Другие языки:
English • русский

## Содержание


- 1 Руководство по разработке модулей
  - 1.1 Определения
  - 1.2 Концепция модуля
  - 1.3 С чего начать разработку модуля
  - 1.4 Структура модуля
  - 1.5 Создание интерфейса модуля
    - 1.5.1 Widget
    - 1.5.2 Remote и прочие графические окна
  - 1.6 Стандарт проектирования интерфейса
    - 1.6.1 Основные принципы
    - 1.6.2 Модульная сетка и блоки
    - 1.6.3 Кнопки
    - 1.6.4 Кнопка-стрелка
    - 1.6.5 Слайдеры
    - 1.6.6 Описание
    - 1.6.7 Иконки
    - 1.6.8 Виджет «header»
    - 1.6.9 Расположение кнопок
  - 1.7 Основы работы со скриптами
  - 1.8 Создание подустройств
  - 1.9 Добавление виджетов к подустройству
  - 1.10 Работа с каналами и тегами
    - 1.10.1 Пример создания каналов и тегов при создании подустройства:
  - 1.11 Установка модуля в приложение
    - 1.11.1 Разработка Setup
    - 1.11.2 Установка модуля вручную
    - 1.11.3 Установка модуля с помощью сканера
    - 1.11.4 Ручное добавление подустройств
    - 1.11.5 Авторизация
    - 1.11.6 Смена настроек
  - 1.12 Типирование модуля с помощью Smart API


# Руководство по разработке модулей

## Определения

**Модуль** - это компонент платформы, который позволяет управлять одним типом оборудования по его уникальному идентификатору. Модуль расположен в магазине модулей.

**Управляющая панель (мобильное устройство)** - устройство, с которого запускается проект i3 lite.(планшет, смартфон)

**Магазин модулей** - это облачный сервер для загрузки готовых модулей. Магазин модулей

доступен из приложения i3 lite или на нашем сайте

**iRidium Studio** - редактор, который позволяет создавать модуль управления устройством для приложения i3 lite. Для разработки модуля надо использовать специальную сборку редактора для разработчиков модулей

**i3 lite** - это приложение для создания, настройки и использования проектов i3 lite.

**Проект i3 lite** – проект управления, составленный из готовых модулей управления оборудованием. Отличие i3 lite проекта от обычного проекта i3 Pro в том, что проект i3 lite может изменяться динамически без предварительного изменения в iRidium Studio. i3 lite проект состоит из небольших частей и при помощи встроенного конструктора в приложении i3 lite может собираться в единое целое и легко редактироваться. Пропадает ряд промежуточных процессов, таких как установка программ на ПК, загрузка проекта на панель. Все происходит в одном месте - на панели управления.

**SubDevice (под-устройство)** - это виртуальное представление рабочей зоны физического устройства. Если взять для примера 6-ти канальный диммер, то у него будет 6 под-устойств, потому что на каждый канал подключается 1 независимое устройство, которое требует отдельный интерфейс управления

**Комната (в i3 lite)** - визуальная часть i3 lite, содержащая виджеты управления оборудованием. Проект в i3 lite состоит из комнат, расположенных на этажах

**Widget** - визуальный компонент модуля, размещенный в комнате, имеет ограничения по размерам и фиксированную позицию в интерфейсе комнаты. Содержит основной функционал модуля. Для дополнительного функционала используются Remote.

**Remote** - визуальный компонент модуля, который по умолчанию скрыт и открывается только при нажатии на соответствующий элемент widget'a. Не имеет ограничений по размеру (не более размера дисплея панели) и может располагаться в любой части интерфейса комнаты (обычно располагается посередине дисплея панели). Содержит дополнительный функционал модуля.

**Макрос** - последовательность команд различных модулей, которая активируется нажатием. Содержит Action (Действия) и Condition (Условие). Например макрос "выключить весь свет в доме" отправит команду выключения всем устройствам, отвечающим за освещение.

**Сценарий** - последовательность команд различных модулей, которая активируется автоматически при срабатывании определенного события. Содержит Event (Событие), Action (Действие) и Condition (условие). Например, "включить свет, если сработал датчик движения"

**Event (Событие)** - компонент сценариев, описывает управляющую команду и границы значения. При отправке значения входящего в указанные границы выполнится сценарий. Например, событие "появилось движение" активируется, если сработает датчик движения

**Action (Действие)** - компонент макросов и сценариев, описывает управляющую команду и значение, которое должно на неё (команду) отправиться. Например "включить свет" Бывает 2 типов: Simple Action и Advanced Action.

