirectHandler: RedirectHandler? = nil,

        cachedResponseHandler: CachedResponseHandler? = nil,

        eventMonitors: [EventMonitor] = []) {

    precondition(session.configuration.identifier == nil,

        "Alamofire does not support background URLSessionConfigurations.")

    precondition(session.delegateQueue.underlyingQueue === rootQueue,

        "Session(session:) initializer must be passed the DispatchQueue used as the
        delegateQueue's underlyingQueue as rootQueue.")

    self.session = session

    self.delegate = delegate

    self.rootQueue = rootQueue

    self.startRequestsImmediately = startRequestsImmediately

    self.requestQueue = requestQueue ?? DispatchQueue(label:
    "\(rootQueue.label).requestQueue", target: rootQueue)

    self.serializationQueue = serializationQueue ?? DispatchQueue(label:
    "\(rootQueue.label).serializationQueue", target: rootQueue)

```

```

        self.interceptor = interceptor

        self.serverTrustManager = serverTrustManager

        self.redirectHandler = redirectHandler

        self.cachedResponseHandler = cachedResponseHandler

        eventMonitor = CompositeEventMonitor(monitors: defaultEventMonitors +
eventMonitors)

        delegate.eventMonitor = eventMonitor

        delegate.stateProvider = self
    }

    /// Creates a `Session` from a `URLSessionConfiguration`.

    ///

    /// - Note: This initializer lets Alamofire handle the creation of the underlying
`URLSession` and its

    ///     `delegateQueue`, and is the recommended initializer for most uses.

    ///

    /// - Parameters:

    ///     - configuration: `URLSessionConfiguration` to be used to create the underlying
`URLSession`. Changes

```

/// to this value after being passed to this initializer will have no effect.

/// `URLSessionConfiguration.af.default` by default.

/// - delegate: `SessionDelegate` that handles `session`'s delegate callbacks as well as `Request` interaction. `SessionDelegate()` by default.

/// - rootQueue: Root `DispatchQueue` for all internal callbacks and state updates. ****MUST**** be a serial queue. `DispatchQueue(label: "org.alamofire.session.rootQueue")` by default.

/// - startRequestsImmediately: Determines whether this instance will automatically start all `Request`'s. `true` by default. If set to `false`, all `Request`'s created must have `.resume()` called.

/// on them for them to start.

/// - requestQueue: `DispatchQueue` on which to perform `URLRequest` creation. By default this queue will use the `rootQueue` as its `target`. A separate queue can be used if it's determined request creation is a bottleneck, but that should only be done after careful testing and profiling. `nil` by default.

/// - serializationQueue: `DispatchQueue` on which to perform all response serialization. By default this

/// queue will use the `rootQueue` as its `target`. A separate queue can be used if

/// it's determined response serialization is a bottleneck, but that should only be

/// done after careful testing and profiling. `nil` by default.

/// - interceptor: `RequestInterceptor` to be used for all `Request`s created by this instance. `nil`

/// by default.

/// - serverTrustManager: `ServerTrustManager` to be used for all trust evaluations by this instance. `nil`

/// by default.

/// - redirectHandler: `RedirectHandler` to be used by all `Request`s created by this instance. `nil` by

/// default.

/// - cachedResponseHandler: `CachedResponseHandler` to be used by all `Request`s created by this instance.

/// `nil` by default.

/// - eventMonitors: Additional `EventMonitor`s used by the instance. Alamofire always adds a

```
/// 'AlamofireNotifications' 'EventMonitor' to the array passed here.  
'[]' by default.
```

```
public convenience init(configuration: URLSessionConfiguration =  
URLSessionConfiguration.af.default,
```

```
    delegate: SessionDelegate = SessionDelegate(),
```

```
    rootQueue: DispatchQueue = DispatchQueue(label:  
"org.alamofire.session.rootQueue"),
```

```
    startRequestsImmediately: Bool = true,
```

```
    requestQueue: DispatchQueue? = nil,
```

```
    serializationQueue: DispatchQueue? = nil,
```

```
    interceptor: RequestInterceptor? = nil,
```

```
    serverTrustManager: ServerTrustManager? = nil,
```

```
    redirectHandler: RedirectHandler? = nil,
```

```
    cachedResponseHandler: CachedResponseHandler? = nil,
```

```
    eventMonitors: [EventMonitor] = []) {
```

```
    precondition(configuration.identifier == nil, "Alamofire does not support background  
URLSessionConfigurations.")
```

```
    let delegateQueue = OperationQueue(maxConcurrentOperationCount: 1,  
    underlyingQueue: rootQueue, name: "org.alamofire.session.sessionDelegateQueue")
```

```
let session = URLSession(configuration: configuration, delegate: delegate,  
delegateQueue: delegateQueue)
```

```
self.init(session: session,  
  
          delegate: delegate,  
  
          rootQueue: rootQueue,  
  
          startRequestsImmediately: startRequestsImmediately,  
  
          requestQueue: requestQueue,  
  
          serializationQueue: serializationQueue,  
  
          interceptor: interceptor,  
  
          serverTrustManager: serverTrustManager,  
  
          redirectHandler: redirectHandler,  
  
          cachedResponseHandler: cachedResponseHandler,  
  
          eventMonitors: eventMonitors)  
  
}
```

```
deinit {  
  
    finishRequestsForDeinit()  
}
```

```

        session.invalidateAndCancel()

    }

// MARK: - Cancellation

/// Cancel all active `Request`s, optionally calling a completion handler when complete.

///

/// - Note: This is an asynchronous operation and does not block the creation of future
`Request`s. Cancelled

///     `Request`s may not cancel immediately due internal work, and may not cancel
at all if they are close to

///     completion when cancelled.

///

/// - Parameters:

/// - queue:     `DispatchQueue` on which the completion handler is run. `.main` by
default.

/// - completion: Closure to be called when all `Request`s have been cancelled.

public func cancelAllRequests(completingOnQueue queue: DispatchQueue = .main,
completion: (() -> Void)? = nil) {

```



```

rootQueue.async {

    self.activeRequests.forEach { $0.cancel() }

    queue.async { completion?() }

}

}

```

// MARK: - DataRequest

```

struct RequestConvertible: URLRequestConvertible {

    let url: URLConvertible

    let method: HTTPMethod

    let parameters: Parameters?

    let encoding: ParameterEncoding

    let headers: HTTPHeaders?

    func asURLRequest() throws -> URLRequest {

        let request = try URLRequest(url: url, method: method, headers: headers)

```

```

        return try encoding.encode(request, with: parameters)

    }

}

/// Creates a `DataRequest` from a `URLRequest` created using the passed components
and a `RequestInterceptor`.

///

/// - Parameters:

/// - convertible: `URLConvertible` value to be used as the `URLRequest`'s `URL`.

/// - method: `HTTPMethod` for the `URLRequest`. `.get` by default.

/// - parameters: `Parameters` (a.k.a. `[String: Any]`) value to be encoded into the
`URLRequest`. `nil` by default.

/// - encoding: `ParameterEncoding` to be used to encode the `parameters` value into
the `URLRequest`.

/// `URLEncoding.default` by default.

/// - headers: `HTTPHeaders` value to be added to the `URLRequest`. `nil` by default.

/// - interceptor: `RequestInterceptor` value to be used by the returned `DataRequest`.
`nil` by default.

///

```

/// - Returns: The created `DataRequest`.

```
open fun request(_ convertible: URLConvertible,

    method: HTTPMethod = .get,

    parameters: Parameters? = nil,

    encoding: ParameterEncoding = URLEncoding.default,

    headers: HTTPHeaders? = nil,

    interceptor: RequestInterceptor? = nil) -> DataRequest {

    let convertible = RequestConvertible(url: convertible,

        method: method,

        parameters: parameters,

        encoding: encoding,

        headers: headers)

    return request(convertible, interceptor: interceptor)

}
```

```
struct RequestEncodableConvertible<Parameters: Encodable>:
URLRequestConvertible {
```

```
let url: URLConvertible
```

```
let method: HTTPMethod
```

```
let parameters: Parameters?
```

```
let encoder: ParameterEncoder
```

```
let headers: HTTPHeaders?
```

```
func asURLRequest() throws -> URLRequest {
```

```
    let request = try URLRequest(url: url, method: method, headers: headers)
```

```
    return try parameters.map { try encoder.encode($0, into: request) } ?? request
```

```
}
```

```
}
```

```
/// Creates a `DataRequest` from a `URLRequest` created using the passed components,  
/// `Encodable` parameters, and a
```

```
/// `RequestInterceptor`.
```

```
///
```

```
/// - Parameters:
```

```

/// - convertible: `URLConvertible` value to be used as the `URLRequest`'s `URL`.

/// - method:    `HTTPMethod` for the `URLRequest`. `.get` by default.

/// - parameters: `Encodable` value to be encoded into the `URLRequest`. `nil` by
default.

/// - encoder:    `ParameterEncoder` to be used to encode the `parameters` value into
the `URLRequest`.

///              `URLEncodedFormParameterEncoder.default` by default.

/// - headers:    `HTTPHeaders` value to be added to the `URLRequest`. `nil` by default.

/// - interceptor: `RequestInterceptor` value to be used by the returned `DataRequest`.
`nil` by default.

///

/// - Returns:    The created `DataRequest`.

open func request<Parameters: Encodable>(_ convertible: URLConvertible,

                                         method: HTTPMethod = .get,

                                         parameters: Parameters? = nil,

                                         encoder:           ParameterEncoder           =
URLEncodedFormParameterEncoder.default,

                                         headers: HTTPHeaders? = nil,

                                         interceptor: RequestInterceptor? = nil) -> DataRequest {

```

```

let convertible = RequestEncodableConvertible(url: convertible,

                                              method: method,

                                              parameters: parameters,

                                              encoder: encoder,

                                              headers: headers)

return request(convertible, interceptor: interceptor)

}

```

/// Creates a `DataRequest` from a `URLRequestConvertible` value and a `RequestInterceptor`.

///

/// - Parameters:

/// - convertible: `URLRequestConvertible` value to be used to create the `URLRequest`.

/// - interceptor: `RequestInterceptor` value to be used by the returned `DataRequest`.
`nil` by default.

///

/// - Returns: The created `DataRequest`.

```
open func request(_ convertible: URLRequestConvertible, interceptor:
RequestInterceptor? = nil) -> DataRequest {
```

```
    let request = DataRequest(convertible: convertible,
```

```
                               underlyingQueue: rootQueue,
```

```
                               serializationQueue: serializationQueue,
```

```
                               eventMonitor: eventMonitor,
```

```
                               interceptor: interceptor,
```

```
                               delegate: self)
```

```
    perform(request)
```

```
    return request
```

```
}
```

```
// MARK: - DownloadRequest
```

```
/// Creates a `DownloadRequest` using a `URLRequest` created using the passed
components, `RequestInterceptor`, and
```

```

/// `Destination`.

///

/// - Parameters:

/// - convertible: `URLConvertible` value to be used as the `URLRequest`'s `URL`.

/// - method: `HTTPMethod` for the `URLRequest`. `.get` by default.

/// - parameters: `Parameters` (a.k.a. `[String: Any]`) value to be encoded into the
`URLRequest`. `nil` by default.

/// - encoding: `ParameterEncoding` to be used to encode the `parameters` value into
the `URLRequest`. Defaults

/// to `URLEncoding.default`.

/// - headers: `HTTPHeaders` value to be added to the `URLRequest`. `nil` by default.

/// - interceptor: `RequestInterceptor` value to be used by the returned `DataRequest`.
`nil` by default.

/// - destination: `DownloadRequest.Destination` closure used to determine how and
where the downloaded file

/// should be moved. `nil` by default.

///

/// - Returns: The created `DownloadRequest`.

open func download(_ convertible: URLConvertible,

```



```

        method: HTTPMethod = .get,

        parameters: Parameters? = nil,

        encoding: ParameterEncoding = URLEncoding.default,

        headers: HTTPHeaders? = nil,

        interceptor: RequestInterceptor? = nil,

        to destination: DownloadRequest.Destination? = nil) -> DownloadRequest
    {

        let convertible = RequestConvertible(url: convertible,

            method: method,

            parameters: parameters,

            encoding: encoding,

            headers: headers)

        return download(convertible, interceptor: interceptor, to: destination)

    }

```

/// Creates a `DownloadRequest` from a `URLRequest` created using the passed components, `Encodable` parameters, and

```

    /// a `RequestInterceptor`.

    ///

    /// - Parameters:

    /// - convertible: `URLConvertible` value to be used as the `URLRequest`'s `URL`.

    /// - method: `HTTPMethod` for the `URLRequest`. `.get` by default.

    /// - parameters: Value conforming to `Encodable` to be encoded into the
    `URLRequest`. `nil` by default.

    /// - encoder: `ParameterEncoder` to be used to encode the `parameters` value into
    the `URLRequest`. Defaults

    /// to `URLEncodedFormParameterEncoder.default`.

    /// - headers: `HTTPHeaders` value to be added to the `URLRequest`. `nil` by default.

    /// - interceptor: `RequestInterceptor` value to be used by the returned `DataRequest`.
    `nil` by default.

    /// - destination: `DownloadRequest.Destination` closure used to determine how and
    where the downloaded file

    /// should be moved. `nil` by default.

    ///

    /// - Returns: The created `DownloadRequest`.

    open func download<Parameters: Encodable>(_ convertible: URLConvertible,

```

```

        method: HTTPMethod = .get,

        parameters: Parameters? = nil,

        encoder: ParameterEncoder =
URLEncodedFormParameterEncoder.default,

        headers: HTTPHeaders? = nil,

        interceptor: RequestInterceptor? = nil,

        to destination: DownloadRequest.Destination? = nil) ->
DownloadRequest {

    let convertible = RequestEncodableConvertible(url: convertible,

        method: method,

        parameters: parameters,

        encoder: encoder,

        headers: headers)

    return download(convertible, interceptor: interceptor, to: destination)

}

```

/// Creates a `DownloadRequest` from a `URLRequestConvertible` value, a `RequestInterceptor`, and a `Destination`.

```

///

/// - Parameters:

/// - convertible: `URLRequestConvertible` value to be used to create the `URLRequest`.

/// - interceptor: `RequestInterceptor` value to be used by the returned `DataRequest`.
`nil` by default.

/// - destination: `DownloadRequest.Destination` closure used to determine how and
where the downloaded file

/// should be moved. `nil` by default.

///

/// - Returns: The created `DownloadRequest`.

open func download(_ convertible: URLRequestConvertible,

                  interceptor: RequestInterceptor? = nil,

                  to destination: DownloadRequest.Destination? = nil) -> DownloadRequest
{

    let request = DownloadRequest(downloadable: .request(convertible),

                                  underlyingQueue: rootQueue,

                                  serializationQueue: serializationQueue,

                                  eventMonitor: eventMonitor,

                                  interceptor: interceptor,

```

delegate: self,

destination: destination ?? DownloadRequest.defaultDestination)

perform(request)

return request

}

/// Creates a `DownloadRequest` from the `resumeData` produced from a previously cancelled `DownloadRequest`, as

/// well as a `RequestInterceptor`, and a `Destination`.

///

/// - Note: If `destination` is not specified, the download will be moved to a temporary location determined by

/// Alamofire. The file will not be deleted until the system purges the temporary files.

///

/// - Note: On some versions of all Apple platforms (iOS 10 - 10.2, macOS 10.12 - 10.12.2, tvOS 10 - 10.1, watchOS 3 - 3.1.1),

/// `resumeData` is broken on background URL session configurations. There's an underlying bug in the `resumeData`

/// generation logic where the data is written incorrectly and will always fail to resume the download. For more

/// information about the bug and possible workarounds, please refer to the [this Stack Overflow post](<http://stackoverflow.com/a/39347461/1342462>).

///

/// - Parameters:

/// - data: The resume data from a previously cancelled `DownloadRequest` or `URLSessionDownloadTask`.

/// - interceptor: `RequestInterceptor` value to be used by the returned `DataRequest`. `nil` by default.

/// - destination: `DownloadRequest.Destination` closure used to determine how and where the downloaded file

/// should be moved. `nil` by default.

///

/// - Returns: The created `DownloadRequest`.

```
open func download(resumingWith data: Data,  
  
                  interceptor: RequestInterceptor? = nil,  
  
                  to destination: DownloadRequest.Destination? = nil) -> DownloadRequest  
{
```

```
let request = DownloadRequest(downloadable: .resumeData(data),  
  
    underlyingQueue: rootQueue,  
  
    serializationQueue: serializationQueue,  
  
    eventMonitor: eventMonitor,  
  
    interceptor: interceptor,  
  
    delegate: self,  
  
    destination: destination ?? DownloadRequest.defaultDestination)
```

```
perform(request)
```

```
return request
```

```
}
```

```
// MARK: - UploadRequest
```

```
struct ParameterlessRequestConvertible: URLRequestConvertible {
```

```
    let url: URLConvertible
```

```
let method: HTTPMethod
```

```
let headers: HTTPHeaders?
```

```
func asURLRequest() throws -> URLRequest {
```

```
    return try URLRequest(url: url, method: method, headers: headers)
```

```
}
```

```
}
```

```
struct Upload: UploadConvertible {
```

```
    let request: URLRequestConvertible
```

```
    let uploadable: UploadableConvertible
```

```
func createUploadable() throws -> UploadRequest.Uploadable {
```

```
    return try uploadable.createUploadable()
```

```
}
```

```
func asURLRequest() throws -> URLRequest {
```



```

        return try request.asURLRequest()

    }

}

// MARK: Data

/// Creates an `UploadRequest` for the given `Data`, `URLRequest` components, and
`RequestInterceptor`.

///

/// - Parameters:

/// - data:      The `Data` to upload.

/// - convertible: `URLConvertible` value to be used as the `URLRequest`'s `URL`.

/// - method:    `HTTPMethod` for the `URLRequest`. `.post` by default.

/// - headers:   `HTTPHeaders` value to be added to the `URLRequest`. `nil` by default.

/// - interceptor: `RequestInterceptor` value to be used by the returned `DataRequest`.
`nil` by default.

/// - fileManager: `FileManager` instance to be used by the returned `UploadRequest`.
`.default` instance by

///          default.

```

```

///

/// - Returns:    The created `UploadRequest`.

open fun upload(_ data: Data,

                to convertible: URLConvertible,

                method: HTTPMethod = .post,

                headers: HTTPHeaders? = nil,

                interceptor: RequestInterceptor? = nil,

                fileManager: FileManager = .default) -> UploadRequest {

    let convertible = ParameterlessRequestConvertible(url: convertible, method: method,
headers: headers)

    return upload(data, with: convertible, interceptor: interceptor, fileManager:
fileManager)

}

/// Creates an `UploadRequest` for the given `Data` using the `URLRequestConvertible`
value and `RequestInterceptor`.

///

/// - Parameters:

```

```

/// - data:      The `Data` to upload.

/// - convertible: `URLRequestConvertible` value to be used to create the `URLRequest`.

/// - interceptor: `RequestInterceptor` value to be used by the returned `DataRequest`.
`nil` by default.

/// - fileManager: `FileManager` instance to be used by the returned `UploadRequest`.
`.default` instance by

///          default.

///

/// - Returns:    The created `UploadRequest`.

open func upload(_ data: Data,

                with convertible: URLRequestConvertible,

                interceptor: RequestInterceptor? = nil,

                fileManager: FileManager = .default) -> UploadRequest {

    return upload(.data(data), with: convertible, interceptor: interceptor, fileManager:
fileManager)

}

// MARK: File

```

/// Creates an `UploadRequest` for the file at the given file `URL`, using a `URLRequest` from the provided

/// components and `RequestInterceptor`.

///

/// - Parameters:

/// - fileURL: The `URL` of the file to upload.

/// - convertible: `URLConvertible` value to be used as the `URLRequest`'s `URL`.

/// - method: `HTTPMethod` for the `URLRequest`. `.post` by default.

/// - headers: `HTTPHeaders` value to be added to the `URLRequest`. `nil` by default.

/// - interceptor: `RequestInterceptor` value to be used by the returned `UploadRequest`. `nil` by default.

/// - fileManager: `FileManager` instance to be used by the returned `UploadRequest`. `.default` instance by

/// default.

///

/// - Returns: The created `UploadRequest`.

open func upload(_ fileURL: URL,

to convertible: URLConvertible,

method: HTTPMethod = .post,

```

        headers: HTTPHeaders? = nil,

        interceptor: RequestInterceptor? = nil,

        fileManager: FileManager = .default) -> UploadRequest {

    let convertible = ParameterlessRequestConvertible(url: convertible, method: method,
headers: headers)

    return upload(fileURL, with: convertible, interceptor: interceptor, fileManager:
fileManager)

}

/// Creates an `UploadRequest` for the file at the given file `URL` using the
`URLRequestConvertible` value and

/// `RequestInterceptor`.

///

/// - Parameters:

/// - fileURL: The `URL` of the file to upload.

/// - convertible: `URLRequestConvertible` value to be used to create the `URLRequest`.

/// - interceptor: `RequestInterceptor` value to be used by the returned `DataRequest`.
`nil` by default.

```

```
/// - fileManager: `FileManager` instance to be used by the returned `UploadRequest`.  
`.default` instance by
```

```
///         default.
```

```
///
```

```
/// - Returns:    The created `UploadRequest`.
```

```
open func upload(_ fileURL: URL,
```

```
                with convertible: URLRequestConvertible,
```

```
                interceptor: RequestInterceptor? = nil,
```

```
                fileManager: FileManager = .default) -> UploadRequest {
```

```
    return upload(.file(fileURL, shouldRemove: false), with: convertible, interceptor:  
interceptor, fileManager: fileManager)
```

```
}
```

```
// MARK: InputStream
```

```
/// Creates an `UploadRequest` from the `InputStream` provided using a `URLRequest`  
from the provided components and
```

```
/// `RequestInterceptor`.
```

```
///
```

```

/// - Parameters:

/// - stream:    The `InputStream` that provides the data to upload.

/// - convertible: `URLConvertible` value to be used as the `URLRequest`'s `URL`.

/// - method:    `HTTPMethod` for the `URLRequest`. `.post` by default.

/// - headers:   `HTTPHeaders` value to be added to the `URLRequest`. `nil` by default.

/// - interceptor: `RequestInterceptor` value to be used by the returned `DataRequest`.
`nil` by default.

/// - fileManager: `FileManager` instance to be used by the returned `UploadRequest`.
`.default` instance by

///             default.

///

/// - Returns:    The created `UploadRequest`.

open func upload(_ stream: InputStream,

                to convertible: URLConvertible,

                method: HTTPMethod = .post,

                headers: HTTPHeaders? = nil,

                interceptor: RequestInterceptor? = nil,

                fileManager: FileManager = .default) -> UploadRequest {

```

```
let convertible = ParameterlessRequestConvertible(url: convertible, method: method,
headers: headers)
```

```
return upload(stream, with: convertible, interceptor: interceptor, fileManager:
fileManager)
```

```
}
```

```
/// Creates an `UploadRequest` from the provided `InputStream` using the
`URLRequestConvertible` value and
```

```
/// `RequestInterceptor`.
```

```
///
```

```
/// - Parameters:
```

```
/// - stream: The `InputStream` that provides the data to upload.
```

```
/// - convertible: `URLRequestConvertible` value to be used to create the `URLRequest`.
```

```
/// - interceptor: `RequestInterceptor` value to be used by the returned `DataRequest`.
`nil` by default.
```

```
/// - fileManager: `FileManager` instance to be used by the returned `UploadRequest`.
`.default` instance by
```

```
/// default.
```

```
///
```



```

/// - Returns:    The created `UploadRequest`.

open fun upload(_ stream: InputStream,

                with convertible: URLRequestConvertible,

                interceptor: RequestInterceptor? = nil,

                fileManager: FileManager = .default) -> UploadRequest {

    return upload(.stream(stream), with: convertible, interceptor: interceptor, fileManager:
fileManager)

}

// MARK: MultipartFormData

/// Creates an `UploadRequest` for the multipart form data built using a closure and sent
using the provided

/// `URLRequest` components and `RequestInterceptor`.

///

/// It is important to understand the memory implications of uploading
`MultipartFormData`. If the cumulative

/// payload is small, encoding the data in-memory and directly uploading to a server is
the by far the most

```

/// efficient approach. However, if the payload is too large, encoding the data in-memory could cause your app to

/// be terminated. Larger payloads must first be written to disk using input and output streams to keep the memory

/// footprint low, then the data can be uploaded as a stream from the resulting file. Streaming from disk MUST be

/// used for larger payloads such as video content.

///

/// The ``encodingMemoryThreshold`` parameter allows Alamofire to automatically determine whether to encode in-memory

/// or stream from disk. If the content length of the ``MultipartFormData`` is below the ``encodingMemoryThreshold``,

/// encoding takes place in-memory. If the content length exceeds the threshold, the data is streamed to disk

/// during the encoding process. Then the result is uploaded as data or as a stream depending on which encoding

/// technique was used.

///

/// - Parameters:

/// - multipartFormData: ``MultipartFormData`` building closure.


```

        method: HTTPMethod = .post,

        headers: HTTPHeaders? = nil,

        interceptor: RequestInterceptor? = nil,

        fileManager: FileManager = .default) -> UploadRequest {

    let convertible = ParameterlessRequestConvertible(url: url, method: method, headers:
headers)

    let formData = MultipartFormData(fileManager: fileManager)

    multipartFormData(formData)

    return upload(multipartFormData: formData,

        with: convertible,

        usingThreshold: encodingMemoryThreshold,

        interceptor: interceptor,

        fileManager: fileManager)

}

```

/// Creates an `UploadRequest` using a `MultipartFormData` building closure, the provided `URLRequestConvertible`

/// value, and a `RequestInterceptor`.

///

/// It is important to understand the memory implications of uploading `MultipartFormData`. If the cumulative

/// payload is small, encoding the data in-memory and directly uploading to a server is the by far the most

/// efficient approach. However, if the payload is too large, encoding the data in-memory could cause your app to

/// be terminated. Larger payloads must first be written to disk using input and output streams to keep the memory

/// footprint low, then the data can be uploaded as a stream from the resulting file. Streaming from disk **MUST** be

/// used for larger payloads such as video content.

///

/// The `encodingMemoryThreshold` parameter allows Alamofire to automatically determine whether to encode in-memory

/// or stream from disk. If the content length of the `MultipartFormData` is below the `encodingMemoryThreshold`,

/// encoding takes place in-memory. If the content length exceeds the threshold, the data is streamed to disk

/// during the encoding process. Then the result is uploaded as data or as a stream depending on which encoding

/// technique was used.

///

/// - Parameters:

/// - multipartFormData: `MultipartFormData` building closure.

/// - request: `URLRequestConvertible` value to be used to create the `URLRequest`.

/// - encodingMemoryThreshold: Byte threshold used to determine whether the form data is encoded into memory or

/// onto disk before being uploaded.
`MultipartFormData.encodingMemoryThreshold` by

/// default.

/// - interceptor: `RequestInterceptor` value to be used by the returned `DataRequest`. `nil` by default.

/// - fileManager: `FileManager` to be used if the form data exceeds the memory threshold and is

/// written to disk before being uploaded. `.default` instance by default.

///

/// - Returns: The created `UploadRequest`.

```

open func upload(multipartFormData: @escaping (MultipartFormData) -> Void,

                with request: URLRequestConvertible,

                usingThreshold          encodingMemoryThreshold: UInt64          =
MultipartFormData.encodingMemoryThreshold,

                interceptor: RequestInterceptor? = nil,

                fileManager: FileManager = .default) -> UploadRequest {

    let formData = MultipartFormData(fileManager: fileManager)

    multipartFormData(formData)

    return upload(multipartFormData: formData,

                with: request,

                usingThreshold: encodingMemoryThreshold,

                interceptor: interceptor,

                fileManager: fileManager)

}

```

/// Creates an 'UploadRequest' for the prebuilt 'MultipartFormData' value using the provided 'URLRequest' components

/// and `'RequestInterceptor'`.

///

/// It is important to understand the memory implications of uploading `'MultipartFormData'`. If the cumulative

/// payload is small, encoding the data in-memory and directly uploading to a server is the by far the most

/// efficient approach. However, if the payload is too large, encoding the data in-memory could cause your app to

/// be terminated. Larger payloads must first be written to disk using input and output streams to keep the memory

/// footprint low, then the data can be uploaded as a stream from the resulting file. Streaming from disk MUST be

/// used for larger payloads such as video content.

///

/// The `'encodingMemoryThreshold'` parameter allows Alamofire to automatically determine whether to encode in-memory

/// or stream from disk. If the content length of the `'MultipartFormData'` is below the `'encodingMemoryThreshold'`,

/// encoding takes place in-memory. If the content length exceeds the threshold, the data is streamed to disk

/// during the encoding process. Then the result is uploaded as data or as a stream depending on which encoding


```

/// technique was used.

///

/// - Parameters:

/// - multipartFormData:    `MultipartFormData` instance to upload.

/// - url:                  `URLConvertible` value to be used as the `URLRequest`'s `URL`.

/// - encodingMemoryThreshold: Byte threshold used to determine whether the form
data is encoded into memory or

///                          onto disk before being uploaded.
`MultipartFormData.encodingMemoryThreshold` by

///                          default.

/// - method:               `HTTPMethod` for the `URLRequest`. `.post` by default.

/// - headers:              `HTTPHeaders` value to be added to the `URLRequest`. `nil`
by default.

/// - interceptor:          `RequestInterceptor` value to be used by the returned
`DataRequest`. `nil` by default.

/// - fileManager:          `FileManager` to be used if the form data exceeds the memory
threshold and is

///                          written to disk before being uploaded. `.default` instance by default.

///

/// - Returns:              The created `UploadRequest`.

```

```

open func upload(multipartFormData: MultipartFormData,

                to url: URLConvertible,

                usingThreshold          encodingMemoryThreshold: UInt64 =
MultipartFormData.encodingMemoryThreshold,

                method: HTTPMethod = .post,

                headers: HTTPHeaders? = nil,

                interceptor: RequestInterceptor? = nil,

                fileManager: FileManager = .default) -> UploadRequest {

    let convertible = ParameterlessRequestConvertible(url: url, method: method, headers:
headers)

    let multipartUpload = MultipartUpload(isInBackgroundSession:
session.configuration.identifier != nil,

                                         encodingMemoryThreshold: encodingMemoryThreshold,

                                         request: convertible,

                                         multipartFormData: multipartFormData)

    return upload(multipartUpload, interceptor: interceptor, fileManager: fileManager)
}

```

/// Creates an `UploadRequest` for the prebuilt `MultipartFormData` value using the providing `URLRequestConvertible`

/// value and `RequestInterceptor`.

///

/// It is important to understand the memory implications of uploading `MultipartFormData`. If the cumulative

/// payload is small, encoding the data in-memory and directly uploading to a server is the by far the most

/// efficient approach. However, if the payload is too large, encoding the data in-memory could cause your app to

/// be terminated. Larger payloads must first be written to disk using input and output streams to keep the memory

/// footprint low, then the data can be uploaded as a stream from the resulting file. Streaming from disk **MUST** be

/// used for larger payloads such as video content.

///

/// The `encodingMemoryThreshold` parameter allows Alamofire to automatically determine whether to encode in-memory

/// or stream from disk. If the content length of the `MultipartFormData` is below the `encodingMemoryThreshold`,

/// encoding takes place in-memory. If the content length exceeds the threshold, the data is streamed to disk

/// during the encoding process. Then the result is uploaded as data or as a stream depending on which encoding

/// technique was used.

///

/// - Parameters:

/// - multipartFormData: `MultipartFormData` instance to upload.

/// - request: `URLRequestConvertible` value to be used to create the `URLRequest`.

/// - encodingMemoryThreshold: Byte threshold used to determine whether the form data is encoded into memory or

/// onto disk before being uploaded.
`MultipartFormData.encodingMemoryThreshold` by

/// default.

/// - interceptor: `RequestInterceptor` value to be used by the returned `DataRequest`. `nil` by default.

/// - fileManager: `FileManager` instance to be used by the returned `UploadRequest`. `.default` instance by

/// default.

///

/// - Returns: The created `UploadRequest`.

```
open fun upload(multipartFormData: MultipartFormData,

                with request: URLRequestConvertible,

                usingThreshold           encodingMemoryThreshold:            UInt64            =
MultipartFormData.encodingMemoryThreshold,

                interceptor: RequestInterceptor? = nil,

                fileManager: FileManager = .default) -> UploadRequest {

    let           multipartUpload            =            MultipartUpload(isInBackgroundSession:
session.configuration.identifier != nil,

                                encodingMemoryThreshold: encodingMemoryThreshold,

                                request: request,

                                multipartFormData: multipartFormData)

    return upload(multipartUpload, interceptor: interceptor, fileManager: fileManager)

}
```

// MARK: - Internal API

```
// MARK: Uploadable
```

```
func upload(_ uploadable: UploadRequest.Uploadable,  
  
           with convertible: URLRequestConvertible,  
  
           interceptor: RequestInterceptor?,  
  
           fileManager: FileManager) -> UploadRequest {  
  
    let uploadable = Upload(request: convertible, uploadable: uploadable)  
  
    return upload(uploadable, interceptor: interceptor, fileManager: fileManager)  
  
}
```

```
func upload(_ upload: UploadConvertible, interceptor: RequestInterceptor?,  
fileManager: FileManager) -> UploadRequest {  
  
    let request = UploadRequest(convertible: upload,  
  
                                underlyingQueue: rootQueue,  
  
                                serializationQueue: serializationQueue,  
  
                                eventMonitor: eventMonitor,  
  
                                interceptor: interceptor,
```

```

        fileManager: fileManager,

        delegate: self)

perform(request)

return request

}

// MARK: Perform

/// Perform `Request`.

///

/// - Note: Called during retry.

///

/// - Parameter request: The `Request` to perform.

func perform(_ request: Request) {

    // Leaf types must come first, otherwise they will cast as their superclass.

```

```

switch request {

    case let r as UploadRequest: perform(r) // UploadRequest must come before
DataRequest due to subtype relationship.

    case let r as DataRequest: perform(r)

    case let r as DownloadRequest: perform(r)

    default: fatalError("Attempted to perform unsupported Request subclass: \(type(of:
request))")

}

}

```

```

func perform(_ request: DataRequest) {

    requestQueue.async {

        guard !request.isCancelled else { return }

        self.activeRequests.insert(request)

        self.performSetupOperations(for: request, convertible: request.convertible)

    }
}

```



```
}
```

```
func perform(_ request: UploadRequest) {

    requestQueue.async {

        guard !request.isCancelled else { return }

        self.activeRequests.insert(request)

        do {

            let uploadable = try request.upload.createUploadable()

            self.rootQueue.async { request.didCreateUploadable(uploadable) }

            self.performSetupOperations(for: request, convertible: request.convertible)

        } catch {

            self.rootQueue.async { request.didFailToCreateUploadable(with:
error.asAFError(or: .createUploadableFailed(error: error))) }

        }

    }

}
```

```
}
```

```
func perform(_ request: DownloadRequest) {
```

```
    requestQueue.async {
```

```
        guard !request.isCancelled else { return }
```

```
        self.activeRequests.insert(request)
```

```
        switch request.downloadable {
```

```
        case let .request(convertible):
```

```
            self.performSetupOperations(for: request, convertible: convertible)
```

```
        case let .resumeData(resumeData):
```

```
            self.rootQueue.async { self.didReceiveResumeData(resumeData, for: request) }
```

```
        }
```

```
    }
```

```
}
```

```

func performSetupOperations(for request: Request, convertible:
URLRequestConvertible) {

    let initialRequest: URLRequest

    do {

        initialRequest = try convertible.asURLRequest()

        try initialRequest.validate()

    } catch {

        rootQueue.async { request.didFailToCreateURLRequest(with:
error.asAFError(or: .createURLRequestFailed(error: error))) }

        return

    }

    rootQueue.async { request.didCreateInitialURLRequest(initialRequest) }

    guard !request.isCancelled else { return }

    guard let adapter = adapter(for: request) else {

```

```

rootQueue.async { self.didCreateURLRequest(initialRequest, for: request) }

return

}

adapter.adapt(initialRequest, for: self) { result in

do {

    let adaptedRequest = try result.get()

    try adaptedRequest.validate()

    self.rootQueue.async {

        request.didAdaptInitialRequest(initialRequest, to: adaptedRequest)

        self.didCreateURLRequest(adaptedRequest, for: request)

    }

} catch {

    self.rootQueue.async { request.didFailToAdaptURLRequest(initialRequest,
withError: .requestAdaptationFailed(error: error)) }

}

}

```

```
}
```

```
// MARK: - Task Handling
```

```
func didCreateURLRequest(_ urlRequest: URLRequest, for request: Request) {
```

```
    request.didCreateURLRequest(urlRequest)
```

```
    guard !request.isCancelled else { return }
```

```
    let task = request.task(for: urlRequest, using: session)
```

```
    requestTaskMap[request] = task
```

```
    request.didCreateTask(task)
```

```
    updateStatesForTask(task, request: request)
```

```
}
```

```
func didReceiveResumeData(_ data: Data, for request: DownloadRequest) {
```

```
guard !request.isCancelled else { return }
```

```
let task = request.task(forResumeData: data, using: session)
```

```
requestTaskMap[request] = task
```

```
request.didCreateTask(task)
```

```
updateStatesForTask(task, request: request)
```

```
}
```

```
func updateStatesForTask(_ task: URLSessionTask, request: Request) {
```

```
    request.withState { state in
```

```
        switch state {
```

```
            case .initialized, .finished:
```

```
                // Do nothing.
```

```
                break
```

```
            case .resumed:
```

```
                task.resume()
```

```

        rootQueue.async { request.didResumeTask(task) }

    case .suspended:

        task.suspend()

        rootQueue.async { request.didSuspendTask(task) }

    case .cancelled:

        // Resume to ensure metrics are gathered.

        task.resume()

        task.cancel()

        rootQueue.async { request.didCancelTask(task) }

    }

}

}

```

// MARK: - Adapters and Retrievers

```

func adapter(for request: Request) -> RequestAdapter? {

    if let requestInterceptor = request.interceptor, let sessionInterceptor = interceptor {

```

```

        return Interceptor(adapters: [requestInterceptor, sessionInterceptor])

    } else {

        return request.interceptor ?? interceptor

    }

}

func retrier(for request: Request) -> RequestRetrier? {

    if let requestInterceptor = request.interceptor, let sessionInterceptor = interceptor {

        return Interceptor(retriers: [requestInterceptor, sessionInterceptor])

    } else {

        return request.interceptor ?? interceptor

    }

}

```

// MARK: - Invalidation

```

func finishRequestsForDeinit() {

```



```

requestTaskMap.requests.forEach { request in

    rootQueue.async {

        request.finish(error: AFError.sessionDeinitialized)

    }

}

}

}

```

```

// MARK: - RequestDelegate

```

```

extension Session: RequestDelegate {

    public var sessionConfiguration: URLSessionConfiguration {

        return session.configuration

    }

    public var startImmediately: Bool { return startRequestsImmediately }

```

```

public func cleanup(after request: Request) {

    activeRequests.remove(request)

}

```

```

public func retryResult(for request: Request, dueTo error: NSError, completion:
@escaping (RetryResult) -> Void) {

    guard let retrier = retrier(for: request) else {

        rootQueue.async { completion(.doNotRetry) }

        return

    }

```

```

    retrier.retry(request, for: self, dueTo: error) { retryResult in

        self.rootQueue.async {

            guard let retryResultError = retryResult.error else { completion(retryResult);
return }

```

```

            let retryError = NSError.requestRetryFailed(retryError: retryResultError,
originalError: error)

            completion(.doNotRetryWithError(retryError))

```

```
    }  
  }  
}
```

```
public func retryRequest(_ request: Request, withDelay timeDelay: TimeInterval?) {  
  
    rootQueue.async {  
  
        let retry: () -> Void = {  
  
            guard !request.isCancelled else { return }  
  
            request.prepareForRetry()  
  
            self.perform(request)  
  
        }  
  
        if let retryDelay = timeDelay {  
  
            self.rootQueue.after(retryDelay) { retry() }  
  
        } else {  
  
            retry()  
  
        }  
    }  
}
```

```
    }  
  }  
}  
}
```

```
// MARK: - SessionStateProvider
```

```
extension Session: SessionStateProvider {  
  
    func request(for task: URLSessionTask) -> Request? {  
  
        dispatchPrecondition(condition: .onQueue(rootQueue))  
  
        return requestTaskMap[task]  
    }  
  
    func didGatherMetricsForTask(_ task: URLSessionTask) {  
  
        dispatchPrecondition(condition: .onQueue(rootQueue))  
  
    }  
}
```

```

let                                     didDisassociate                                     =
requestTaskMap.disassociateIfNecessaryAfterGatheringMetricsForTask(task)

```

```

    if didDisassociate {

        waitingCompletions[task]?()

        waitingCompletions[task] = nil

    }

}

```

```

func didCompleteTask(_ task: URLSessionTask, completion: @escaping () -> Void) {

    dispatchPrecondition(condition: .onQueue(rootQueue))

}

```

```

let                                     didDisassociate                                     =
requestTaskMap.disassociateIfNecessaryAfterCompletingTask(task)

```

```

    if didDisassociate {

        completion()

    } else {

}

```

```

        waitingCompletions[task] = completion
    }
}

```

```

func credential(for task: URLSessionTask, in protectionSpace: URLProtectionSpace) ->
URLCredential? {

```

```

    dispatchPrecondition(condition: .onQueue(rootQueue))

```

```

    return requestTaskMap[task]?.credential ??

```

```

        session.configuration.urlCredentialStorage?.defaultCredential(for:
protectionSpace)
    }

```

```

func cancelRequestsForSessionInvalidation(with error: Error?) {

```

```

    dispatchPrecondition(condition: .onQueue(rootQueue))

```

```

    requestTaskMap.requests.forEach { $0.finish(error:
AFError.sessionInvalidated(error: error)) }
}

```

```
}

//

// SessionDelegate.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,

// EXPRESS OR
```

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

import Foundation

/// Class which implements the various `URLSessionDelegate` methods to connect various
Alamofire features.

open class SessionDelegate: NSObject {

private let fileManager: FileManager


```
weak var stateProvider: SessionStateProvider?
```

```
var eventMonitor: EventMonitor?
```

```
/// Creates an instance from the given `FileManager`.
```

```
///
```

```
/// - Parameter fileManager: `FileManager` to use for underlying file management, such  
as moving downloaded files.
```

```
///          `.default` by default.
```

```
public init(fileManager: FileManager = .default) {
```

```
    self.fileManager = fileManager
```

```
}
```

```
/// Internal method to find and cast requests while maintaining some integrity checking.
```

```
///
```

```
/// - Parameters:
```

```
///   - task: The `URLSessionTask` for which to find the associated `Request`.
```

```
///   - type: The `Request` subclass type to cast any `Request` associate with `task`.
```

```
func request<R: Request>(for task: URLSessionTask, as type: R.Type) -> R? {
```

```

guard let provider = stateProvider else {

    assertionFailure("stateProvider is nil.")

    return nil

}

guard let request = provider.request(for: task) as? R else {

    fatalError("Returned Request is not of expected type: \(R.self).")