**Simple Action** - простой Action используется когда заранее известна команда и значение Action'a.

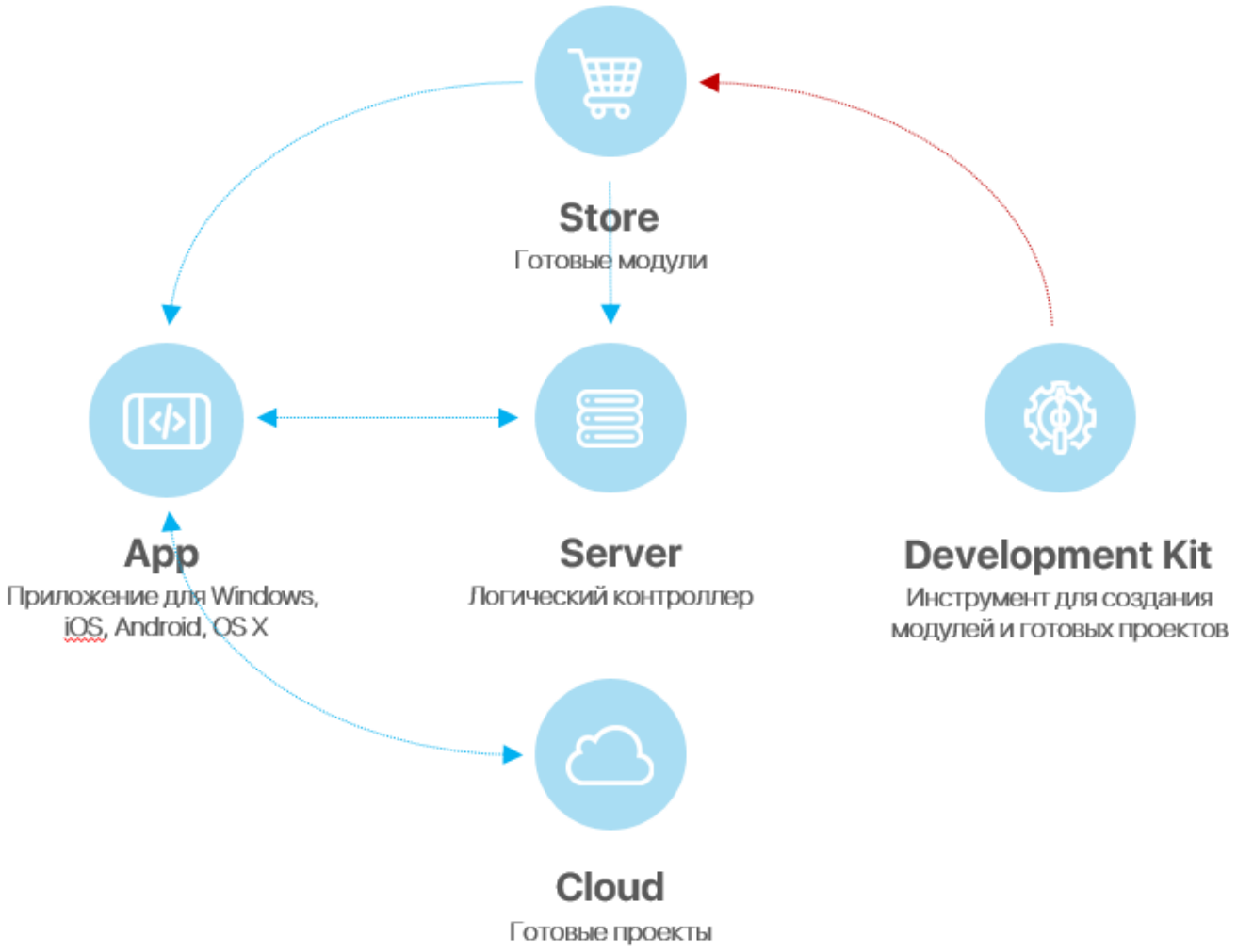
**Advanced Action** - расширенный Action используется, когда заранее известна команда, но неизвестно значение. Значение задается динамически при использовании Action'а в i3 lite (но не при создании в iRidium Studio).

**Condition (Условие)** - компонент макросов и сценариев, описывает управляющую команду и границы значения. При отправке значения входящего в указанные границы может выполняться Action.

**Scanner (сканер)** - это программная часть, которая анализирует какую-либо шину на подключенные устройства. Подключенные устройства формируются в список, для выбранного устройства качается модуль и становится доступен в i3 lite.

## Концепция модуля

Модуль это самостоятельная подпрограмма приложения i3 lite, которая встраивается в проект и имеет самостоятельный интерфейс, драйвер и логику работы. Модули разрабатываются в редакторе iRidium Studio и размещаются в магазине модулей iRidium Store. Интегратор скачивает модуль из магазина модулей для каждого отдельного проекта.



Модули бывают двух типов:

- Управляющий модуль - модуль для управления оборудованием
- Сканер - модуль для поиска оборудования в локальной сети и загрузку модулей для найденных устройств

## С чего начать разработку модуля

---

Для начала рекомендуется ознакомиться с документацией iRidium. В ней описаны основы работы в iRidium. В i3 lite документации описываются особенности при разработке модулей под i3 lite платформу, рассчитанные на пользователя знакомого с iRidium. При написании скрипта используйте i3 lite API.

---

Рассмотрим из каких частей состоит жизненный цикл модуля:

- Разработка графической части в iRidium Studio
- Разработка драйверной части (скрипты, устройства)
- Тестирование модуля
- Публикация в магазине (iRidium Store)
- Использование (Загрузка модуля через приложение i3 из облака и добавление в проект автоматизации)

## Структура модуля

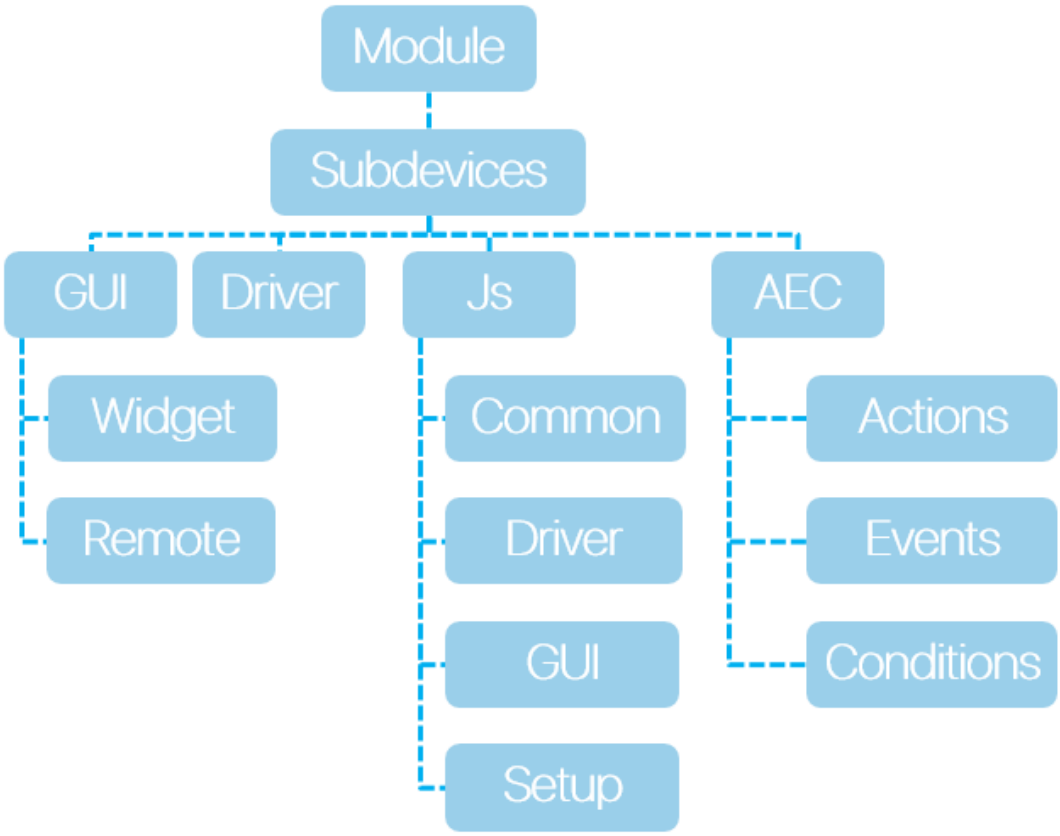
Каждый модуль должен состоять из подустройств(Subdevices). Подустройство это виртуальное представление рабочей зоны физического устройства. Многие устройства поддерживают несколько рабочих групп с одинаковыми функциями. Каждая рабочая группа может находиться в отдельной зоне. Для полноценного управления зоной необходимо создавать SubDevice для каждой рабочей группы устройства. Например 6-канальный диммер HDL, содержит в себе 6 диммируемых каналов, с помощью которых можно управлять 6 рабочими группами. Поэтому целесообразно выделить в модуле 6 SubDevice, для возможности создания действий, событий и условий для каждой рабочей группы по отдельности Наличие SubDevice в i3 lite модуле обязательно и их может быть от 1 до множества. Обычно количество SubDevice равно количеству каналов контроллера, количеству его рабочих зон или логических устройств.

Рассмотрим из чего состоит SubDevice:

- Условий (Conditions)
- Действий (Actions)
- Событий (Events)
- Виджетов (Widgets)

Условия, действия и события являются логическими элементами SubDevice, на их основе создаются макросы и сценарии. Widget (виджет) является графическим элементом, с помощью

него пользователь будет управлять SubDevice. Обычно 1 SubDevice содержит 1 виджет, но количество условий, действий и событий сильно варьируется (от 0 до множества). Количество подустройств в модуле может быть как фиксированным, так и нет. Например, можно разработать модуль только для 6-ти канального диммера, а можно сделать модуль, который будет запрашивать количество подустройств у оборудования и создавать нужное количество подустройств.




## Создание интерфейса модуля

интерфейсная часть модуля состоит из:

- виджетов - интерфейсная часть подустройства, которая располагается в комнатах и имеет основные функции управления
- пультов управления(Remote) - интерфейсная часть модуля, которая имеет полный набор функций устройства и открывается на весь экран, при нажатии на кнопку виджета
- Окно авторизации - Для работы с некоторыми устройствами требуется авторизация. Для этого разработчик может создать специальное окно авторизации, которое доступно в настройках модуля
- Окно настроек модуля - окно, в котором требуется ввести параметры для работы модуля

картинка окна настроек модуля и кнопки открытия Вся графическая часть рисуется в графическом интерфейсе редактора iRidium studio. В отличии от i3 pro, интерфейс модуля надо составлять из готовых графических компонентов, находящихся в галерее.

**Widget**

Widget (виджет), визуальный компонент модуля, разновидность Popup'а, который будет

отображен в комнате. Основная задача виджета это отображение основных элементов управления SubDevice и обеспечить переход на пульт управления (Remote) SubDevice'ом модуля. Например, для управления медиа плеером можно поместить слайдер громкости или кнопку Mute, для быстрого доступа к регулировке громкости. Погодный виджет может отображать текущую погоду, а сам пульт управления будет содержать прогноз на все 5 дней. Как и без SubDevice без виджета i3 lite модуль не имеет смысла, так как не будет иметь графического представления в i3 lite проекте.

Виджет ограничен размерами:

1. Ширина **640**
2. Высота не может превышать **1200**

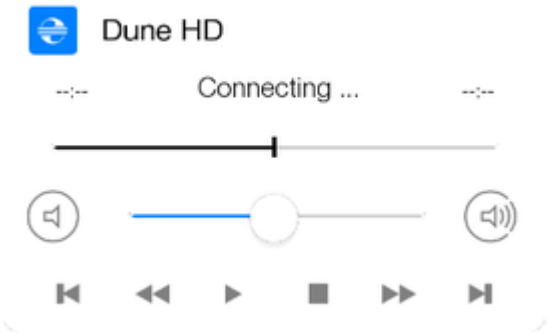
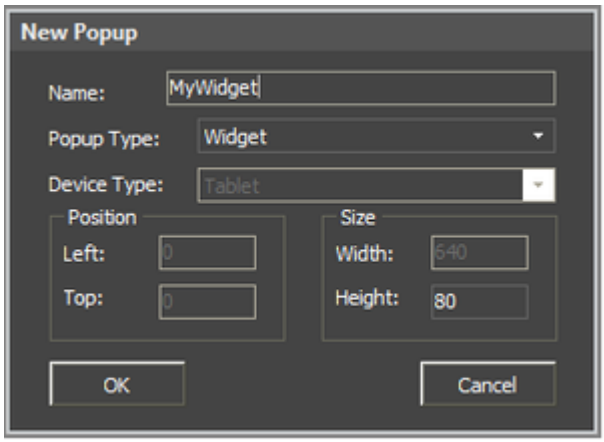