}

return request

}

}

/// Type which provides various `Session` state values.

protocol SessionStateProvider: AnyObject {

    var serverTrustManager: ServerTrustManager? { get }

    var redirectHandler: RedirectHandler? { get }

```

```

var cachedResponseHandler: CachedResponseHandler? { get }

func request(for task: URLSessionTask) -> Request?

func didGatherMetricsForTask(_ task: URLSessionTask)

func didCompleteTask(_ task: URLSessionTask, completion: @escaping () -> Void)

func credential(for task: URLSessionTask, in protectionSpace: URLProtectionSpace) ->
URLCredential?

func cancelRequestsForSessionInvalidation(with error: Error?)

}

// MARK: URLSessionDelegate

extension SessionDelegate: URLSessionDelegate {

    open func urlSession(_ session: URLSession, didBecomeInvalidWithError error: Error?)
    {

        eventMonitor?.urlSession(session, didBecomeInvalidWithError: error)

        stateProvider?.cancelRequestsForSessionInvalidation(with: error)
    }
}

```

```

    }

}

// MARK: URLSessionTaskDelegate

extension SessionDelegate: URLSessionTaskDelegate {

    /// Result of a `URLAuthenticationChallenge` evaluation.

    typealias ChallengeEvaluation = (disposition: URLSession.AuthChallengeDisposition,
    credential: URLCredential?, error: NSError?)

    open func urlSession(_ session: URLSession,

        task: URLSessionTask,

        didReceive challenge: URLAuthenticationChallenge,

        completionHandler: @escaping (URLSession.AuthChallengeDisposition,
    URLCredential?) -> Void) {

        eventMonitor?.urlSession(session, task: task, didReceive: challenge)

        let evaluation: ChallengeEvaluation

```

```

switch challenge.protectionSpace.authenticationMethod {

case NSURLAuthenticationMethodServerTrust:

    evaluation = attemptServerTrustAuthentication(with: challenge)

case
                                NSURLAuthenticationMethodHTTPBasic,
NSURLAuthenticationMethodHTTPDigest, NSURLAuthenticationMethodNTLM,

                                NSURLAuthenticationMethodNegotiate,
NSURLAuthenticationMethodClientCertificate:

    evaluation = attemptCredentialAuthentication(for: challenge, belongingTo: task)

default:

    evaluation = (.performDefaultHandling, nil, nil)

}

if let error = evaluation.error {

    stateProvider?.request(for: task)?.didFailTask(task, earlyWithError: error)

}

completionHandler(evaluation.disposition, evaluation.credential)

}

```

```

/// Evaluates the server trust `URLAuthenticationChallenge` received.

///

/// - Parameter challenge: The `URLAuthenticationChallenge`.

///

/// - Returns:      The `ChallengeEvaluation`.

func attemptServerTrustAuthentication(with challenge: URLAuthenticationChallenge)
-> ChallengeEvaluation {

    let host = challenge.protectionSpace.host

    guard challenge.protectionSpace.authenticationMethod ==
NSURLAuthenticationMethodServerTrust,

        let trust = challenge.protectionSpace.serverTrust

    else {

        return (.performDefaultHandling, nil, nil)

    }

    do {

```

```

guard let evaluator = try
stateProvider?.serverTrustManager?.serverTrustEvaluator(forHost: host) else {

    return (.performDefaultHandling, nil, nil)

}

try evaluator.evaluate(trust, forHost: host)

return (.useCredential, URLCredential(trust: trust), nil)

} catch {

    return (.cancelAuthenticationChallenge, nil,
error.asAFError(or: .serverTrustEvaluationFailed(reason: .customEvaluationFailed(error:
error))))

}

}

```

/// Evaluates the credential-based authentication `URLAuthenticationChallenge`
received for `task`.

///

/// - Parameters:

```
/// - challenge: The `URLAuthenticationChallenge`.
```

```
/// - task: The `URLSessionTask` which received the challenge.
```

```
///
```

```
/// - Returns: The `ChallengeEvaluation`.
```

```
func attemptCredentialAuthentication(for challenge: URLAuthenticationChallenge,  
  
                                   belongingTo task: URLSessionTask) -> ChallengeEvaluation {  
  
    guard challenge.previousFailureCount == 0 else {  
  
        return (.rejectProtectionSpace, nil, nil)  
  
    }  
  
    guard let credential = stateProvider?.credential(for: task, in:  
challenge.protectionSpace) else {  
  
        return (.performDefaultHandling, nil, nil)  
  
    }  
  
    return (.useCredential, credential, nil)  
  
}
```



```

open func urlSession(_ session: URLSession,

                        task: URLSessionTask,

                        didSendBodyData bytesSent: Int64,

                        totalBytesSent: Int64,

                        totalBytesExpectedToSend: Int64) {

    eventMonitor?.urlSession(session,

                                task: task,

                                didSendBodyData: bytesSent,

                                totalBytesSent: totalBytesSent,

                                totalBytesExpectedToSend: totalBytesExpectedToSend)

    stateProvider?.request(for: task)?.updateUploadProgress(totalBytesSent:
totalBytesSent,

                                                                totalBytesExpectedToSend:
totalBytesExpectedToSend)

}

```

```

open func urlSession(_ session: URLSession,

```

```

        task: URLSessionTask,

        needNewBodyStream completionHandler: @escaping (InputStream?) ->
Void) {

    eventMonitor?.urlSession(session, taskNeedsNewBodyStream: task)

    guard let request = request(for: task, as: UploadRequest.self) else {

        assertionFailure("needNewBodyStream did not find UploadRequest.")

        completionHandler(nil)

        return

    }

    completionHandler(request.inputStream())

}

open func urlSession(_ session: URLSession,

        task: URLSessionTask,

        willPerformHTTPRedirection response: HTTPURLResponse,

        newRequest request: URLRequest,

```

```

        completionHandler: @escaping (URLRequest?) -> Void) {

    eventMonitor?.urlSession(session, task: task, willPerformHTTPRedirection:
response, newRequest: request)

    if let redirectHandler = stateProvider?.request(for: task)?.redirectHandler ??
stateProvider?.redirectHandler {

        redirectHandler.task(task, willBeRedirectedTo: request, for: response, completion:
completionHandler)

    } else {

        completionHandler(request)

    }

}

open func urlSession(_ session: URLSession, task: URLSessionTask,
didFinishCollecting metrics: URLSessionTaskMetrics) {

    eventMonitor?.urlSession(session, task: task, didFinishCollecting: metrics)

    stateProvider?.request(for: task)?.didGatherMetrics(metrics)

```

```

stateProvider?.didGatherMetricsForTask(task)

}

```

```

open func urlSession(_ session: URLSession, task: URLSessionTask,
didCompleteWithError error: Error?) {

```

```

    eventMonitor?.urlSession(session, task: task, didCompleteWithError: error)

```

```

    let request = stateProvider?.request(for: task)

```

```

    stateProvider?.didCompleteTask(task) {

```

```

        request?.didCompleteTask(task, with: error.map
{ $0.asAFError(or: .sessionTaskFailed(error: $0)) })

```

```

    }

```

```

}

```

```

@available(macOS 10.13, iOS 11.0, tvOS 11.0, watchOS 4.0, *)

```

```

open func urlSession(_ session: URLSession, taskIsWaitingForConnectivity task:
URLSessionTask) {

```

```

    eventMonitor?.urlSession(session, taskIsWaitingForConnectivity: task)

```

```

    }

}

// MARK: URLSessionDataDelegate

extension SessionDelegate: URLSessionDataDelegate {

    open func urlSession(_ session: URLSession, dataTask: URLSessionDataTask,
didReceive data: Data) {

        eventMonitor?.urlSession(session, dataTask: dataTask, didReceive: data)

        guard let request = request(for: dataTask, as: DataRequest.self) else {

            assertionFailure("dataTask did not find DataRequest.")

            return

        }

        request.didReceive(data: data)

    }

```

```

open func urlSession(_ session: URLSession,

                        dataTask: URLSessionDataTask,

                        willCacheResponse proposedResponse: CachedURLResponse,

                        completionHandler: @escaping (CachedURLResponse?) -> Void) {

    eventMonitor?.urlSession(session,   dataTask:   dataTask,   willCacheResponse:
proposedResponse)

    if let handler = stateProvider?.request(for: dataTask)?.cachedResponseHandler ??
stateProvider?.cachedResponseHandler {

        handler.dataTask(dataTask, willCacheResponse: proposedResponse, completion:
completionHandler)

    } else {

        completionHandler(proposedResponse)

    }

}

}

}

}

// MARK: URLSessionDownloadDelegate

```

```

extension SessionDelegate: URLSessionDownloadDelegate {

    open func urlSession(_ session: URLSession,

                        downloadTask: URLSessionDownloadTask,

                        didResumeAtOffset fileOffset: Int64,

                        expectedTotalBytes: Int64) {

        eventMonitor?.urlSession(session,

                                downloadTask: downloadTask,

                                didResumeAtOffset: fileOffset,

                                expectedTotalBytes: expectedTotalBytes)

        guard let downloadRequest = request(for: downloadTask, as: DownloadRequest.self)
    else {

        assertionFailure("downloadTask did not find DownloadRequest.")

        return

    }

    downloadRequest.updateDownloadProgress(bytesWritten: fileOffset,

                                          totalBytesExpectedToWrite: expectedTotalBytes)

}

```

```

open func urlSession(_ session: URLSession,

                        downloadTask: URLSessionDownloadTask,

                        didWriteData bytesWritten: Int64,

                        totalBytesWritten: Int64,

                        totalBytesExpectedToWrite: Int64) {

    eventMonitor?.urlSession(session,

                                downloadTask: downloadTask,

                                didWriteData: bytesWritten,

                                totalBytesWritten: totalBytesWritten,

                                totalBytesExpectedToWrite: totalBytesExpectedToWrite)

    guard let downloadRequest = request(for: downloadTask, as: DownloadRequest.self)
else {

        assertionFailure("downloadTask did not find DownloadRequest.")

        return

    }

    downloadRequest.updateDownloadProgress(bytesWritten: bytesWritten,

```



```

        totalBytesExpectedToWrite: totalBytesExpectedToWrite)

    }

    open func URLSession(_ session: URLSession, downloadTask:
    URLSessionDownloadTask, didFinishDownloadingTo location: URL) {

        eventMonitor?.urlSession(session, downloadTask: downloadTask,
        didFinishDownloadingTo: location)

        guard let request = request(for: downloadTask, as: DownloadRequest.self) else {

            assertionFailure("downloadTask did not find DownloadRequest.")

            return

        }

        guard let response = request.response else {

            fatalError("URLSessionDownloadTask finished downloading with no response.")

        }

        let (destination, options) = (request.destination)(location, response)

```

```

eventMonitor?.request(request, didCreateDestinationURL: destination)

do {

    if options.contains(.removePreviousFile),    fileManager.fileExists(atPath:
destination.path) {

        try fileManager.removeItem(at: destination)

    }

    if options.contains(.createIntermediateDirectories) {

        let directory = destination.deletingLastPathComponent()

        try fileManager.createDirectory(at: directory, withIntermediateDirectories: true)

    }

    try fileManager.moveItem(at: location, to: destination)

    request.didFinishDownloading(using: downloadTask, with: .success(destination))

} catch {

```

```
        request.didFinishDownloading(using: downloadTask,  
with: .failure(.downloadedFileMoveFailed(error: error, source: location, destination:  
destination)))
```

```
    }
```

```
}
```

```
}
```

```
import UIKit
```

```
private let reuseIdentifier = "SettingsCell"
```

```
class SettingViewController: UIViewController {
```

```
    // MARK: - Properties
```

```
    var tableView: UITableView!
```

```
    var userInfoHeader: UserInfoHeader!
```

```
    // MARK: - Init
```

```

override func viewDidLoad() {

    super.viewDidLoad()

    navigationController?.navigationBar.prefersLargeTitles = true

    if #available(iOS 13.0, *) {

        let navBarAppearance = UINavigationBarAppearance()

        navBarAppearance.configureWithOpaqueBackground()

        navBarAppearance.titleTextAttributes = [.foregroundColor: UIColor.white]

        navBarAppearance.largeTitleTextAttributes = [.foregroundColor: UIColor.black]

        navBarAppearance.backgroundColor = UIColor.init(hexString: "#a1edab")

        navigationController?.navigationBar.standardAppearance = navBarAppearance

        navigationController?.navigationBar.scrollEdgeAppearance = navBarAppearance

    }

    configureUI()

}

```

// MARK: - Helper Functions

```
func configureTableView() {  
  
    tableView = UITableView()  
  
    tableView.delegate = self  
  
    tableView.dataSource = self  
  
    tableView.rowHeight = 60  
  
    tableView.contentInset = UIEdgeInsets(top: 5,left: 0,bottom: 0,right: 0);  
  
  
    tableView.register(SettingsCell.self, forCellReuseIdentifier: reuseIdentifier)  
  
    view.addSubview(tableView)  
  
    tableView.frame = view.frame  
  
  
  
    let frame = CGRect(x: 0, y: 88, width: view.frame.width, height: 100)  
  
    userInfoHeader = UserInfoHeader(frame: frame)  
  
    tableView.tableHeaderView = userInfoHeader  
  
    tableView.tableFooterView = UIView()  
  
}
```

```

func configureUI() {

    configureTableView()


    //    navigationController?.navigationBar.isTranslucent = false

    //    navigationController?.navigationBar.barStyle = .black

    //    navigationController?.navigationBar.barTintColor = UIColor(red: 55/255, green:
120/255, blue: 250/255, alpha: 1)

    navigationItem.title = "Settings"

}

}

extension SettingViewController: UITableViewDelegate, UITableViewDataSource {

func numberOfSections(in tableView: UITableView) -> Int {

    return SettingsSection.allCases.count

```

```

    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) ->
    Int {

        guard let section = SettingsSection(rawValue: section) else { return 0 }

        switch section {

            case .Graph: return GraphOptions.allCases.count

            case .Account: return AccountOptions.allCases.count

        }

    }
}

```

```

    func tableView(_ tableView: UITableView, viewForHeaderInSection section: Int) ->
    UIView? {

        let view = UIView()

        view.backgroundColor = UIColor.gray
    }
}

```

```

        print("Section is \(section)")

        let title = UILabel()

        title.font = UIFont.boldSystemFont(ofSize: 16)

        title.textColor = .white

        title.text = SettingsSection(rawValue: section)?.description

        view.addSubview(title)

        title.translatesAutoresizingMaskIntoConstraints = false

        title.centerYAnchor.constraint(equalTo: view.centerYAnchor).isActive = true

        title.leftAnchor.constraint(equalTo: view.leftAnchor, constant: 16).isActive = true

        return view
    }

    func tableView(_ tableView: UITableView, heightForHeaderInSection section: Int) ->
    CGFloat {

        return 40
    }

```



```

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {

    let cell = tableView.dequeueReusableCell(withIdentifier: reuseIdentifier, for:
indexPath) as! SettingsCell

    guard let section = SettingsSection(rawValue: indexPath.section) else { return
UITableViewCell() }

    switch section {

    case .Graph:

        let communications = GraphOptions(rawValue: indexPath.row)

        cell.textLabel?.text = communications?.description

    case .Account:

        let social = AccountOptions(rawValue: indexPath.row)

        cell.textLabel?.text = social?.description

    }

    return cell
}

```

```

    }

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {

        guard let section = SettingsSection(rawValue: indexPath.section) else { return }

        switch section {

        case .Graph:

            print(GraphOptions(rawValue: indexPath.row)?.description)

        case .Account:

            print(AccountOptions(rawValue: indexPath.row)?.description)

            let alert = UIAlertController(title: "Are you sure you want to logout?", message: "",
            preferredStyle: UIAlertController.Style.alert)

            alert.addAction(UIAlertAction(title: "Cancel", style:
            UIAlertAction.Style.destructive, handler: { _ in

                //Cancel Action

            })))

            alert.addAction(UIAlertAction(title: "Logout", style:
            UIAlertAction.Style.default, handler: {(_ : UIAlertAction!) in

                let view = ATCClassicLandingScreenViewController(nibName: nil, bundle: nil)

```

```

        self.hidesBottomBarWhenPushed = true

        self.navigationController?.pushViewController(view,animated: true)

    )))

    self.present(alert, animated: true, completion: nil)

}

}

}

// Copyright (c) 2016-present, Facebook, Inc. All rights reserved.

//

// You are hereby granted a non-exclusive, worldwide, royalty-free license to use,

// copy, modify, and distribute this software in source code or binary form for use

// in connection with the web services and APIs provided by Facebook.

```

//

// As with any software that integrates with the Facebook platform, your use of

// this software is subject to the Facebook Developer Principles and Policies

// [<http://developers.facebook.com/policy/>]. This copyright notice shall be

// included in all copies or substantial portions of the software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS

// FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR

// COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER

// IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN

// CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

import Foundation

```
//-----
```

```
// MARK: - SDKSettings
```

```
//-----
```

```
/**
```

Provides access to settings and configuration used by the entire SDK.

```
*/
```

```
public extension Settings {
```

```
//-----
```

```
// MARK: - SDKSettings + Logging Behavior
```

```
//-----
```

```
/**
```

Current logging behaviors of Facebook SDK.

The default enabled behavior is `.DeveloperErrors`` only.

```
*/
```

```

static var loggingBehaviors: Set<LoggingBehavior> {

    get {

        return Set(Settings.__loggingBehaviors.map { LoggingBehavior(rawValue: $0) })

    }

    set {

        Settings.__loggingBehaviors = Set(newValue.map { $0.rawValue })

    }

}

//

// SettingsCell.swift

// SettingsTemplate

//

// Created by Stephen Dowless on 2/10/19.

// Copyright © 2019 Stephan Dowless. All rights reserved.

//

```

```
import UIKit
```

```
class SettingsCell: UITableViewCell {
```

```
    // MARK: - Properties
```

```
    // MARK: - Init
```

```
    override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
```

```
        super.init(style: style, reuseIdentifier: reuseIdentifier)
```

```
    }
```

```
    required init?(coder aDecoder: NSCoder) {
```

```
        fatalError("init(coder:) has not been implemented")
```

```
    }
```

```
}
```

```
//

// SettingsSections.swift

// SettingsTemplate

//

// Created by user on 1/5/2563 BE.

// Copyright © 2563 Stephan Dowless. All rights reserved.

//

enum SettingsSection: Int, CaseIterable, CustomStringConvertible{

    case Graph

    case Account

    var description: String{

        switch self{

            case .Graph: return "Mornitor"

            case .Account: return "Account"

        }

    }

}
```



```

    }

}

enum GraphOptions: Int, CaseIterable, CustomStringConvertible{

    case delete

    var description: String{

        switch self{

            case .delete: return "Reset Mornitor"

        }

    }

}

enum AccountOptions: Int, CaseIterable, CustomStringConvertible{

    case logout

    var description: String{

```

```
switch self{  
  
    case .logout: return "Log Out"  
  
    }  
  
}  
  
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The 'SliderCharacteristicCell' displays characteristics with a continuous range of options.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
/**
```

A `CharacteristicCell` subclass that contains a slider.

Used for numeric characteristics that have a continuous range of options.

```
*/
```

```
class SliderCharacteristicCell: CharacteristicCell {
```

```
    // MARK: Properties
```

```
    @IBOutlet weak var valueSlider: UISlider!
```

```
    override var characteristic: HMCharacteristic! {
```

```
        didSet {
```

```
            valueSlider.alpha = enabled ? 1.0 : CharacteristicCell.DisabledAlpha
```

```
            valueSlider.isUserInteractionEnabled = enabled
```

```
        }
```

```
        willSet(newCharacteristic) {
```

```
// These are sane defaults in case the max and min are not set.
```

```
valueSlider.minimumValue = newCharacteristic.metadata?.minimumValue as?  
Float ?? 0.0
```

```
valueSlider.maximumValue = newCharacteristic.metadata?.maximumValue as?  
Float ?? 100.0
```

```
}
```

```
}
```

```
/// If notify is false, sets the valueSlider's represented value.
```

```
override fun setValue(_ newValue: CellValueType?, notify: Bool) {
```

```
    super.setValue(newValue, notify: notify)
```

```
    if let newValue = newValue as? NSNumber , !notify {
```

```
        valueSlider.value = newValue.floatValue
```

```
    }
```

```
}
```

```
/**
```

```
    Restricts a value to the step value provided in the cell's
```

characteristic's metadata.

- parameter sliderValue: The provided value.

- returns: The value adjusted to align with a step value.

*/

```
func roundedValueForSliderValue(_ value: Float) -> Float {
```

```
    if let metadata = characteristic.metadata,
```

```
        let stepValue = metadata.stepValue as? Float
```

```
        , stepValue > 0 {
```

```
            let newStep = roundf(value / stepValue)
```

```
            let stepped = newStep * stepValue
```

```
            return stepped
```

```
        }
```

```
    return value
```

```
}
```

```

// Sliders don't update immediately, because sliders generate many updates.

override class var updatesImmediately: Bool {

    return false

}

/**

Responds to a slider change and sets the cell's value.

- parameter slider: The slider that changed.

*/

@IBAction func didChangeSliderValue(_ slider: UISlider) {

    let value = roundedValueForSliderValue(slider.value)

    setValue(value as NSNumber?, notify: true)

}

}

```

```
//

// SquareShapeRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class SquareShapeRenderer : NSObject, IShapeRenderer

{

    open func renderShape(

        context: CGContext,
```

```

    dataSet: IScatterChartDataSet,

    viewPortHandler: ViewPortHandler,

    point: CGPoint,

    color: NSUIColor)

{

    let shapeSize = dataSet.scatterShapeSize

    let shapeHalf = shapeSize / 2.0

    let shapeHoleSizeHalf = dataSet.scatterShapeHoleRadius

    let shapeHoleSize = shapeHoleSizeHalf * 2.0

    let shapeHoleColor = dataSet.scatterShapeHoleColor

    let shapeStrokeSize = (shapeSize - shapeHoleSize) / 2.0

    let shapeStrokeSizeHalf = shapeStrokeSize / 2.0


    if shapeHoleSize > 0.0

    {

        context.setStrokeColor(color.cgColor)

        context.setLineWidth(shapeStrokeSize)

```



```

var rect = CGRect()

rect.origin.x = point.x - shapeHoleSizeHalf - shapeStrokeSizeHalf

rect.origin.y = point.y - shapeHoleSizeHalf - shapeStrokeSizeHalf

rect.size.width = shapeHoleSize + shapeStrokeSize

rect.size.height = shapeHoleSize + shapeStrokeSize

context.stroke(rect)


if let shapeHoleColor = shapeHoleColor
{
    context.setFillColor(shapeHoleColor.cgColor)

    rect.origin.x = point.x - shapeHoleSizeHalf

    rect.origin.y = point.y - shapeHoleSizeHalf

    rect.size.width = shapeHoleSize

    rect.size.height = shapeHoleSize

    context.fill(rect)
}
}

```

```

else

{

    context.setFillColor(color.cgColor)

    var rect = CGRect()

    rect.origin.x = point.x - shapeHalf

    rect.origin.y = point.y - shapeHalf

    rect.size.width = shapeSize

    rect.size.height = shapeSize

    context.fill(rect)

}

}

}

//

//  MIT License

//

//  Copyright (c) 2017 Robert-Hein Hooijmans <rh.hooijmans@gmail.com>

//

```

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
KIND, EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM,

```
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER  
DEALINGS IN
```

```
// THE SOFTWARE.
```

```
//
```

```
#if os(OSX)
```

```
import AppKit
```

```
#else
```

```
import UIKit
```

```
#endif
```

```
public extension View {
```

```
    @discardableResult
```

```
    func stack(_ views: [View], axis: ConstraintAxis = .vertical, width: CGFloat? = nil,  
height: CGFloat? = nil, spacing: CGFloat = 0) -> Constraints {
```

```
        translatesAutoresizingMaskIntoConstraints = false
```

```

var offset: CGFloat = 0

var previous: View?

var constraints: Constraints = []

for view in views {

    view.translatesAutoresizingMaskIntoConstraints = false

    addSubview(view)

    switch axis {

    case .vertical:

        constraints.append(view.top(to: previous ?? self, previous?.bottomAnchor ??
topAnchor, offset: offset))

        constraints.append(view.leftToSuperview())

        constraints.append(view.rightToSuperview())

        if let lastView = views.last, view == lastView {

            constraints.append(view.bottomToSuperview())

```

```

    }

    case .horizontal:

        constraints.append(view.topToSuperview())

        constraints.append(view.bottomToSuperview())

        constraints.append(view.left(to: previous ?? self, previous?.rightAnchor ??
leftAnchor, offset: offset))

        if let lastView = views.last, view == lastView {

            constraints.append(view.rightToSuperview())

        }

    @unknown default:

        fatalError()

}

if let width = width {

    constraints.append(view.width(width))

}

```

```

        if let height = height {

            constraints.append(view.height(height))

        }

        offset = spacing

        previous = view

    }

    return constraints

}

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `SwitchCharacteristicCell` displays Boolean characteristics.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
/**
```

```
    A `CharacteristicCell` subclass that contains a single switch.
```

```
    Used for Boolean characteristics.
```

```
*/
```

```
class SwitchCharacteristicCell: CharacteristicCell {
```

```
    // MARK: Properties
```

```
    @IBOutlet weak var valueSwitch: UISwitch!
```

```
    override var characteristic: HMCharacteristic! {
```

```
        didSet {
```

```
            valueSwitch.alpha = enabled ? 1.0 : CharacteristicCell.DisabledAlpha
```



```

        valueSwitch.isUserInteractionEnabled = enabled
    }
}

/// If notify is false, sets the switch to the value.

override func setValue(_ newValue: CellValueType?, notify: Bool) {

    super.setValue(newValue, notify: notify)

    if !notify {

        if let boolValue = newValue as? Bool {

            valueSwitch.setOn(boolValue, animated: true)

        }

    }

}

/**

```

Responds to the switch updating and sets the

value to the switch's value.

- parameter valueSwitch: The switch that updated.

*/

```
@IBAction func didChangeSwitchValue(_ valueSwitch: UISwitch) {
```

```
    setValue(valueSwitch.isOn as NSNumber?, notify: true)
```

```
}
```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `TextCharacteristicCell` represents text-input characteristics.

*/

```
import UIKit
```

```
import HomeKit
```

```
/**
```

```
    A `CharacteristicCell` subclass that contains a text field.
```

```
    Used for text-input characteristics.
```

```
*/
```

```
class TextCharacteristicCell: CharacteristicCell, UITextFieldDelegate {
```

```
    // MARK: Properties
```

```
    @IBOutlet weak var textField: UITextField!
```

```
    override var characteristic: HMCharacteristic! {
```

```
        didSet {
```

```
            textField.alpha = enabled ? 1.0 : CharacteristicCell.DisabledAlpha
```

```
            textField.isUserInteractionEnabled = enabled
```

```
        }
```

```
}
```

```
/// If notify is false, sets the text field's text from the value.
```

```
override func setValue(_ newValue: CellValueType?, notify: Bool) {
```

```
    super.setValue(newValue, notify: notify)
```

```
    if !notify {
```

```
        if let newStringValue = newValue as? String {
```

```
            textField.text = newStringValue
```

```
        }
```

```
    }
```

```
}
```

```
/// Dismiss the keyboard when "Go" is clicked
```

```
func textFieldShouldReturn(_ textField: UITextField) -> Bool {
```

```
    textField.resignFirstResponder()
```

```
    return true
```

```
}
```

```

    /// Sets the value of the characteristic when editing is complete.

    func textFieldDidEndEditing(_ textField: UITextField) {

        setValue(textField.text as NSString?, notify: true)

    }

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `TimeConditionViewController` allows the user to create a new time condition.

```

*/

```

```

import UIKit

```

```

import HomeKit

```

/// Represents a section in the `TimeConditionViewController`.

enum TimeConditionTableViewSection: Int {

/**

This section contains the segmented control to

choose a time condition type.

*/

case timeOrSun

/**

This section contains cells to allow the selection

of 'before', 'after', or 'at'. 'At' is only available

when the exact time is specified.

*/

case beforeOrAfter

/**

If the condition type is exact time, this section will

only have one cell, the date picker cell.

If the condition type is relative to a solar event,

this section will have two cells, one for 'sunrise' and

one for 'sunset.

*/

case value

static let count = 3

}

/**

Represents the type of time condition.

The condition can be an exact time, or relative to a solar event.

*/

enum TimeConditionType: Int {

case time, sun

}

/**

Represents the type of solar event.

This can be sunrise or sunset.

*/

enum TimeConditionSunState: Int {

case sunrise, sunset

}

/**

Represents the condition order.

Conditions can be before, after, or exactly at a given time.

*/


```

enum TimeConditionOrder: Int {

    case before, after, at

}

/// A view controller that facilitates the creation of time conditions for triggers.

class TimeConditionViewController: HMCatalogViewController {

    // MARK: Types

    struct Identifiers {

        static let selectionCell = "SelectionCell"

        static let timePickerCell = "TimePickerCell"

        static let segmentedTimeCell = "SegmentedTimeCell"

    }

    static let timeOrSunTitles = [

        NSLocalizedString("Relative to time", comment: "Relative to time"),

        NSLocalizedString("Relative to sun", comment: "Relative to sun")
    ]

```

```
]
```

```
static let beforeOrAfterTitles = [  
  
    NSLocalizedString("Before", comment: "Before"),  
  
    NSLocalizedString("After", comment: "After"),  
  
    NSLocalizedString("At", comment: "At")  
]
```

```
static let sunriseSunsetTitles = [  
  
    NSLocalizedString("Sunrise", comment: "Sunrise"),  
  
    NSLocalizedString("Sunset", comment: "Sunset")  
]
```

```
// MARK: Properties
```

```
private var timeType: TimeConditionType = .time
```

```
private var order: TimeConditionOrder = .before
```

```
private var sunState: TimeConditionSunState = .sunrise
```

```
private var datePicker: UIDatePicker?
```

```
var triggerCreator: EventTriggerCreator?
```

```
// MARK: View Methods
```

```
/// Configures the table view.
```

```
override func viewDidLoad() {
```

```
    super.viewDidLoad()
```

```
    tableView.rowHeight = UITableView.automaticDimension
```

```
    tableView.estimatedRowHeight = 44.0
```

```
}
```

```
// MARK: Table View Methods
```

```

/// - returns: The number of `TimeConditionTableViewSection`s.

override func numberOfSections(in tableView: UITableView) -> Int {

    return TimeConditionTableViewSection.count

}

/**

- returns: The number rows based on the `TimeConditionTableViewSection`

and the `timeType`.

*/

override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {

    switch TimeConditionTableViewSection(rawValue: section) {

        case .timeOrSun?:

            return 1

        case .beforeOrAfter?:

            // If we're choosing an exact time, we add the 'At' row.

            return (timeType == .time) ? 3 : 2

```

```
case .value?:
```

```
    // Date picker cell or sunrise/sunset selection cells
```

```
    return (timeType == .time) ? 1 : 2
```

```
case nil:
```

```
    fatalError("Unexpected `TimeConditionTableViewSection` raw value.")
```

```
}
```

```
}
```

```
/// Switches based on the section to generate a cell.
```

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
```

```
    switch TimeConditionTableViewSection(rawValue: indexPath.section) {
```

```
        case .timeOrSun?:
```

```
            return self.tableView(tableView, segmentedCellForRowAtIndexPath: indexPath)
```

```
        case .beforeOrAfter?:
```

```

        return self.tableView(tableView, selectionCellForRowAtIndexPath: indexPath)

    case .value?:

        switch timeType {

        case .time:

            return self.tableView(tableView, datePickerCellForRowAtIndexPath:
indexPath)

        case .sun:

            return self.tableView(tableView, selectionCellForRowAtIndexPath:
indexPath)

        }

    case nil:

        fatalError("Unexpected `TimeConditionTableViewSection` raw value.")

    }

}

```

/// - returns: A localized string describing the section.

```
override func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
```

```
    switch TimeConditionTableViewSection(rawValue: section) {
```

```
        case .timeOrSun?:
```

```
            return NSLocalizedString("Condition Type", comment: "Condition Type")
```

```
        case .beforeOrAfter?:
```

```
            return nil
```

```
        case .value?:
```

```
            if timeType == .time {
```

```
                return NSLocalizedString("Time", comment: "Time")
```

```
            }
```

```
        else {
```

```
            return NSLocalizedString("Event", comment: "Event")
```

```
        }
```

```
        case nil:
```

```

        fatalError("Unexpected `TimeConditionTableViewSection` raw value.")
    }
}

/// - returns: A localized description for condition type section; `nil` otherwise.

override func tableView(_ tableView: UITableView, titleForFooterInSection section:
Int) -> String? {

    switch TimeConditionTableViewSection(rawValue: section) {

        case .timeOrSun?:

            return NSLocalizedString("Time conditions can relate to specific times or
special events, like sunrise and sunset.", comment: "Condition Type Description")


        case .beforeOrAfter?:

            return nil


        case .value?:

            return nil

```



```

        case nil:

            fatalError("Unexpected `TimeConditionTableViewSection` raw value.")

        }

    }

    /// Updates internal values based on row selection.

    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {

        let cell = tableView.cellForRow(at: indexPath)!

        if cell.selectionStyle == .none {

            return

        }

        tableView.deselectRow(at: indexPath, animated: true)

        switch TimeConditionTableViewSection(rawValue: indexPath.section) {

            case .timeOrSun?:

                timeType = TimeConditionType(rawValue: indexPath.row)!

```

```
reloadDynamicSections()
```

```
return
```

```
case .beforeOrAfter?:
```

```
order = TimeConditionOrder(rawValue: indexPath.row)!
```

```
tableView.reloadSections(IndexSet(integer: indexPath.section), with: .automatic)
```

```
case .value?:
```

```
if timeType == .sun {
```

```
sunState = TimeConditionSunState(rawValue: indexPath.row)!
```

```
}
```

```
tableView.reloadSections(IndexSet(integer: indexPath.section), with: .automatic)
```

```
case nil:
```

```
fatalError("Unexpected `TimeConditionTableViewSection` raw value.")
```

```
}
```

```
}
```

```

// MARK: Helper Methods

/**

Generates a selection cell based on the section.

Ordering and sun-state sections have selections.

*/

private func tableView(_ tableView: UITableView, selectionCellForRowAtIndexPath
indexPath: IndexPath) -> UITableViewCell {

    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.selectionCell,
for: indexPath)

    switch TimeConditionTableViewSection(rawValue: indexPath.section) {

        case .beforeOrAfter?:

            cell.textLabel?.text =
TimeConditionViewController.beforeOrAfterTitles[indexPath.row]

            cell.accessoryType = (order.rawValue == indexPath.row) ? .checkmark : .none


        case .value?:

            if timeType == .sun {

```

```

        cell.textLabel?.text =
TimeConditionViewController.sunriseSunsetTitles[indexPath.row]

        cell.accessoryType = (sunState.rawValue ==
indexPath.row) ? .checkmark : .none

    }

case nil:

    fatalError("Unexpected `TimeConditionTableViewSection` raw value.")

default:

    break

}

return cell

}

```

/// Generates a date picker cell and sets the internal date picker when created.