Рис. Пример виджета

Для создания виджета надо нажати кнопку добавления попапа в панели Project overview и в появившемся меню выбрать popup type: "Widget", задаём имя и свойства:




В созданном виджете можно менять параметры с помощью **Object Properties**.
Связывание графического элемента и канала происходит как в обычном Popup'е:



Рис. Связывание канала команды и графического элемента

Виджет будет отображаться:

- В комнате, при использовании пользователем проекта:

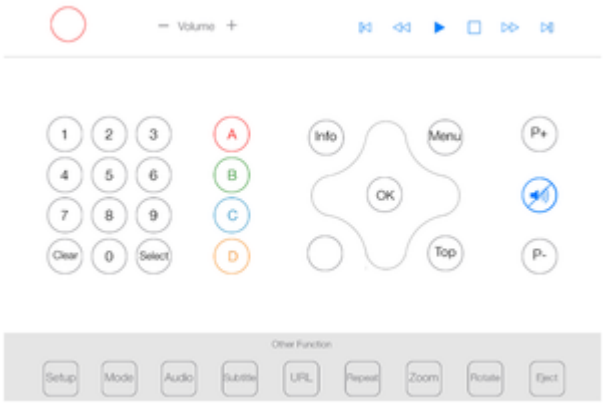



- И в настройках модуля, при добавлении его в проект:




**Remote и прочие графические окна**

Remote (пульт управления) и прочие окна - это графическое пространство для управления SubDevice, в отличии от виджета они могут иметь произвольные размеры, позицию и отображается по вызову. Пульт отображается в момент, когда на виджете была нажата кнопка перехода к пульту. На пульте управления размещаются элементы управления оборудованием (кнопки, списки, переключатели, уровни). Пультов управления может быть от 0 до нескольких. Пример пульта:

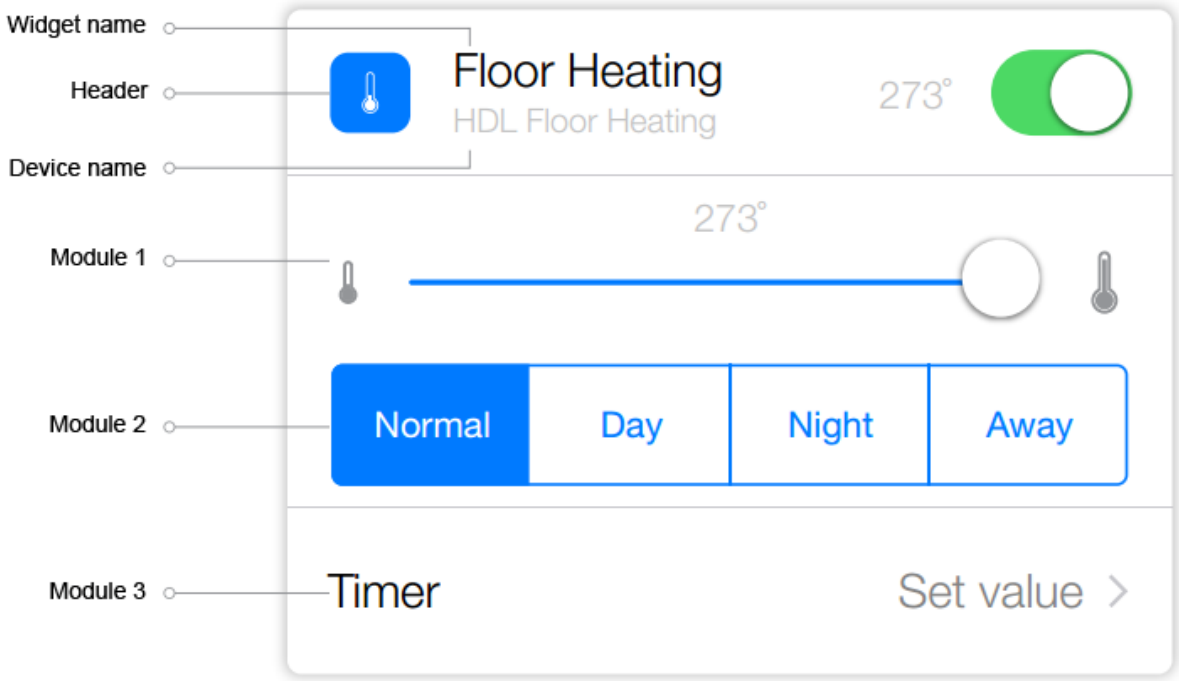



Окно создаётся аналогично виджету, с одним лишь отличием, что при нажатии "Add Popup Page" в параметре "Device Type" надо выбрать "Remote".

# Стандарт проектирования интерфейса

Для удобства проектирования интерфейса был разработан стандарт, в котором следует разрабатывать графические окна в модуле, чтобы модули разных разработчиков выглядели единообразно в рамках одного проекта.

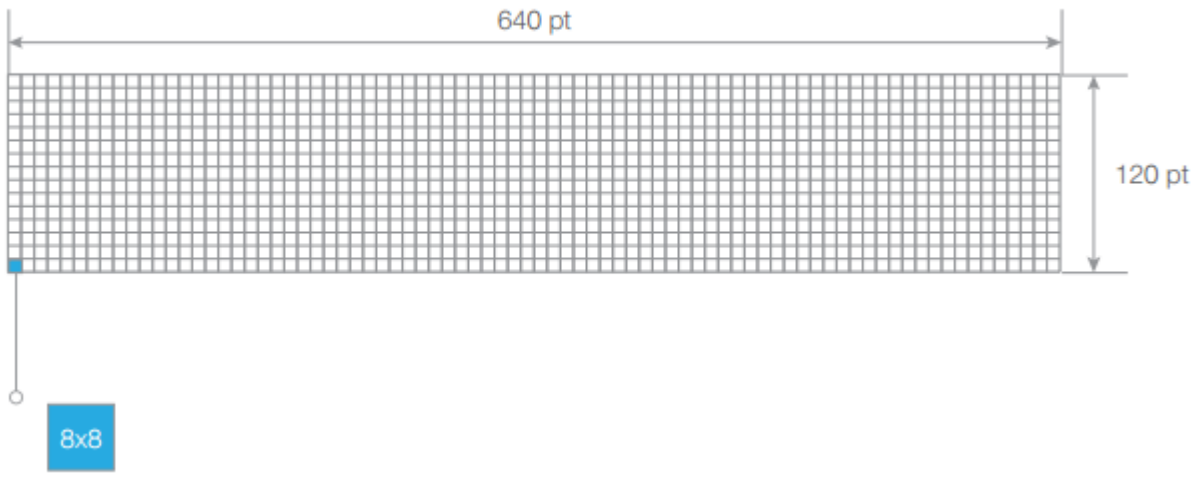
## Основные принципы



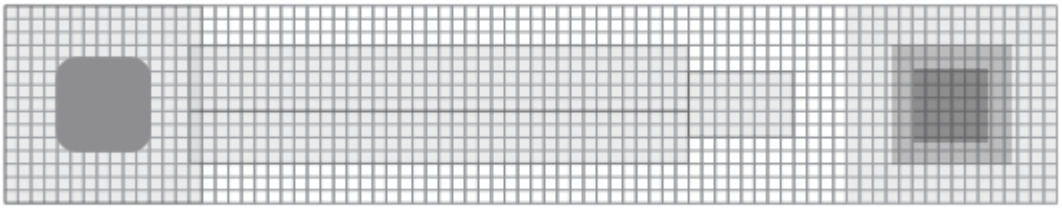

Виджет состоит из «Header» и дополнительных модулей основного функционала.
«Header» — верхний модуль виджета.
«Header» содержит:
- наименование виджета; - наименование устройства, которым управляет виджет;
- иконка категории устройства.

Дополнительные модули содержат элементы управления устройством.

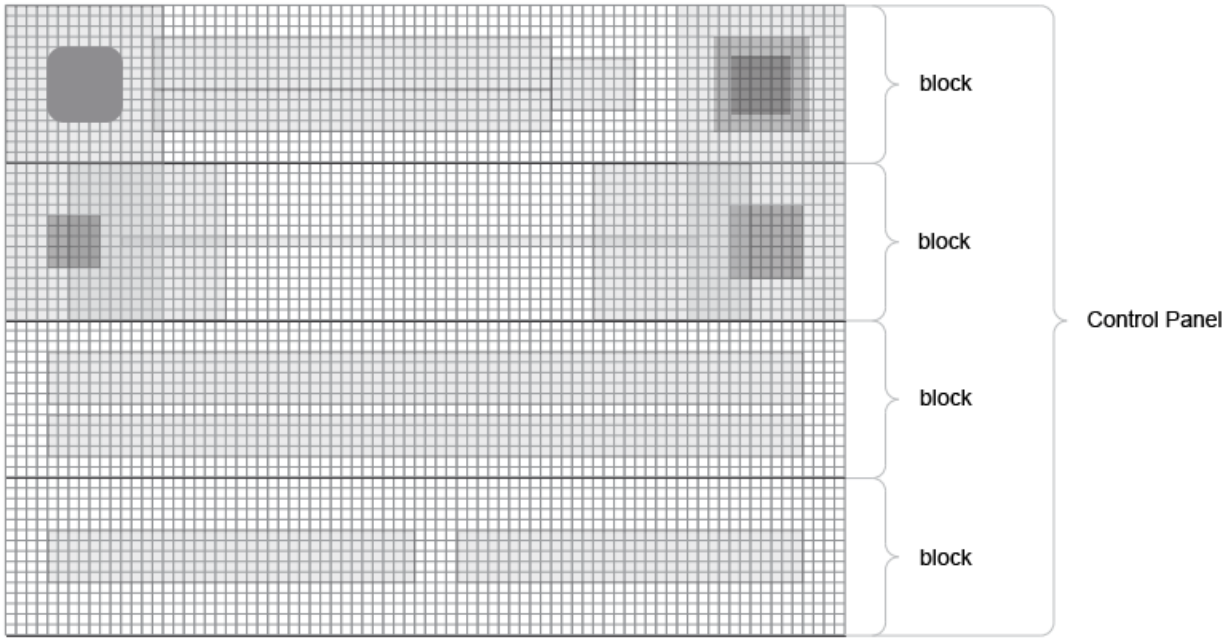
## Модульная сетка и блоки



Модульная сетка виджета представляет собой блоки 640 pt в ширину и 120 pt в высоту. Каждый блок имеет сетку 8x8 pt. Скругления углов равно 12,5 ut. (Circle 25)



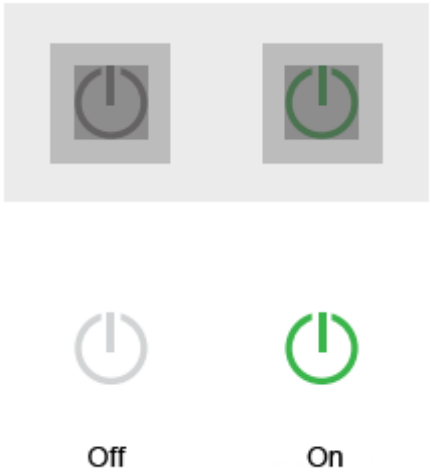
Принцип построения блока виджета представляет собой конструктор из элементов управления (кнопок, свитчей, лэйблов) от меньшего к большему.