```

private func tableView(_ tableView: UITableView, datePickerCellForRowAtIndexPath
indexPath: IndexPath) -> UITableViewCell {

```

```
let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.timePickerCell,
for: indexPath) as! TimePickerCell
```

```
// Save the date picker so we can get the result later.
```

```
datePicker = cell.datePicker
```

```
return cell
```

```
}
```

```
/// Generates a segmented cell and sets its target when created.
```

```
private func tableView(_ tableView: UITableView, segmentedCellForRowAtIndexPath
indexPath: IndexPath) -> UITableViewCell {
```

```
let cell = tableView.dequeueReusableCell(withIdentifier:
Identifiers.segmentedTimeCell, for: indexPath) as! SegmentedTimeCell
```

```
cell.segmentedControl.selectedSegmentIndex = timeType.rawValue
```

```
cell.segmentedControl.removeTarget(nil, action: nil, for: .allEvents)
```

```
cell.segmentedControl.addTarget(self, action:
#selector(self.segmentedControlDidChange(_)), for: .valueChanged)
```

```
return cell
```

```
}
```

```
/// Creates date components from the date picker's date.
```

```
var dateComponents: DateComponents? {
```

```
    guard let datePicker = datePicker else { return nil }
```

```
    let flags: Set<Calendar.Component> = [.hour, .minute]
```

```
    return Calendar.current.dateComponents(flags, from: datePicker.date)
```

```
}
```

```
/**
```

```
    Updates the time type and reloads dynamic sections.
```

```
    - parameter segmentedControl: The segmented control that changed.
```

```
*/
```

```
@objc func segmentedControlDidChange(_ segmentedControl: UISegmentedControl)
```

```
{
```

```
    if let segmentedControlType = TimeConditionType(rawValue:
        segmentedControl.selectedSegmentIndex) {
```

```
        timeType = segmentedControlType
```

```
}
```

```

        reloadDataDynamicSections()

    }

    /// Reloads the BeforeOrAfter and Value section.

    private func reloadDataDynamicSections() {

        if timeType == .sun && order == .at {

            order = .before

        }

        let          reloadIndexSet          =          IndexSet(integersIn:
Range(NSMakeRange(TimeConditionTableViewSection.beforeOrAfter.rawValue, 2)) ??
0..<0)

        tableView.reloadDataSections(reloadIndexSet, with: .automatic)

    }

    // MARK: IBAction Methods

    /**

    Generates a predicate based on the stored values, adds

```

the condition to the trigger, then dismisses the view.

```
*/
```

```
@IBAction func saveAndDismiss(_ sender: UIBarButtonItem) {
```

```
    var predicate: NSPredicate?
```

```
    switch timeType {
```

```
        case .time:
```

```
            switch order {
```

```
                case .before:
```

```
                    predicate
```

```
=
```

```
HMEventTrigger.predicateForEvaluatingTrigger(occurringBefore: dateComponents!)
```

```
                case .after:
```

```
                    predicate
```

```
=
```

```
HMEventTrigger.predicateForEvaluatingTrigger(occurringAfter: dateComponents!)
```

```
                case .at:
```

```
                    predicate = HMEventTrigger.predicateForEvaluatingTrigger(occurringOn:
```

```
dateComponents!)
```

```
            }
```



```

        case .sun:

            let    significantEventString    =    (sunState    ==    .sunrise)    ?
HMSignificantEvent.sunrise : HMSignificantEvent.sunset

            switch order {

                case .before:

                    predicate                                =
HMEventTrigger.predicateForEvaluatingTrigger(occurringBefore:
significantEventString.rawValue, applyingOffset: nil)

                case .after:

                    predicate                                =
HMEventTrigger.predicateForEvaluatingTrigger(occurringAfter:
significantEventString.rawValue, applyingOffset: nil)

                case .at:

                    // Significant events must be specified 'before' or 'after'.

                    break

            }

    }

```

```

    if let predicate = predicate {

        triggerCreator?.addCondition(predicate)

    }

    self.dismiss(animated: true, completion: nil)

}

/// Cancels the creation of the conditions and exits.

@IBAction func dismiss(_ sender: UIBarButtonItem) {

    self.dismiss(animated: true, completion: nil)

}

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The 'TimePickerCell' has a date picker, used for selecting a specific time of day.

*/

import UIKit

/// A `UITableViewCell` subclass with a `UIDatePicker`, used for selecting a specific time of day.

class TimePickerCell: UITableViewCell {

// MARK: Properties

@IBOutlet weak var datePicker: UIDatePicker!

}/*

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `TimerTriggerCreator` creates Timer triggers.

*/

```
import HomeKit
```

```
/**
```

```
    A `TriggerCreator` subclass which allows for the creation  
    of timer triggers.
```

```
*/
```

```
class TimerTriggerCreator: TriggerCreator {
```

```
    static let RecurrenceComponents: [Calendar.Component] = [
```

```
        .hour,
```

```
        .day,
```

```
        .weekOfYear
```

```
    ]
```

```
    // MARK: Properties
```

```
    var timerTrigger: HMTimerTrigger? {
```

```
        return trigger as? HMTimerTrigger
```

```
}
```

```
var selectedRecurrenceIndex: Int? = nil
```

```
var rawFireDate = Date()
```

```
var fireDate: Date {
```

```
    let flags: Set<Calendar.Component> = [.year, .weekday, .month, .day, .hour, .minute]
```

```
    let dateComponents = Calendar.current.dateComponents(flags, from: self.rawFireDate)
```

```
    let probableDate = Calendar.current.date(from: dateComponents)
```

```
    return probableDate ?? rawFireDate
```

```
}
```

```
// MARK: Trigger Creator Methods
```

```
/// Configures raw fire date and selected recurrence index.
```

```
required init(trigger: HMTrigger?, home: HMHome) {
```

```
    super.init(trigger: trigger, home: home)
```

```

    if let timerTrigger = timerTrigger {

        rawFireDate = timerTrigger.fireDate

        selectedRecurrenceIndex =
recurrenceIndexFromDateComponents(timerTrigger.recurrence)

    }

}

/// - returns: A new `HMTimerTrigger` with the stored configurations.

override func newTrigger() -> HMTrigger? {

    return HMTimerTrigger(name: name, fireDate: fireDate, timeZone:
Calendar.current.timeZone, recurrence: recurrenceComponents, recurrenceCalendar: nil)

}

/// Updates the fire date and recurrence of the trigger.

override func updateTrigger() {

    updateFireDateIfNecessary()

    updateRecurrenceIfNecessary()

}

```

```
// MARK: Helper Methods
```

```
/**
```

```
    Creates an NSDateComponent for the selected recurrence type.
```

```
    - returns: An NSDateComponent where either `weekOfYear`,
```

```
              `hour`, or `day` is set to 1.
```

```
*/
```

```
var recurrenceComponents:DateComponents? {
```

```
    guard let selectedRecurrenceIndex = selectedRecurrenceIndex else {
```

```
        return nil
```

```
    }
```

```
    var recurrenceComponents = DateComponents()
```

```
    let unit = TimerTriggerCreator.RecurrenceComponents[selectedRecurrenceIndex]
```

```
    switch unit {
```

```
        case .weekOfYear:
```

```
recurrenceComponents.weekOfYear = 1
```

```
case .hour:
```

```
    recurrenceComponents.hour = 1
```

```
case .day:
```

```
    recurrenceComponents.day = 1
```

```
default:
```

```
    break
```

```
}
```

```
return recurrenceComponents
```

```
}
```

```
/**
```

Maps the possible calendar units associated with recurrence titles, so we can properly

set our recurrenceUnit when an index is selected.

- parameter components: An optional `NSDateComponents` to query.

- returns: An index for the date components.

*/

```
func recurrenceIndexFromDateComponents(_ components: DateComponents?) -> Int?
{
    guard let components = components else { return nil }

    var unit: Calendar.Component?

    if components.day == 1 {
        unit = .day
    }

    else if components.weekOfYear == 1 {
        unit = .weekOfYear
    }

    else if components.hour == 1 {
        unit = .hour
    }
}
```

```

    if let unit = unit {

        return TimerTriggerCreator.RecurrenceComponents.firstIndex(of: unit)

    }

    return nil

}

```

```
/**
```

Updates the trigger's fire date, entering and leaving the dispatch group if necessary.

If the trigger's fire date is already equal to the passed-in fire date, this method does nothing.

- parameter fireDate: The trigger's new fire date.

```
*/
```

```

private func updateFireDateIfNecessary() {

    if timerTrigger?.fireDate == fireDate {

        return

    }

    saveTriggerGroup.enter()

```

```

timerTrigger?.updateFireDate(fireDate) { error in

    if let error = error {

        self.errors.append(error)

    }

    self.saveTriggerGroup.leave()

}

}

```

```

/**

```

Updates the trigger's recurrence components, entering and leaving the dispatch group if necessary.

If the trigger's components are already equal to the passed-in components, this method does nothing.

- parameter recurrenceComponents: The trigger's new recurrence components.

```

*/

```

```

private func updateRecurrenceIfNecessary() {

    if recurrenceComponents == timerTrigger?.recurrence {

```

```

        return

    }

    saveTriggerGroup.enter()

    timerTrigger?.updateRecurrence(recurrenceComponents) { error in

        if let error = error {

            self.errors.append(error)

        }

        self.saveTriggerGroup.leave()

    }

}

}

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The 'TimerTriggerViewController' allows the user to create Timer triggers.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
/// A view controller which facilitates the creation of timer triggers.
```

```
class TimerTriggerViewController: TriggerViewController {
```

```
    // MARK: Types
```

```
    struct Identifiers {
```

```
        static let recurrenceCell = "RecurrenceCell"
```

```
    }
```

```
    static let RecurrenceTitles = [
```

```
        NSLocalizedString("Every Hour", comment: "Every Hour"),
```

```
        NSLocalizedString("Every Day", comment: "Every Day"),
```

```
        NSLocalizedString("Every Week", comment: "Every Week")
```

```
]
```

```
// MARK: Properties
```

```
@IBOutlet weak var datePicker: UIDatePicker!
```

```
/**
```

Sets the stored fireDate to the new value.

HomeKit only accepts dates aligned with minute boundaries,

so we use NSDateComponents to only get the appropriate pieces of information from that date.

Eventually we will end up with a date following this format: "MM/dd/yyyy hh:mm"

```
*/
```

```
var timerTrigger: HMTimerTrigger? {
```

```
    return trigger as? HMTimerTrigger
```

```
}
```

```

var timerTriggerCreator: TimerTriggerCreator {

    return triggerCreator as! TimerTriggerCreator

}

// MARK: View Methods

/// Configures the views and registers for table view cells.

override func viewDidLoad() {

    super.viewDidLoad()

    tableView.rowHeight = UITableView.automaticDimension

    tableView.estimatedRowHeight = 44.0

    triggerCreator = TimerTriggerCreator(trigger: trigger, home: home)

    datePicker.date = timerTriggerCreator.fireDate as Date

    tableView.register(UITableViewCell.self,                       forCellReuseIdentifier:
Identifiers.recurrenceCell)

}

// MARK: IBAction Methods

```

```

    /// Reset our saved fire date to the date in the picker.

    @IBAction func didChangeDate(_ picker: UIDatePicker) {

        timerTriggerCreator.rawFireDate = picker.date

    }

// MARK: Table View Methods

/**

    - returns: The number of rows in the Recurrence section;

               defaults to the super implementation for other sections

    */

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {

        switch sectionForIndex(section) {

            case .recurrence?:

                return TimerTriggerViewController.RecurrenceTitles.count

```


case nil:

fatalError("Unexpected `TriggerTableViewSection` raw value.")

default:

return super.tableView(tableView, numberOfRowsInSection: section)

}

}

/**

Generates a recurrence cell.

Defaults to the super implementation for other sections

*/

override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

switch sectionForIndex(indexPath.section) {

case .recurrence?:

return self.tableView(tableView, recurrenceCellForRowAtIndexPath: indexPath)

```
case nil:
```

```
    fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
default:
```

```
    return super.tableView(tableView, cellForRowAt: indexPath)
```

```
}
```

```
}
```

```
/// Creates a cell that represents a recurrence type.
```

```
func tableView(_ tableView: UITableView, recurrenceCellForRowAtIndexPath  
indexPath: IndexPath) -> UITableViewCell {
```

```
    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.recurrenceCell,  
for: indexPath)
```

```
    let title = TimerTriggerViewController.RecurrenceTitles[indexPath.row]
```

```
    cell.textLabel?.text = title
```

```
// The current preferred recurrence style should have a check mark.
```

```
if indexPath.row == timerTriggerCreator.selectedRecurrenceIndex {
```

```

        cell.accessoryType = .checkmark

    }

    else {

        cell.accessoryType = .none

    }

    return cell

}

```

```

/**

```

Tell the tableView to automatically size the custom rows, while using the superclass's static sizing for the static cells.

```

*/

```

```

override func tableView(_ tableView: UITableView, heightForRowAt indexPath:
IndexPath) -> CGFloat {

```

```

    switch sectionForIndex(indexPath.section) {

```

```

        case .recurrence?:

```

```

            return UITableView.automaticDimension

```

```

        case nil:

            fatalError("Unexpected `TriggerTableViewSection` raw value.")

        default:

            return super.tableView(tableView, heightForRowAt: indexPath)

    }

}

/**

Handles recurrence cell selection.

Defaults to the super implementation for other sections

*/

override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {

    tableView.deselectRow(at: indexPath, animated: true)

    switch sectionForIndex(indexPath.section) {

        case .recurrence?:

```

```
        self.tableView(tableView, didSelectRecurrenceComponentAtIndex:
indexPath)
```

```
    case nil:
```

```
        fatalError("Unexpected `TriggerTableViewSection` raw value.")
```

```
    default:
```

```
        super.tableView(tableView, didSelectRowAt: indexPath)
```

```
    }
```

```
}
```

```
/**
```

Handles selection of a recurrence cell.

If the newly selected recurrence component is the previously selected

recurrence component, reset the current selected component to `NSNotFound`

and deselect that row.

```
*/
```

```

func tableView(_ tableView: UITableView,
didSelectRecurrenceComponentAtIndex indexPath: IndexPath) {

    if indexPath.row == timerTriggerCreator.selectedRecurrenceIndex {

        timerTriggerCreator.selectedRecurrenceIndex = nil

    }

    else {

        timerTriggerCreator.selectedRecurrenceIndex = indexPath.row

    }

    tableView.reloadSections(IndexSet(integer: indexPath.section), with: .automatic)

}

/**

- parameter index: The section index.

- returns: The `TriggerTableViewSection` for the given index.

*/

override func sectionForIndex(_ index: Int) -> TriggerTableViewSection? {

    switch index {

```

case 0:

return .name

case 1:

return .enabled

case 2:

return .dateAndTime

case 3:

return .recurrence

case 4:

return .actionSets

default:

return nil

```
}

}

}

//

// MIT License

//

// Copyright (c) 2017 Robert-Hein Hooijmans <rh.hooijmans@gmail.com>

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in
```



```
// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
// KIND, EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
// MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
// EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
// OTHERWISE, ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
// DEALINGS IN

// THE SOFTWARE.

//

#if os(OSX)

import AppKit
```

```

public extension View {

    @discardableResult

    func edgesToSuperview(excluding excludedEdge: LayoutEdge = .none, insets:
TinyEdgeInsets = .zero, usingSafeArea: Bool = false) -> Constraints {

        var constraints = Constraints()

        if !excludedEdge.contains(.top) {

            constraints.append(topToSuperview(offset: insets.top, usingSafeArea:
usingSafeArea))

        }

        if !excludedEdge.contains(.left) {

            constraints.append(leftToSuperview(offset: insets.left, usingSafeArea:
usingSafeArea))

        }

        if !excludedEdge.contains(.right) {

```

```
        constraints.append(rightToSuperview(offset: -insets.right, usingSafeArea:
usingSafeArea))
```

```
    }
```

```
    if !excludedEdge.contains(.bottom) {
```

```
        constraints.append(bottomToSuperview(offset: -insets.bottom, usingSafeArea:
usingSafeArea))
```

```
    }
```

```
    return constraints
```

```
}
```

```
}
```

```
#else
```

```
import UIKit
```

```
public extension View {
```

```
    @available(tvOS 10.0, *)
```

```
@available(iOS 10.0, *)
```

```
@discardableResult
```

```
func edgesToSuperview(excluding excludedEdge: LayoutEdge = .none, insets: TinyEdgeInsets = .zero, relation: ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool = true, usingSafeArea: Bool = false) -> Constraints {
```

```
    var constraints = Constraints()
```

```
    if !excludedEdge.contains(.top) {
```

```
        constraints.append(topToSuperview(offset: insets.top, relation: relation, priority: priority, isActive: isActive, usingSafeArea: usingSafeArea))
```

```
    }
```

```
    if effectiveUserInterfaceLayoutDirection == .leftToRight {
```

```
        if !(excludedEdge.contains(.leading) || excludedEdge.contains(.left)) {
```

```
            constraints.append(leftToSuperview(offset: insets.left, relation: relation, priority: priority, isActive: isActive, usingSafeArea: usingSafeArea))
```

```
        }
```

```

        if !(excludedEdge.contains(.trailing) || excludedEdge.contains(.right)) {

            constraints.append(rightToSuperview(offset: -insets.right, relation: relation,
priority: priority, isActive: isActive, usingSafeArea: usingSafeArea))

        }

    } else {

        if !(excludedEdge.contains(.leading) || excludedEdge.contains(.right)) {

            constraints.append(rightToSuperview(offset: -insets.right, relation: relation,
priority: priority, isActive: isActive, usingSafeArea: usingSafeArea))

        }

        if !(excludedEdge.contains(.trailing) || excludedEdge.contains(.left)) {

            constraints.append(leftToSuperview(offset: insets.left, relation: relation,
priority: priority, isActive: isActive, usingSafeArea: usingSafeArea))

        }

    }

    if !excludedEdge.contains(.bottom) {

```

```
        constraints.append(bottomToSuperview(offset: -insets.bottom, relation: relation,
priority: priority, isActive: isActive, usingSafeArea: usingSafeArea))
```

```
    }
```

```
    return constraints
```

```
}
```

```
@available(tvOS 10.0, *)
```

```
@available(iOS 10.0, *)
```

```
@discardableResult
```

```
func leadingToSuperview(_ anchor: NSLayoutXAxisAnchor? = nil, offset: CGFloat
= 0, relation: ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive:
Bool = true, usingSafeArea: Bool = false) -> Constraint {
```

```
    let constrainable = safeConstrainable(for: superview, usingSafeArea:
usingSafeArea)
```

```
    if effectiveUserInterfaceLayoutDirection == .rightToLeft {
```

```
        return leading(to: constrainable, anchor, offset: -offset, relation: relation, priority:
priority, isActive: isActive)
```

```
    } else {
```

```
        return leading(to: constrainable, anchor, offset: offset, relation: relation, priority:
priority, isActive: isActive)
```

```
    }
```

```
}
```

```
@available(tvOS 10.0, *)
```

```
@available(iOS 10.0, *)
```

```
@discardableResult
```

```
func trailingToSuperview(_ anchor: NSLayoutXAxisAnchor? = nil, offset: CGFloat
= 0, relation: ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive:
Bool = true, usingSafeArea: Bool = false) -> Constraint {
```

```
    let constrainable = safeConstrainable(for: superview, usingSafeArea:
usingSafeArea)
```

```
    if effectiveUserInterfaceLayoutDirection == .rightToLeft {
```

```
        return trailing(to: constrainable, anchor, offset: offset, relation: relation, priority:
priority, isActive: isActive)
```

```
    } else {
```

```
        return trailing(to: constrainable, anchor, offset: -offset, relation: relation, priority:
priority, isActive: isActive)
```

```
}  
  
}
```

```
@available(tvOS 10.0, *)
```

```
@available(iOS 10.0, *)
```

```
@discardableResult
```

```
func horizontalToSuperview(insets: TinyEdgeInsets = .zero, relation:  
ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool = true,  
usingSafeArea: Bool = false) -> Constraints {
```

```
    var constraints = Constraints()
```

```
    if effectiveUserInterfaceLayoutDirection == .leftToRight {
```

```
        constraints.append(leftToSuperview(offset: insets.left, relation: relation, priority:  
priority, isActive: isActive, usingSafeArea: usingSafeArea))
```

```
        constraints.append(rightToSuperview(offset: -insets.right, relation: relation,  
priority: priority, isActive: isActive, usingSafeArea: usingSafeArea))
```

```
    } else {
```

```
        constraints.append(rightToSuperview(offset: -insets.right, relation: relation,  
priority: priority, isActive: isActive, usingSafeArea: usingSafeArea))
```



```
constraints.append(leftToSuperview(offset: insets.left, relation: relation, priority:
priority, isActive: isActive, usingSafeArea: usingSafeArea))
```

```
}
```

```
return constraints
```

```
}
```

```
@available(tvOS 10.0, *)
```

```
@available(iOS 10.0, *)
```

```
@discardableResult
```

```
func verticalToSuperview(insets: TinyEdgeInsets = .zero, relation:
ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool = true,
usingSafeArea: Bool = false) -> Constraints {
```

```
let constraints = Constraints(arrayLiteral:
```

```
topToSuperview(offset: insets.top, relation: relation, priority: priority, isActive:
isActive, usingSafeArea: usingSafeArea),
```

```
bottomToSuperview(offset: -insets.bottom, relation: relation, priority: priority,
isActive: isActive, usingSafeArea: usingSafeArea)
```

```
)
```

```

        return constraints

    }

}

#endif

public struct LayoutEdge: OptionSet {

    public let rawValue: UInt8

    public init(rawValue: UInt8) {

        self.rawValue = rawValue

    }

    public static let top = LayoutEdge(rawValue: 1 << 0)

    public static let bottom = LayoutEdge(rawValue: 1 << 1)

    public static let trailing = LayoutEdge(rawValue: 1 << 2)

    public static let leading = LayoutEdge(rawValue: 1 << 3)

    public static let left = LayoutEdge(rawValue: 1 << 4)

    public static let right = LayoutEdge(rawValue: 1 << 5)

    public static let none = LayoutEdge(rawValue: 1 << 6)

```

```
}
```

```
public extension View {
```

```
    private fun safeConstrainable(for superview: View?, usingSafeArea: Bool) ->  
    Constrainable {
```

```
        guard let superview = superview else {
```

```
            fatalError("Unable to create this constraint to it's superview, because it has no  
superview.")
```

```
        }
```

```
    prepareForLayout()
```

```
    #if os(iOS) || os(tvOS)
```

```
        if #available(iOS 11, tvOS 11, *){
```

```
            if usingSafeArea {
```

```
                return superview.safeAreaLayoutGuide
```

```
            }
```

```
}
```

```
#endif
```

```
return superview
```

```
}
```

```
@discardableResult
```

```
func centerInSuperview(offset: CGPoint = .zero, priority: LayoutPriority = .required,  
isActive: Bool = true, usingSafeArea: Bool = false) -> Constraints {
```

```
    let constrainable = safeConstrainable(for: superview, usingSafeArea: usingSafeArea)
```

```
    return center(in: constrainable, offset: offset, priority: priority, isActive: isActive)
```

```
}
```

```
@discardableResult
```

```
func originToSuperview(insets: TinyEdgeInsets = .zero, relation: ConstraintRelation  
= .equal, priority: LayoutPriority = .required, isActive: Bool = true, usingSafeArea: Bool  
= false) -> Constraints {
```

```
    let constrainable = safeConstrainable(for: superview, usingSafeArea: usingSafeArea)
```

```

        return origin(to: constrainable, insets: insets, relation: relation, priority: priority,
isActive: isActive)

    }

```

@discardableResult

```

func widthToSuperview(_ dimension: NSLayoutDimension? = nil, multiplier: CGFloat
= 1, offset: CGFloat = 0, relation: ConstraintRelation = .equal, priority: LayoutPriority
= .required, isActive: Bool = true, usingSafeArea: Bool = false) -> Constraint {

    let constrainable = safeConstrainable(for: superview, usingSafeArea: usingSafeArea)

    return width(to: constrainable, dimension, multiplier: multiplier, offset: offset,
relation: relation, priority: priority, isActive: isActive)

}

```

@discardableResult

```

func heightToSuperview(_ dimension: NSLayoutDimension? = nil, multiplier: CGFloat
= 1, offset: CGFloat = 0, relation: ConstraintRelation = .equal, priority: LayoutPriority
= .required, isActive: Bool = true, usingSafeArea: Bool = false) -> Constraint {

    let constrainable = safeConstrainable(for: superview, usingSafeArea: usingSafeArea)

    return height(to: constrainable, dimension, multiplier: multiplier, offset: offset,
relation: relation, priority: priority, isActive: isActive)

}

```

@discardableResult

```
func leftToSuperview( _ anchor: NSLayoutXAxisAnchor? = nil, offset: CGFloat = 0,
relation: ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool
= true, usingSafeArea: Bool = false) -> Constraint {
```

```
    let constrainable = safeConstrainable(for: superview, usingSafeArea: usingSafeArea)
```

```
    return left(to: constrainable, anchor, offset: offset, relation: relation, priority: priority,
isActive: isActive)
```

```
}
```

@discardableResult

```
func rightToSuperview( _ anchor: NSLayoutXAxisAnchor? = nil, offset: CGFloat = 0,
relation: ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool
= true, usingSafeArea: Bool = false) -> Constraint {
```

```
    let constrainable = safeConstrainable(for: superview, usingSafeArea: usingSafeArea)
```

```
    return right(to: constrainable, anchor, offset: offset, relation: relation, priority: priority,
isActive: isActive)
```

```
}
```

@discardableResult

```
func topToSuperview( _ anchor: NSLayoutYAxisAnchor? = nil, offset: CGFloat = 0,
relation: ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool
= true, usingSafeArea: Bool = false) -> Constraint {
```

```
    let constrainable = safeConstrainable(for: superview, usingSafeArea: usingSafeArea)
```

```
    return top(to: constrainable, anchor, offset: offset, relation: relation, priority: priority,
isActive: isActive)
```

```
}
```

```
@discardableResult
```

```
func bottomToSuperview( _ anchor: NSLayoutYAxisAnchor? = nil, offset: CGFloat =
0, relation: ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool
= true, usingSafeArea: Bool = false) -> Constraint {
```

```
    let constrainable = safeConstrainable(for: superview, usingSafeArea: usingSafeArea)
```

```
    return bottom(to: constrainable, anchor, offset: offset, relation: relation, priority:
priority, isActive: isActive)
```

```
}
```

```
@discardableResult
```

```
func centerXToSuperview( _ anchor: NSLayoutXAxisAnchor? = nil, multiplier:
CGFloat = 1, offset: CGFloat = 0, priority: LayoutPriority = .required, isActive: Bool =
true, usingSafeArea: Bool = false) -> Constraint {
```

```

        let constrainable = safeConstrainable(for: superview, usingSafeArea: usingSafeArea)

        return centerX(to: constrainable, anchor, multiplier: multiplier, offset: offset, priority:
priority, isActive: isActive)

    }

```

@discardableResult

```

func centerYToSuperview( _ anchor: NSLayoutYAxisAnchor? = nil, multiplier:
CGFloat = 1, offset: CGFloat = 0, priority: LayoutPriority = .required, isActive: Bool =
true, usingSafeArea: Bool = false) -> Constraint {

```

```

    let constrainable = safeConstrainable(for: superview, usingSafeArea: usingSafeArea)

    return centerY(to: constrainable, anchor, multiplier: multiplier, offset: offset, priority:
priority, isActive: isActive)

}

```

```

}

```

```

//

```

```

// MIT License

```

```

//

```

```

// Copyright (c) 2017 Robert-Hein Hooijmans <rh.hooijmans@gmail.com>

```

```

//

```


// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
KIND, EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM,

```
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER  
DEALINGS IN
```

```
// THE SOFTWARE.
```

```
//
```

```
#if os(OSX)
```

```
import AppKit
```

```
#else
```

```
import UIKit
```

```
#endif
```

```
public extension Constrainable {
```

```
    @discardableResult
```

```
    func center(in view: Constrainable, offset: CGPoint = .zero, priority: LayoutPriority  
= .required, isActive: Bool = true) -> Constraints {
```

```
        prepareForLayout()
```

```

let constraints = [

    centerX(to: view, offset: offset.x, priority: priority, isActive: isActive),

    centerY(to: view, offset: offset.y, priority: priority, isActive: isActive)

]

return constraints

}

@discardableResult

func edges(to view: Constrainable, excluding excludedEdge: LayoutEdge = .none, insets:
TinyEdgeInsets = .zero, relation: ConstraintRelation = .equal, priority: LayoutPriority
= .required, isActive: Bool = true) -> Constraints {

    prepareForLayout()

    var constraints = Constraints()

    if !excludedEdge.contains(.top) {

        constraints.append(top(to: view, offset: insets.top, relation: relation, priority:
priority, isActive: isActive))

```

```
}
```

```
if !excludedEdge.contains(.left) {
```

```
    constraints.append(left(to: view, offset: insets.left, relation: relation, priority:  
priority, isActive: isActive))
```

```
}
```

```
if !excludedEdge.contains(.bottom) {
```

```
    constraints.append(bottom(to: view, offset: -insets.bottom, relation: relation,  
priority: priority, isActive: isActive))
```

```
}
```

```
if !excludedEdge.contains(.right) {
```

```
    constraints.append(right(to: view, offset: -insets.right, relation: relation, priority:  
priority, isActive: isActive))
```

```
}
```

```
return constraints
```

```
}
```

@discardableResult

```
func size(_ size: CGSize, relation: ConstraintRelation = .equal, priority: LayoutPriority  
= .required, isActive: Bool = true) -> Constraints {
```

```
    prepareForLayout()
```

```
    let constraints = [
```

```
        width(size.width, relation: relation, priority: priority, isActive: isActive),
```

```
        height(size.height, relation: relation, priority: priority, isActive: isActive)
```

```
    ]
```

```
    return constraints
```

```
}
```

@discardableResult

```
func size(to view: Constrainable, multiplier: CGFloat = 1, insets: CGSize = .zero,  
relation: ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool  
= true) -> Constraints {
```

```
    prepareForLayout()
```

```

let constraints = [

    width(to: view, multiplier: multiplier, offset: insets.width, relation: relation,
priority: priority, isActive: isActive),

    height(to: view, multiplier: multiplier, offset: insets.height, relation: relation,
priority: priority, isActive: isActive)

]

return constraints

}

```

@discardableResult

```

func origin(to view: Constrainable, insets: TinyEdgeInsets = .zero, relation:
ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool = true) ->
Constraints {

```

```

    prepareForLayout()

```

```

let constraints = [

    left(to: view, offset: insets.left, relation: relation, priority: priority, isActive:
isActive),

```

```
        top(to: view, offset: insets.top, relation: relation, priority: priority, isActive:
isActive)
```

```
    ]
```

```
    return constraints
```

```
    }
```

```
@discardableResult
```

```
func width(_ width: CGFloat, relation: ConstraintRelation = .equal, priority:
LayoutPriority = .required, isActive: Bool = true) -> Constraint {
```

```
    prepareForLayout()
```

```
    switch relation {
```

```
        case .equal: return widthAnchor.constraint(equalToConstant:
width).with(priority).set(isActive)
```

```
        case .equalOrLess: return widthAnchor.constraint(lessThanOrEqualToConstant:
width).with(priority).set(isActive)
```

```
        case .equalOrGreater: return
widthAnchor.constraint(greaterThanOrEqualToConstant:
width).with(priority).set(isActive)
```

```
}
```

```
}
```

```
@discardableResult
```

```
func width(to view: Constrainable, _ dimension: NSLayoutDimension? = nil, multiplier: CGFloat = 1, offset: CGFloat = 0, relation: ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool = true) -> Constraint {
```

```
    prepareForLayout()
```

```
    switch relation {
```

```
        case .equal: return widthAnchor.constraint(equalTo: dimension ?? view.widthAnchor, multiplier: multiplier, constant: offset).with(priority).set(isActive)
```

```
        case .equalOrLess: return widthAnchor.constraint(lessThanOrEqualTo: dimension ?? view.widthAnchor, multiplier: multiplier, constant: offset).with(priority).set(isActive)
```

```
        case .equalOrGreater: return widthAnchor.constraint(greaterThanOrEqualTo: dimension ?? view.widthAnchor, multiplier: multiplier, constant: offset).with(priority).set(isActive)
```

```
    }
```

```
}
```


@discardableResult

```
func widthToHeight(of view: Constrainable, multiplier: CGFloat = 1, offset: CGFloat =
0, relation: ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool
= true) -> Constraint {

    return width(to: view, view.heightAnchor, multiplier: multiplier, offset: offset,
relation: relation, priority: priority, isActive: isActive)

}
```

@discardableResult

```
func width(min: CGFloat? = nil, max: CGFloat? = nil, priority: LayoutPriority
= .required, isActive: Bool = true) -> Constraints {

    prepareForLayout()

    var constraints: Constraints = []

    if let min = min {

        let constraint = widthAnchor.constraint(greaterThanOrEqualToConstant:
min).with(priority)

        constraint.isActive = isActive

        constraints.append(constraint)

    }
```

```
}
```

```
if let max = max {
```

```
    let constraint = widthAnchor.constraint(lessThanOrEqualToConstant:  
max).with(priority)
```

```
    constraint.isActive = isActive
```

```
    constraints.append(constraint)
```

```
}
```

```
return constraints
```

```
}
```

```
@discardableResult
```

```
func height(_ height: CGFloat, relation: ConstraintRelation = .equal, priority:  
LayoutPriority = .required, isActive: Bool = true) -> Constraint {
```

```
    prepareForLayout()
```

```
    switch relation {
```

```

        case .equal: return heightAnchor.constraint(equalToConstant:
height).with(priority).set(isActive)

```

```

        case .equalOrLess: return heightAnchor.constraint(lessThanOrEqualToConstant:
height).with(priority).set(isActive)

```

```

        case .equalOrGreater: return
heightAnchor.constraint(greaterThanOrEqualToConstant:
height).with(priority).set(isActive)

```

```

    }

```

```

}

```

@discardableResult

```

func height(to view: Constrainable, _ dimension: NSLayoutDimension? = nil, multiplier:
CGFloat = 1, offset: CGFloat = 0, relation: ConstraintRelation = .equal, priority:
LayoutPriority = .required, isActive: Bool = true) -> Constraint {

```

```

    prepareForLayout()

```

```

    switch relation {

```

```

        case .equal: return heightAnchor.constraint(equalTo: dimension ??
view.heightAnchor, multiplier: multiplier, constant: offset).with(priority).set(isActive)

```

```

        case .equalOrLess: return heightAnchor.constraint(lessThanOrEqualTo: dimension ??
view.heightAnchor, multiplier: multiplier, constant: offset).with(priority).set(isActive)

```

```

        case .equalOrGreater: return heightAnchor.constraint(greaterThanOrEqualTo:
dimension ?? view.heightAnchor, multiplier: multiplier, constant:
offset).with(priority).set(isActive)

    }

}

```

@discardableResult

```

func heightToWidth(of view: Constrainable, multiplier: CGFloat = 1, offset: CGFloat =
0, relation: ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool
= true) -> Constraint {

    return height(to: view, view.widthAnchor, multiplier: multiplier, offset: offset,
relation: relation, priority: priority, isActive: isActive)

}

```

@discardableResult

```

func height(min: CGFloat? = nil, max: CGFloat? = nil, priority: LayoutPriority
= .required, isActive: Bool = true) -> Constraints {

    prepareForLayout()

```

```

    var constraints: Constraints = []

```

```

    if let min = min {

        let constraint = heightAnchor.constraint(greaterThanOrEqualToConstant:
min).with(priority)

        constraint.isActive = isActive

        constraints.append(constraint)

    }

    if let max = max {

        let constraint = heightAnchor.constraint(lessThanOrEqualToConstant:
max).with(priority)

        constraint.isActive = isActive

        constraints.append(constraint)

    }

    return constraints

}

```

@discardableResult

```
func aspectRatio(_ ratio: CGFloat, relation: ConstraintRelation = .equal, priority:
LayoutPriority = .required, isActive: Bool = true) -> Constraint {

    return widthToHeight(of: self, multiplier: ratio, offset: 0, relation: relation, priority:
priority, isActive: isActive)

}
```

@discardableResult

```
func leadingToTrailing(of view: Constraintable, offset: CGFloat = 0, relation:
ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool = true) ->
Constraint {

    prepareForLayout()

    return leading(to: view, view.trailingAnchor, offset: offset, relation: relation, priority:
priority, isActive: isActive)

}
```

@discardableResult

```
func leading(to view: Constraintable, _ anchor: NSLayoutXAxisAnchor? = nil, offset:
CGFloat = 0, relation: ConstraintRelation = .equal, priority: LayoutPriority = .required,
isActive: Bool = true) -> Constraint {

    prepareForLayout()

}
```

```

switch relation {

    case .equal: return leadingAnchor.constraint(equalTo: anchor ?? view.leadingAnchor,
constant: offset).with(priority).set(isActive)

    case .equalOrLess: return leadingAnchor.constraint(lessThanOrEqualTo: anchor ??
view.leadingAnchor, constant: offset).with(priority).set(isActive)

    case .equalOrGreater: return leadingAnchor.constraint(greaterThanOrEqualTo:
anchor ?? view.leadingAnchor, constant: offset).with(priority).set(isActive)

}

}

```

@discardableResult

```

func leftToRight(of view: Constrainable, offset: CGFloat = 0, relation:
ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool = true) ->
Constraint {

    prepareForLayout()

    return left(to: view, view.rightAnchor, offset: offset, relation: relation, priority:
priority, isActive: isActive)

}

```

@discardableResult

```
func left(to view: Constraintable, _ anchor: NSLayoutXAxisAnchor? = nil, offset:
CGFloat = 0, relation: ConstraintRelation = .equal, priority: LayoutPriority = .required,
isActive: Bool = true) -> Constraint {
```

```
    prepareForLayout()
```

```
    switch relation {
```

```
        case .equal: return leftAnchor.constraint(equalTo: anchor ?? view.leftAnchor,
constant: offset).with(priority).set(isActive)
```

```
        case .equalOrLess: return leftAnchor.constraint(lessThanOrEqualTo: anchor ??
view.leftAnchor, constant: offset).with(priority).set(isActive)
```

```
        case .equalOrGreater: return leftAnchor.constraint(greaterThanOrEqualTo: anchor ??
view.leftAnchor, constant: offset).with(priority).set(isActive)
```

```
    }
```

```
}
```

@discardableResult

```
func trailingToLeading(of view: Constraintable, offset: CGFloat = 0, relation:
ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool = true) ->
Constraint {
```

```
    prepareForLayout()
```



```
        return trailing(to: view, view.leadingAnchor, offset: offset, relation: relation, priority:
priority, isActive: isActive)
```

```
    }
```

```
@discardableResult
```

```
func trailing(to view: Constraintable, _ anchor: NSLayoutXAxisAnchor? = nil, offset:
CGFloat = 0, relation: ConstraintRelation = .equal, priority: LayoutPriority = .required,
isActive: Bool = true) -> Constraint {
```

```
    prepareForLayout()
```

```
    switch relation {
```

```
        case .equal: return trailingAnchor.constraint(equalTo: anchor ?? view.trailingAnchor,
constant: offset).with(priority).set(isActive)
```

```
        case .equalOrLess: return trailingAnchor.constraint(lessThanOrEqualTo: anchor ??
view.trailingAnchor, constant: offset).with(priority).set(isActive)
```

```
        case .equalOrGreater: return trailingAnchor.constraint(greaterThanOrEqualTo:
anchor ?? view.trailingAnchor, constant: offset).with(priority).set(isActive)
```

```
    }
```

```
}
```