Элементы управления комбинируются и объединяются в блок, а блоки объединяются в панель управления.

## Кнопки

## Toggle




Кнопка изменения значения «вкл/выкл». Применяется для отправки команды на устройство, вне зависимости от его текущего состояния.

### Switch




Кнопка изменения значения «вкл/выкл». Применяется только в случаях с известным состоянием работы устройства.

### Multistate

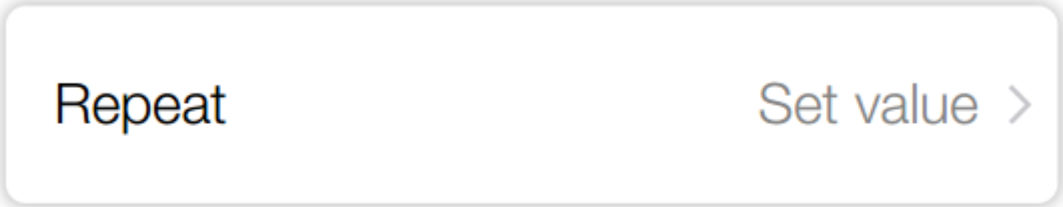


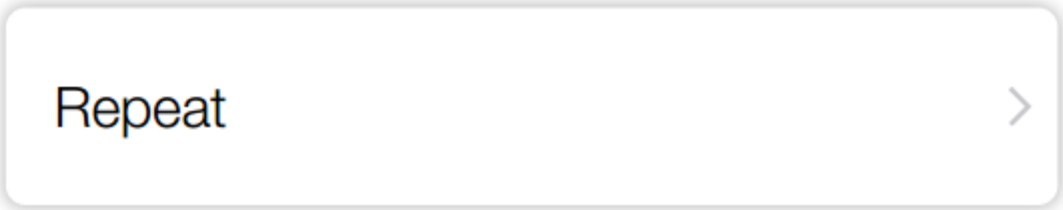


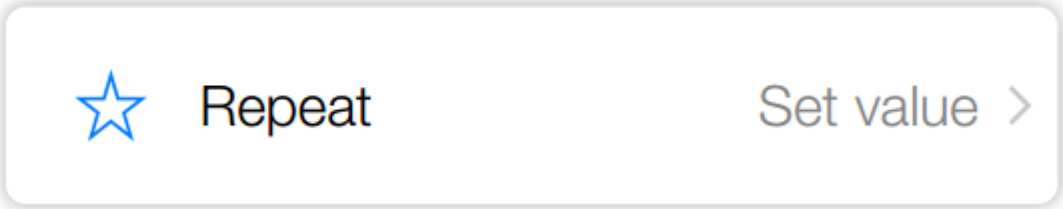


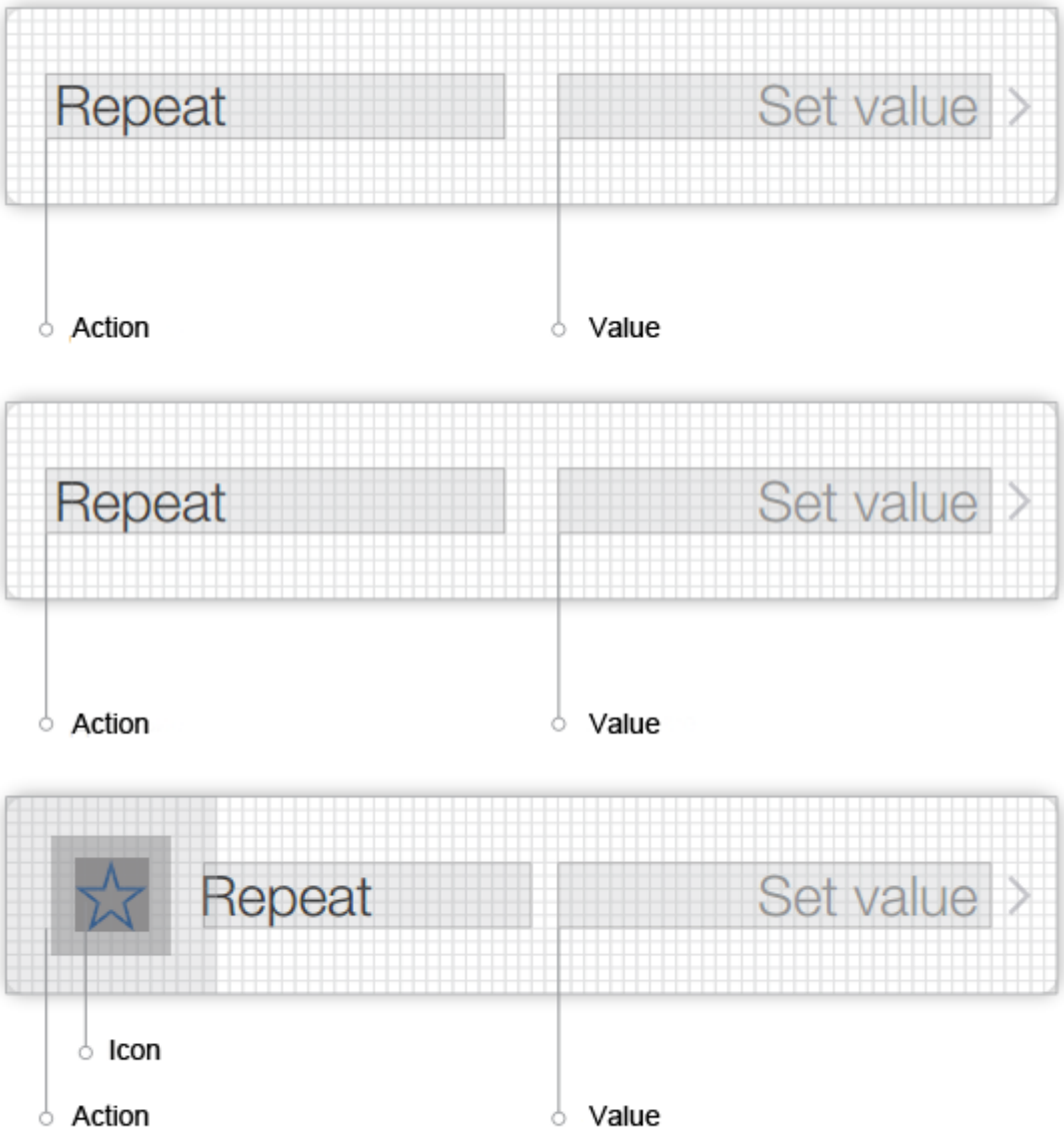
Кнопка изменения значения предустановленного состояния устройства применяется только в случаях с известным состоянием работы устройства.