@discardableResult

```
func rightToLeft(of view: Constrainable, offset: CGFloat = 0, relation:
ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool = true) ->
Constraint {

    prepareForLayout()

    return right(to: view, view.leftAnchor, offset: offset, relation: relation, priority:
priority, isActive: isActive)

}
```

@discardableResult

```
func right(to view: Constrainable, _ anchor: NSLayoutXAxisAnchor? = nil, offset:
CGFloat = 0, relation: ConstraintRelation = .equal, priority: LayoutPriority = .required,
isActive: Bool = true) -> Constraint {

    prepareForLayout()

    switch relation {

        case .equal: return rightAnchor.constraint(equalTo: anchor ?? view.rightAnchor,
constant: offset).with(priority).set(isActive)

        case .equalOrLess: return rightAnchor.constraint(lessThanOrEqualTo: anchor ??
view.rightAnchor, constant: offset).with(priority).set(isActive)
```

```

        case .equalOrGreater: return rightAnchor.constraint(greaterThanOrEqualTo: anchor ??
view.rightAnchor, constant: offset).with(priority).set(isActive)

    }

}

```

@discardableResult

```

func topToBottom(of view: Constrainable, offset: CGFloat = 0, relation:
ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool = true) ->
Constraint {

    prepareForLayout()

    return top(to: view, view.bottomAnchor, offset: offset, relation: relation, priority:
priority, isActive: isActive)

}

```

@discardableResult

```

func top(to view: Constrainable, _ anchor: NSLayoutYAxisAnchor? = nil, offset:
CGFloat = 0, relation: ConstraintRelation = .equal, priority: LayoutPriority = .required,
isActive: Bool = true) -> Constraint {

    prepareForLayout()

```

```

switch relation {

    case .equal: return topAnchor.constraint(equalTo: anchor ?? view.topAnchor,
constant: offset).with(priority).set(isActive)

    case .equalOrLess: return topAnchor.constraint(lessThanOrEqualTo: anchor ??
view.topAnchor, constant: offset).with(priority).set(isActive)

    case .equalOrGreater: return topAnchor.constraint(greaterThanOrEqualTo: anchor ??
view.topAnchor, constant: offset).with(priority).set(isActive)

}

}

```

@discardableResult

```

func bottomToTop(of view: Constrainable, offset: CGFloat = 0, relation:
ConstraintRelation = .equal, priority: LayoutPriority = .required, isActive: Bool = true) ->
Constraint {

    prepareForLayout()

    return bottom(to: view, view.topAnchor, offset: offset, relation: relation, priority:
priority, isActive: isActive)

}

```

@discardableResult

```
func bottom(to view: Constrainable, _ anchor: NSLayoutYAxisAnchor? = nil, offset:
CGFloat = 0, relation: ConstraintRelation = .equal, priority: LayoutPriority = .required,
isActive: Bool = true) -> Constraint {
```

```
    prepareForLayout()
```

```
    switch relation {
```

```
        case .equal: return bottomAnchor.constraint(equalTo: anchor ?? view.bottomAnchor,
constant: offset).with(priority).set(isActive)
```

```
        case .equalOrLess: return bottomAnchor.constraint(lessThanOrEqualTo: anchor ??
view.bottomAnchor, constant: offset).with(priority).set(isActive)
```

```
        case .equalOrGreater: return bottomAnchor.constraint(greaterThanOrEqualTo:
anchor ?? view.bottomAnchor, constant: offset).with(priority).set(isActive)
```

```
    }
```

```
}
```

```
@discardableResult
```

```
func centerX(to view: Constrainable, _ anchor: NSLayoutXAxisAnchor? = nil,
multiplier: CGFloat = 1, offset: CGFloat = 0, priority: LayoutPriority = .required, isActive:
Bool = true) -> Constraint {
```

```
    prepareForLayout()
```

```

let constraint: Constraint

if let anchor = anchor {

    constraint = centerXAnchor.constraint(equalTo: anchor, constant:
offset).with(priority)

} else {

    constraint = NSLayoutConstraint(item: self, attribute: .centerX, relatedBy: .equal,
toItem: view, attribute: .centerX, multiplier: multiplier, constant: offset).with(priority)

}

constraint.isActive = isActive

return constraint

}

@discardableResult

func centerY(to view: Constrainable, _ anchor: NSLayoutYAxisAnchor? = nil,
multiplier: CGFloat = 1, offset: CGFloat = 0, priority: LayoutPriority = .required, isActive:
Bool = true) -> Constraint {

    prepareForLayout()

```

```

let constraint: Constraint

if let anchor = anchor {

    constraint = centerYAnchor.constraint(equalTo: anchor, constant:
offset).with(priority)

} else {

    constraint = NSLayoutConstraint(item: self, attribute: .centerY, relatedBy: .equal,
toItem: view, attribute: .centerY, multiplier: multiplier, constant: offset).with(priority)

}

constraint.isActive = isActive

return constraint

}

}

public extension View {

```

```

func setHugging(_ priority: LayoutPriority, for axis: ConstraintAxis) {

    setContentHuggingPriority(priority, for: axis)

}

func setCompressionResistance(_ priority: LayoutPriority, for axis: ConstraintAxis) {

    setContentCompressionResistancePriority(priority, for: axis)

}

}

//

//  MIT License

//

//  Copyright (c) 2017 Robert-Hein Hooijmans <rh.hooijmans@gmail.com>

//

//  Permission is hereby granted, free of charge, to any person obtaining a copy

//  of this software and associated documentation files (the "Software"), to deal

//  in the Software without restriction, including without limitation the rights

//  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

```


// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
KIND, EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

```
#if os(OSX)
```

```
    import AppKit
```

```
#else
```

```
    import UIKit
```

```
#endif
```

```
extension TinyEdgeInsets {
```

```
    public static func uniform(_ value: CGFloat) -> TinyEdgeInsets {
```

```
        return TinyEdgeInsets(top: value, left: value, bottom: value, right: value)
```

```
    }
```

```
    public static func top(_ value: CGFloat) -> TinyEdgeInsets {
```

```
        return TinyEdgeInsets(top: value, left: 0, bottom: 0, right: 0)
```

```
    }
```

```
public static func left(_ value: CGFloat) -> TinyEdgeInsets {  
  
    return TinyEdgeInsets(top: 0, left: value, bottom: 0, right: 0)  
  
}
```

```
public static func bottom(_ value: CGFloat) -> TinyEdgeInsets {  
  
    return TinyEdgeInsets(top: 0, left: 0, bottom: value, right: 0)  
  
}
```

```
public static func right(_ value: CGFloat) -> TinyEdgeInsets {  
  
    return TinyEdgeInsets(top: 0, left: 0, bottom: 0, right: value)  
  
}
```

```
public static func horizontal(_ value: CGFloat) -> TinyEdgeInsets {  
  
    return TinyEdgeInsets(top: 0, left: value, bottom: 0, right: value)  
  
}
```

```
public static func vertical(_ value: CGFloat) -> TinyEdgeInsets {
```

```

        return TinyEdgeInsets(top: value, left: 0, bottom: value, right: 0)

    }

}

public func + (lhs: TinyEdgeInsets, rhs: TinyEdgeInsets) -> TinyEdgeInsets {

    return .init(top: lhs.top + rhs.top, left: lhs.left + rhs.left, bottom: lhs.bottom + rhs.bottom,
right: lhs.right + rhs.right)

}

//

// Transformer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

```

```
import Foundation
```

```
import CoreGraphics
```

```
/// Transformer class that contains all matrices and is responsible for transforming values  
into pixels on the screen and backwards.
```

```
@objc(CharTransformer)
```

```
open class Transformer: NSObject
```

```
{
```

```
    /// matrix to map the values to the screen pixels
```

```
    internal var _matrixValueToPx = CGAffineTransform.identity
```

```
    /// matrix for handling the different offsets of the chart
```

```
    internal var _matrixOffset = CGAffineTransform.identity
```

```
    internal var _viewPortHandler: ViewPortHandler
```

```
@objc public init(viewPortHandler: ViewPortHandler)
```

```

{

    _viewPortHandler = viewPortHandler

}

/// Prepares the matrix that transforms values to pixels. Calculates the scale factors from
the charts size and offsets.

@objc open func prepareMatrixValuePx(chartXMin: Double, deltaX: CGFloat, deltaY:
CGFloat, chartYMin: Double)

{

    var scaleX = (_viewPortHandler.contentWidth / deltaX)

    var scaleY = (_viewPortHandler.contentHeight / deltaY)

    if CGFloat.infinity == scaleX

    {

        scaleX = 0.0

    }

    if CGFloat.infinity == scaleY

    {

```

```

        scaleY = 0.0

    }

    // setup all matrices

    _matrixValueToPx = CGAffineTransform.identity

    _matrixValueToPx = _matrixValueToPx.scaledBy(x: scaleX, y: -scaleY)

    _matrixValueToPx = _matrixValueToPx.translatedBy(x: CGFloat(-chartXMin), y:
CGFloat(-chartYMin))

}

/// Prepares the matrix that contains all offsets.

@objc open func prepareMatrixOffset(inverted: Bool)

{

    if !inverted

    {

        _matrixOffset = CGAffineTransform(translationX: _viewPortHandler.offsetLeft, y:
_viewPortHandler.chartHeight - _viewPortHandler.offsetBottom)

    }

```

```

else

{

    _matrixOffset = CGAffineTransform(scaleX: 1.0, y: -1.0)

    _matrixOffset = _matrixOffset.translatedBy(x: _viewPortHandler.offsetLeft, y: -
_viewPortHandler.offsetTop)

}

}

```

/// Transform an array of points with all matrices.

// VERY IMPORTANT: Keep matrix order "value-touch-offset" when transforming.

```

open func pointValuesToPixel(_ points: inout [CGPoint])

```

```

{

    let trans = valueToPixelMatrix

    points = points.map { $0.applying(trans) }

}

```

```

open func pointValueToPixel(_ point: inout CGPoint)

```

```

{

```



```

    point = point.applying(valueToPixelMatrix)

}

@objc open func pixelForValues(x: Double, y: Double) -> CGPoint

{

    return CGPoint(x: x, y: y).applying(valueToPixelMatrix)

}

/// Transform a rectangle with all matrices.

open func rectValueToPixel(_ r: inout CGRect)

{

    r = r.applying(valueToPixelMatrix)

}

/// Transform a rectangle with all matrices with potential animation phases.

open func rectValueToPixel(_ r: inout CGRect, phaseY: Double)

{

```

```

// multiply the height of the rect with the phase

var bottom = r.origin.y + r.size.height

bottom *= CGFloat(phaseY)

let top = r.origin.y * CGFloat(phaseY)

r.size.height = bottom - top

r.origin.y = top


r = r.applying(valueToPixelMatrix)

}


/// Transform a rectangle with all matrices.

open func rectValueToPixelHorizontal(_ r: inout CGRect)

{

    r = r.applying(valueToPixelMatrix)

}


/// Transform a rectangle with all matrices with potential animation phases.

```

```

open func rectValueToPixelHorizontal(_ r: inout CGRect, phaseY: Double)

{

    // multiply the height of the rect with the phase

    let left = r.origin.x * CGFloat(phaseY)

    let right = (r.origin.x + r.size.width) * CGFloat(phaseY)

    r.size.width = right - left

    r.origin.x = left


    r = r.applying(valueToPixelMatrix)

}


/// transforms multiple rects with all matrices

open func rectValuesToPixel(_ rects: inout [CGRect])

{

    let trans = valueToPixelMatrix

    rects = rects.map { $0.applying(trans) }

}

```

/// Transforms the given array of touch points (pixels) into values on the chart.

```
open func pixelsToValues(_ pixels: inout [CGPoint])
```

```
{
```

```
    let trans = pixelToValueMatrix
```

```
    pixels = pixels.map { $0.applying(trans) }
```

```
}
```

/// Transforms the given touch point (pixels) into a value on the chart.

```
open func pixelToValues(_ pixel: inout CGPoint)
```

```
{
```

```
    pixel = pixel.applying(pixelToValueMatrix)
```

```
}
```

/// - Returns: The x and y values in the chart at the given touch point

/// (encapsulated in a CGPoint). This method transforms pixel coordinates to

/// coordinates / values in the chart.

```
@objc open func valueForTouchPoint(_ point: CGPoint) -> CGPoint
```

```
{
```

```
    return point.applying(pixelToValueMatrix)
```

```
}
```

```
/// - Returns: The x and y values in the chart at the given touch point
```

```
/// (x/y). This method transforms pixel coordinates to
```

```
/// coordinates / values in the chart.
```

```
@objc open func valueForTouchPoint(x: CGFloat, y: CGFloat) -> CGPoint
```

```
{
```

```
    return CGPoint(x: x, y: y).applying(pixelToValueMatrix)
```

```
}
```

```
@objc open var valueToPixelMatrix: CGAffineTransform
```

```
{
```

```
    return
```

```
        _matrixValueToPx.concatenating(_viewPortHandler.touchMatrix
```

```

        ).concatenating(_matrixOffset

    )

}

@objc open var pixelToValueMatrix: CGAffineTransform

{

    return valueToPixelMatrix.inverted()

}

}

//

// TransformerHorizontalBarChart.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
@objc(ChartTransformerHorizontalBarChart)
```

```
open class TransformerHorizontalBarChart: Transformer
```

```
{
```

```
    /// Prepares the matrix that contains all offsets.
```

```
    open override func prepareMatrixOffset(inverted: Bool)
```

```
    {
```

```
        if !inverted
```

```
        {
```

```
            _matrixOffset = CGAffineTransform(translationX: _viewPortHandler.offsetLeft, y:  
            _viewPortHandler.chartHeight - _viewPortHandler.offsetBottom)
```

```
        }
```

```
    else
```

```

    {

        _matrixOffset = CGAffineTransform(scaleX: -1.0, y: 1.0)

        _matrixOffset = _matrixOffset.translatedBy(x: -(_viewPortHandler.chartWidth -
_viewPortHandler.offsetRight),

            y: _viewPortHandler.chartHeight - _viewPortHandler.offsetBottom)

    }

}

}

//

// TriangleShapeRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

```



```
import Foundation
```

```
import CoreGraphics
```

```
open class TriangleShapeRenderer : NSObject, IShapeRenderer
```

```
{
```

```
    open func renderShape(
```

```
        context: CGContext,
```

```
        dataSet: IScatterChartDataSet,
```

```
        viewPortHandler: ViewPortHandler,
```

```
        point: CGPoint,
```

```
        color: NSUIColor)
```

```
{
```

```
    let shapeSize = dataSet.scatterShapeSize
```

```
    let shapeHalf = shapeSize / 2.0
```

```
    let shapeHoleSizeHalf = dataSet.scatterShapeHoleRadius
```

```
    let shapeHoleSize = shapeHoleSizeHalf * 2.0
```

```
    let shapeHoleColor = dataSet.scatterShapeHoleColor
```

```
let shapeStrokeSize = (shapeSize - shapeHoleSize) / 2.0
```

```
context.setFillColor(color.cgColor)
```

```
// create a triangle path
```

```
context.beginPath()
```

```
context.move(to: CGPoint(x: point.x, y: point.y - shapeHalf))
```

```
context.addLine(to: CGPoint(x: point.x + shapeHalf, y: point.y + shapeHalf))
```

```
context.addLine(to: CGPoint(x: point.x - shapeHalf, y: point.y + shapeHalf))
```

```
if shapeHoleSize > 0.0
```

```
{
```

```
    context.addLine(to: CGPoint(x: point.x, y: point.y - shapeHalf))
```

```
    context.move(to: CGPoint(x: point.x - shapeHalf + shapeStrokeSize, y: point.y +  
shapeHalf - shapeStrokeSize))
```

```
    context.addLine(to: CGPoint(x: point.x + shapeHalf - shapeStrokeSize, y: point.y  
+ shapeHalf - shapeStrokeSize))
```

```

        context.addLine(to: CGPoint(x: point.x, y: point.y - shapeHalf + shapeStrokeSize))

        context.addLine(to: CGPoint(x: point.x - shapeHalf + shapeStrokeSize, y: point.y
+ shapeHalf - shapeStrokeSize))

    }

    context.closePath()

    context.fillPath()

    if shapeHoleSize > 0.0 && shapeHoleColor != nil
    {

        context.setFillColor(shapeHoleColor!.cgColor)

        // create a triangle path

        context.beginPath()

        context.move(to: CGPoint(x: point.x, y: point.y - shapeHalf + shapeStrokeSize))

        context.addLine(to: CGPoint(x: point.x + shapeHalf - shapeStrokeSize, y: point.y
+ shapeHalf - shapeStrokeSize))

```

```
context.addLine(to: CGPoint(x: point.x - shapeHalf + shapeStrokeSize, y: point.y  
+ shapeHalf - shapeStrokeSize))
```

```
context.closePath()
```

```
context.fillPath()
```

```
}
```

```
}
```

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `TriggerCreator` is a superclass that builds triggers.

```
*/
```

```
import HomeKit
```

```
/**
```

A superclass for all trigger creators.

These classes manage the temporary state of the trigger

and unify some of the saving processes.

```
*/
```

```
class TriggerCreator {
```

```
    // MARK: Properties
```

```
    internal var home: HMHome
```

```
    internal var trigger: HMTrigger?
```

```
    internal var name = ""
```

```
    internal let saveTriggerGroup = DispatchGroup()
```

```
    internal var errors = [Error]()
```

```
/**
```

Initializes a trigger creator from an existing trigger (if it exists),

and the current home.

- parameter trigger: An `HMTrigger` or `nil`, if creation is desired.

- parameter home: The `HMHome` into which this trigger will go.

*/

```
required init(trigger: HMTrigger?, home: HMHome) {
```

```
    self.home = home
```

```
    self.trigger = trigger
```

```
}
```

/**

Completes one of two actions based on the current status of the `trigger` object:

1. Updates the existing trigger.

2. Creates a new trigger.

- parameter name: The name to set for the new or updated trigger.

- parameter actionSets: The new list of action sets to set for the trigger
- parameter completion: The closure to call when all configurations have been completed.

*/

```
func saveTriggerWithName(_ name: String, actionSets: [HMAActionSet], completion:
@escaping (HMTrigger?, [Error]) -> Void) {
```

```
    self.name = name
```

```
    if trigger != nil {
```

```
        // Let the subclass update the trigger.
```

```
        updateTrigger()
```

```
        updateNameIfNecessary()
```

```
        configureWithActionSets(actionSets)
```

```
    }
```

```
    else {
```

```
        self.trigger = newTrigger()
```

```
        saveTriggerGroup.enter()
```

```
        home.addTrigger(trigger!) { error in
```

```
            if let error = error {
```

```

        self.errors.append(error)

    }

    else {

        self.configureWithActionSets(actionSets)

    }

    self.saveTriggerGroup.leave()

}

}

```

```

/*

```

Call the completion block with our event trigger and any accumulated errors from the saving process.

```

*/

```

```

saveTriggerGroup.notify(queue: DispatchQueue.main) {

    self.cleanUp()

    completion(self.trigger, self.errors)

}

```



```
}
```

```
/**
```

Updates the trigger's internals.

Action sets and the trigger name need not be configured.

Implemented by subclasses.

```
*/
```

```
internal func updateTrigger() { }
```

```
/**
```

Creates a new trigger to be added to the home.

Action sets and the trigger name need not be configured.

Implemented by subclasses.

- returns: A new, generated `HMTrigger`.

```
*/
```

```
internal func newTrigger() -> HMTrigger? {
```

```
    return nil
```

```
}
```

```
/**
```

```
    Cleans up an internal structures after the trigger has been saved.
```

```
    Implemented by subclasses.
```

```
*/
```

```
internal func cleanUp() {}
```

```
// MARK: Helper Methods
```

```
/**
```

```
    Syncs the trigger's action sets with the specified array of action sets.
```

- parameter actionSets: Array of `HMActionSet`s to match.

*/

```
private func configureWithActionSets(_ actionSets: [HMActionSet]) {
```

```
    guard let trigger = trigger else { return }
```

```
    /*
```

```
        Save a standard completion handler to use when we either add or remove  
        an action set.
```

```
    */
```

```
    let defaultCompletion: (Error?) -> Void = { error in
```

```
        // Leave the dispatch group, to notify that we've finished this task.
```

```
        if let error = error {
```

```
            self.errors.append(error)
```

```
        }
```

```
        self.saveTriggerGroup.leave()
```

```
    }
```

```

// First pass, remove the action sets that have been deselected.

for actionSet in trigger.actionSets {

    if actionSets.contains(actionSet) {

        continue

    }

    saveTriggerGroup.enter()

    trigger.removeActionSet(actionSet, completionHandler: defaultCompletion)

}


// Second pass, add the new action sets that were just selected.

for actionSet in actionSets {

    if trigger.actionSets.contains(actionSet) {

        continue

    }

    saveTriggerGroup.enter()

    trigger.addActionSet(actionSet, completionHandler: defaultCompletion)

}

```

```

    }

    /// Updates the trigger's name from the stored name, entering and leaving the dispatch
    group if necessary.

    func updateNameIfNecessary() {

        if trigger?.name == self.name {

            return

        }

        saveTriggerGroup.enter()

        trigger?.updateName(name) { error in

            if let error = error {

                self.errors.append(error)

            }

            self.saveTriggerGroup.leave()

        }

    }

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `TriggerViewController` is a superclass which allows users to create triggers.

*/

import UIKit

import HomeKit

/// Represents all possible sections in a `TriggerViewController` subclass.

enum TriggerTableViewSection: Int {

// All triggers have these sections.

case name, enabled, actionSets

// Timer triggers only.

case dateAndTime, recurrence

```
// Location and Characteristic triggers only.
```

```
case conditions
```

```
// Location triggers only.
```

```
case location, region
```

```
// Characteristic triggers only.
```

```
case characteristics
```

```
}
```

```
/**
```

A superclass for all trigger view controllers.

It manages the name, enabled state, and action set components of the view,

as these are shared components.

```
*/
```

```
class TriggerViewController: HMCatalogViewController {
```

```
    // MARK: Types
```

```
    struct Identifiers {
```

```
        static let actionSetCell = "ActionSetCell"
```

```
    }
```

```
    // MARK: Properties
```

```
    @IBOutlet weak var saveButton: UIBarButtonItem!
```

```
    @IBOutlet weak var nameField: UITextField!
```

```
    @IBOutlet weak var enabledSwitch: UISwitch!
```

```
    var trigger: HMTrigger?
```

```
    var triggerCreator: TriggerCreator?
```

```
    /// An internal array of all action sets in the home.
```



```
var actionSets: [HMActionSet]!
```

```
/**
```

```
    An array of all action sets that the user has selected.
```

```
    This will be used to save the trigger when it is finalized.
```

```
*/
```

```
lazy var selectedActionSets = [HMActionSet]()
```

```
// MARK: View Methods
```

```
/// Resets internal data, sets initial UI, and configures the table view.
```

```
override func viewDidLoad() {
```

```
    super.viewDidLoad()
```

```
    let filteredActionSets = home.actionSets.filter { actionSet in
```

```
        return !actionSet.actions.isEmpty
```

```
    }
```

```
actionSets = filteredActionSets.sortByTypeAndLocalizedName()
```

```
tableView.rowHeight = UITableView.automaticDimension
```

```
tableView.estimatedRowHeight = 44.0
```

```
/*
```

If we have a trigger, set the saved properties to the current properties

of the passed-in trigger.

```
*/
```

```
if let trigger = trigger {
```

```
    selectedActionSets = trigger.actionSets
```

```
    nameField.text = trigger.name
```

```
    enabledSwitch.isOn = trigger.isEnabled
```

```
}
```

```
enableSaveButtonIfApplicable()
```

```
tableView.register(UITableViewCell.self,  
Identifiers.actionSetCell)
```

```
forCellReuseIdentifier:
```

```
}
```

```
// MARK: IBAction Methods
```

```
/**
```

```
Any time the name field changed, reevaluate whether or not
```

```
to enable the save button.
```

```
*/
```

```
@IBAction func nameFieldDidChange(_ sender: UITextField) {
```

```
    enableSaveButtonIfApplicable()
```

```
}
```

```
/// Saves the trigger and dismisses this view controller.
```

```
@IBAction func saveAndDismiss() {
```

```
    saveButton.isEnabled = false
```

```
    triggerCreator?.saveTriggerWithName(trimmedName,                      actionSets:
selectedActionSets) { trigger, errors in
```

```
        self.trigger = trigger
```

```
self.saveButton.isEnabled = true
```

```
if !errors.isEmpty {
```

```
    self.displayErrors(errors)
```

```
    return
```

```
}
```

```
self.enableTrigger(self.trigger!) {
```

```
    self.dismiss()
```

```
}
```

```
}
```

```
}
```

```
@IBAction func dismiss() {
```

```
    self.dismiss(animated: true, completion: nil)
```

```
}
```

```
// MARK: Subclass Methods
```

```
/**
```

Generates the section for the index.

This allows for the subclasses to lay out their content in different sections

while still maintaining common code in the `TriggerViewController`.

- parameter index: The index of the section

- returns: The `TriggerTableViewSection` for the provided index.

```
*/
```

```
func sectionForIndex(_ index: Int) -> TriggerTableViewSection? {
```

```
    return nil
```

```
}
```

```
// MARK: Helper Methods
```

```
/// Enable the trigger if necessary.
```

```
func enableTrigger(_ trigger: HMTrigger, completion: @escaping () -> Void) {
```

```
    if trigger.isEnabled == enabledSwitch.isOn {
```

```
        completion()
```

```
        return
```

```
    }
```

```
    trigger.enable(enabledSwitch.isOn) { error in
```

```
        if let error = error {
```

```
            self.displayError(error)
```

```
        }
```

```
    } else {
```

```
        completion()
```

```
    }
```

```
}
```

```
}
```

```
/**
```

Enables the save button if:

1. The name field is not empty, and
2. There will be at least one action set in the trigger after saving.

```
*/
```

```
private fun enableSaveButtonIfApplicable() {
```

```
    saveButton.isEnabled = !trimmedName.isEmpty &&
```

```
        (!selectedActionSets.isEmpty || trigger?.actionSets.count ?? 0 > 0)
```

```
}
```

/// - returns: The name from the `nameField`, stripping newline and whitespace characters.

```
var trimmedName: String {
```

```
    return nameField.text!.trimmingCharacters(in:
        CharacterSet.whitespacesAndNewlines)
```

```
}
```

```
// MARK: Table View Methods
```

```
/// Creates a cell that represents either a selected or unselected action set cell.
```

```
private func tableView(_ tableView: UITableView, actionSetCellForRowAtIndexPath  
indexPath: IndexPath) -> UITableViewCell {
```

```
    let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.actionSetCell,  
for: indexPath)
```

```
    let actionSet = actionSets[indexPath.row]
```

```
    if selectedActionSets.contains(actionSet) {
```

```
        cell.accessoryType = .checkmark
```

```
    }
```

```
    else {
```

```
        cell.accessoryType = .none
```

```
    }
```

```
    cell.textLabel?.text = actionSet.name
```



```

    return cell
}

```

/// Only handles the ActionSets case, defaults to super.

```

override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {

    if sectionForIndex(section) == .actionSets {

        return actionSets?.count ?? 0

    }

    return super.tableView(tableView, numberOfRowsInSection: section)

}

```

/// Only handles the ActionSets case, defaults to super.

```

override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

```

```

if sectionForIndex(indexPath.section) == .actionSets {

    return self.tableView(tableView, actionSetCellForRowAtIndexPath: indexPath)

}

```

```

return super.tableView(tableView, cellForRowAt: indexPath)

}

```

```

/**

```

This is necessary for mixing static and dynamic table view cells.

We return a fake index path because otherwise the superclass's implementation (which does not

know about the extra cells we're adding) will cause an error.

- returns: The superclass's indentationLevel for the first row in the provided section,
instead of the provided row.

```

*/

```

```

override func tableView(_ tableView: UITableView, indentationLevelForRowAt
indexPath: IndexPath) -> Int {

```

```

    let newIndexPath = IndexPath(row: 0, section: indexPath.section)

    return super.tableView(tableView, indentationLevelForRowAt: newIndexPath)

}

/**

Tell the tableView to automatically size the custom rows, while using the superclass's

static sizing for the static cells.

*/

override func tableView(_ tableView: UITableView, heightForRowAt indexPath:
IndexPath) -> CGFloat {

    switch sectionForIndex(indexPath.section) {

        case .name?, .enabled?:

            return super.tableView(tableView, heightForRowAt: indexPath)

        case nil:

            fatalError("Unexpected `TriggerTableViewSection` raw value.")

```

default:

```
        return UITableView.automaticDimension
```

```
    }
```

```
}
```

```
/// Handles row selection for action sets, defaults to super implementation.
```

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
```

```
    tableView.deselectRow(at: indexPath, animated: true)
```

```
    if sectionForIndex(indexPath.section) == .actionSets {
```

```
        self.tableView(tableView, didSelectActionSetAtIndex: indexPath)
```

```
    }
```

```
}
```

```
/**
```

```
    Manages footer titles for higher-level sections. Superclasses should fall back
```

```
    on this implementation after attempting to handle any special trigger sections.
```

```
*/
```

```

    override func tableView(_ tableView: UITableView, titleForFooterInSection section:
Int) -> String? {

        switch sectionForIndex(section) {

            case .actionSets?:

                return NSLocalizedString("When this trigger is activated, it will set these scenes.
You can only select scenes which have at least one action.", comment: "Scene Trigger
Description")

            case .enabled?:

                return NSLocalizedString("This trigger will only activate if it is enabled. You
can disable triggers to temporarily stop them from running.", comment: "Trigger Enabled
Description")

            case nil:

                fatalError("Unexpected `TriggerTableViewSection` raw value.")

            default:

                return super.tableView(tableView, titleForFooterInSection: section)

        }

    }
}

```

```

/**

    Handle selection of an action set cell. If the action set is already part of the selected
    action sets,

        then remove it from the selected list. Otherwise, add it to the selected list.

*/

func tableView(_ tableView: UITableView, didSelectActionSetAtIndex indexPath:
IndexPath) {

    let actionSet = actionSets[indexPath.row]

    if let index = selectedActionSets.firstIndex(of: actionSet) {

        selectedActionSets.remove(at: index)

    }

    else {

        selectedActionSets.append(actionSet)

    }

}

enableSaveButtonIfApplicable()

tableView.reloadRows(at: [indexPath], with: .automatic)

```

```
}
```

```
// MARK: HMHomeDelegate Methods
```

```
/**
```

```
    If our trigger has been removed from the home,
```

```
    dismiss the view controller.
```

```
*/
```

```
func home(_ home: HMHome, didRemoveTrigger trigger: HMTrigger) {
```

```
    if self.trigger == trigger{
```

```
        self.dismiss(animated: true, completion: nil)
```

```
    }
```

```
}
```

```
/// If our trigger has been updated, reload our data.
```

```
func home(_ home: HMHome, didUpdateTrigger trigger: HMTrigger) {
```

```
    if self.trigger == trigger{
```

```
tableView.reloadData()

    }

}

}

/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `UIAlertController+Convenience` methods allow for quick construction of `UIAlertController`s with common structures.

```
*/
```

```
import UIKit
```

```
extension UIAlertController {
```

```
/**
```


A simple `UIAlertController` that prompts for a name, then runs a completion block passing in the name.

- parameter `attributeType`: The type of object that will be named.
- parameter `completion`: A block to call, passing in the provided text.
- parameter `placeholder`: An optional string used as text field's placeholder text.
- parameter `shortType`: An optional string used as to form the alert's action title.

- returns: A `UIAlertController` instance with a `UITextField`, cancel button, and add button.

*/

```
convenience init(attributeType: String, completionHandler: @escaping (_ name: String)
-> Void, placeholder: String? = nil, shortType: String? = nil) {
```

```
    let title = NSLocalizedString("New", comment: "New") + " \(attributeType)"
```

```
    let message = NSLocalizedString("Enter a name.", comment: "Enter a name.")
```

```
    self.init(title: title, message: message, preferredStyle: .alert)
```

```
    self.addTextField { textField in
```

```
        textField.placeholder = placeholder ?? attributeType
```

```
        textField.autocapitalizationType = .words
```

```

    }

    let cancelAction = UIAlertAction(title: NSLocalizedString("Cancel", comment:
"Cancel"), style: .cancel) { action in

        self.dismiss(animated: true, completion: nil)

    }

    let add = NSLocalizedString("Add", comment: "Add")

    let actionTitle = "\(add) \(shortType ?? attributeType)"

    let addNewObject = UIAlertAction(title: actionTitle, style: .default) { action in

        if let name = self.textFields!.first!.text {

            let          trimmedName          =          name.trimmingCharacters(in:
CharacterSet.whitespacesAndNewlines)

            completionHandler(trimmedName)

        }

        self.dismiss(animated: true, completion: nil)

    }

    self.addAction(cancelAction)

    self.addAction(addNewObject)

}

```

```

/**

A simple `UIAlertController` made to show an error message that's passed in.

- parameter body: The body of the alert.

- returns: A `UIAlertController` with an 'Okay' button.

*/

convenience init(title: String, body: String) {

    self.init(title: title, message: body, preferredStyle: .alert)

    let okayAction = UIAlertAction(title: NSLocalizedString("Okay", comment: "Okay"),
style: .default) { action in

        self.dismiss(animated: true, completion: nil)

    }

    self.addAction(okayAction)

}

}

//

```

```

// UIButton.swift

// FirebaseStarterApp

//

// Created by Florian Marcu on 2/21/19.

// Copyright © 2019 Instamobile. All rights reserved.

//

import UIKit

extension UIButton {

    func configure(color: UIColor = .blue, font: UIFont = UIFont.boldSystemFont(ofSize:
12)) {

        self.setTitleColor(color, for: .normal)

        self.titleLabel?.font = font

        self.layer.borderColor = color.cgColor

        self.layer.borderWidth = 1.0

        self.layer.cornerRadius = 5.0

    }

```

```

func configure(icon: UIImage, color: UIColor? = nil) {

    self.setImage(icon, for: .normal)

    if let color = color {

        tint_color = color

    }

}

```

```

func configure(color: UIColor = .blue,

    font: UIFont = UIFont.boldSystemFont(ofSize: 12),

    cornerRadius: CGFloat,

    borderColor: UIColor? = nil,

    backgroundColor: UIColor,

    borderWidth: CGFloat? = nil) {

    self.setTitleColor(color, for: .normal)

    self.titleLabel?.font = font

    self.backgroundColor = backgroundColor

```

```

    if let borderColor = borderColor {

        self.layer.borderColor = borderColor.cgColor

    }

    if let borderWidth = borderWidth {

        self.layer.borderWidth = borderWidth

    }

    self.layer.cornerRadius = cornerRadius

}

}

/*

```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `UIColor+Custom` method provides a generator method for a custom color.

```

*/

```

```
import UIKit
```

```
extension UIColor {
```

```
    /// - returns: A nice blue color which suggests editability.
```

```
    static let editableBlue = UIColor(red:0/255.0, green:122/255.0, blue:255/255.0,  
alpha:1.0)
```

```
}
```

```
//
```

```
// UIColor.swift
```

```
// FirebaseStarterApp
```

```
//
```

```
// Created by Florian Marcu on 2/21/19.
```

```
// Copyright © 2019 Instamobile. All rights reserved.
```

```
//
```

```
import UIKit
```

```

class HelperDarkMode {

    static let mainThemeBackgroundColor: UIColor = {

        if #available(iOS 13.0, *) {

            return UIColor { (traitCollection: UITraitCollection) -> UIColor in

                switch traitCollection.userInterfaceStyle {

                    case

                        .unspecified,

                        .light:

                        return .white

                    case .dark:

                        return UIColor(hexString: "#252525")

                        @unknown default:

                            fatalError()

                }

            }

        } else {

            return UIColor.white

```



```

    }

}0

static let mainThemeForegroundColor: UIColor = UIColor(hexString: "#0997E8")

static let mainTextColor: UIColor = {

    if #available(iOS 13.0, *) {

        return UIColor { (traitCollection: UITraitCollection) -> UIColor in

            switch traitCollection.userInterfaceStyle {

                case

                    .unspecified,

                    .light: return UIColor(hexString: "#464646")

                case .dark: return UIColor(hexString: "#b9b9b9")

                @unknown default:

                    fatalError()

            }

        }

    }

} else {

```

```

        return UIColor(hexString: "#464646")

    }

}

static let mainSubtextColor: UIColor = {

    if #available(iOS 13.0, *) {

        return UIColor { (traitCollection: UITraitCollection) -> UIColor in

            switch traitCollection.userInterfaceStyle {

                case

                    .unspecified,

                    .light: return UIColor(hexString: "#999999")

                case .dark: return UIColor(hexString: "#666666")

                @unknown default:

                    fatalError()

            }

        }

    }

} else {

```

```

        return UIColor(hexString: "#999999")

    }

}

static let statusBarStyle: UIStatusBarStyle = .default

static let hairlineColor: UIColor = {

    if #available(iOS 13.0, *) {

        return UIColor { (traitCollection: UITraitCollection) -> UIColor in

            switch traitCollection.userInterfaceStyle {

                case

                    .unspecified,

                    .light: return UIColor(hexString: "#d6d6d6")

                case .dark: return UIColor(hexString: "#292929")

                @unknown default:

                    fatalError()

            }

        }

    }

```

```

    }

    } else {

        return UIColor(hexString: "#d6d6d6")

    }

}()

```

```

static let colorGray0: UIColor = {

    if #available(iOS 13.0, *) {

        return UIColor { (traitCollection: UITraitCollection) -> UIColor in

            switch traitCollection.userInterfaceStyle {

                case

                    .unspecified,

                    .light: return .black

                case .dark: return .white

                @unknown default:

                    fatalError()

            }

        }

    }

```

```

    }

    } else {

        return .black

    }

}()

```

```

static var colorGray3: UIColor = {

    if #available(iOS 13.0, *) {

        return UIColor { (traitCollection: UITraitCollection) -> UIColor in

            switch traitCollection.userInterfaceStyle {

                case

                    .unspecified,

                    .light: return UIColor(hexString: "#333333")

                case .dark: return UIColor(hexString: "#cccccc")

                @unknown default:

                    fatalError()

            }

        }

    }

```

```

    }

    } else {

        return UIColor(hexString: "#333333")

    }

}()

```

```

static var colorGray9: UIColor = {

    if #available(iOS 13.0, *) {

        return UIColor { (traitCollection: UITraitCollection) -> UIColor in

            switch traitCollection.userInterfaceStyle {

                case

                    .unspecified,

                    .light: return UIColor(hexString: "#f4f4f4")

                case .dark: return UIColor(hexString: "#0b0b0b")

                @unknown default:

                    return UIColor(hexString: "#f4f4f4")

            }

        }

    }

}()

```

```

    }

    } else {

        return UIColor(hexString: "#f4f4f4")

    }

}()

```

```

static func darkModeEnabled() -> Bool {

    return {

        if #available(iOS 13.0, *) {

            let color = UIColor { (traitCollection: UITraitCollection) -> UIColor in

                switch traitCollection.userInterfaceStyle {

                    case

                        .unspecified,

                        .light: return .white

                    case .dark: return .black

                    @unknown default:

                        return .white
                }
            }
        }
    }
}

```

```

        }

    }

    return color.toHexString() == "#000000"

} else {

    return false

}

}()

}

}

```

```

extension UIColor {

    convenience init(hexString: String, alpha: CGFloat = 1.0) {

        let hexString: String = hexString.trimmingCharacters(in:
CharacterSet.whitespacesAndNewlines)

        let scanner = Scanner(string: hexString)

        if (hexString.hasPrefix("#")) {

            scanner.scanLocation = 1

```



```

    }

    var color: UInt32 = 0

    scanner.scanHexInt32(&color)

    let mask = 0x000000FF

    let r = Int(color >> 16) & mask

    let g = Int(color >> 8) & mask

    let b = Int(color) & mask

    let red  = CGFloat(r) / 255.0

    let green = CGFloat(g) / 255.0

    let blue = CGFloat(b) / 255.0

    self.init(red:red, green:green, blue:blue, alpha:alpha)

    }

```

```

func toHexString() -> String {

    var r:CGFloat = 0

    var g:CGFloat = 0

    var b:CGFloat = 0

    var a:CGFloat = 0


    getRed(&r, green: &g, blue: &b, alpha: &a)


    let rgb:Int = (Int)(r*255)<<16 | (Int)(g*255)<<8 | (Int)(b*255)<<0


    return String(format:"#%06x", rgb)

}

}

//

// UIImage.swift

// FirebaseStarterApp

//

```

```
// Created by Florian Marcu on 2/21/19.
```

```
// Copyright © 2019 Instamobile. All rights reserved.
```

```
//
```

```
import UIKit
```

```
extension UIImage {
```

```
    static func localImage(_ name: String, template: Bool = false) -> UIImage {
```

```
        var image = UIImage(named: name)!
```

```
        if template {
```

```
            image = image.withRenderingMode(.alwaysTemplate)
```

```
        }
```

```
        return image
```

```
    }
```

```
}
```

```
/*
```

```
Copyright (C) 2016 Apple Inc. All Rights Reserved.
```

See LICENSE.txt for this sample's licensing information

Abstract:

The `UIStoryboardSegue+IntendedDestination` method allows for the selection of a segue's destination.

*/

import UIKit

extension UIStoryboardSegue {

/// - returns: The intended `UIViewController` from the segue's destination.

var intendedDestinationViewController: UIViewController {

if let navigationController = self.destination as? UINavigationController {

return navigationController.topViewController!

}

return self.destination

}

```
}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The `UITableViewController+Convenience` methods allow for the configuration of a background label.

```
*/
```

```
import HomeKit
```

```
import UIKit
```

```
extension UITableViewController {
```

```
/**
```

Displays or hides a label in the background of the table view.

- parameter message: The String message to display. The message is hidden

if `nil` is provided.

*/

```
func setBackgroundMessage(_ message: String?) {  
  
    if let message = message {  
  
        // Display a message when the table is empty  
  
        let messageLabel = UILabel()  
  
        messageLabel.text = message  
  
        messageLabel.font = .preferredFont(forTextStyle: UIFont.TextStyle.body)  
  
        messageLabel.textColor = .lightGray  
  
        messageLabel.textAlignment = .center  
  
        messageLabel.sizeToFit()  
  
        tableView.backgroundView = messageLabel  
  
        tableView.separatorStyle = .none  
  
    }
```

```

else {

    tableView.backgroundColor = nil

    tableView.separatorStyle = .singleLine

}

}

}

//

// UITextField.swift

// FirebaseStarterApp

//

// Created by Florian Marcu on 2/21/19.

// Copyright © 2019 Instamobile. All rights reserved.

//

import UIKit

extension UITextField {

```

```

func configure(color: UIColor = .blue,

               font: UIFont = UIFont.boldSystemFont(ofSize: 12),

               cornerRadius: CGFloat,

               borderColor: UIColor? = nil,

               backgroundColor: UIColor,

               borderWidth: CGFloat? = nil) {

    if let borderWidth = borderWidth {

        self.layer.borderWidth = borderWidth

    }

    if let borderColor = borderColor {

        self.layer.borderColor = borderColor.cgColor

    }

    self.layer.cornerRadius = cornerRadius

    self.font = font

    self.textColor = color

    self.backgroundColor = backgroundColor

}

```



```
}
```

```
//
```

```
// fewfw.swift
```

```
// FirebaseStarterApp
```

```
//
```

```
// Created by Florian Marcu on 2/21/19.
```

```
// Copyright © 2019 Instamobile. All rights reserved.
```

```
//
```

```
import UIKit
```

```
extension UIViewController {
```

```
    func hideKeyboardWhenTappedAround() {
```

```
        let tap: UITapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(UIViewController.dismissKeyboard))
```

```
        tap.cancelsTouchesInView = false
```

```
view.addGestureRecognizer(tap)

}
```

```
@objc func dismissKeyboard() {

    view.endEditing(true)

}

}
```

```
/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The 'UIViewController+Convenience' methods allow for easy presentation of common views.

```
*/
```

```
import HomeKit
```

```
import UIKit
```

```

extension UIViewController {

    /**

    Displays a `UIAlertController` on the main thread with the error's
    `localizedDescription` at the body.

    - parameter error: The error to display.

    */

    func displayError(_ error: Error) {

        if error is CocoaError {

            let errorCode = HLError(_nsError: error as NSError).code

            if self.presentedViewController != nil || errorCode == .operationCancelled ||
            errorCode == .userDeclinedAddingUser {

                print(error.localizedDescription)

            }

            else {

                self.displayErrorMessage(error.localizedDescription)
            }
        }
    }
}

```

```

    }

}

else {

    self.displayErrorMessage(String(describing: error))

}

}

/**

Displays a collection of errors, separated by newlines.

- parameter errors: An array of `NSError`'s to display.

*/

func displayErrors(_ errors: [Error]) {

    var messages = [String]()

    for error in errors {

        let errorCode = HLError(_nsError: error as NSError).code

        if self.presentedViewController != nil || errorCode == .operationCancelled ||
errorCode == .userDeclinedAddingUser {

```

```

        print(error.localizedDescription)

    }

    else {

        messages.append(error.localizedDescription)

    }

}

if messages.count > 0 {

    // There were errors in the list, reduce the messages into a single one.

    let collectedMessage = messages.reduce("", { (accumulator, message) -> String in

        return accumulator + "\n" + message

    })

    self.displayErrorMessage(collectedMessage)

}

}

```

/// Displays a `UIAlertController` with the passed-in text and an 'Okay' button.

```

func displayMessage(_ title: String, message: String) {

    DispatchQueue.main.async {

        let alert = UIAlertController(title: title, body: message)

        self.present(alert, animated: true, completion: nil)

    }

}

```

/**

Displays `UIAlertController` with a message and a localized "Error" title.

- parameter message: The message to display.

*/

```

private func displayErrorMessage(_ message: String) {

    let errorTitle = NSLocalizedString("Error", comment: "Error")

    displayMessage(errorTitle, message: message)

}

```

```
/**
```

Presents a simple `UIAlertController` with a textField, set up to accept a name. Once the name is entered, the completion handler will be called and the name will be passed in.

- parameter attributeType: The kind of object being added

- parameter completion: The block to run when the user taps the add button.

```
*/
```

```
func presentAddAlertWithAttributeType(_ type: String, placeholder: String? = nil,
shortType: String? = nil, completion: @escaping (String) -> Void) {
```

```
    let alertController = UIAlertController(attributeType: type, completionHandler:
completion, placeholder: placeholder, shortType: shortType)
```

```
    self.present(alertController, animated: true, completion: nil)
```

```
}
```

```
}
```

```
//
```

```
// URLConvertible+URLRequestConvertible.swift
```

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (<http://alamofire.org/>)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE


```
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,  
DAMAGES OR OTHER
```

```
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,  
ARISING FROM,
```

```
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER  
DEALINGS IN
```

```
// THE SOFTWARE.
```

```
//
```

```
import Foundation
```

```
/// Types adopting the `URLConvertible` protocol can be used to construct `URL`s, which  
can then be used to construct
```

```
/// `URLRequests`.
```

```
public protocol URLConvertible {
```

```
    /// Returns a `URL` from the conforming instance or throws.
```

```
    ///
```

```
    /// - Returns: The `URL` created from the instance.
```

```
    /// - Throws: Any error thrown while creating the `URL`.
```

```

    func asURL() throws -> URL

}

extension String: URLConvertible {

    /// Returns a `URL` if `self` can be used to initialize a `URL` instance, otherwise throws.

    ///

    /// - Returns: The `URL` initialized with `self`.

    /// - Throws: An `AFError.invalidURL` instance.

    public func asURL() throws -> URL {

        guard let url = URL(string: self) else { throw AFError.invalidURL(url: self) }

        return url

    }

}

extension URL: URLConvertible {

    /// Returns `self`.

```

```

    public func asURL() throws -> URL { return self }

}

extension URLComponents: URLConvertible {

    /// Returns a `URL` if the `self`'s `url` is not nil, otherwise throws.

    ///

    /// - Returns: The `URL` from the `url` property.

    /// - Throws: An `AFError.invalidURL` instance.

    public func asURL() throws -> URL {

        guard let url = url else { throw AFError.invalidURL(url: self) }

        return url

    }

}

// MARK: -

```

/// Types adopting the `URLRequestConvertible` protocol can be used to safely construct `URLRequest`s.

```
public protocol URLRequestConvertible {
```

```
    /// Returns a `URLRequest` or throws if an `Error` was encountered.
```

```
    ///
```

```
    /// - Returns: A `URLRequest`.
```

```
    /// - Throws: Any error thrown while constructing the `URLRequest`.
```

```
    func asURLRequest() throws -> URLRequest
```

```
}
```

```
extension URLRequestConvertible {
```

```
    /// The `URLRequest` returned by discarding any `Error` encountered.
```

```
    public var urlRequest: URLRequest? { return try? asURLRequest() }
```

```
}
```

```
extension URLRequest: URLRequestConvertible {
```

```
    /// Returns `self`.
```

```
    public func asURLRequest() throws -> URLRequest { return self }
```

```
}
```

```
// MARK: -
```

```
extension URLRequest {
```

```
    /// Creates an instance with the specified `url`, `method`, and `headers`.
```

```
    ///
```

```
    /// - Parameters:
```

```
    /// - url: The `URLConvertible` value.
```

```
    /// - method: The `HTTPMethod`.
```

```
    /// - headers: The `HTTPHeaders`, `nil` by default.
```

```
    /// - Throws: Any error thrown while converting the `URLConvertible` to a `URL`.
```

```
    public init(url: URLConvertible, method: HTTPMethod, headers: HTTPHeaders? = nil)
    throws {
```

```
        let url = try url.asURL()
```

```
        self.init(url: url)
```

```

    httpMethod = method.rawValue

    allHTTPHeaderFields = headers?.dictionary

}

}

//

// URLEncodedFormEncoder.swift

//

// Copyright (c) 2019 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

```

```
// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
// EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
// MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
// EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
// ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
// DEALINGS IN

// THE SOFTWARE.

//
```

```
import Foundation
```

```
/// An object that encodes instances into URL-encoded query strings.
```

///

/// There is no published specification for how to encode collection types. By default, the convention of appending

/// `[ ]` to the key for array values (`foo[]=1&foo[]=2`), and appending the key surrounded by square brackets for

/// nested dictionary values (`foo[bar]=baz`) is used. Optionally, `ArrayEncoding` can be used to omit the

/// square brackets appended to array keys.

///

/// `BoolEncoding` can be used to configure how `Bool` values are encoded. The default behavior is to encode

/// `true` as 1 and `false` as 0.

///

/// `DateEncoding` can be used to configure how `Date` values are encoded. By default, the `.deferredToDate`

/// strategy is used, which formats dates from their structure.

///

/// `SpaceEncoding` can be used to configure how spaces are encoded. Modern encodings use percent replacement (`%20`),

/// while older encodings may expect spaces to be replaced with `+`.


```

///

/// This type is largely based on Vapor's [`url-encoded-form`](https://github.com/vapor/url-
encoded-form) project.

public final class URLEncodedFormEncoder {

    /// Encoding to use for `Array` values.

    public enum ArrayEncoding {

        /// An empty set of square brackets ("[]") are appended to the key for every value. This
        is the default encoding.

        case brackets

        /// No brackets are appended to the key and the key is encoded as is.

        case noBrackets

        /// Encodes the key according to the encoding.

        ///

        /// - Parameter key: The `key` to encode.

        /// - Returns: The encoded key.

        func encode(_ key: String) -> String {

            switch self {

```

```

        case .brackets: return "\(key)[]"

        case .noBrackets: return key

    }

}

}

```

/// Encoding to use for `Bool` values.

```
public enum BoolEncoding {
```

```
    /// Encodes `true` as `1`, `false` as `0`.
```

```
    case numeric
```

```
    /// Encodes `true` as "true", `false` as "false". This is the default encoding.
```

```
    case literal
```

```
    /// Encodes the given `Bool` as a `String`.
```

```
    ///
```

```
    /// - Parameter value: The `Bool` to encode.
```

```
    ///
```

/// - Returns: The encoded `String`.

```
func encode(_ value: Bool) -> String {  
  
    switch self {  
  
        case .numeric: return value ? "1" : "0"  
  
        case .literal: return value ? "true" : "false"  
  
    }  
  
}  
  
}
```

/// Encoding to use for `Data` values.

```
public enum DataEncoding {  
  
    /// Defers encoding to the `Data` type.  
  
    case deferredToData  
  
    /// Encodes `Data` as a Base64-encoded string. This is the default encoding.  
  
    case base64  
  
    /// Encode the `Data` as a custom value encoded by the given closure.  
  
    case custom((Data) throws -> String)
```

```

    /// Encodes `Data` according to the encoding.

    ///

    /// - Parameter data: The `Data` to encode.

    ///

    /// - Returns:      The encoded `String`, or `nil` if the `Data` should be encoded
according to its

    ///      `Encodable` implementation.

    func encode(_ data: Data) throws -> String? {

        switch self {

        case .deferredToData: return nil

        case .base64: return data.base64EncodedString()

        case let .custom(encoding): return try encoding(data)

        }

    }

}

/// Encoding to use for `Date` values.

```

```

public enum DateEncoding {

    /// ISO8601 and RFC3339 formatter.

    private static let iso8601Formatter: ISO8601DateFormatter = {

        let formatter = ISO8601DateFormatter()

        formatter.formatOptions = .withInternetDateTime

        return formatter

    }()

    /// Defers encoding to the `Date` type. This is the default encoding.

    case deferredToDate

    /// Encodes `Date`'s as seconds since midnight UTC on January 1, 1970.

    case secondsSince1970

    /// Encodes `Date`'s as milliseconds since midnight UTC on January 1, 1970.

    case millisecondsSince1970

    /// Encodes `Date`'s according to the ISO8601 and RFC3339 standards.

    case iso8601

    /// Encodes `Date`'s using the given `DateFormatter`.

```

```

case formatted(DateFormatter)

/// Encodes `Date`s using the given closure.

case custom((Date) throws -> String)

/// Encodes the date according to the encoding.

///

/// - Parameter date: The `Date` to encode.

///

/// - Returns:      The encoded `String`, or `nil` if the `Date` should be encoded
according to its

///      `Encodable` implementation.

func encode(_ date: Date) throws -> String? {

    switch self {

    case .deferredToDate:

        return nil

    case .secondsSince1970:

        return String(date.timeIntervalSince1970)

    case .millisecondsSince1970:

```

```

        return String(date.timeIntervalSince1970 * 1000.0)

    case .iso8601:

        return DateEncoding.iso8601Formatter.string(from: date)

    case let .formatted(formatter):

        return formatter.string(from: date)

    case let .custom(closure):

        return try closure(date)

    }

}

}

/// Encoding to use for keys.

///

/// This type is derived from [JSONEncoder's
`KeyEncodingStrategy`](https://github.com/apple/swift/blob/6aa313b8dd5f05135f7f878e
ccc1db6f9fbe34ff/stdlib/public/Darwin/Foundation/JSONEncoder.swift#L128)

/// and [XMLEncoder's
`KeyEncodingStrategy`](https://github.com/MaxDesiatov/XMLCoder/blob/master/Sourc
es/XMLCoder/Encoder/XMLEncoder.swift#L102).

```

```

public enum KeyEncoding {

    /// Use the keys specified by each type. This is the default encoding.

    case useDefaultKeys

    /// Convert from "camelCaseKeys" to "snake_case_keys" before writing a key.

    ///

    /// Capital characters are determined by testing membership in

    /// `CharacterSet.uppercaseLetters` and `CharacterSet.lowercaseLetters`

    /// (Unicode General Categories Lu and Lt).

    /// The conversion to lower case uses `Locale.system`, also known as

    /// the ICU "root" locale. This means the result is consistent

    /// regardless of the current user's locale and language preferences.

    ///

    /// Converting from camel case to snake case:

    /// 1. Splits words at the boundary of lower-case to upper-case

    /// 2. Inserts `_` between words

    /// 3. Lowercases the entire string

    /// 4. Preserves starting and ending `_`.

```



```
///
```

```
/// For example, `oneTwoThree` becomes `one_two_three`. `_oneTwoThree`  
becomes `_one_two_three`.
```

```
///
```

```
/// - Note: Using a key encoding strategy has a nominal performance cost, as each  
string key has to be converted.
```

```
case convertToSnakeCase
```

```
/// Same as convertToSnakeCase, but using `-` instead of `_`.
```

```
/// For example `oneTwoThree` becomes `one-two-three`.
```

```
case convertToKebabCase
```

```
/// Capitalize the first letter only.
```

```
/// For example `oneTwoThree` becomes `OneTwoThree`.
```

```
case capitalized
```

```
/// Uppercase all letters.
```

```
/// For example `oneTwoThree` becomes `ONETWOTHREE`.
```

```
case uppercased
```

```
/// Lowercase all letters.
```

```
/// For example `oneTwoThree` becomes `onetwothree`.
```

```
case lowercased
```

```
/// A custom encoding using the provided closure.
```

```
case custom((String) -> String)
```

```
func encode(_ key: String) -> String {
```

```
    switch self {
```

```
        case .useDefaultKeys: return key
```

```
        case .convertToSnakeCase: return convertToSnakeCase(key)
```

```
        case .convertToKebabCase: return convertToKebabCase(key)
```

```
        case .capitalized: return String(key.prefix(1).uppercased() + key.dropFirst())
```

```
        case .uppercased: return key.uppercased()
```

```
        case .lowercased: return key.lowercased()
```

```
        case let .custom(encoding): return encoding(key)
```

```
    }
```

```
}
```

```
private func convertToSnakeCase(_ key: String) -> String {
```

```
    return convert(key, usingSeparator: "_")
}
```

```
private fun convertToKebabCase(_ key: String) -> String {

    return convert(key, usingSeparator: "-")

}
```

```
private fun convert(_ key: String, usingSeparator separator: String) -> String {

    guard !key.isEmpty else { return key }

}
```

```
var words: [Range<String.Index>] = []
```

```
// The general idea of this algorithm is to split words on
```

```
// transition from lower to upper case, then on transition of >1
```

```
// upper case characters to lowercase
```

```
//
```

```
// myProperty -> my_property
```

```
// myURLProperty -> my_url_property
```

```

//

// It is assumed, per Swift naming conventions, that the first character of the key is
lowercase.

var wordStart = key.startIndex

var searchRange = key.index(after: wordStart)..

```

```

    }

    // Is the next lowercase letter more than 1 after the uppercase?

    // If so, we encountered a group of uppercase letters that we

    // should treat as its own word

    let nextCharacterAfterCapital = key.index(after: upperCaseRange.lowerBound)

    if lowerCaseRange.lowerBound == nextCharacterAfterCapital {

        // The next character after capital is a lower case character and therefore not a
        word boundary.

        // Continue searching for the next upper case for the boundary.

        wordStart = upperCaseRange.lowerBound

    } else {

        // There was a range of >1 capital letters. Turn those into a word, stopping at
        the capital before the lower case character.

        let beforeLowerIndex = key.index(before: lowerCaseRange.lowerBound)

        words.append(upperCaseRange.lowerBound..

```

```

        wordStart = beforeLowerIndex

    }

    searchRange = lowerCaseRange.upperBound..

```

```

    /// Encodes spaces as `+`,

    case plusReplaced

    /// Encodes the string according to the encoding.

    ///

    /// - Parameter string: The `String` to encode.

    ///

    /// - Returns:      The encoded `String`.

    func encode(_ string: String) -> String {

        switch self {

            case .percentEscaped: return string.replacingOccurrences(of: " ", with: "%20")

            case .plusReplaced: return string.replacingOccurrences(of: " ", with: "+")

            }

        }

    }

    /// `URLEncodedFormEncoder` error.

```

```

public enum Error: Swift.Error {

    /// An invalid root object was created by the encoder. Only keyed values are valid.

    case invalidRootObject(String)

    var localizedDescription: String {

        switch self {

            case let .invalidRootObject(object):

                return "URLEncodedFormEncoder requires keyed root object. Received \(object)
instead."

            }

        }

    }

    /// Whether or not to sort the encoded key value pairs.

    ///

    /// - Note: This setting ensures a consistent ordering for all encodings of the same
parameters. When set to `false`,

    ///     encoded `Dictionary` values may have a different encoded order each time they're
encoded due to

```



```
///      `Dictionary`'s random storage order, but `Encodable` types will maintain their  
encoded order.
```

```
public let alphabetizeKeyValuePairs: Bool
```

```
/// The `ArrayEncoding` to use.
```

```
public let arrayEncoding: ArrayEncoding
```

```
/// The `BoolEncoding` to use.
```

```
public let boolEncoding: BoolEncoding
```

```
/// The `DataEncoding` to use.
```

```
public let dataEncoding: DataEncoding
```

```
/// The `DateEncoding` to use.
```

```
public let dateEncoding: DateEncoding
```

```
/// The `KeyEncoding` to use.
```

```
public let keyEncoding: KeyEncoding
```

```
/// The `SpaceEncoding` to use.
```

```
public let spaceEncoding: SpaceEncoding
```

```
/// The `CharacterSet` of allowed (non-escaped) characters.
```

```
public var allowedCharacters: CharacterSet
```

```

/// Creates an instance from the supplied parameters.

///

/// - Parameters:

///   - alphabetizeKeyValuePairs: Whether or not to sort the encoded key value pairs.
  `true` by default.

///   - arrayEncoding:      The `ArrayEncoding` to use. `.brackets` by default.

///   - boolEncoding:       The `BoolEncoding` to use. `.numeric` by default.

///   - dataEncoding:       The `DataEncoding` to use. `.base64` by default.

///   - dateEncoding:       The `DateEncoding` to use. `.deferredToDate` by default.

///   - keyEncoding:        The `KeyEncoding` to use. `.useDefaultKeys` by default.

///   - spaceEncoding:      The `SpaceEncoding` to use. `.percentEscaped` by default.

///   - allowedCharacters:   The `CharacterSet` of allowed (non-escaped) characters.
  `.afURLQueryAllowed` by
  default.

public init(alphabetizeKeyValuePairs: Bool = true,

            arrayEncoding: ArrayEncoding = .brackets,

            boolEncoding: BoolEncoding = .numeric,

            dataEncoding: DataEncoding = .base64,

```

```

        dateEncoding: DateEncoding = .deferredToDate,

        keyEncoding: KeyEncoding = .useDefaultKeys,

        spaceEncoding: SpaceEncoding = .percentEscaped,

        allowedCharacters: CharacterSet = .afURLQueryAllowed) {

    self.alphabetizeKeyValuePairs = alphabetizeKeyValuePairs

    self.arrayEncoding = arrayEncoding

    self.boolEncoding = boolEncoding

    self.dataEncoding = dataEncoding

    self.dateEncoding = dateEncoding

    self.keyEncoding = keyEncoding

    self.spaceEncoding = spaceEncoding

    self.allowedCharacters = allowedCharacters

}

func encode(_ value: Encodable) throws -> URLEncodedFormComponent {

    let context = URLEncodedFormContext(.object([]))

    let encoder = _URLEncodedFormEncoder(context: context,

```

```

        boolEncoding: boolEncoding,

        dataEncoding: dataEncoding,

        dateEncoding: dateEncoding)

    try value.encode(to: encoder)

    return context.component
}

/// Encodes the `value` as a URL form encoded `String`.

///

/// - Parameter value: The `Encodable` value.`

///

/// - Returns:      The encoded `String`.

/// - Throws:      An `Error` or `EncodingError` instance if encoding fails.

public func encode(_ value: Encodable) throws -> String {

    let component: URLEncodedFormComponent = try encode(value)

```

```

guard case let .object(object) = component else {

    throw Error.invalidRootObject("\(component)")

}

```

```

let    serializer    =    URLEncodedFormSerializer(alphabetizeKeyValuePairs:
alphabetizeKeyValuePairs,

                                arrayEncoding: arrayEncoding,

                                keyEncoding: keyEncoding,

                                spaceEncoding: spaceEncoding,

                                allowedCharacters: allowedCharacters)

let query = serializer.serialize(object)


return query

}

```

/// Encodes the value as `Data`. This is performed by first creating an encoded `String` and then returning the

```

/// `.utf8` data.

```

```

///

/// - Parameter value: The `Encodable` value.

///

/// - Returns:      The encoded `Data`.

///

/// - Throws:      An `Error` or `EncodingError` instance if encoding fails.

public func encode(_ value: Encodable) throws -> Data {

    let string: String = try encode(value)

    return Data(string.utf8)

}

}

final class _URLEncodedFormEncoder {

    var codingPath: [CodingKey]

    // Returns an empty dictionary, as this encoder doesn't support userInfo.

    var userInfo: [CodingUserInfoKey: Any] { return [:] }

```

```

let context: URLEncodedFormContext

private let boolEncoding: URLEncodedFormEncoder.BoolEncoding

private let dataEncoding: URLEncodedFormEncoder.DataEncoding

private let dateEncoding: URLEncodedFormEncoder.DateEncoding


init(context: URLEncodedFormContext,

    codingPath: [CodingKey] = [],

    boolEncoding: URLEncodedFormEncoder.BoolEncoding,

    dataEncoding: URLEncodedFormEncoder.DataEncoding,

    dateEncoding: URLEncodedFormEncoder.DateEncoding) {

    self.context = context

    self.codingPath = codingPath

    self.boolEncoding = boolEncoding

    self.dataEncoding = dataEncoding

    self.dateEncoding = dateEncoding

```

```
}  
  
}
```

```
extension _URLEncodedFormEncoder: Encoder {  
  
    func container<Key>(keyedBy type: Key.Type) -> KeyedEncodingContainer<Key>  
    where Key: CodingKey {  
  
        let container = _URLEncodedFormEncoder.KeyedContainer<Key>(context: context,  
  
                                                                    codingPath: codingPath,  
  
                                                                    boolEncoding: boolEncoding,  
  
                                                                    dataEncoding: dataEncoding,  
  
                                                                    dateEncoding: dateEncoding)  
  
        return KeyedEncodingContainer(container)  
  
    }
```

```
func unkeyedContainer() -> UnkeyedEncodingContainer {  
  
    return _URLEncodedFormEncoder.UnkeyedContainer(context: context,  
  
                                                    codingPath: codingPath,  
  
                                                    boolEncoding: boolEncoding,
```



```

        dataEncoding: dataEncoding,

        dateEncoding: dateEncoding)

    }

func singleValueContainer() -> SingleValueEncodingContainer {

    return _URLEncodedFormEncoder.SingleValueContainer(context: context,

        codingPath: codingPath,

        boolEncoding: boolEncoding,

        dataEncoding: dataEncoding,

        dateEncoding: dateEncoding)

    }

}

```

```

final class URLEncodedFormContext {

    var component: URLEncodedFormComponent

    init(_ component: URLEncodedFormComponent) {

```

```

        self.component = component
    }
}

```

```

enum URLEncodedFormComponent {

    typealias Object = [(key: String, value: URLEncodedFormComponent)]

    case string(String)

    case array([URLEncodedFormComponent])

    case object(Object)

    /// Converts self to an `[URLEncodedFormData]` or returns `nil` if not convertible.

    var array: [URLEncodedFormComponent]? {

        switch self {

            case let .array(array): return array

            default: return nil

        }
    }
}

```

```
}
```

```
/// Converts self to an `Object` or returns `nil` if not convertible.
```

```
var object: Object? {
```

```
    switch self {
```

```
        case let .object(object): return object
```

```
        default: return nil
```

```
    }
```

```
}
```

```
/// Sets self to the supplied value at a given path.
```

```
///
```

```
///     data.set(to: "hello", at: ["path", "to", "value"])
```

```
///
```

```
/// - parameters:
```

```
///     - value: Value of `Self` to set at the supplied path.
```

```
///     - path: `CodingKey` path to update with the supplied value.
```

```

public mutating func set(to value: URLEncodedFormComponent, at path: [CodingKey])
{

    set(&self, to: value, at: path)

}

```

```

/// Recursive backing method to `set(to:at:)`.

```

```

private func set(_ context: inout URLEncodedFormComponent, to value:
URLEncodedFormComponent, at path: [CodingKey]) {

    guard path.count >= 1 else {

        context = value

        return

    }

```

```

    let end = path[0]

```

```

    var child: URLEncodedFormComponent

```

```

    switch path.count {

```

```

    case 1:

```

```

        child = value

```

case 2....:

```
    if let index = end.intValue {

        let array = context.array ?? []

        if array.count > index {

            child = array[index]

        } else {

            child = .array([])

        }

        set(&child, to: value, at: Array(path[1...]))

    } else {

        child = context.object?.first { $0.key ==
end.stringValue }?.value ?? .object(.init())

        set(&child, to: value, at: Array(path[1...]))

    }

    default: fatalError("Unreachable")

}
```



```
    if let index = end.intValue {
```

```

if var array = context.array {

    if array.count > index {

        array[index] = child

    } else {

        array.append(child)

    }

    context = .array(array)

} else {

    context = .array([child])

}

} else {

    if var object = context.object {

        if let index = object.firstIndex(where: { $0.key == end.stringValue }) {

            object[index] = (key: end.stringValue, value: child)

        } else {

            object.append((key: end.stringValue, value: child))

        }

    }

}

```

```

        context = .object(object)

    } else {

        context = .object([(key: end.stringValue, value: child)])

    }

}

}

}

}

```

```

struct AnyCodingKey: CodingKey, Hashable {

```

```

    let stringValue: String

```

```

    let intValue: Int?

```

```

    init?(stringValue: String) {

```

```

        self.stringValue = stringValue

```

```

        intValue = nil

```

```

    }

```

```

init?(intValue: Int) {

    stringValue = "\(intValue)"

    self.intValue = intValue

}

init<Key>(_ base: Key) where Key: CodingKey {

    if let intValue = base.intValue {

        self.init(intValue: intValue)!

    } else {

        self.init(stringValue: base.stringValue)!

    }

}

}

extension _URLEncodedFormEncoder {

    final class KeyedContainer<Key> where Key: CodingKey {

        var codingPath: [CodingKey]
    }
}

```



```

private let context: URLEncodedFormContext

private let boolEncoding: URLEncodedFormEncoder.BoolEncoding

private let dataEncoding: URLEncodedFormEncoder.DataEncoding

private let dateEncoding: URLEncodedFormEncoder.DateEncoding


init(context: URLEncodedFormContext,

    codingPath: [CodingKey],

    boolEncoding: URLEncodedFormEncoder.BoolEncoding,

    dataEncoding: URLEncodedFormEncoder.DataEncoding,

    dateEncoding: URLEncodedFormEncoder.DateEncoding) {

    self.context = context

    self.codingPath = codingPath

    self.boolEncoding = boolEncoding

    self.dataEncoding = dataEncoding

    self.dateEncoding = dateEncoding

}

```

```

private func nestedCodingPath(for key: CodingKey) -> [CodingKey] {

    return codingPath + [key]

}

}

}

extension _URLEncodedFormEncoder.KeyedContainer:
KeyedEncodingContainerProtocol {

    func encodeNil(forKey key: Key) throws {

        let context = EncodingError.Context(codingPath: codingPath,

                                            debugDescription: "URLEncodedFormEncoder cannot encode
nil values.")

        throw EncodingError.invalidValue("\(key): nil", context)

    }

    func encode<T>(_ value: T, forKey key: Key) throws where T: Encodable {

        var container = nestedSingleValueEncoder(for: key)

```

```
        try container.encode(value)
    }
}
```

```
func nestedSingleValueEncoder(for key: Key) -> SingleValueEncodingContainer {

    let container = _URLEncodedFormEncoder.SingleValueContainer(context: context,

                                                                codingPath: nestedCodingPath(for: key),

                                                                boolEncoding: boolEncoding,

                                                                dataEncoding: dataEncoding,

                                                                dateEncoding: dateEncoding)

    return container

}
```

```
func nestedUnkeyedContainer(forKey key: Key) -> UnkeyedEncodingContainer {

    let container = _URLEncodedFormEncoder.UnkeyedContainer(context: context,

                                                            codingPath: nestedCodingPath(for: key),

                                                            boolEncoding: boolEncoding,
```

```

        dataEncoding: dataEncoding,

        dateEncoding: dateEncoding)

    return container

}

func nestedContainer<NestedKey>(keyedBy keyType: NestedKey.Type, forKey key:
Key) -> KeyedEncodingContainer<NestedKey> where NestedKey: CodingKey {

    let container = _URLEncodedFormEncoder.KeyedContainer<NestedKey>(context:
context,

        codingPath: nestedCodingPath(for: key),

        boolEncoding: boolEncoding,

        dataEncoding: dataEncoding,

        dateEncoding: dateEncoding)

    return KeyedEncodingContainer(container)

}

```

```

func superEncoder() -> Encoder {

    return _URLEncodedFormEncoder(context: context,

                                   codingPath: codingPath,

                                   boolEncoding: boolEncoding,

                                   dataEncoding: dataEncoding,

                                   dateEncoding: dateEncoding)

}

func superEncoder(forKey key: Key) -> Encoder {

    return _URLEncodedFormEncoder(context: context,

                                   codingPath: nestedCodingPath(for: key),

                                   boolEncoding: boolEncoding,

                                   dataEncoding: dataEncoding,

                                   dateEncoding: dateEncoding)

}
}

```

```

extension _URLEncodedFormEncoder {

  final class SingleValueContainer {

    var codingPath: [CodingKey]

    private var canEncodeNewValue = true

    private let context: URLEncodedFormContext

    private let boolEncoding: URLEncodedFormEncoder.BoolEncoding

    private let dataEncoding: URLEncodedFormEncoder.DataEncoding

    private let dateEncoding: URLEncodedFormEncoder.DateEncoding

    init(context: URLEncodedFormContext,

         codingPath: [CodingKey],

         boolEncoding: URLEncodedFormEncoder.BoolEncoding,

         dataEncoding: URLEncodedFormEncoder.DataEncoding,

         dateEncoding: URLEncodedFormEncoder.DateEncoding) {

      self.context = context
    }
  }
}

```

```

        self.codingPath = codingPath

        self.boolEncoding = boolEncoding

        self.dataEncoding = dataEncoding

        self.dateEncoding = dateEncoding

    }

    private func checkCanEncode(value: Any?) throws {

        guard canEncodeNewValue else {

            let context = EncodingError.Context(codingPath: codingPath,

                                                debugDescription: "Attempt to encode value through
single value container when previously value already encoded.")

            throw EncodingError.invalidValue(value as Any, context)

        }

    }

}

```

```

extension _URLEncodedFormEncoder.SingleValueContainer:
SingleValueEncodingContainer {

    func encodeNil() throws {

        try checkCanEncode(value: nil)

        defer { canEncodeNewValue = false }

        let context = EncodingError.Context(codingPath: codingPath,

                                             debugDescription: "URLEncodedFormEncoder cannot encode
nil values.")

        throw EncodingError.invalidValue("nil", context)

    }

    func encode(_ value: Bool) throws {

        try encode(value, as: String(boolEncoding.encode(value)))

    }

    func encode(_ value: String) throws {

        try encode(value, as: value)

    }

```



```
}
```

```
func encode(_ value: Double) throws {  
  
    try encode(value, as: String(value))
```

```
}
```

```
func encode(_ value: Float) throws {  
  
    try encode(value, as: String(value))
```

```
}
```

```
func encode(_ value: Int) throws {  
  
    try encode(value, as: String(value))
```

```
}
```

```
func encode(_ value: Int8) throws {  
  
    try encode(value, as: String(value))
```

```
}
```

```
func encode(_ value: Int16) throws {  
  
    try encode(value, as: String(value))  
  
}
```

```
func encode(_ value: Int32) throws {  
  
    try encode(value, as: String(value))  
  
}
```

```
func encode(_ value: Int64) throws {  
  
    try encode(value, as: String(value))  
  
}
```

```
func encode(_ value: UInt) throws {  
  
    try encode(value, as: String(value))  
  
}
```

```
func encode(_ value: UInt8) throws {  
  
    try encode(value, as: String(value))  
  
}
```

```
func encode(_ value: UInt16) throws {  
  
    try encode(value, as: String(value))  
  
}
```

```
func encode(_ value: UInt32) throws {  
  
    try encode(value, as: String(value))  
  
}
```

```
func encode(_ value: UInt64) throws {  
  
    try encode(value, as: String(value))  
  
}
```

```
private func encode<T>(_ value: T, as string: String) throws where T: Encodable {
```

```

try checkCanEncode(value: value)

defer { canEncodeNewValue = false }

context.component.set(to: .string(string), at: codingPath)

}

```

```

func encode<T>(_ value: T) throws where T: Encodable {

    switch value {

    case let date as Date:

        guard let string = try dateEncoding.encode(date) else {

            try attemptToEncode(value)

            return

        }

```

```

        try encode(value, as: string)

    case let data as Data:

        guard let string = try dataEncoding.encode(data) else {

```

```

        try attemptToEncode(value)

        return

    }

```

```

        try encode(value, as: string)

```

```

default:

```

```

        try attemptToEncode(value)

    }

```

```

}

```

```

private func attemptToEncode<T>(_ value: T) throws where T: Encodable {

```

```

    try checkCanEncode(value: value)

```

```

    defer { canEncodeNewValue = false }

```

```

let encoder = _URLEncodedFormEncoder(context: context,

```

```

    codingPath: codingPath,

```

```

    boolEncoding: boolEncoding,

```

```

        dataEncoding: dataEncoding,

        dateEncoding: dateEncoding)

    try value.encode(to: encoder)

}

}

extension _URLEncodedFormEncoder {

    final class UnkeyedContainer {

        var codingPath: [CodingKey]

        var count = 0

        var nestedCodingPath: [CodingKey] {

            return codingPath + [AnyCodingKey(intValue: count)!]