**Кнопка-стрелка**



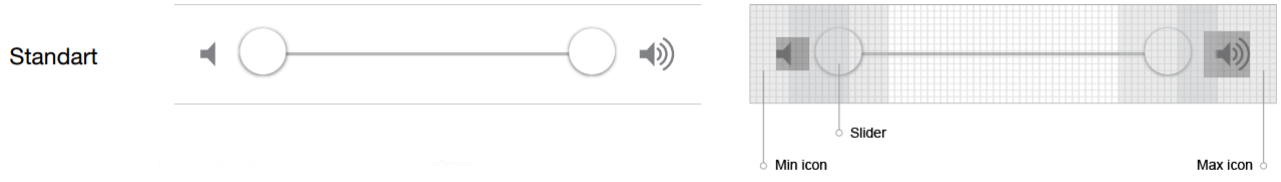





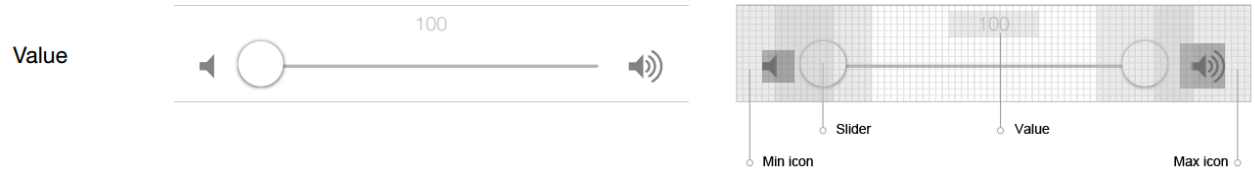

**Действие кнопки**
При нажатии на кнопку происходит вызов «PopUp».
**Содержимое «PopUp»**
«Input field», «Single selection», «Multiple selection».