        }

        private let context: URLEncodedFormContext

        private let boolEncoding: URLEncodedFormEncoder.BoolEncoding

```

```

private let dataEncoding: URLEncodedFormEncoder.DataEncoding

private let dateEncoding: URLEncodedFormEncoder.DateEncoding


init(context: URLEncodedFormContext,

    codingPath: [CodingKey],

    boolEncoding: URLEncodedFormEncoder.BoolEncoding,

    dataEncoding: URLEncodedFormEncoder.DataEncoding,

    dateEncoding: URLEncodedFormEncoder.DateEncoding) {

    self.context = context

    self.codingPath = codingPath

    self.boolEncoding = boolEncoding

    self.dataEncoding = dataEncoding

    self.dateEncoding = dateEncoding

}

}

}

```

```

extension _URLEncodedFormEncoder.UnkeyedContainer: UnkeyedEncodingContainer {

    func encodeNil() throws {

        let context = EncodingError.Context(codingPath: codingPath,

                                             debugDescription: "URLEncodedFormEncoder cannot encode
nil values.")

        throw EncodingError.invalidValue("nil", context)

    }

    func encode<T>(_ value: T) throws where T: Encodable {

        var container = nestedSingleValueContainer()

        try container.encode(value)

    }

    func nestedSingleValueContainer() -> SingleValueEncodingContainer {

        defer { count += 1 }

        return _URLEncodedFormEncoder.SingleValueContainer(context: context,

                                                             codingPath: nestedCodingPath,

```



```

        boolEncoding: boolEncoding,

        dataEncoding: dataEncoding,

        dateEncoding: dateEncoding)

    }

func nestedContainer<NestedKey>(keyedBy keyType: NestedKey.Type) ->
KeyedEncodingContainer<NestedKey> where NestedKey: CodingKey {

    defer { count += 1 }

    let container = _URLEncodedFormEncoder.KeyedContainer<NestedKey>(context:
context,

        codingPath: nestedCodingPath,

        boolEncoding: boolEncoding,

        dataEncoding: dataEncoding,

        dateEncoding: dateEncoding)

    return KeyedEncodingContainer(container)

}

```

```

func nestedUnkeyedContainer() -> UnkeyedEncodingContainer {

    defer { count += 1 }

    return _URLEncodedFormEncoder.UnkeyedContainer(context: context,

                                                    codingPath: nestedCodingPath,

                                                    boolEncoding: boolEncoding,

                                                    dataEncoding: dataEncoding,

                                                    dateEncoding: dateEncoding)

}

```

```

func superEncoder() -> Encoder {

    defer { count += 1 }

    return _URLEncodedFormEncoder(context: context,

                                    codingPath: codingPath,

                                    boolEncoding: boolEncoding,

                                    dataEncoding: dataEncoding,

```

```

        dateEncoding: dateEncoding)

    }

}

final class URLEncodedFormSerializer {

    private let alphabetizeKeyValuePairs: Bool

    private let arrayEncoding: URLEncodedFormEncoder.ArrayEncoding

    private let keyEncoding: URLEncodedFormEncoder.KeyEncoding

    private let spaceEncoding: URLEncodedFormEncoder.SpaceEncoding

    private let allowedCharacters: CharacterSet

    init(alphabetizeKeyValuePairs: Bool,

        arrayEncoding: URLEncodedFormEncoder.ArrayEncoding,

        keyEncoding: URLEncodedFormEncoder.KeyEncoding,

        spaceEncoding: URLEncodedFormEncoder.SpaceEncoding,

        allowedCharacters: CharacterSet) {

        self.alphabetizeKeyValuePairs = alphabetizeKeyValuePairs

```

```

self.arrayEncoding = arrayEncoding

self.keyEncoding = keyEncoding

self.spaceEncoding = spaceEncoding

self.allowedCharacters = allowedCharacters

}

func serialize(_ object: URLEncodedFormComponent.Object) -> String {

    var output: [String] = []

    for (key, component) in object {

        let value = serialize(component, forKey: key)

        output.append(value)

    }

    output = alphabetizeKeyValuePairs ? output.sorted() : output

    return output.joinedWithAmpersands()

}

```

```

func serialize(_ component: URLEncodedFormComponent, forKey key: String) ->
String {

    switch component {

        case let .string(string): return
            "\"(escape(keyEncoding.encode(key)))=\"(escape(string))\""

        case let .array(array): return serialize(array, forKey: key)

        case let .object(object): return serialize(object, forKey: key)

    }

}

```

```

func serialize(_ object: URLEncodedFormComponent.Object, forKey key: String) ->
String {

    var segments: [String] = object.map { subKey, value in

        let keyPath = "\"(subKey)\""

        return serialize(value, forKey: key + keyPath)

    }

    segments = alphabetizeKeyValuePairs ? segments.sorted() : segments

    return segments.joinWithAmpersands()
}

```

```
}
```

```
func serialize(_ array: [URLEncodedFormComponent], forKey key: String) -> String {
```

```
    var segments: [String] = array.map { component in
```

```
        let keyPath = arrayEncoding.encode(key)
```

```
        return serialize(component, forKey: keyPath)
```

```
    }
```

```
    segments = alphabetizeKeyValuePairs ? segments.sorted() : segments
```

```
    return segments.joinedWithAmpersands()
```

```
}
```

```
func escape(_ query: String) -> String {
```

```
    var allowedCharactersWithSpace = allowedCharacters
```

```
    allowedCharactersWithSpace.insert(charactersIn: " ")
```

```
    let escapedQuery = query.addingPercentEncoding(withAllowedCharacters:
allowedCharactersWithSpace) ?? query
```

```
    let spaceEncodedQuery = spaceEncoding.encode(escapedQuery)
```

```

        return spaceEncodedQuery
    }
}

```

```

extension Array where Element == String {

    func joinedWithAmpersands() -> String {

        return joined(separator: "&")

    }

}

```

```

public extension CharacterSet {

    /// Creates a CharacterSet from RFC 3986 allowed characters.

    ///

    /// RFC 3986 states that the following characters are "reserved" characters.

    ///

    /// - General Delimiters: ":", "#", "[", "]", "@", "?", "/"

```

```

/// - Sub-Delimiters: "!", "$", "&", "'", "(", ")", "*", "+", ",", ";", "="

///

/// In RFC 3986 - Section 3.4, it states that the "?" and "/" characters should not be
escaped to allow

/// query strings to include a URL. Therefore, all "reserved" characters with the exception
of "?" and "/"

/// should be percent-escaped in the query string.

static let afURLQueryAllowed: CharacterSet = {

    let generalDelimitersToEncode = ":[ ]@" // does not include "?" or "/" due to RFC
3986 - Section 3.4

    let subDelimitersToEncode = "!$&'()*+,-;="

    let          encodableDelimiters          =          CharacterSet(charactersIn:
"\(generalDelimitersToEncode)\(subDelimitersToEncode)")

    return CharacterSet.urlQueryAllowed.subtracting(encodableDelimiters)

}()

}

//

// URLRequest+Alamofire.swift

```


//

// Copyright (c) 2019 Alamofire Software Foundation (<http://alamofire.org/>)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

```
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,  
DAMAGES OR OTHER
```

```
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,  
ARISING FROM,
```

```
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER  
DEALINGS IN
```

```
// THE SOFTWARE.
```

```
//
```

```
import Foundation
```

```
public extension URLRequest {
```

```
    /// Returns the `httpMethod` as Alamofire's `HTTPMethod` type.
```

```
    var method: HTTPMethod? {
```

```
        get { return httpMethod.flatMap(HTTPMethod.init) }
```

```
        set { httpMethod = newValue?.rawValue }
```

```
    }
```

```
    func validate() throws {
```

```

        if method == .get, let bodyData = httpBody {

            throw
            NSError.urlRequestValidationFailed(reason: .bodyDataInGETRequest(bodyData))

        }

    }

}

//

// URLSessionConfiguration+Alamofire.swift

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (http://alamofire.org/)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

```

```
// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//
```

import Foundation

```

extension URLSessionConfiguration: AlamofireExtended {}

extension AlamofireExtension where ExtendedType: URLSessionConfiguration {

    /// Alamofire's default configuration. Same as `URLSessionConfiguration.default` but
    adds Alamofire default

    /// `Accept-Language`, `Accept-Encoding`, and `User-Agent` headers.

    public static var `default`: URLSessionConfiguration {

        let configuration = URLSessionConfiguration.default

        configuration.headers = .default

        return configuration

    }

}

//

// UserInfoHeader.swift

// SettingsTemplate

//

// Created by Stephen Dowless on 2/10/19.

// Copyright © 2019 Stephan Dowless. All rights reserved.

```

```
//

import UIKit

class UserInfoHeader: UIView {

    // MARK: - Properties

    let profileImageView: UIImageView = {

        let iv = UIImageView()

        iv.contentMode = .scaleAspectFill

        iv.clipsToBounds = true

        iv.translatesAutoresizingMaskIntoConstraints = false

        iv.image = UIImage(named: "ironman")

        return iv

    }()
}
```

```
let usernameLabel: UILabel = {  
  
    let label = UILabel()  
  
    label.text = "Ruengruj Vivatthanamukda"  
  
    label.font = UIFont.systemFont(ofSize: 16)  
  
    label.translatesAutoresizingMaskIntoConstraints = false  
  
    return label  
  
}()
```

```
let emailLabel: UILabel = {  
  
    let label = UILabel()  
  
    label.text = "tiger@gmail.com"  
  
    label.font = UIFont.systemFont(ofSize: 14)  
  
    label.textColor = .lightGray  
  
    label.translatesAutoresizingMaskIntoConstraints = false  
  
    return label  
  
}()
```

```

// MARK: - Init

override init(frame: CGRect) {

    super.init(frame: frame)

    let profileImageDimension: CGFloat = 60

    addSubview(profileImageView)

    profileImageView.centerYAnchor.constraint(equalTo: centerYAnchor).isActive =
true

    profileImageView.leftAnchor.constraint(equalTo: leftAnchor, constant: 16).isActive
= true

    profileImageView.widthAnchor.constraint(equalToConstant:
profileImageDimension).isActive = true

    profileImageView.heightAnchor.constraint(equalToConstant:
profileImageDimension).isActive = true

    profileImageView.layer.cornerRadius = profileImageDimension / 2

    addSubview(usernameLabel)

```



```
        usernameLabel.centerYAnchor.constraint(equalTo:
profileImageView.centerYAnchor, constant: -10).isActive = true
```

```
        usernameLabel.leftAnchor.constraint(equalTo:    profileImageView.rightAnchor,
constant: 12).isActive = true
```

```
        addSubview(emailLabel)
```

```
        emailLabel.centerYAnchor.constraint(equalTo:  profileImageView.centerYAnchor,
constant: 10).isActive = true
```

```
        emailLabel.leftAnchor.constraint(equalTo: profileImageView.rightAnchor, constant:
12).isActive = true
```

```
    }
```

```
required init?(coder aDecoder: NSCoder) {
```

```
    fatalError("init(coder:) has not been implemented")
```

```
}
```

```
}
```

```
//
```

```
// Validation.swift
```

//

// Copyright (c) 2014-2018 Alamofire Software Foundation (<http://alamofire.org/>)

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in

// all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN

// THE SOFTWARE.

//

import Foundation

extension Request {

// MARK: Helper Types

fileprivate typealias ErrorReason = AFError.ResponseValidationFailureReason

/// Used to represent whether a validation succeeded or failed.

public typealias ValidationResult = Result<Void, Error>

```

fileprivate struct MIMEType {

    let type: String

    let subtype: String

    var isWildcard: Bool { return type == "*" && subtype == "*" }

    init?(_ string: String) {

        let components: [String] = {

            let stripped = string.trimmingCharacters(in: .whitespacesAndNewlines)

            let split = stripped[..<(stripped.range(of: ";")?.lowerBound ?? stripped.endIndex)]

            return split.components(separatedBy: "/")

        }()

        if let type = components.first, let subtype = components.last {

            self.type = type

            self.subtype = subtype
        }
    }
}

```

```
    } else {  
  
        return nil  
  
    }  
  
}
```

```
func matches(_ mime: MIMEType) -> Bool {  
  
    switch (type, subtype) {  
  
        case (mime.type, mime.subtype), (mime.type, "*"), ("*", mime.subtype), ("*", "*"):  
  
            return true  
  
        default:  
  
            return false  
  
    }  
  
}  
  
}
```

```
// MARK: Properties
```

```
fileprivate var acceptableStatusCodes: Range<Int> { return 200..<300 }
```

```
fileprivate var acceptableContentTypes: [String] {
```

```
    if let accept = request?.value(forHTTPHeaderField: "Accept") {
```

```
        return accept.components(separatedBy: ",")
```

```
    }
```

```
    return ["*/*"]
```

```
}
```

```
// MARK: Status Code
```

```
fileprivate func validate<S: Sequence>(statusCode acceptableStatusCodes: S,
```

```
    response: HTTPURLResponse)
```

```
-> ValidationResult
```

```
where S.Iterator.Element == Int {
```

```
    if acceptableStatusCodes.contains(response.statusCode) {
```

```

        return .success(Void())

    } else {

        let reason: ErrorReason = .unacceptableStatusCode(code: response.statusCode)

        return .failure(AFError.responseValidationFailed(reason: reason))

    }

}

```

```

// MARK: Content Type

```

```

fileprivate func validate<S: Sequence>(contentType acceptableContentTypes: S,

                                       response: HTTPURLResponse,

                                       data: Data?)

-> ValidationResult

where S.Iterator.Element == String {

    guard let data = data, !data.isEmpty else { return .success(Void()) }

    guard

```

```

        let responseContentType = response.mimeType,

        let responseMIMETYPE = MIMETYPE(responseContentType)

    else {

        for contentType in acceptableContentTypes {

            if let mimeType = MIMETYPE(contentType), mimeType.isWildcard {

                return .success(Void())

            }

        }

        let error: AFError = {

            let reason: ErrorReason = .missingContentType(acceptableContentTypes:
Array(acceptableContentTypes))

            return AFError.responseValidationFailed(reason: reason)

        }()

        return .failure(error)

    }

```



```

    for contentType in acceptableContentTypes {

        if let acceptableMIMETYPE = MIMETYPE(contentType),
        acceptableMIMETYPE.matches(responseMIMETYPE) {

            return .success(Void())

        }

    }

    let error: AFError = {

        let reason: ErrorReason = .unacceptableContentType(acceptableContentTypes:
        Array(acceptableContentTypes),

                                responseContentType: responseContentType)

        return AFError.responseValidationFailed(reason: reason)

    }()

    return .failure(error)

}

}

```

// MARK: -

extension DataRequest {

/// A closure used to validate a request that takes a URL request, a URL response and data, and returns whether the

/// request was valid.

public typealias Validation = (URLRequest?, HTTPURLResponse, Data?) ->
ValidationResult

/// Validates that the response has a status code in the specified sequence.

///

/// If validation fails, subsequent calls to response handlers will have an associated error.

///

/// - parameter range: The range of acceptable status codes.

///

/// - returns: The request.

@discardableResult

```

public func validate<S: Sequence>(statusCode acceptableStatusCodes: S) -> Self where
S.Iterator.Element == Int {

```

```

    return validate { [unowned self] _, response, _ in

```

```

        self.validate(statusCode: acceptableStatusCodes, response: response)

```

```

    }

```

```

}

```

```

/// Validates that the response has a content type in the specified sequence.

```

```

///

```

```

/// If validation fails, subsequent calls to response handlers will have an associated error.

```

```

///

```

```

/// - parameter contentType: The acceptable content types, which may specify wildcard
types and/or subtypes.

```

```

///

```

```

/// - returns: The request.

```

```

@discardableResult

```

```

public func validate<S: Sequence>(contentType acceptableContentTypes: @escaping
@autoclosure () -> S) -> Self where S.Iterator.Element == String {

```

```

    return validate { [unowned self] _, response, data in

```

```

        self.validate(contentType: acceptableContentTypes(), response: response, data:
data)

    }

}

```

/// Validates that the response has a status code in the default acceptable range of 200...299, and that the content

/// type matches any specified in the Accept HTTP header field.

///

/// If validation fails, subsequent calls to response handlers will have an associated error.

///

/// - returns: The request.

@discardableResult

public fun validate() -> Self {

let contentTypes: () -> [String] = { [unowned self] in

self.acceptableContentTypes

}

return validate(statusCode: acceptableStatusCodes).validate(contentType: contentTypes())

```
}  
  
}
```

```
// MARK: -
```

```
extension DownloadRequest {
```

```
    /// A closure used to validate a request that takes a URL request, a URL response, a  
    temporary URL and a
```

```
    /// destination URL, and returns whether the request was valid.
```

```
    public typealias Validation = (_ request: URLRequest?,
```

```
                                   _ response: HTTPURLResponse,
```

```
                                   _ fileURL: URL?)
```

```
    -> ValidationResult
```

```
    /// Validates that the response has a status code in the specified sequence.
```

```
    ///
```

```
    /// If validation fails, subsequent calls to response handlers will have an associated error.
```

```
    ///
```

/// - parameter range: The range of acceptable status codes.

///

/// - returns: The request.

@discardableResult

```
public func validate<S: Sequence>(statusCode acceptableStatusCodes: S) -> Self where  
S.Iterator.Element == Int {
```

```
    return validate { [unowned self] _, response, _ in
```

```
        self.validate(statusCode: acceptableStatusCodes, response: response)
```

```
    }
```

```
}
```

/// Validates that the response has a content type in the specified sequence.

///

/// If validation fails, subsequent calls to response handlers will have an associated error.

///

/// - parameter contentType: The acceptable content types, which may specify wildcard types and/or subtypes.

///

```

/// - returns: The request.

@discardableResult

public func validate<S: Sequence>(contentType acceptableContentTypes: @escaping
@autoclosure () -> S) -> Self where S.Iterator.Element == String {

    return validate { [unowned self] _, response, fileURL in

        guard let validFileURL = fileURL else {

            return .failure(AFError.responseValidationFailed(reason: .dataFileNil))

        }

        do {

            let data = try Data(contentsOf: validFileURL)

            return self.validate(contentType: acceptableContentTypes(), response: response,
data: data)

        } catch {

            return .failure(AFError.responseValidationFailed(reason: .dataFileReadFailed(at:
validFileURL)))

        }

    }
}

```

```

    }

    /// Validates that the response has a status code in the default acceptable range of
    200...299, and that the content

    /// type matches any specified in the Accept HTTP header field.

    ///

    /// If validation fails, subsequent calls to response handlers will have an associated error.

    ///

    /// - returns: The request.

    @discardableResult

    public func validate() -> Self {

        let contentTypes = { [unowned self] in

            self.acceptableContentTypes

        }

        return validate(statusCode: acceptableStatusCodes).validate(contentType:
        contentTypes())

    }

}

```



```

//

// ViewPortHandler.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

/// Class that contains information about the charts current viewport settings, including
/// offsets, scale & translation levels, ...

@objc(ChartViewPortHandler)

open class ViewPortHandler: NSObject

```

```

{

    /// matrix used for touch events

    private var _touchMatrix = CGAffineTransform.identity


    /// this rectangle defines the area in which graph values can be drawn

    private var _contentRect = CGRect()


    private var _chartWidth = CGFloat(0.0)

    private var _chartHeight = CGFloat(0.0)


    /// minimum scale value on the y-axis

    private var _minScaleY = CGFloat(1.0)


    /// maximum scale value on the y-axis

    private var _maxScaleY = CGFloat.greatestFiniteMagnitude


    /// minimum scale value on the x-axis

```

```
private var _minScaleX = CGFloat(1.0)

/// maximum scale value on the x-axis

private var _maxScaleX = CGFloat.greatestFiniteMagnitude

/// contains the current scale factor of the x-axis

private var _scaleX = CGFloat(1.0)

/// contains the current scale factor of the y-axis

private var _scaleY = CGFloat(1.0)

/// current translation (drag distance) on the x-axis

private var _transX = CGFloat(0.0)

/// current translation (drag distance) on the y-axis

private var _transY = CGFloat(0.0)
```

```
/// offset that allows the chart to be dragged over its bounds on the x-axis
```

```
private var _transOffsetX = CGFloat(0.0)
```

```
/// offset that allows the chart to be dragged over its bounds on the x-axis
```

```
private var _transOffsetY = CGFloat(0.0)
```

```
/// Constructor - don't forget calling setChartDimens(...)
```

```
@objc public init(width: CGFloat, height: CGFloat)
```

```
{
```

```
    super.init()
```

```
    setChartDimens(width: width, height: height)
```

```
}
```

```
@objc open func setChartDimens(width: CGFloat, height: CGFloat)
```

```
{
```

```
    let offsetLeft = self.offsetLeft
```

```
let offsetTop = self.offsetTop
```

```
let offsetRight = self.offsetRight
```

```
let offsetBottom = self.offsetBottom
```

```
_chartHeight = height
```

```
_chartWidth = width
```

```
restrainViewPort(offsetLeft: offsetLeft, offsetTop: offsetTop, offsetRight:  
offsetRight, offsetBottom: offsetBottom)
```

```
}
```

```
@objc open var hasChartDimens: Bool
```

```
{
```

```
if _chartHeight > 0.0 && _chartWidth > 0.0
```

```
{
```

```
    return true
```

```
}
```

```
else
```

```

{

    return false

}

}

```

```

@objc open func restrainViewPort(offsetLeft: CGFloat, offsetTop: CGFloat,
offsetRight: CGFloat, offsetBottom: CGFloat)

```

```

{

    _contentRect.origin.x = offsetLeft

    _contentRect.origin.y = offsetTop

    _contentRect.size.width = _chartWidth - offsetLeft - offsetRight

    _contentRect.size.height = _chartHeight - offsetBottom - offsetTop

}

```

```

@objc open var offsetLeft: CGFloat

```

```

{

    return _contentRect.origin.x

}

```

```
@objc open var offsetRight: CGFloat

{

    return _chartWidth - _contentRect.size.width - _contentRect.origin.x

}
```

```
@objc open var offsetTop: CGFloat

{

    return _contentRect.origin.y

}
```

```
@objc open var offsetBottom: CGFloat

{

    return _chartHeight - _contentRect.size.height - _contentRect.origin.y

}
```

```
@objc open var contentTop: CGFloat
```

```
{  
  
    return _contentRect.origin.y  
  
}
```

@objc open var contentLeft: CGFloat

```
{  
  
    return _contentRect.origin.x  
  
}
```

@objc open var contentRight: CGFloat

```
{  
  
    return _contentRect.origin.x + _contentRect.size.width  
  
}
```

@objc open var contentBottom: CGFloat

```
{  
  
    return _contentRect.origin.y + _contentRect.size.height  
  
}
```



```
}
```

```
@objc open var contentWidth: CGFloat
```

```
{
```

```
    return _contentRect.size.width
```

```
}
```

```
@objc open var contentHeight: CGFloat
```

```
{
```

```
    return _contentRect.size.height
```

```
}
```

```
@objc open var contentRect: CGRect
```

```
{
```

```
    return _contentRect
```

```
}
```

```
@objc open var contentCenter: CGPoint

{

    return CGPoint(x: _contentRect.origin.x + _contentRect.size.width / 2.0, y:
_contentRect.origin.y + _contentRect.size.height / 2.0)

}
```

```
@objc open var chartHeight: CGFloat

{

    return _chartHeight

}
```

```
@objc open var chartWidth: CGFloat

{

    return _chartWidth

}
```

```
// MARK: - Scaling/Panning etc.
```

/// Zooms by the specified zoom factors.

```
@objc open func zoom(scaleX: CGFloat, scaleY: CGFloat) -> CGAffineTransform  
  
{  
  
    return _touchMatrix.scaledBy(x: scaleX, y: scaleY)  
  
}
```

/// Zooms around the specified center

```
@objc open func zoom(scaleX: CGFloat, scaleY: CGFloat, x: CGFloat, y: CGFloat) ->  
CGAffineTransform  
  
{  
  
    var matrix = _touchMatrix.translatedBy(x: x, y: y)  
  
    matrix = matrix.scaledBy(x: scaleX, y: scaleY)  
  
    matrix = matrix.translatedBy(x: -x, y: -y)  
  
    return matrix  
  
}
```

/// Zooms in by 1.4, x and y are the coordinates (in pixels) of the zoom center.

```
@objc open func zoomIn(x: CGFloat, y: CGFloat) -> CGAffineTransform
```

```
{  
  
    return zoom(scaleX: 1.4, scaleY: 1.4, x: x, y: y)  
  
}
```

/// Zooms out by 0.7, x and y are the coordinates (in pixels) of the zoom center.

```
@objc open func zoomOut(x: CGFloat, y: CGFloat) -> CGAffineTransform  
  
{  
  
    return zoom(scaleX: 0.7, scaleY: 0.7, x: x, y: y)  
  
}
```

/// Zooms out to original size.

```
@objc open func resetZoom() -> CGAffineTransform  
  
{  
  
    return zoom(scaleX: 1.0, scaleY: 1.0, x: 0.0, y: 0.0)  
  
}
```

/// Sets the scale factor to the specified values.

```
@objc open func setZoom(scaleX: CGFloat, scaleY: CGFloat) -> CGAffineTransform
```

```
{
```

```
    var matrix = _touchMatrix
```

```
    matrix.a = scaleX
```

```
    matrix.d = scaleY
```

```
    return matrix
```

```
}
```

```
/// Sets the scale factor to the specified values. x and y is pivot.
```

```
@objc open func setZoom(scaleX: CGFloat, scaleY: CGFloat, x: CGFloat, y: CGFloat)
```

```
-> CGAffineTransform
```

```
{
```

```
    var matrix = _touchMatrix
```

```
    matrix.a = 1.0
```

```
    matrix.d = 1.0
```

```
    matrix = matrix.translatedBy(x: x, y: y)
```

```
    matrix = matrix.scaledBy(x: scaleX, y: scaleY)
```

```
    matrix = matrix.translatedBy(x: -x, y: -y)
```

```

    return matrix
}

/// Resets all zooming and dragging and makes the chart fit exactly it's bounds.

@objc open func fitScreen() -> CGAffineTransform

{

    _minScaleX = 1.0

    _minScaleY = 1.0

    return CGAffineTransform.identity
}

/// Translates to the specified point.

@objc open func translate(pt: CGPoint) -> CGAffineTransform

{

    let translateX = pt.x - offsetLeft

    let translateY = pt.y - offsetTop

```

```
        let matrix = _touchMatrix.concatenating(CGAffineTransform(translationX: -  
translateX, y: -translateY))
```

```
        return matrix
```

```
    }
```

```
/// Centers the viewport around the specified position (x-index and y-value) in the chart.
```

```
/// Centering the viewport outside the bounds of the chart is not possible.
```

```
/// Makes most sense in combination with the setScaleMinima(...) method.
```

```
@objc open func centerViewPort(pt: CGPoint, chart: ChartViewBase)
```

```
{
```

```
    let translateX = pt.x - offsetLeft
```

```
    let translateY = pt.y - offsetTop
```

```
        let matrix = _touchMatrix.concatenating(CGAffineTransform(translationX: -  
translateX, y: -translateY))
```

```
        refresh(newMatrix: matrix, chart: chart, invalidate: true)
```

```
}
```

```
/// call this method to refresh the graph with a given matrix
```

```
@objc @discardableResult open func refresh(newMatrix: CGAffineTransform, chart:  
ChartViewBase, invalidate: Bool) -> CGAffineTransform
```

```
{
```

```
    _touchMatrix = newMatrix
```

```
    // make sure scale and translation are within their bounds
```

```
    limitTransAndScale(matrix: &_touchMatrix, content: _contentRect)
```

```
    chart.setNeedsDisplay()
```

```
    return _touchMatrix
```

```
}
```

```
/// limits the maximum scale and X translation of the given matrix
```

```
private func limitTransAndScale(matrix: inout CGAffineTransform, content: CGRect?)
```



```

{

    // min scale-x is 1

    _scaleX = min(max(_minScaleX, matrix.a), _maxScaleX)


    // min scale-y is 1

    _scaleY = min(max(_minScaleY, matrix.d), _maxScaleY)


    var width: CGFloat = 0.0

    var height: CGFloat = 0.0


    if content != nil

    {

        width = content!.width

        height = content!.height

    }

```

```
let maxTransX = -width * (_scaleX - 1.0)
```

```
_transX = min(max(matrix.tx, maxTransX - _transOffsetX), _transOffsetX)
```

```
let maxTransY = height * (_scaleY - 1.0)
```

```
_transY = max(min(matrix.ty, maxTransY + _transOffsetY), -_transOffsetY)
```

```
matrix.tx = _transX
```

```
matrix.a = _scaleX
```

```
matrix.ty = _transY
```

```
matrix.d = _scaleY
```

```
}
```

```
/// Sets the minimum scale factor for the x-axis
```

```
@objc open func setMinimumScaleX(_ xScale: CGFloat)
```

```
{
```

```
var newValue = xScale
```

```

    if newValue < 1.0

    {

        newValue = 1.0

    }

    _minScaleX = newValue

    limitTransAndScale(matrix: &_touchMatrix, content: _contentRect)

}

/// Sets the maximum scale factor for the x-axis

@objc open func setMaximumScaleX(_ xScale: CGFloat)

{

    var newValue = xScale

    if newValue == 0.0

    {

```

```

        newValue = CGFloat.greatestFiniteMagnitude
    }

    _maxScaleX = newValue

    limitTransAndScale(matrix: &_touchMatrix, content: _contentRect)
}

/// Sets the minimum and maximum scale factors for the x-axis

@objc open func setMinMaxScaleX(minScaleX: CGFloat, maxScaleX: CGFloat)
{
    var newMin = minScaleX

    var newMax = maxScaleX

    if newMin < 1.0
    {
        newMin = 1.0
    }
}

```

```

    }

    if newMax == 0.0

    {

        newMax = CGFloat.greatestFiniteMagnitude

    }

    _minScaleX = newMin

    _maxScaleX = maxScaleX

    limitTransAndScale(matrix: &_touchMatrix, content: _contentRect)

}

/// Sets the minimum scale factor for the y-axis

@objc open func setMinimumScaleY(_ yScale: CGFloat)

{

    var newValue = yScale

```

```

    if newValue < 1.0

    {

        newValue = 1.0

    }

    _minScaleY = newValue

    limitTransAndScale(matrix: &_touchMatrix, content: _contentRect)

}

/// Sets the maximum scale factor for the y-axis

@objc open func setMaximumScaleY(_ yScale: CGFloat)

{

    var newValue = yScale

    if newValue == 0.0

    {

```

```

        newValue = CGFloat.greatestFiniteMagnitude
    }

    _maxScaleY = newValue

    limitTransAndScale(matrix: &_touchMatrix, content: _contentRect)
}

@objc open func setMinMaxScaleY(minScaleY: CGFloat, maxScaleY: CGFloat)
{
    var minScaleY = minScaleY, maxScaleY = maxScaleY

    if minScaleY < 1.0
    {
        minScaleY = 1.0
    }
}

```

```

if maxScaleY == 0.0

{

    maxScaleY = CGFloat.greatestFiniteMagnitude

}

_minScaleY = minScaleY

_maxScaleY = maxScaleY

limitTransAndScale(matrix: &_touchMatrix, content: _contentRect)

}

@objc open var touchMatrix: CGAffineTransform

{

    return _touchMatrix

}

// MARK: - Boundaries Check

```



```
@objc open func isInBoundsX(_ x: CGFloat) -> Bool
```

```
{
```

```
    return isInBoundsLeft(x) && isInBoundsRight(x)
```

```
}
```

```
@objc open func isInBoundsY(_ y: CGFloat) -> Bool
```

```
{
```

```
    return isInBoundsTop(y) && isInBoundsBottom(y)
```

```
}
```

```
/**
```

A method to check whether coordinate lies within the viewport.

- Parameters:

- point: a coordinate.

```
*/
```

```
@objc open func isInBounds(point: CGPoint) -> Bool
```

```
{  
  
    return isInBounds(x: point.x, y: point.y)  
  
}
```

```
@objc open func isInBounds(x: CGFloat, y: CGFloat) -> Bool
```

```
{  
  
    return isInBoundsX(x) && isInBoundsY(y)  
  
}
```

```
@objc open func isInBoundsLeft(_ x: CGFloat) -> Bool
```

```
{  
  
    return _contentRect.origin.x <= x + 1.0  
  
}
```

```
@objc open func isInBoundsRight(_ x: CGFloat) -> Bool
```

```
{
```

```

    let x = floor(x * 100.0) / 100.0

    return (_contentRect.origin.x + _contentRect.size.width) >= x - 1.0

}

```

```

@objc open func isInBoundsTop(_ y: CGFloat) -> Bool

{

    return _contentRect.origin.y <= y

}

```

```

@objc open func isInBoundsBottom(_ y: CGFloat) -> Bool

{

    let normalizedY = floor(y * 100.0) / 100.0

    return (_contentRect.origin.y + _contentRect.size.height) >= normalizedY

}

```

```

/**

```

A method to check whether a line between two coordinates intersects with the view port by using a linear function.

Linear function (calculus): $y = ax + b$

Note: this method will not check for collision with the right edge of the view port, as we assume lines run from left

to right (e.g. `startPoint < endPoint`).

- Parameters:

- `startPoint`: the start coordinate of the line.

- `endPoint`: the end coordinate of the line.

*/

@objc open func isIntersectingLine(from startPoint: CGPoint, to endPoint: CGPoint) -> Bool

{

// If start- and/or endpoint fall within the viewport, bail out early.

if isInBounds(point: startPoint) || isInBounds(point: endPoint) { return true }

// check if x in bound when it's a vertical line

if startPoint.x == endPoint.x { return isInBoundsX(startPoint.x) }

```

// Calculate the slope (`a`) of the line (e.g. `a = (y2 - y1) / (x2 - x1)`).

let a = (endPoint.y - startPoint.y) / (endPoint.x - startPoint.x)

// Calculate the y-correction (`b`) of the line (e.g. `b = y1 - (a * x1)`).

let b = startPoint.y - (a * startPoint.x)


// Check for collision with the left edge of the view port (e.g. `y = (a * minX) + b`).

// if a is 0, it's a horizontal line; checking b here is still valid, as b is `point.y` all the
time

if isInBoundsY((a * contentRect.minX) + b) { return true }


// Skip unnecessary check for collision with the right edge of the view port

// (e.g. `y = (a * maxX) + b`), as such a line will either begin inside the view port,

// or intersect the left, top or bottom edges of the view port. Leaving this logic here
for clarity's sake:

// if isInBoundsY((a * contentRect.maxX) + b) { return true }


// While slope `a` can theoretically never be `0`, we should protect against division by
zero.

```

```
guard a != 0 else { return false }
```

```
// Check for collision with the bottom edge of the view port (e.g. `x = (maxY - b) / a`).
```

```
if isInBoundsX((contentRect.maxY - b) / a) { return true }
```

```
// Check for collision with the top edge of the view port (e.g. `x = (minY - b) / a`).
```

```
if isInBoundsX((contentRect.minY - b) / a) { return true }
```

```
// This line does not intersect the view port.
```

```
return false
```

```
}
```

```
/// The current x-scale factor
```

```
@objc open var scaleX: CGFloat
```

```
{
```

```
    return _scaleX
```

```
}
```

```
/// The current y-scale factor
```

```
@objc open var scaleY: CGFloat
```

```
{
```

```
    return _scaleY
```

```
}
```

```
/// The minimum x-scale factor
```

```
@objc open var minScaleX: CGFloat
```

```
{
```

```
    return _minScaleX
```

```
}
```

```
/// The minimum y-scale factor
```

```
@objc open var minScaleY: CGFloat
```

```
{
```

```
    return _minScaleY
```

```
}
```

```
/// The minimum x-scale factor
```

```
@objc open var maxScaleX: CGFloat
```

```
{
```

```
    return _maxScaleX
```

```
}
```

```
/// The minimum y-scale factor
```

```
@objc open var maxScaleY: CGFloat
```

```
{
```

```
    return _maxScaleY
```

```
}
```

```
/// The translation (drag / pan) distance on the x-axis
```

```
@objc open var transX: CGFloat
```

```
{
```



```

        return _transX
    }

    /// The translation (drag / pan) distance on the y-axis

    @objc open var transY: CGFloat

    {

        return _transY
    }


    /// if the chart is fully zoomed out, return true

    @objc open var isFullyZoomedOut: Bool

    {

        return isFullyZoomedOutX && isFullyZoomedOutY
    }


    /// `true` if the chart is fully zoomed out on it's y-axis (vertical).

    @objc open var isFullyZoomedOutY: Bool

```

```

{

    return !(_scaleY > _minScaleY || _minScaleY > 1.0)

}

/// `true` if the chart is fully zoomed out on it's x-axis (horizontal).

@objc open var isFullyZoomedOutX: Bool

{

    return !(_scaleX > _minScaleX || _minScaleX > 1.0)

}

/// Set an offset in pixels that allows the user to drag the chart over it's bounds on the x-
axis.

@objc open func setDragOffsetX(_ offset: CGFloat)

{

    _transOffsetX = offset

}

```

/// Set an offset in pixels that allows the user to drag the chart over it's bounds on the y-axis.

```
@objc open func setDragOffsetY(_ offset: CGFloat)
```

```
{
```

```
    _transOffsetY = offset
```

```
}
```

/// `true` if both drag offsets (x and y) are zero or smaller.

```
@objc open var hasNoDragOffset: Bool
```

```
{
```

```
    return _transOffsetX <= 0.0 && _transOffsetY <= 0.0
```

```
}
```

/// `true` if the chart is not yet fully zoomed out on the x-axis

```
@objc open var canZoomOutMoreX: Bool
```

```
{
```

```
    return _scaleX > _minScaleX
```

```
}
```

```
/// `true` if the chart is not yet fully zoomed in on the x-axis
```

```
@objc open var canZoomInMoreX: Bool
```

```
{
```

```
    return _scaleX < _maxScaleX
```

```
}
```

```
/// `true` if the chart is not yet fully zoomed out on the y-axis
```

```
@objc open var canZoomOutMoreY: Bool
```

```
{
```

```
    return _scaleY > _minScaleY
```

```
}
```

```
/// `true` if the chart is not yet fully zoomed in on the y-axis
```

```
@objc open var canZoomInMoreY: Bool
```

```
{
```

```
    return _scaleY < _maxScaleY
```

```

    }

}

//

// ViewPortJob.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

// This defines a viewport modification job, used for delaying or animating viewport
changes

```

```

@objc(ChartViewPortJob)

open class ViewPortJob: NSObject

{

    internal var point: CGPoint = CGPoint()

    internal weak var viewPortHandler: ViewPortHandler?

    internal var xValue: Double = 0.0

    internal var yValue: Double = 0.0

    internal weak var transformer: Transformer?

    internal weak var view: ChartViewBase?


    @objc public init(

        viewPortHandler: ViewPortHandler,

        xValue: Double,

        yValue: Double,

        transformer: Transformer,

        view: ChartViewBase)

    {

```

```

    super.init()

    self.viewPortHandler = viewPortHandler

    self.xValue = xValue

    self.yValue = yValue

    self.transformer = transformer

    self.view = view
}

@objc open func doJob()

{

    fatalError("`doJob()` must be overridden by subclasses")

}

}

//

// XAxis.swift

// Charts

```

```
//  
  
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda  
  
// A port of MPAndroidChart for iOS  
  
// Licensed under Apache License 2.0  
  
//  
  
// https://github.com/danielgindi/Charts  
  
//
```

```
import Foundation
```

```
import CoreGraphics
```

```
@objc(ChartXAxis)
```

```
open class XAxis: AxisBase
```

```
{
```

```
    @objc(XAxisLabelPosition)
```

```
    public enum LabelPosition: Int
```

```
{
```



```
case top

case bottom

case bothSided

case topInside

case bottomInside

}
```

```
/// width of the x-axis labels in pixels - this is automatically calculated by the
`computeSize()` methods in the renderers
```

```
@objc open var labelWidth = CGFloat(1.0)
```

```
/// height of the x-axis labels in pixels - this is automatically calculated by the
`computeSize()` methods in the renderers
```

```
@objc open var labelHeight = CGFloat(1.0)
```

```
/// width of the (rotated) x-axis labels in pixels - this is automatically calculated by the
`computeSize()` methods in the renderers
```

```
@objc open var labelRotatedWidth = CGFloat(1.0)
```

/// height of the (rotated) x-axis labels in pixels - this is automatically calculated by the
`computeSize()` methods in the renderers

@objc open var labelRotatedHeight = CGFloat(1.0)

/// This is the angle for drawing the X axis labels (in degrees)

@objc open var labelRotationAngle = CGFloat(0.0)

/// if set to true, the chart will avoid that the first and last label entry in the chart "clip"
off the edge of the chart

@objc open var avoidFirstLastClippingEnabled = false

/// the position of the x-labels relative to the chart

@objc open var labelPosition = LabelPosition.top

/// if set to true, word wrapping the labels will be enabled.

/// word wrapping is done using `(value width \* labelRotatedWidth)`

///

/// - Note: currently supports all charts except pie/radar/horizontal-bar*

```
@objc open var wordWrapEnabled = false
```

```
/// `true` if word wrapping the labels is enabled
```

```
@objc open var isWordWrapEnabled: Bool { return wordWrapEnabled }
```

```
/// the width for wrapping the labels, as percentage out of one value width.
```

```
/// used only when isWordWrapEnabled = true.
```

```
///
```

```
/// default: 1.0
```

```
@objc open var wordWrapWidthPercent: CGFloat = 1.0
```

```
public override init()
```

```
{
```

```
    super.init()
```

```
    self.yOffset = 4.0
```

```
}
```

```
@objc open var isAvoidFirstLastClippingEnabled: Bool

{

    return avoidFirstLastClippingEnabled

}

}

//

// XAxisRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//
```

```
import Foundation
```

```
import CoreGraphics
```

```
#if canImport(Uikit)
```

```
    import UIKit
```

```
#endif
```

```
#if canImport(Cocoa)
```

```
import Cocoa
```

```
#endif
```

```
@objc(ChartXAxisRenderer)
```

```
open class XAxisRenderer: AxisRendererBase
```

```
{
```

```
    @objc public init(viewPortHandler: ViewPortHandler, xAxis: XAxis?, transformer:  
Transformer?)
```

```
{
```

```
        super.init(viewPortHandler: viewPortHandler, transformer: transformer, axis: xAxis)
```

```
}
```

```
open override fun computeAxis(min: Double, max: Double, inverted: Bool)
```

```
{
```

```
    var min = min, max = max
```

```
    if let transformer = self.transformer
```

```
    {
```

```
        // calculate the starting and entry point of the y-labels (depending on
```

```
        // zoom / contentrect bounds)
```

```
        if viewPortHandler.contentWidth > 10 && !viewPortHandler.isFullyZoomedOutX
```

```
        {
```

```
            let p1 = transformer.valueForTouchPoint(CGPoint(x:
viewPortHandler.contentLeft, y: viewPortHandler.contentTop))
```

```
            let p2 = transformer.valueForTouchPoint(CGPoint(x:
viewPortHandler.contentRight, y: viewPortHandler.contentTop))
```

```
            if inverted
```

```

    {

        min = Double(p2.x)

        max = Double(p1.x)

    }

    else

    {

        min = Double(p1.x)

        max = Double(p2.x)

    }

}

computeAxisValues(min: min, max: max)

}

```

```

open override fun computeAxisValues(min: Double, max: Double)

{

```

```
super.computeAxisValues(min: min, max: max)
```

```
computeSize()
```

```
}
```

```
@objc open func computeSize()
```

```
{
```

```
    guard let
```

```
        xAxis = self.axis as? XAxis
```

```
    else { return }
```

```
    let longest = xAxis.getLongestLabel()
```

```
    let labelSize = longest.size(withAttributes: [NSAttributedString.Key.font:
xAxis.labelFont])
```

```
    let labelWidth = labelSize.width
```

```
    let labelHeight = labelSize.height
```



```

let labelRotatedSize = labelSize.rotatedBy(degrees: xAxis.labelRotationAngle)

xAxis.labelWidth = labelWidth

xAxis.labelHeight = labelHeight

xAxis.labelRotatedWidth = labelRotatedSize.width

xAxis.labelRotatedHeight = labelRotatedSize.height
}

open override func renderAxisLabels(context: CGContext)

{

    guard let xAxis = self.axis as? XAxis else { return }

    if !xAxis.isEnabled || !xAxis.isDrawLabelsEnabled

    {

        return

    }
}

```

```

let yOffset = xAxis.yOffset

if xAxis.labelPosition == .top

{

    drawLabels(context: context, pos: viewPortHandler.contentTop - yOffset, anchor:
CGPoint(x: 0.5, y: 1.0))

}

else if xAxis.labelPosition == .topInside

{

    drawLabels(context: context, pos: viewPortHandler.contentTop + yOffset +
xAxis.labelRotatedHeight, anchor: CGPoint(x: 0.5, y: 1.0))

}

else if xAxis.labelPosition == .bottom

{

    drawLabels(context: context, pos: viewPortHandler.contentBottom + yOffset,
anchor: CGPoint(x: 0.5, y: 0.0))

}

else if xAxis.labelPosition == .bottomInside

```

```

    {

        drawLabels(context: context, pos: viewPortHandler.contentBottom - yOffset -
xAxis.labelRotatedHeight, anchor: CGPoint(x: 0.5, y: 0.0))

    }

    else

    { // BOTH SIDED

        drawLabels(context: context, pos: viewPortHandler.contentTop - yOffset, anchor:
CGPoint(x: 0.5, y: 1.0))

        drawLabels(context: context, pos: viewPortHandler.contentBottom + yOffset,
anchor: CGPoint(x: 0.5, y: 0.0))

    }

}

private var _axisLineSegmentsBuffer = [CGPoint](repeating: CGPoint(), count: 2)

open override func renderAxisLine(context: CGContext)

{

    guard let xAxis = self.axis as? XAxis else { return }

```

```

if !xAxis.isEnabled || !xAxis.isDrawAxisLineEnabled

{

    return

}

context.saveGState()

context.setStrokeColor(xAxis.axisLineColor.cgColor)

context.setLineWidth(xAxis.axisLineWidth)

if xAxis.axisLineDashLengths != nil

{

    context.setLineDash(phase:          xAxis.axisLineDashPhase,          lengths:
xAxis.axisLineDashLengths)

}

else

{

    context.setLineDash(phase: 0.0, lengths: [])

}

```

```

if xAxis.labelPosition == .top

    || xAxis.labelPosition == .topInside

    || xAxis.labelPosition == .bothSided

{

    _axisLineSegmentsBuffer[0].x = viewPortHandler.contentLeft

    _axisLineSegmentsBuffer[0].y = viewPortHandler.contentTop

    _axisLineSegmentsBuffer[1].x = viewPortHandler.contentRight

    _axisLineSegmentsBuffer[1].y = viewPortHandler.contentTop

    context.strokeLineSegments(between: _axisLineSegmentsBuffer)

}

```

```

if xAxis.labelPosition == .bottom

    || xAxis.labelPosition == .bottomInside

    || xAxis.labelPosition == .bothSided

{

    _axisLineSegmentsBuffer[0].x = viewPortHandler.contentLeft

```

```

        _axisLineSegmentsBuffer[0].y = viewPortHandler.contentBottom

        _axisLineSegmentsBuffer[1].x = viewPortHandler.contentRight

        _axisLineSegmentsBuffer[1].y = viewPortHandler.contentBottom

        context.strokeLineSegments(between: _axisLineSegmentsBuffer)

    }

    context.restoreGState()

}

/// draws the x-labels on the specified y-position

@objc open func drawLabels(context: CGContext, pos: CGFloat, anchor: CGPoint)

{

    guard

        let xAxis = self.axis as? XAxis,

        let transformer = self.transformer

    else { return }

```

```
let paraStyle = NSParagraphStyle.default.mutableCopy() as!
NSMutableParagraphStyle
```

```
paraStyle.alignment = .center
```

```
let labelAttrs: [NSAttributedString.Key : Any] = [
```

```
    .font: xAxis.labelFont,
```

```
    .foregroundColor: xAxis.labelTextColor,
```

```
    .paragraphStyle: paraStyle
```

```
]
```

```
let labelRotationAngleRadians = xAxis.labelRotationAngle.DEG2RAD
```

```
let centeringEnabled = xAxis.isCenterAxisLabelsEnabled
```

```
let valueToPixelMatrix = transformer.valueToPixelMatrix
```

```
var position = CGPoint(x: 0.0, y: 0.0)
```

```
var labelMaxSize = CGSize()
```

```

if xAxis.isWordWrapEnabled

{

    labelMaxSize.width = xAxis.wordWrapWidthPercent * valueToPixelMatrix.a

}

let entries = xAxis.entries

for i in stride(from: 0, to: entries.count, by: 1)

{

    if centeringEnabled

    {

        position.x = CGFloat(xAxis.centeredEntries[i])

    }

    else

    {

        position.x = CGFloat(entries[i])
    }
}

```



```
}
```

```
position.y = 0.0
```

```
position = position.applying(valueToPixelMatrix)
```

```
if viewportHandler.isInBoundsX(position.x)
```

```
{
```

```
    let label = xAxis.valueFormatter?.stringForValue(xAxis.entries[i], axis:  
xAxis) ?? ""
```

```
    let labelns = label as NSString
```

```
if xAxis.isAvoidFirstLastClippingEnabled
```

```
{
```

```
    // avoid clipping of the last
```

```
if i == xAxis.entryCount - 1 && xAxis.entryCount > 1
```

```
{
```

```

        let width = labelns.boundingRect(with: labelMaxSize,
options: .usesLineFragmentOrigin, attributes: labelAttrs, context: nil).size.width

```

```

        if width > viewPortHandler.offsetRight * 2.0

```

```

            && position.x + width > viewPortHandler.chartWidth

```

```

        {

```

```

            position.x -= width / 2.0

```

```

        }

```

```

    }

```

```

    else if i == 0

```

```

    { // avoid clipping of the first

```

```

        let width = labelns.boundingRect(with: labelMaxSize,
options: .usesLineFragmentOrigin, attributes: labelAttrs, context: nil).size.width

```

```

        position.x += width / 2.0

```

```

    }

```

```

}

```

```

drawLabel(context: context,

```

```

        formattedLabel: label,

        x: position.x,

        y: pos,

        attributes: labelAttrs,

        constrainedToSize: labelMaxSize,

        anchor: anchor,

        angleRadians: labelRotationAngleRadians)

    }

}

}

```

```

@objc open func drawLabel(

    context: CGContext,

    formattedLabel: String,

    x: CGFloat,

    y: CGFloat,

    attributes: [NSAttributedString.Key : Any],

```

```

        constrainedToSize: CGSize,

        anchor: CGPoint,

        angleRadians: CGFloat)

{

    ChartUtils.drawMultilineText(

        context: context,

        text: formattedLabel,

        point: CGPoint(x: x, y: y),

        attributes: attributes,

        constrainedToSize: constrainedToSize,

        anchor: anchor,

        angleRadians: angleRadians)

    }

open override func renderGridLines(context: CGContext)

{

    guard

```

```

    let xAxis = self.axis as? XAxis,

    let transformer = self.transformer

    else { return }

    if !xAxis.isDrawGridLinesEnabled || !xAxis.isEnabled

    {

        return

    }

    context.saveGState()

    defer { context.restoreGState() }

    context.clip(to: self.gridClippingRect)

    context.setShouldAntialias(xAxis.gridAntialiasEnabled)

    context.setStrokeColor(xAxis.gridColor.cgColor)

    context.setLineWidth(xAxis.gridLineWidth)

    context.setLineCap(xAxis.gridLineCap)

```

```

if xAxis.gridLineDashLengths != nil

{

    context.setLineDash(phase:      xAxis.gridLineDashPhase,      lengths:
xAxis.gridLineDashLengths)

}

else

{

    context.setLineDash(phase: 0.0, lengths: [])

}


let valueToPixelMatrix = transformer.valueToPixelMatrix


var position = CGPoint(x: 0.0, y: 0.0)


let entries = xAxis.entries


for i in stride(from: 0, to: entries.count, by: 1)

```

```

{

    position.x = CGFloat(entries[i])

    position.y = position.x

    position = position.applying(valueToPixelMatrix)

    drawGridLine(context: context, x: position.x, y: position.y)

}

}

```

```

@objc open var gridClippingRect: CGRect

{

    var contentRect = viewPortHandler.contentRect

    let dx = self.axis?.gridLineWidth ?? 0.0

    contentRect.origin.x -= dx / 2.0

    contentRect.size.width += dx

    return contentRect

}

```

```

@objc open func drawGridLine(context: CGContext, x: CGFloat, y: CGFloat)

{

    if x >= viewPortHandler.offsetLeft

        && x <= viewPortHandler.chartWidth

    {

        context.beginPath()

        context.move(to: CGPoint(x: x, y: viewPortHandler.contentTop))

        context.addLine(to: CGPoint(x: x, y: viewPortHandler.contentBottom))

        context.strokePath()

    }

}

```

```

open override func renderLimitLines(context: CGContext)

{

    guard

        let xAxis = self.axis as? XAxis,

```



```

let transformer = self.transformer,

!xAxis.limitLines.isEmpty

else { return }


let trans = transformer.valueToPixelMatrix


var position = CGPoint(x: 0.0, y: 0.0)


for l in xAxis.limitLines where l.isEnabled

{

context.saveGState()

defer { context.restoreGState() }


var clippingRect = viewPortHandler.contentRect

clippingRect.origin.x -= l.lineWidth / 2.0

clippingRect.size.width += l.lineWidth

context.clip(to: clippingRect)

```

```
position.x = CGFloat(l.limit)
```

```
position.y = 0.0
```

```
position = position.applying(trans)
```

```
renderLimitLineLine(context: context, limitLine: l, position: position)
```

```
renderLimitLineLabel(context: context, limitLine: l, position: position, yOffset: 2.0  
+ l.yOffset)
```

```
}
```

```
}
```

```
@objc open func renderLimitLineLine(context: CGContext, limitLine: ChartLimitLine,  
position: CGPoint)
```

```
{
```

```
context.beginPath()
```

```
context.move(to: CGPoint(x: position.x, y: viewPortHandler.contentTop))
```

```
context.addLine(to: CGPoint(x: position.x, y: viewPortHandler.contentBottom))
```

```

context.setStrokeColor(limitLine.lineColor.cgColor)

context.setLineWidth(limitLine.lineWidth)

if limitLine.lineDashLengths != nil

{

    context.setLineDash(phase:        limitLine.lineDashPhase,        lengths:
limitLine.lineDashLengths!)

}

else

{

    context.setLineDash(phase: 0.0, lengths: [])

}

context.strokePath()

}

```

```

@objc open func renderLimitLineLabel(context: CGContext, limitLine:
ChartLimitLine, position: CGPoint, yOffset: CGFloat)

```

```

{

    let label = limitLine.label

    guard limitLine.drawLabelEnabled, !label.isEmpty else { return }

    let labelLineHeight = limitLine.valueFont.lineHeight

    let xOffset: CGFloat = limitLine.lineWidth + limitLine.xOffset

    let attributes: [NSAttributedString.Key : Any] = [

        .font : limitLine.valueFont,

        .foregroundColor : limitLine.valueTextColor

    ]

    let (point, align): (CGPoint, NSTextAlignment)

    switch limitLine.labelPosition {

    case .topRight:

        point = CGPoint(

```

```

        x: position.x + xOffset,

        y: viewPortHandler.contentTop + yOffset

    )

    align = .left

case .bottomRight:

    point = CGPoint(

        x: position.x + xOffset,

        y: viewPortHandler.contentBottom - labelLineHeight - yOffset

    )

    align = .left

case .topLeft:

    point = CGPoint(

        x: position.x - xOffset,

        y: viewPortHandler.contentTop + yOffset

    )

```

```
align = .right
```

```
case .bottomLeft:
```

```
point = CGPoint(
```

```
    x: position.x - xOffset,
```

```
    y: viewPortHandler.contentBottom - labelLineHeight - yOffset
```

```
)
```

```
align = .right
```

```
}
```

```
ChartUtils.drawText(
```

```
    context: context,
```

```
    text: label,
```

```
    point: point,
```

```
    align: align,
```

```
    attributes: attributes
```

```
)
```

```
}

}

//

// XAxisRendererHorizontalBarChart.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class XAxisRendererHorizontalBarChart: XAxisRenderer
```

```

{

    internal weak var chart: BarChartView?

    @objc public init(viewPortHandler: ViewPortHandler, xAxis: XAxis?, transformer:
Transformer?, chart: BarChartView)

    {

        super.init(viewPortHandler: viewPortHandler, xAxis: xAxis, transformer:
transformer)

        self.chart = chart

    }

    open override func computeAxis(min: Double, max: Double, inverted: Bool)

    {

        var min = min, max = max

        if let transformer = self.transformer

        {

```



```

// calculate the starting and entry point of the y-labels (depending on

// zoom / contentrect bounds)

if viewPortHandler.contentWidth > 10 && !viewPortHandler.isFullyZoomedOutY

{

    let      p1      =      transformer.valueForTouchPoint(CGPoint(x:
viewPortHandler.contentLeft, y: viewPortHandler.contentBottom))

    let      p2      =      transformer.valueForTouchPoint(CGPoint(x:
viewPortHandler.contentLeft, y: viewPortHandler.contentTop))

    if inverted

    {

        min = Double(p2.y)

        max = Double(p1.y)

    }

    else

    {

        min = Double(p1.y)

        max = Double(p2.y)

```

```
}
```

```
}
```

```
}
```

```
computeAxisValues(min: min, max: max)
```

```
}
```

```
open override fun computeSize()
```

```
{
```

```
    guard let
```

```
        xAxis = self.axis as? XAxis
```

```
    else { return }
```

```
    let longest = xAxis.getLongestLabel() as NSString
```

```
    let labelSize = longest.size(withAttributes: [NSAttributedString.Key.font:
xAxis.labelFont])
```

```

        let labelWidth = floor(labelSize.width + xAxis.xOffset * 3.5)

        let labelHeight = labelSize.height

        let labelRotatedSize = CGSize(width: labelSize.width, height:
labelHeight).rotatedBy(degrees: xAxis.labelRotationAngle)

        xAxis.labelWidth = labelWidth

        xAxis.labelHeight = labelHeight

        xAxis.labelRotatedWidth = round(labelRotatedSize.width + xAxis.xOffset * 3.5)

        xAxis.labelRotatedHeight = round(labelRotatedSize.height)

    }

    open override func renderAxisLabels(context: CGContext)

    {

        guard

            let xAxis = self.axis as? XAxis

        else { return }

        if !xAxis.isEnabled || !xAxis.isDrawLabelsEnabled || chart?.data === nil

```

```

    {

        return

    }

    let xoffset = xAxis.xOffset

    if xAxis.labelPosition == .top

    {

        drawLabels(context: context, pos: viewPortHandler.contentRight + xoffset, anchor:
CGPoint(x: 0.0, y: 0.5))

    }

    else if xAxis.labelPosition == .topInside

    {

        drawLabels(context: context, pos: viewPortHandler.contentRight - xoffset, anchor:
CGPoint(x: 1.0, y: 0.5))

    }

    else if xAxis.labelPosition == .bottom

    {

```

```

        drawLabels(context: context, pos: viewPortHandler.contentLeft - xoffset, anchor:
CGPoint(x: 1.0, y: 0.5))

    }

    else if xAxis.labelPosition == .bottomInside

    {

        drawLabels(context: context, pos: viewPortHandler.contentLeft + xoffset, anchor:
CGPoint(x: 0.0, y: 0.5))

    }

    else

    { // BOTH SIDED

        drawLabels(context: context, pos: viewPortHandler.contentRight + xoffset, anchor:
CGPoint(x: 0.0, y: 0.5))

        drawLabels(context: context, pos: viewPortHandler.contentLeft - xoffset, anchor:
CGPoint(x: 1.0, y: 0.5))

    }

}

```

/// draws the x-labels on the specified y-position

open override func drawLabels(context: CGContext, pos: CGFloat, anchor: CGPoint)

```

{

    guard

        let xAxis = self.axis as? XAxis,

        let transformer = self.transformer

        else { return }

    let labelFont = xAxis.labelFont

    let labelTextColor = xAxis.labelTextColor

    let labelRotationAngleRadians = xAxis.labelRotationAngle.DEG2RAD

    let centeringEnabled = xAxis.isCenterAxisLabelsEnabled

    // pre allocate to save performance (dont allocate in loop)

    var position = CGPoint(x: 0.0, y: 0.0)

    for i in stride(from: 0, to: xAxis.entryCount, by: 1)

    {

```

```
// only fill x values

position.x = 0.0

if centeringEnabled

{

    position.y = CGFloat(xAxis.centeredEntries[i])

}

else

{

    position.y = CGFloat(xAxis.entries[i])

}

transformer.pointValueToPixel(&position)

if viewPortHandler.isInBoundsY(position.y)

{
```

```

        if let label = xAxis.valueFormatter?.stringForValue(xAxis.entries[i], axis: xAxis)

        {

            drawLabel(

                context: context,

                formattedLabel: label,

                x: pos,

                y: position.y,

                attributes: [NSAttributedString.Key.font: labelFont,
NSAttributedString.Key.foregroundColor: labelTextColor],

                anchor: anchor,

                angleRadians: labelRotationAngleRadians)

        }

    }

}

}

```

```

@objc open func drawLabel(

```

```

    context: CGContext,

```



```

        formattedLabel: String,

        x: CGFloat,

        y: CGFloat,

        attributes: [NSAttributedString.Key : Any],

        anchor: CGPoint,

        angleRadians: CGFloat)

{

    ChartUtils.drawText(

        context: context,

        text: formattedLabel,

        point: CGPoint(x: x, y: y),

        attributes: attributes,

        anchor: anchor,

        angleRadians: angleRadians)

}

```

```

open override var gridClippingRect: CGRect

```

```

{

    var contentRect = viewPortHandler.contentRect

    let dy = self.axis?.gridLineWidth ?? 0.0

    contentRect.origin.y -= dy / 2.0

    contentRect.size.height += dy

    return contentRect

}

private var _gridLineSegmentsBuffer = [CGPoint](repeating: CGPoint(), count: 2)

open override func drawGridLine(context: CGContext, x: CGFloat, y: CGFloat)

{

    if viewPortHandler.isInBoundsY(y)

    {

        context.beginPath()

        context.move(to: CGPoint(x: viewPortHandler.contentLeft, y: y))

        context.addLine(to: CGPoint(x: viewPortHandler.contentRight, y: y))
    }
}

```

```

        context.strokePath()

    }

}

open override fun renderAxisLine(context: CGContext)

{

    guard let xAxis = self.axis as? XAxis else { return }

    if !xAxis.isEnabled || !xAxis.isDrawAxisLineEnabled

    {

        return

    }

    context.saveGState()

    context.setStrokeColor(xAxis.axisLineColor.cgColor)

    context.setLineWidth(xAxis.axisLineWidth)

```

```

if xAxis.axisLineDashLengths != nil

{

    context.setLineDash(phase:      xAxis.axisLineDashPhase,      lengths:
xAxis.axisLineDashLengths)

}

else

{

    context.setLineDash(phase: 0.0, lengths: [])

}


if xAxis.labelPosition == .top ||

    xAxis.labelPosition == .topInside ||

    xAxis.labelPosition == .bothSided

{

    context.beginPath()

    context.move(to:      CGPoint(x:      viewPortHandler.contentRight,      y:
viewPortHandler.contentTop))

    context.addLine(to:      CGPoint(x:      viewPortHandler.contentRight,      y:
viewPortHandler.contentBottom))

```

```

        context.strokePath()

    }

    if xAxis.labelPosition == .bottom ||

        xAxis.labelPosition == .bottomInside ||

        xAxis.labelPosition == .bothSided
    {

        context.beginPath()

        context.move(to: CGPoint(x: viewPortHandler.contentLeft, y:
viewPortHandler.contentTop))

        context.addLine(to: CGPoint(x: viewPortHandler.contentLeft, y:
viewPortHandler.contentBottom))

        context.strokePath()

    }

    context.restoreGState()

}

```

```
open override func renderLimitLines(context: CGContext)
```

```
{
```

```
    guard
```

```
        let xAxis = self.axis as? XAxis,
```

```
        let transformer = self.transformer
```

```
    else { return }
```

```
    let limitLines = xAxis.limitLines
```

```
    if limitLines.count == 0
```

```
    {
```

```
        return
```

```
    }
```

```
    let trans = transformer.valueToPixelMatrix
```

```
    var position = CGPoint(x: 0.0, y: 0.0)
```

```

for i in 0 ..< limitLines.count

{

    let l = limitLines[i]

    if !l.isEnabled

    {

        continue

    }

    context.saveGState()

    defer { context.restoreGState() }

    var clippingRect = viewPortHandler.contentRect

    clippingRect.origin.y -= l.lineWidth / 2.0

    clippingRect.size.height += l.lineWidth

    context.clip(to: clippingRect)

```

```
position.x = 0.0
```

```
position.y = CGFloat(1.limit)
```

```
position = position.applying(trans)
```

```
context.beginPath()
```

```
context.move(to: CGPoint(x: viewPortHandler.contentLeft, y: position.y))
```

```
context.addLine(to: CGPoint(x: viewPortHandler.contentRight, y: position.y))
```

```
context.setStrokeColor(l.lineColor.cgColor)
```

```
context.setLineWidth(l.lineWidth)
```

```
if l.lineDashLengths != nil
```

```
{
```

```
    context.setLineDash(phase: l.lineDashPhase, lengths: l.lineDashLengths!)
```

```
}
```

```
else
```

```
{
```



```

        context.setLineDash(phase: 0.0, lengths: [])

    }

    context.strokePath()

    let label = l.label

    // if drawing the limit-value label is enabled

    if l.drawLabelEnabled && label.count > 0

    {

        let labelLineHeight = l.valueFont.lineHeight

        let xOffset: CGFloat = 4.0 + l.xOffset

        let yOffset: CGFloat = l.lineWidth + labelLineHeight + l.yOffset

        if l.labelPosition == .topRight

        {

```

```

        ChartUtils.drawText(context: context,

            text: label,

            point: CGPoint(

                x: viewPortHandler.contentRight - xOffset,

                y: position.y - yOffset),

            align: .right,

            attributes: [NSAttributedString.Key.font: l.valueFont,
NSAttributedString.Key.foregroundColor: l.valueTextColor])

    }

    else if l.labelPosition == .bottomRight

    {

        ChartUtils.drawText(context: context,

            text: label,

            point: CGPoint(

                x: viewPortHandler.contentRight - xOffset,

                y: position.y + yOffset - labelLineHeight),

            align: .right,

```

```

        attributes: [NSAttributedString.Key.font: l.valueFont,
NSAttributedString.Key.foregroundColor: l.valueTextColor])

    }

    else if l.labelPosition == .topLeft

    {

        ChartUtils.drawText(context: context,

            text: label,

            point: CGPoint(

                x: viewPortHandler.contentLeft + xOffset,

                y: position.y - yOffset),

            align: .left,

            attributes: [NSAttributedString.Key.font: l.valueFont,
NSAttributedString.Key.foregroundColor: l.valueTextColor])

        }

    else

    {

        ChartUtils.drawText(context: context,

            text: label,

```

```

        point: CGPoint(

            x: viewPortHandler.contentLeft + xOffset,

            y: position.y + yOffset - labelLineHeight),

        align: .left,

        attributes: [NSAttributedString.Key.font: l.valueFont,
NSAttributedString.Key.foregroundColor: l.valueTextColor])

    }

}

}

}

}

//

// XAxisRendererRadarChart.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

```

```

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class XAxisRendererRadarChart: XAxisRenderer

{

    @objc open weak var chart: RadarChartView?

    @objc public init(viewPortHandler: ViewPortHandler, xAxis: XAxis?, chart:
RadarChartView)

    {

        super.init(viewPortHandler: viewPortHandler, xAxis: xAxis, transformer: nil)

        self.chart = chart

    }

```

```

open override func renderAxisLabels(context: CGContext)

{

    guard let

        xAxis = axis as? XAxis,

        let chart = chart

    else { return }


    if !xAxis.isEnabled || !xAxis.isDrawLabelsEnabled

    {

        return

    }


    let labelFont = xAxis.labelFont

    let labelTextColor = xAxis.labelTextColor

    let labelRotationAngleRadians = xAxis.labelRotationAngle.RAD2DEG

    let drawLabelAnchor = CGPoint(x: 0.5, y: 0.25)

```

```

let sliceangle = chart.sliceAngle

// calculate the factor that is needed for transforming the value to pixels

let factor = chart.factor

let center = chart.centerOffsets

for i in stride(from: 0, to: chart.data?.maxEntryCountSet?.entryCount ?? 0, by: 1)
{

    let label = xAxis.valueFormatter?.stringForValue(Double(i), axis: xAxis) ?? ""

    let angle = (sliceangle * CGFloat(i) +
chart.rotationAngle).truncatingRemainder(dividingBy: 360.0)

    let p = center.moving(distance: CGFloat(chart.yRange) * factor +
xAxis.labelRotatedWidth / 2.0, atAngle: angle)

```

```

drawLabel(context: context,

    formattedLabel: label,

    x: p.x,

    y: p.y - xAxis.labelRotatedHeight / 2.0,

    attributes: [NSAttributedString.Key.font: labelFont,
NSAttributedString.Key.foregroundColor: labelTextColor],

    anchor: drawLabelAnchor,

    angleRadians: labelRotationAngleRadians)

}

}

```

```

@objc open func drawLabel(

    context: CGContext,

    formattedLabel: String,

    x: CGFloat,

    y: CGFloat,

    attributes: [NSAttributedString.Key : Any],

```



```

        anchor: CGPoint,

        angleRadians: CGFloat)

{

    ChartUtils.drawText(

        context: context,

        text: formattedLabel,

        point: CGPoint(x: x, y: y),

        attributes: attributes,

        anchor: anchor,

        angleRadians: angleRadians)

}

open override func renderLimitLines(context: CGContext)

{

    /// XAxis LimitLines on RadarChart not yet supported.

}

}

```

```

//

// XShapeRenderer.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class XShapeRenderer : NSObject, IShapeRenderer

{

    open func renderShape(

        context: CGContext,

```

```
dataSet: IScatterChartDataSet,  
  
viewPortHandler: ViewPortHandler,  
  
point: CGPoint,  
  
color: NSUIColor)  
  
{  
  
    let shapeSize = dataSet.scatterShapeSize  
  
    let shapeHalf = shapeSize / 2.0  
  
  
    context.setLineWidth(1.0)  
  
    context.setStrokeColor(color.cgColor)  
  
  
    context.beginPath()  
  
    context.move(to: CGPoint(x: point.x - shapeHalf, y: point.y - shapeHalf))  
  
    context.addLine(to: CGPoint(x: point.x + shapeHalf, y: point.y + shapeHalf))  
  
    context.move(to: CGPoint(x: point.x + shapeHalf, y: point.y - shapeHalf))  
  
    context.addLine(to: CGPoint(x: point.x - shapeHalf, y: point.y + shapeHalf))  
  
    context.strokePath()
```

```
}

}

//

// YAxis.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

#if canImport(Uikit)
```

```

import UIKit

#endif

#if canImport(Cocoa)

import Cocoa

#endif

/// Class representing the y-axis labels settings and its entries.

/// Be aware that not all features the YLabels class provides are suitable for the RadarChart.

/// Customizations that affect the value range of the axis need to be applied before setting
data for the chart.

@objc(ChartYAxis)

open class YAxis: AxisBase

{

    @objc(YAxisLabelPosition)

    public enum LabelPosition: Int

    {

```

```
        case outsideChart

        case insideChart

    }
```

/// Enum that specifies the axis a DataSet should be plotted against, either Left or Right.

```
@objc
```

```
public enum AxisDependency: Int
```

```
{
```

```
    case left
```

```
    case right
```

```
}
```

/// indicates if the bottom y-label entry is drawn or not

```
@objc open var drawBottomYLabelEntryEnabled = true
```

/// indicates if the top y-label entry is drawn or not

```
@objc open var drawTopYLabelEntryEnabled = true
```

```
/// flag that indicates if the axis is inverted or not
```

```
@objc open var inverted = false
```

```
/// flag that indicates if the zero-line should be drawn regardless of other grid lines
```

```
@objc open var drawZeroLineEnabled = false
```

```
/// Color of the zero line
```

```
@objc open var zeroLineColor: NSUIColor? = NSUIColor.gray
```

```
/// Width of the zero line
```

```
@objc open var zeroLineWidth: CGFloat = 1.0
```

```
/// This is how much (in pixels) into the dash pattern are we starting from.
```

```
@objc open var zeroLineDashPhase = CGFloat(0.0)
```

```
/// This is the actual dash pattern.
```

```
/// I.e. [2, 3] will paint [-- -- ]
```

```
/// [1, 3, 4, 2] will paint [- ---- - ---- ]
```

```
@objc open var zeroLineDashLengths: [CGFloat]?
```

```
/// axis space from the largest value to the top in percent of the total axis range
```

```
@objc open var spaceTop = CGFloat(0.1)
```

```
/// axis space from the smallest value to the bottom in percent of the total axis range
```

```
@objc open var spaceBottom = CGFloat(0.1)
```

```
/// the position of the y-labels relative to the chart
```

```
@objc open var labelPosition = LabelPosition.outsideChart
```

```
/// the alignment of the text in the y-label
```

```
@objc open var labelAlignment: NSTextAlignment = .left
```

```
/// the horizontal offset of the y-label
```



```

@objc open var labelXOffset: CGFloat = 0.0

/// the side this axis object represents

private var _axisDependency = AxisDependency.left

/// the minimum width that the axis should take

///

/// **default**: 0.0

@objc open var minWidth = CGFloat(0)

/// the maximum width that the axis can take.

/// use Infinity for disabling the maximum.

///

/// **default**: CGFloat.infinity

@objc open var maxWidth = CGFloat(CGFloat.infinity)

public override init()

```

```
{  
  
    super.init()  
  
    self.yOffset = 0.0  
  
}
```

```
@objc public init(position: AxisDependency)
```

```
{  
  
    super.init()  
  
    _axisDependency = position
```

```
    self.yOffset = 0.0  
  
}
```

```
@objc open var axisDependency: AxisDependency
```

```
{
```

```

        return _axisDependency
    }

@objc open func requiredSize() -> CGSize
{
    let label = getLongestLabel() as NSString

    var size = label.size(withAttributes: [NSAttributedString.Key.font: labelFont])

    size.width += xOffset * 2.0

    size.height += yOffset * 2.0

    size.width = max(minWidth, min(size.width, maxWidth > 0.0 ? maxWidth :
size.width))

    return size
}

@objc open func getRequiredHeightSpace() -> CGFloat
{
    return requiredSize().height
}

```

```
/// `true` if this axis needs horizontal offset, `false` if no offset is needed.
```

```
@objc open var needsOffset: Bool
```

```
{  
  
    if isEnabled && isDrawLabelsEnabled && labelPosition == .outsideChart  
  
    {  
  
        return true  
  
    }  
  
    else  
  
    {  
  
        return false  
  
    }  
  
}
```

```
@objc open var isInverted: Bool { return inverted }
```

```
open override func calculate(min dataMin: Double, max dataMax: Double)
```

```

{

    // if custom, use value as is, else use data value

    var min = _customAxisMin ? _axisMinimum : dataMin

    var max = _customAxisMax ? _axisMaximum : dataMax


    // Make sure max is greater than min


    //
    https://github.com/danielgindi/Charts/pull/3650#discussion_r221409991

    if min > max

    {

        switch(_customAxisMax, _customAxisMin)

        {

            case(true, true):

                (min, max) = (max, min)

            case(true, false):

                min = max < 0 ? max * 1.5 : max * 0.5

            case(false, true):

                max = min < 0 ? min * 0.5 : min * 1.5

```

Discussion:

```

    case(false, false):

        break

    }

}

// temporary range (before calculations)

let range = abs(max - min)


// in case all values are equal

if range == 0.0

{

    max = max + 1.0

    min = min - 1.0

}


// bottom-space only effects non-custom min

if !_customAxisMin

```

```

{

    let bottomSpace = range * Double(spaceBottom)

    _axisMinimum = (min - bottomSpace)

}

```

```

// top-space only effects non-custom max

```

```

if !_customAxisMax

```

```

{

    let topSpace = range * Double(spaceTop)

    _axisMaximum = (max + topSpace)

}

```

```

// calc actual range

```

```

axisRange = abs(_axisMaximum - _axisMinimum)

```

```

}

```

```

@objc open var isDrawBottomYLabelEntryEnabled: Bool { return
drawBottomYLabelEntryEnabled }

```

```
        @objc open var isDrawTopYLabelEntryEnabled: Bool { return  
drawTopYLabelEntryEnabled }
```

```
}
```

```
//
```

```
// YAxisRenderer.swift
```

```
// Charts
```

```
//
```

```
// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda
```

```
// A port of MPAndroidChart for iOS
```

```
// Licensed under Apache License 2.0
```

```
//
```

```
// https://github.com/danielgindi/Charts
```

```
//
```

```
import Foundation
```

```
import CoreGraphics
```



```
#if canImport(UiKit)
```

```
import UIKit
```

```
#endif
```

```
#if canImport(Cocoa)
```

```
import Cocoa
```

```
#endif
```

```
@objc(ChartYAxisRenderer)
```

```
open class YAxisRenderer: AxisRendererBase
```

```
{
```

```
    @objc public init(viewPortHandler: ViewPortHandler, yAxis: YAxis?, transformer:  
Transformer?)
```

```
    {
```

```
        super.init(viewPortHandler: viewPortHandler, transformer: transformer, axis: yAxis)
```

```
    }
```

```

/// draws the y-axis labels to the screen

open override func renderAxisLabels(context: CGContext)

{

    guard let yAxis = self.axis as? YAxis else { return }

    if !yAxis.isEnabled || !yAxis.isDrawLabelsEnabled

    {

        return

    }

    let xoffset = yAxis.xOffset

    let yoffset = yAxis.labelFont.lineHeight / 2.5 + yAxis.yOffset

    let dependency = yAxis.axisDependency

    let labelPosition = yAxis.labelPosition

    var xPos = CGFloat(0.0)

```

```

var textAlign: NSTextAlignment

if dependency == .left

{

    if labelPosition == .outsideChart

    {

        textAlign = .right

        xPos = viewPortHandler.offsetLeft - xoffset

    }

    else

    {

        textAlign = .left

        xPos = viewPortHandler.offsetLeft + xoffset

    }

}

```

```

else

{

    if labelPosition == .outsideChart

    {

        textAlign = .left

        xPos = viewPortHandler.contentRight + xoffset

    }

    else

    {

        textAlign = .right

        xPos = viewPortHandler.contentRight - xoffset

    }

}

```

```

drawYLabels(

    context: context,

    fixedPosition: xPos,

```

```

        positions: transformedPositions(),

        offset: yOffset - yAxis.labelFont.lineHeight,

        textAlign: textAlign)
    }

    open override fun renderAxisLine(context: CGContext)
    {

        guard let yAxis = self.axis as? YAxis else { return }

        if !yAxis.isEnabled || !yAxis.drawAxisLineEnabled
        {

            return
        }

        context.saveGState()

        context.setStrokeColor(yAxis.axisLineColor.cgColor)

```

```

context.setLineWidth(yAxis.axisLineWidth)

if yAxis.axisLineDashLengths != nil

{

    context.setLineDash(phase:        yAxis.axisLineDashPhase,        lengths:
yAxis.axisLineDashLengths)

}

else

{

    context.setLineDash(phase: 0.0, lengths: [])

}


if yAxis.axisDependency == .left

{

    context.beginPath()

    context.move(to:        CGPoint(x:        viewPortHandler.contentLeft,        y:
viewPortHandler.contentTop))

    context.addLine(to:        CGPoint(x:        viewPortHandler.contentLeft,        y:
viewPortHandler.contentBottom))

    context.strokePath()

```

```

    }

    else

    {

        context.beginPath()

        context.move(to: CGPoint(x: viewPortHandler.contentRight, y:
viewPortHandler.contentTop))

        context.addLine(to: CGPoint(x: viewPortHandler.contentRight, y:
viewPortHandler.contentBottom))

        context.strokePath()

    }

    context.restoreGState()

}

```

/// draws the y-labels on the specified x-position

```

open func drawYLabels(

    context: CGContext,

    fixedPosition: CGFloat,

```

```

        positions: [CGPoint],

        offset: CGFloat,

        textAlign: NSTextAlignment)

{

    guard

        let yAxis = self.axis as? YAxis

        else { return }

    let labelFont = yAxis.labelFont

    let labelTextColor = yAxis.labelTextColor

    let from = yAxis.isDrawBottomYLabelEntryEnabled ? 0 : 1

    let to = yAxis.isDrawTopYLabelEntryEnabled ? yAxis.entryCount :
(yAxis.entryCount - 1)

    let xOffset = yAxis.labelXOffset

    for i in stride(from: from, to: to, by: 1)

```



```

{

    let text = yAxis.getFormattedLabel(i)

    ChartUtils.drawText(

        context: context,

        text: text,

        point: CGPoint(x: fixedPosition + xOffset, y: positions[i].y + offset),

        align: textAlign,

        attributes: [.font: labelFont, .foregroundColor: labelTextColor]

    )

}

}

```

```

open override func renderGridLines(context: CGContext)

```

```

{

    guard let

        yAxis = self.axis as? YAxis

```

```
else { return }
```

```
if !yAxis.isEnabled
```

```
{
```

```
    return
```

```
}
```

```
if yAxis.drawGridLinesEnabled
```

```
{
```

```
    let positions = transformedPositions()
```

```
    context.saveGState()
```

```
    defer { context.restoreGState() }
```

```
    context.clip(to: self.gridClippingRect)
```

```
    context.setShouldAntialias(yAxis.gridAntialiasEnabled)
```

```
    context.setStrokeColor(yAxis.gridColor.cgColor)
```

```

context.setLineWidth(yAxis.gridLineWidth)

context.setLineCap(yAxis.gridLineCap)

if yAxis.gridLineDashLengths != nil
{
    context.setLineDash(phase:      yAxis.gridLineDashPhase,      lengths:
yAxis.gridLineDashLengths)

}

else

{

    context.setLineDash(phase: 0.0, lengths: [])

}

// draw the grid

positions.forEach { drawGridLine(context: context, position: $0) }

}

```

```

if yAxis.drawZeroLineEnabled

{

    // draw zero line

    drawZeroLine(context: context)

}

}

@objc open var gridClippingRect: CGRect

{

    var contentRect = viewPortHandler.contentRect

    let dy = self.axis?.gridLineWidth ?? 0.0

    contentRect.origin.y -= dy / 2.0

    contentRect.size.height += dy

    return contentRect

}

@objc open func drawGridLine(

```

```

    context: CGContext,

    position: CGPoint)

{

    context.beginPath()

    context.move(to: CGPoint(x: viewPortHandler.contentLeft, y: position.y))

    context.addLine(to: CGPoint(x: viewPortHandler.contentRight, y: position.y))

    context.strokePath()

}

```

```

@objc open func transformedPositions() -> [CGPoint]

```

```

{

    guard

        let yAxis = self.axis as? YAxis,

        let transformer = self.transformer

    else { return [CGPoint]() }


```

```

    var positions = [CGPoint]()

```

```
positions.reserveCapacity(yAxis.entryCount)
```

```
let entries = yAxis.entries
```

```
for i in stride(from: 0, to: yAxis.entryCount, by: 1)
```

```
{
```

```
    positions.append(CGPoint(x: 0.0, y: entries[i]))
```

```
}
```

```
transformer.pointValuesToPixel(&positions)
```

```
return positions
```

```
}
```

```
/// Draws the zero line at the specified position.
```

```
@objc open func drawZeroLine(context: CGContext)
```

```
{
```

```
guard
```

```
    let yAxis = self.axis as? YAxis,
```

```
    let transformer = self.transformer,
```

```
    let zeroLineColor = yAxis.zeroLineColor
```

```
    else { return }
```

```
context.saveGState()
```

```
defer { context.restoreGState() }
```

```
var clippingRect = viewPortHandler.contentRect
```

```
clippingRect.origin.y -= yAxis.zeroLineWidth / 2.0
```

```
clippingRect.size.height += yAxis.zeroLineWidth
```

```
context.clip(to: clippingRect)
```

```
context.setStrokeColor(zeroLineColor.cgColor)
```

```
context.setLineWidth(yAxis.zeroLineWidth)
```

```

let pos = transformer.pixelForValues(x: 0.0, y: 0.0)

if yAxis.zeroLineDashLengths != nil

{

    context.setLineDash(phase:        yAxis.zeroLineDashPhase,        lengths:
yAxis.zeroLineDashLengths!)

}

else

{

    context.setLineDash(phase: 0.0, lengths: [])

}


context.move(to: CGPoint(x: viewPortHandler.contentLeft, y: pos.y))

context.addLine(to: CGPoint(x: viewPortHandler.contentRight, y: pos.y))

context.drawPath(using: CGPathDrawingMode.stroke)

}

open override func renderLimitLines(context: CGContext)

```



```
{  
  
  guard  
  
    let yAxis = self.axis as? YAxis,  
  
    let transformer = self.transformer  
  
    else { return }  
  
  
  let limitLines = yAxis.limitLines  
  
  
  
  if limitLines.count == 0  
  
  {  
  
    return  
  
  }  
  
  
  
  context.saveGState()  
  
  
  
  let trans = transformer.valueToPixelMatrix
```

```

var position = CGPoint(x: 0.0, y: 0.0)

for i in 0 ..< limitLines.count
{
    let l = limitLines[i]

    if !l.isEnabled
    {
        continue
    }

    context.saveGState()

    defer { context.restoreGState() }

    var clippingRect = viewPortHandler.contentRect

    clippingRect.origin.y -= l.lineWidth / 2.0

    clippingRect.size.height += l.lineWidth

```

```
context.clip(to: clippingRect)
```

```
position.x = 0.0
```

```
position.y = CGFloat(l.limit)
```

```
position = position.applying(trans)
```

```
context.beginPath()
```

```
context.move(to: CGPoint(x: viewPortHandler.contentLeft, y: position.y))
```

```
context.addLine(to: CGPoint(x: viewPortHandler.contentRight, y: position.y))
```

```
context.setStrokeColor(l.lineColor.cgColor)
```

```
context.setLineWidth(l.lineWidth)
```

```
if l.lineDashLengths != nil
```

```
{
```

```
    context.setLineDash(phase: l.lineDashPhase, lengths: l.lineDashLengths!)
```

```
}
```

```
else
```

```

{

    context.setLineDash(phase: 0.0, lengths: [])

}

```

```

context.strokePath()

```

```

let label = l.label

```

```

// if drawing the limit-value label is enabled

```

```

if l.drawLabelEnabled && label.count > 0

```

```

{

```

```

    let labelLineHeight = l.valueFont.lineHeight

```

```

    let xOffset: CGFloat = 4.0 + l.xOffset

```

```

    let yOffset: CGFloat = l.lineWidth + labelLineHeight + l.yOffset

```

```

    if l.labelPosition == .topRight

```

```

{

    ChartUtils.drawText(context: context,

        text: label,

        point: CGPoint(

            x: viewPortHandler.contentRight - xOffset,

            y: position.y - yOffset),

        align: .right,

        attributes: [NSAttributedString.Key.font: l.valueFont,
NSAttributedString.Key.foregroundColor: l.valueTextColor])

    }

    else if l.labelPosition == .bottomRight

    {

        ChartUtils.drawText(context: context,

            text: label,

            point: CGPoint(

                x: viewPortHandler.contentRight - xOffset,

                y: position.y + yOffset - labelLineHeight),

            align: .right,

```

```

        attributes: [NSAttributedString.Key.font: l.valueFont,
NSAttributedString.Key.foregroundColor: l.valueTextColor])

    }

    else if l.labelPosition == .topLeft

    {

        ChartUtils.drawText(context: context,

            text: label,

            point: CGPoint(

                x: viewPortHandler.contentLeft + xOffset,

                y: position.y - yOffset),

            align: .left,

            attributes: [NSAttributedString.Key.font: l.valueFont,
NSAttributedString.Key.foregroundColor: l.valueTextColor])

        }

    else

    {

        ChartUtils.drawText(context: context,

            text: label,

```

```

        point: CGPoint(

            x: viewPortHandler.contentLeft + xOffset,

            y: position.y + yOffset - labelLineHeight),

        align: .left,

        attributes: [NSAttributedString.Key.font: l.valueFont,
NSAttributedString.Key.foregroundColor: l.valueTextColor])

    }

}

}

context.restoreGState()

}

}

//

// YAxisRendererHorizontalBarChart.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

```

```

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

open class YAxisRendererHorizontalBarChart: YAxisRenderer

{

    public override init(viewPortHandler: ViewPortHandler, yAxis: YAxis?, transformer:
Transformer?)

    {

        super.init(viewPortHandler: viewPortHandler, yAxis: yAxis, transformer:
transformer)

    }

```



```

/// Computes the axis values.

open override func computeAxis(min: Double, max: Double, inverted: Bool)

{

    guard let transformer = self.transformer else { return }

    var min = min, max = max


    // calculate the starting and entry point of the y-labels (depending on zoom /
    contentrect bounds)

    if viewPortHandler.contentHeight > 10.0 && !viewPortHandler.isFullyZoomedOutX

    {

        let p1 = transformer.valueForTouchPoint(CGPoint(x:
viewPortHandler.contentLeft, y: viewPortHandler.contentTop))

        let p2 = transformer.valueForTouchPoint(CGPoint(x:
viewPortHandler.contentRight, y: viewPortHandler.contentTop))


        if !inverted

        {

            min = Double(p1.x)

```

```

        max = Double(p2.x)

    }

    else

    {

        min = Double(p2.x)

        max = Double(p1.x)

    }

}

computeAxisValues(min: min, max: max)

}

/// draws the y-axis labels to the screen

open override fun renderAxisLabels(context: CGContext)

{

    guard let yAxis = axis as? YAxis else { return }

```

```
if !yAxis.isEnabled || !yAxis.isDrawLabelsEnabled
```

```
{
```

```
    return
```

```
}
```

```
let lineHeight = yAxis.labelFont.lineHeight
```

```
let baseYOffset: CGFloat = 2.5
```

```
let dependency = yAxis.axisDependency
```

```
let labelPosition = yAxis.labelPosition
```

```
var yPos: CGFloat = 0.0
```

```
if dependency == .left
```

```
{
```

```
    if labelPosition == .outsideChart
```

```
    {
```

```

        yPos = viewPortHandler.contentTop - baseYOffset

    }

    else

    {

        yPos = viewPortHandler.contentTop - baseYOffset

    }

}

else

{

    if labelPosition == .outsideChart

    {

        yPos = viewPortHandler.contentBottom + lineHeight + baseYOffset

    }

    else

    {

        yPos = viewPortHandler.contentBottom + lineHeight + baseYOffset

    }

}

```

```
}
```

```
// For compatibility with Android code, we keep above calculation the same,
```

```
// And here we pull the line back up
```

```
yPos -= lineHeight
```

```
drawYLabels(
```

```
    context: context,
```

```
    fixedPosition: yPos,
```

```
    positions: transformedPositions(),
```

```
    offset: yAxis.yOffset)
```

```
}
```

```
open override fun renderAxisLine(context: CGContext)
```

```
{
```

```
    guard let yAxis = axis as? YAxis else { return }
```

```

if !yAxis.isEnabled || !yAxis.drawAxisLineEnabled

{

    return

}

context.saveGState()

context.setStrokeColor(yAxis.axisLineColor.cgColor)

context.setLineWidth(yAxis.axisLineWidth)

if yAxis.axisLineDashLengths != nil

{

    context.setLineDash(phase:          yAxis.axisLineDashPhase,          lengths:
yAxis.axisLineDashLengths)

}

else

{

    context.setLineDash(phase: 0.0, lengths: [])

}

```

```

if yAxis.axisDependency == .left

{

    context.beginPath()

    context.move(to: CGPoint(x: viewPortHandler.contentLeft, y:
viewPortHandler.contentTop))

    context.addLine(to: CGPoint(x: viewPortHandler.contentRight, y:
viewPortHandler.contentTop))

    context.strokePath()

}

else

{

    context.beginPath()

    context.move(to: CGPoint(x: viewPortHandler.contentLeft, y:
viewPortHandler.contentBottom))

    context.addLine(to: CGPoint(x: viewPortHandler.contentRight, y:
viewPortHandler.contentBottom))

    context.strokePath() }

```

```

        context.restoreGState()

    }

    /// draws the y-labels on the specified x-position

    @objc open func drawYLabels(

        context: CGContext,

        fixedPosition: CGFloat,

        positions: [CGPoint],

        offset: CGFloat)

    {

        guard let

            yAxis = axis as? YAxis

        else { return }

        let labelFont = yAxis.labelFont

        let labelTextColor = yAxis.labelTextColor

```



```
let from = yAxis.isDrawBottomYLabelEntryEnabled ? 0 : 1
```

```
let to = yAxis.isDrawTopYLabelEntryEnabled ? yAxis.entryCount :  
(yAxis.entryCount - 1)
```

```
let xOffset = yAxis.labelXOffset
```

```
for i in stride(from: from, to: to, by: 1)
```

```
{
```

```
let text = yAxis.getFormattedLabel(i)
```

```
ChartUtils.drawText(  
    context: context,  
    text: text,  
    point: CGPoint(  
        x: positions[i].x, y:  
        fixedPosition - offset + xOffset  
    ),  
    align: .center,
```

```
        attributes: [NSAttributedString.Key.font: labelFont,  
NSAttributedString.Key.foregroundColor: labelTextColor])
```

```
    }
```

```
}
```

```
open override var gridClippingRect: CGRect
```

```
{
```

```
    var contentRect = viewportHandler.contentRect
```

```
    let dx = self.axis?.gridLineWidth ?? 0.0
```

```
    contentRect.origin.x -= dx / 2.0
```

```
    contentRect.size.width += dx
```

```
    return contentRect
```

```
}
```

```
open override func drawGridLine(
```

```
    context: CGContext,
```

```
    position: CGPoint)
```

```
{
```

```

context.beginPath()

context.move(to: CGPoint(x: position.x, y: viewPortHandler.contentTop))

context.addLine(to: CGPoint(x: position.x, y: viewPortHandler.contentBottom))

context.strokePath()

}

```

```

open override fun transformedPositions() -> [CGPoint]

```

```

{

    guard

        let yAxis = self.axis as? YAxis,

        let transformer = self.transformer

    else { return [CGPoint]() }

```

```

var positions = [CGPoint]()

```

```

positions.reserveCapacity(yAxis.entryCount)

```

```

let entries = yAxis.entries

```

```

    for i in stride(from: 0, to: yAxis.entryCount, by: 1)

    {

        positions.append(CGPoint(x: entries[i], y: 0.0))

    }

    transformer.pointValuesToPixel(&positions)

    return positions

}

/// Draws the zero line at the specified position.

open override func drawZeroLine(context: CGContext)

{

    guard

        let yAxis = self.axis as? YAxis,

        let transformer = self.transformer,

```

```

    let zeroLineColor = yAxis.zeroLineColor

    else { return }

context.saveGState()

defer { context.restoreGState() }

var clippingRect = viewPortHandler.contentRect

clippingRect.origin.x -= yAxis.zeroLineWidth / 2.0

clippingRect.size.width += yAxis.zeroLineWidth

context.clip(to: clippingRect)

context.setStrokeColor(zeroLineColor.cgColor)

context.setLineWidth(yAxis.zeroLineWidth)

let pos = transformer.pixelForValues(x: 0.0, y: 0.0)

if yAxis.zeroLineDashLengths != nil

```

```

    {

        context.setLineDash(phase:          yAxis.zeroLineDashPhase,          lengths:
yAxis.zeroLineDashLengths!)

    }

    else

    {

        context.setLineDash(phase: 0.0, lengths: [])

    }


    context.move(to: CGPoint(x: pos.x - 1.0, y: viewPortHandler.contentTop))

    context.addLine(to: CGPoint(x: pos.x - 1.0, y: viewPortHandler.contentBottom))

    context.drawPath(using: CGPathDrawingMode.stroke)

}


private var _limitLineSegmentsBuffer = [CGPoint](repeating: CGPoint(), count: 2)


open override func renderLimitLines(context: CGContext)

{

```

```
guard

    let yAxis = axis as? YAxis,

    let transformer = self.transformer

    else { return }

let limitLines = yAxis.limitLines

if limitLines.count <= 0

{

    return

}

context.saveGState()

let trans = transformer.valueToPixelMatrix

var position = CGPoint(x: 0.0, y: 0.0)
```

```

for i in 0 ..< limitLines.count

{

    let l = limitLines[i]

    if !l.isEnabled

    {

        continue

    }

    context.saveGState()

    defer { context.restoreGState() }

    var clippingRect = viewPortHandler.contentRect

    clippingRect.origin.x -= l.lineWidth / 2.0

    clippingRect.size.width += l.lineWidth

    context.clip(to: clippingRect)

```



```
position.x = CGFloat(l.limit)
```

```
position.y = 0.0
```

```
position = position.applying(trans)
```

```
context.beginPath()
```

```
context.move(to: CGPoint(x: position.x, y: viewPortHandler.contentTop))
```

```
context.addLine(to: CGPoint(x: position.x, y: viewPortHandler.contentBottom))
```

```
context.setStrokeColor(l.lineColor.cgColor)
```

```
context.setLineWidth(l.lineWidth)
```

```
if l.lineDashLengths != nil
```

```
{
```

```
    context.setLineDash(phase: l.lineDashPhase, lengths: l.lineDashLengths!)
```

```
}
```

```
else
```

```
{
```

```

        context.setLineDash(phase: 0.0, lengths: [])

    }

    context.strokePath()

    let label = l.label

    // if drawing the limit-value label is enabled

    if l.drawLabelEnabled && label.count > 0

    {

        let labelLineHeight = l.valueFont.lineHeight

        let xOffset: CGFloat = l.lineWidth + l.xOffset

        let yOffset: CGFloat = 2.0 + l.yOffset

        if l.labelPosition == .topRight

        {

```

```

        ChartUtils.drawText(context: context,

            text: label,

            point: CGPoint(

                x: position.x + xOffset,

                y: viewPortHandler.contentTop + yOffset),

            align: .left,

            attributes: [NSAttributedString.Key.font: l.valueFont,
NSAttributedString.Key.foregroundColor: l.valueTextColor])

    }

    else if l.labelPosition == .bottomRight

    {

        ChartUtils.drawText(context: context,

            text: label,

            point: CGPoint(

                x: position.x + xOffset,

                y: viewPortHandler.contentBottom - labelLineHeight - yOffset),

            align: .left,

```

```

        attributes: [NSAttributedString.Key.font: l.valueFont,
NSAttributedString.Key.foregroundColor: l.valueTextColor])

    }

    else if l.labelPosition == .topLeft

    {

        ChartUtils.drawText(context: context,

            text: label,

            point: CGPoint(

                x: position.x - xOffset,

                y: viewPortHandler.contentTop + yOffset),

            align: .right,

            attributes: [NSAttributedString.Key.font: l.valueFont,
NSAttributedString.Key.foregroundColor: l.valueTextColor])

        }

    else

    {

        ChartUtils.drawText(context: context,

            text: label,

```

```

        point: CGPoint(

            x: position.x - xOffset,

            y: viewPortHandler.contentBottom - labelLineHeight - yOffset),

        align: .right,

        attributes: [NSAttributedString.Key.font: l.valueFont,
NSAttributedString.Key.foregroundColor: l.valueTextColor])

    }

}

}

context.restoreGState()

}

}

//

// YAxisRendererRadarChart.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

```

```
// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//
```

```
import Foundation
```

```
import CoreGraphics
```

```
#if canImport(Uikit)
```

```
    import UIKit
```

```
#endif
```

```
#if canImport(Cocoa)
```

```
    import Cocoa
```

```
#endif
```

```
open class YAxisRendererRadarChart: YAxisRenderer
```

```
{
```

```
    private weak var chart: RadarChartView?
```

```
    @objc public init(viewPortHandler: ViewPortHandler, yAxis: YAxis?, chart: RadarChartView)
```

```
{
```

```
    super.init(viewPortHandler: viewPortHandler, yAxis: yAxis, transformer: nil)
```

```
    self.chart = chart
```

```
}
```

```
open override func computeAxisValues(min yMin: Double, max yMax: Double)
```

```
{
```

```
    guard let
```

```
        axis = axis as? YAxis
```

```
    else { return }
```

```
let labelCount = axis.labelCount
```

```
let range = abs(yMax - yMin)
```

```
if labelCount == 0 || range <= 0 || range.isInfinite
```

```
{
```

```
    axis.entries = [Double]()
```

```
    axis.centeredEntries = [Double]()
```

```
    return
```

```
}
```

```
// Find out how much spacing (in yValue space) between axis values
```

```
let rawInterval = range / Double(labelCount)
```

```
var interval = rawInterval.roundedToNextSignificant()
```

```
// If granularity is enabled, then do not allow the interval to go below specified  
granularity.
```

```
// This is used to avoid repeated values when rounding values for display.
```

```
if axis.isGranularityEnabled
```



```

{

    interval = interval < axis.granularity ? axis.granularity : interval

}


// Normalize interval

let intervalMagnitude = pow(10.0, floor(log10(interval))).roundedToNextSignificant()

let intervalSigDigit = Int(interval / intervalMagnitude)


if intervalSigDigit > 5

{

    // Use one order of magnitude higher, to avoid intervals like 0.9 or 90

    // if it's 0.0 after floor(), we use the old value

    interval = floor(10.0 * intervalMagnitude) == 0.0 ? interval : floor(10.0 *
intervalMagnitude)

}


let centeringEnabled = axis.isCenterAxisLabelsEnabled

var n = centeringEnabled ? 1 : 0

```

```

// force label count

if axis.isForceLabelsEnabled

{

    let step = Double(range) / Double(labelCount - 1)


    // Ensure stops contains at least n elements.

    axis.entries.removeAll(keepingCapacity: true)

    axis.entries.reserveCapacity(labelCount)


    var v = yMin


    for _ in 0 ..< labelCount

    {

        axis.entries.append(v)

        v += step

    }

```

```

        n = labelCount

    }

    else

    {

        // no forced count

        var first = interval == 0.0 ? 0.0 : ceil(yMin / interval) * interval


        if centeringEnabled

        {

            first -= interval

        }

        let last = interval == 0.0 ? 0.0 : (floor(yMax / interval) * interval).nextUp


        if interval != 0.0

```

```

{
    for _ in stride(from: first, through: last, by: interval)

    {

        n += 1

    }

}

```

```

n += 1

```

```

// Ensure stops contains at least n elements.

```

```

axis.entries.removeAll(keepingCapacity: true)

```

```

axis.entries.reserveCapacity(labelCount)

```

```

var f = first

```

```

var i = 0

```

```

while i < n

```

```

{

```

```

    if f == 0.0

    {

        // Fix for IEEE negative zero case (Where value == -0.0, and 0.0 == -0.0)

        f = 0.0

    }

    axis.entries.append(Double(f))

    f += interval

    i += 1

}

}

// set decimals

if interval < 1

{

    axis.decimals = Int(ceil(-log10(interval)))

```

```

    }

    else

    {

        axis.decimals = 0

    }


    if centeringEnabled

    {

        axis.centeredEntries.reserveCapacity(n)

        axis.centeredEntries.removeAll()

        let offset = (axis.entries[1] - axis.entries[0]) / 2.0

        for i in 0 ..< n

        {

            axis.centeredEntries.append(axis.entries[i] + offset)

        }

```

```
}
```

```
axis._axisMinimum = axis.entries[0]
```

```
axis._axisMaximum = axis.entries[n-1]
```

```
axis.axisRange = abs(axis._axisMaximum - axis._axisMinimum)
```

```
}
```

```
open override func renderAxisLabels(context: CGContext)
```

```
{
```

```
    guard let
```

```
        yAxis = axis as? YAxis,
```

```
        let chart = chart
```

```
    else { return }
```

```
    if !yAxis.isEnabled || !yAxis.isDrawLabelsEnabled
```

```
    {
```

```
        return
```

```
}
```

```
let labelFont = yAxis.labelFont
```

```
let labelTextColor = yAxis.labelTextColor
```

```
let center = chart.centerOffsets
```

```
let factor = chart.factor
```

```
let labelLineHeight = yAxis.labelFont.lineHeight
```

```
let from = yAxis.isDrawBottomYLabelEntryEnabled ? 0 : 1
```

```
let to = yAxis.isDrawTopYLabelEntryEnabled ? yAxis.entryCount :  
(yAxis.entryCount - 1)
```

```
let alignment: NSTextAlignment = yAxis.labelAlignment
```

```
let xOffset = yAxis.labelXOffset
```

```
for j in stride(from: from, to: to, by: 1)
```



```

{

    let r = CGFloat(yAxis.entries[j] - yAxis._axisMinimum) * factor

    let p = center.moving(distance: r, atAngle: chart.rotationAngle)

    let label = yAxis.getFormattedLabel(j)

    ChartUtils.drawText(

        context: context,

        text: label,

        point: CGPoint(x: p.x + xOffset, y: p.y - labelLineHeight),

        align: alignment,

        attributes: [

            NSAttributedString.Key.font: labelFont,

            NSAttributedString.Key.foregroundColor: labelTextColor

        ])

}

```

```
}
```

```
open override func renderLimitLines(context: CGContext)
```

```
{
```

```
    guard
```

```
        let yAxis = axis as? YAxis,
```

```
        let chart = chart,
```

```
        let data = chart.data
```

```
    else { return }
```

```
    let limitLines = yAxis.limitLines
```

```
    if limitLines.count == 0
```

```
    {
```

```
        return
```

```
    }
```

```
context.saveGState()
```

```
let sliceangle = chart.sliceAngle
```

```
// calculate the factor that is needed for transforming the value to pixels
```

```
let factor = chart.factor
```

```
let center = chart.centerOffsets
```

```
for i in 0 ..< limitLines.count
```

```
{
```

```
    let l = limitLines[i]
```

```
    if !l.isEnabled
```

```
    {
```

```
        continue
```

```
    }
```

```

context.setStrokeColor(l.lineColor.cgColor)

context.setLineWidth(l.lineWidth)

if l.lineDashLengths != nil
{
    context.setLineDash(phase: l.lineDashPhase, lengths: l.lineDashLengths!)
}
else
{
    context.setLineDash(phase: 0.0, lengths: [])
}

let r = CGFloat(l.limit - chart.chartYMin) * factor

context.beginPath()

for j in 0 ..< (data.maxEntryCountSet?.entryCount ?? 0)

```

```

    {

        let p = center.moving(distance: r, atAngle: sliceangle * CGFloat(j) +
chart.rotationAngle)

        if j == 0

        {

            context.move(to: CGPoint(x: p.x, y: p.y))

        }

        else

        {

            context.addLine(to: CGPoint(x: p.x, y: p.y))

        }

    }

    context.closePath()

    context.strokePath()

}

```

```
        context.restoreGState()

    }

}

/*
```

Copyright (C) 2016 Apple Inc. All Rights Reserved.

See LICENSE.txt for this sample's licensing information

Abstract:

The 'ZoneViewController' lists the rooms in a zone.

```
*/
```

```
import UIKit
```

```
import HomeKit
```

```
/// A view controller that lists the rooms within a provided zone.
```

```
class ZoneViewController: HMCatalogViewController {
```

```
// MARK: Types
```

```
struct Identifiers {
```

```
    static let roomCell = "RoomCell"
```

```
    static let addCell = "AddCell"
```

```
    static let disabledAddCell = "DisabledAddCell"
```

```
    static let addRoomsSegue = "Add Rooms"
```

```
}
```

```
// MARK: Properties
```

```
var homeZone: HMZone!
```

```
var rooms = [HMRoom]()
```

```
// MARK: View Methods
```

```
/// Reload the data and configure the view.
```

```

override func viewWillAppear(_ animated: Bool) {

    super.viewWillAppear(animated)

    title = homeZone.name

    reloadData()

}

/// If our data is invalid, pop the view controller.

override func viewDidAppear(_ animated: Bool) {

    super.viewDidAppear(animated)

    if shouldPopViewController() {

        _ = navigationController?.popViewController(animated: true)

    }

}

/// Provide the zone to `AddRoomViewController`.

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

    super.prepare(for: segue, sender: sender)

```



```

        if segue.identifier == Identifiers.addRoomsSegue {

            let addViewController = segue.intendedDestinationViewController as!
AddRoomViewController

            addViewController.homeZone = homeZone

        }

    }

```

// MARK: Helper Methods

/// Resets the internal list of rooms and reloads the table view.

```

private func reloadData() {

    rooms = homeZone.rooms.sortByLocalizedNames()

    tableView.reloadData()

}

```

/// Sorts the internal list of rooms by localized name.

```

private func sortRooms() {

    rooms = rooms.sortByLocalizedNames()

```

```
}
```

```
/// - returns: The `NSIndexPath` where the 'Add Cell' should be located.
```

```
private var addIndexPath: IndexPath {
```

```
    return IndexPath(row: rooms.count, section: 0)
```

```
}
```

```
/**
```

```
    - parameter indexPath: The index path in question.
```

```
    - returns: `true` if the indexPath should contain
```

```
        an 'add' cell, `false` otherwise
```

```
*/
```

```
private func indexPathIsAdd(_ indexPath: IndexPath) -> Bool {
```

```
    return indexPath.row == addIndexPath.row
```

```
}
```

```
/**
```

Reloads the `addIndexPath`.

This is typically used when something has changed to allow the user to add a room.

```
*/
```

```
private func reloadAddIndexPath() {
```

```
    tableView.reloadRows(at: [addIndexPath], with: .automatic)
```

```
}
```

```
/**
```

Adds a room to the internal array of rooms and inserts new row into the table view.

- parameter room: The new `HMRoom` to add.

```
*/
```

```
private func didAddRoom(_ room: HMRoom) {
```

```
rooms.append(room)
```

```
sortRooms()
```

```
if let newRoomIndex = rooms.firstIndex(of: room) {
```

```
    let newRoomIndexPath = IndexPath(row: newRoomIndex, section: 0)
```

```
    tableView.insertRows(at: [newRoomIndexPath], with: .automatic)
```

```
}
```

```
reloadAddIndexPath()
```

```
}
```

```
/**
```

Removes a room from the internal array of rooms and deletes

the row from the table view.

- parameter room: The `HMRoom` to remove.

```
*/
```

```
private func didRemoveRoom(_ room: HMRoom) {  
  
    if let roomIndex = rooms.firstIndex(of: room) {  
  
        rooms.remove(at: roomIndex)  
  
        let roomIndexPath = IndexPath(row: roomIndex, section: 0)  
  
        tableView.deleteRows(at: [roomIndexPath], with: .automatic)  
  
    }  
  
    reloadAddIndexPath()  
  
}
```

```
/**
```

Reloads the cell corresponding a given room.

- parameter room: The `HMRoom` to reload.

```
*/
```

```
private func didUpdateRoom(_ room: HMRoom) {
```

```

    if let roomIndex = rooms.firstIndex(of: room) {

        let roomIndexPath = IndexPath(row: roomIndex, section: 0)

        tableView.reloadRows(at: [roomIndexPath], with: .automatic)

    }

}

```

```
/**
```

Removes a room from HomeKit and updates the view.

- parameter room: The `HMRoom` to remove.

```
*/
```

```

private func removeRoom(_ room: HMRoom) {

    didRemoveRoom(room)

    homeZone.removeRoom(room) { error in

        if let error = error {

            self.displayError(error)

            self.didAddRoom(room)

```

```

    }

}

}

/**
- returns: `true` if our current home no longer
    exists, `false` otherwise.

*/

private func shouldPopViewController() -> Bool {

    for zone in home.zones {

        if zone == homeZone {

            return false

        }

    }

    return true

}

```

```

/**

- returns: `true` if more rooms can be added to this zone;

        `false` otherwise.

*/

private var canAddRoom: Bool {

    return rooms.count < home.rooms.count

}

// MARK: Table View Methods

/// - returns: The number of rooms in the zone, plus 1 for the 'add' row.

override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {

    return rooms.count + 1

}

/// - returns: A cell containing the name of an HMRoom.

```



```

        override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

            if indexPathIsAdd(indexPath) {

                let reuseIdentifier = home.isAdmin && canAddRoom ? Identifiers.addCell :
Identifiers.disabledAddCell

                return tableView.dequeueReusableCell(withIdentifier: reuseIdentifier, for:
indexPath)

            }

            let cell = tableView.dequeueReusableCell(withIdentifier: Identifiers.roomCell, for:
indexPath)

            cell.textLabel?.text = rooms[indexPath.row].name

            return cell

        }

    /**

```

- returns: `true` if the cell is anything but an 'add' cell;

`false` otherwise.

*/

```
override func tableView(_ tableView: UITableView, canEditRowAt indexPath:
IndexPath) -> Bool {
```

```
    return home.isAdmin && !indexPathIsAdd(indexPath)
```

```
}
```

/// Deletes the room at the provided index path.

```
override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {
```

```
    if editingStyle == .delete {
```

```
        let room = rooms[indexPath.row]
```

```
        removeRoom(room)
```

```
    }
```

```
}
```

```
// MARK: HMHomeDelegate Methods
```

```
/// If our zone was removed, pop the view controller.
```

```
func home(_ home: HMHome, didRemoveZone zone: HMZone) {
```

```
    if zone == homeZone {
```

```
        _ = navigationController?.popViewController(animated: true)
```

```
    }
```

```
}
```

```
/// If our zone was renamed, update the title.
```

```
func home(_ home: HMHome, didUpdateNameForZone zone: HMZone) {
```

```
    if zone == homeZone {
```

```
        title = zone.name
```

```
    }
```

```
}
```

```
/// Update the row for the room.
```

```
func home(_ home: HMHome, didUpdateNameForRoom room: HMRoom) {

    didUpdateRoom(room)

}
```

```
/**
```

A room has been added, we may be able to add it to the zone.

Reload the 'addIndexPath'

```
*/
```

```
func home(_ home: HMHome, didAddRoom room: HMRoom) {

    reloadAddIndexPath()

}
```

```
/**
```

A room has been removed, attempt to remove it from the room.

This will always reload the 'addIndexPath'.

```
*/
```

```
func home(_ home: HMHome, didRemoveRoom room: HMRoom) {
```

```

        didRemoveRoom(room)

    }

    /// If the room was added to our zone, add it to the view.

    func home(_ home: HMHome, didAddRoom room: HMRoom, toZone zone: HMZone)
    {

        if zone == homeZone {

            didAddRoom(room)

        }

    }

    /// If the room was removed from our zone, remove it from the view.

    func home(_ home: HMHome, didRemoveRoom room: HMRoom, fromZone zone:
    HMZone) {

        if zone == homeZone {

            didRemoveRoom(room)

        }

    }

```

```
}

//

// ZoomViewJob.swift

// Charts

//

// Copyright 2015 Daniel Cohen Gindi & Philipp Jahoda

// A port of MPAndroidChart for iOS

// Licensed under Apache License 2.0

//

// https://github.com/danielgindi/Charts

//

import Foundation

import CoreGraphics

@objc(ZoomChartViewJob)

open class ZoomViewJob: ViewPortJob
```

```

{

    internal var scaleX: CGFloat = 0.0

    internal var scaleY: CGFloat = 0.0

    internal var axisDependency: YAxis.AxisDependency = .left


    @objc public init(

        viewPortHandler: ViewPortHandler,

        scaleX: CGFloat,

        scaleY: CGFloat,

        xValue: Double,

        yValue: Double,

        transformer: Transformer,

        axis: YAxis.AxisDependency,

        view: ChartViewBase)

    {

        super.init(

            viewPortHandler: viewPortHandler,

```

```
xValue: xValue,  
  
yValue: yValue,  
  
transformer: transformer,  
  
view: view)
```

```
self.scaleX = scaleX  
  
self.scaleY = scaleY  
  
self.axisDependency = axis  
  
}
```

```
open override func doJob()  
  
{  
  
    guard  
  
        let viewportHandler = viewportHandler,  
  
        let transformer = transformer,  
  
        let view = view  
  
        else { return }
```



```
var matrix = viewPortHandler.setZoom(scaleX: scaleX, scaleY: scaleY)
```

```
viewPortHandler.refresh(newMatrix: matrix, chart: view, invalidate: false)
```

```
let yValsInView = (view as! BarLineChartViewBase).getAxis(axisDependency).axisRange /  
Double(viewPortHandler.scaleY)
```

```
let xValsInView = (view as! BarLineChartViewBase).xAxis.axisRange /  
Double(viewPortHandler.scaleX)
```

```
var pt = CGPoint(  
  
    x: CGFloat(xValue - xValsInView / 2.0),  
  
    y: CGFloat(yValue + yValsInView / 2.0)  
  
)
```

```
transformer.pointValueToPixel(&pt)
```

```
matrix = viewPortHandler.translate(pt: pt)
```

```
viewPortHandler.refresh(newMatrix: matrix, chart: view, invalidate: false)
```

```
(view as! BarLineChartViewBase).calculateOffsets()

view.setNeedsDisplay()

}

}
```