## Слайдеры

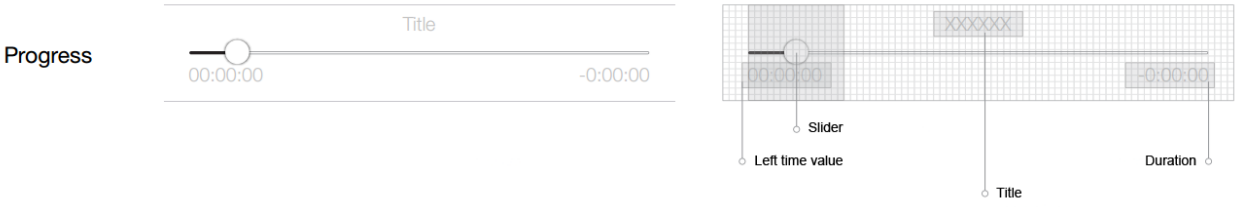
Standart



Value



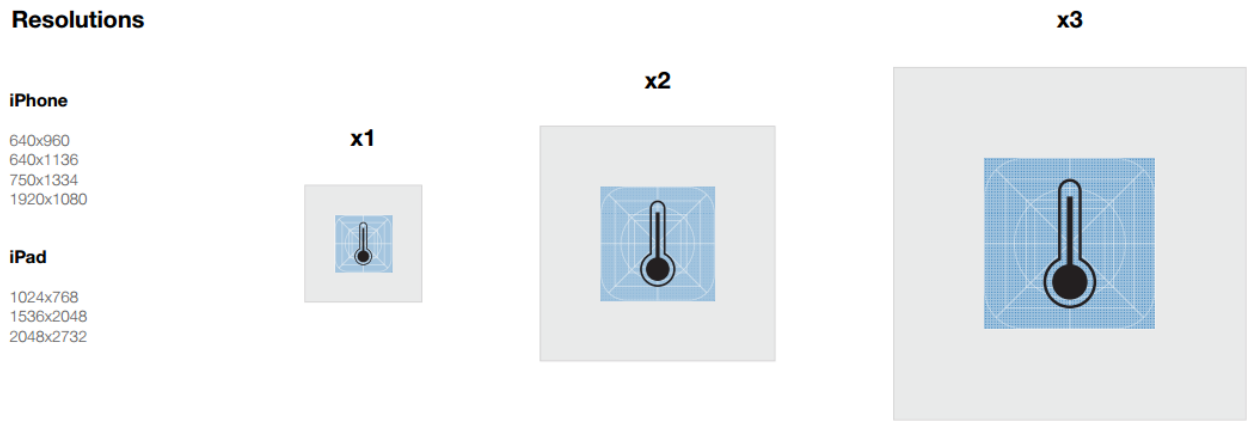

Progress



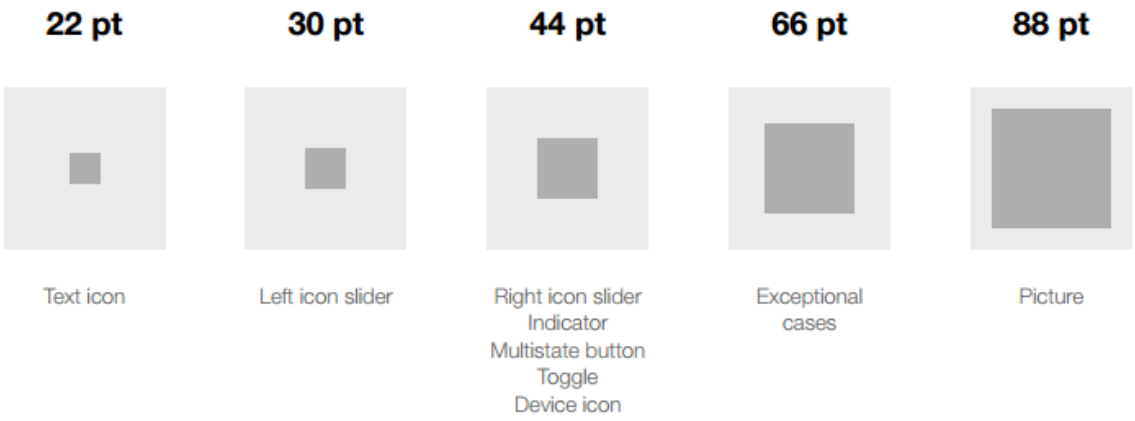

## Описание




## Иконки




Все иконки делятся по сферам применения, и их главное различие — размеры

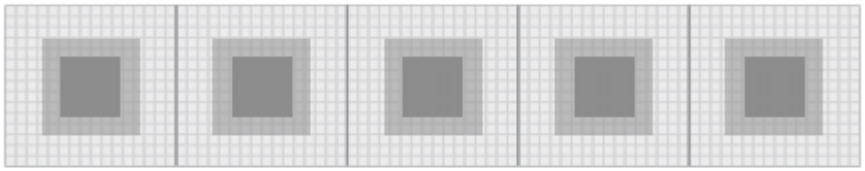



## Виджет «header»




## Расположение кнопок

Форм фактор: 128x120 pt



Все кнопки имеют два состояния — покой и активность.





В состоянии покоя цвет кнопки всегда # 848484 или RGB 132

Активным состояние кнопки считается только тогда, когда действие получило обратную связь является кратковременным или мгновенным (например: «Play» в медиаплеере) или действие, которое находится в состоянии активности продолжительное время и требует дополнительного воздействия для отключения состояния активности (например: «Repeat» в медиаплеере).

## Основы работы со скриптами

После проектирования интерфейса и добавления драйвера, вам понадобится разработать логику работы модуля в скриптах. Скриптовая часть при разработке i3 lite модуля имеет некоторые особенности. Перед разработкой скриптовой части lite модуля рекомендуем ознакомиться с iRidium Pro API и i3 lite API.

Модуль может содержать неограниченное количество скриптов. Существуют разные типы скриптов для модуля:

- Driver - в скриптовых файлах данного типа должны содержаться скрипты, отвечающие за работу с драйвером
- GUI - В этих скриптовых файлах содержатся скрипты, отвечающие за работу с графикой
- Common - в данных скриптовых файлах должны содержаться скрипты общего характера, тут может быть набор дополнительных функций или описание классов второстепенного назначения
- Setup - это специальный скриптовой файл, в нем содержиться информация о настройке модуля. а именно - список полей, обязательных для заполнения, типы этих полей и функции проверки правильности введенных данных.

Разделение на типы необходимо для правильной обработки модуля на сервере. На сервере работает только логика и драверная часть модулей, но сервер ничего не знает о графической части. Поэтому разработчику модуля требуется разбивать скрипты по типам. Сервер игнорирует файлы для работы с графической частью.

При разработке модуля следует использовать пространство имен module вместо IR. Таким образом слушатель будет иметь вид module.AddListener(…

Каждый скриптовой файл должен начинаться со слушателя

```
module.AddListener(IR.EVENT_MODULE_START, , function(){});
```

Это новый вид события, предназначенный специально для разработки модуля. При разработке модуля следует учитывать что скриптовой файл стал закрыт для других скриптовых файлов. Это хзначить что теперь можно использовать одноименные переменные и названия функций в разных скриптовых файлах и они не будут перезаписываться, но это накладывает ограничение: нельзя получить доступ к функции из другого скриптового файла простым способом. Для получения доступа к функции из другого скриптового файла, следует импортировать скриптовый файл с помощью команды module.Import("FileName.js").
Пример:

```
//Name_1.js - первый js файл
this.Text = "Hello, world!"; //Указатель на текстовую переменную
//Name_2.js - второй js файл
var Text = module.Import("Name_1.js").Text; //Импортируем из первого js файла объект с указателем на текстовую переменную
IR.Log(Text); //"Hello, world!" //выводим в лог переменную Text
```

## Создание подустройств

Основой любого модуля служат подустройства и задачей разработчика модуля является определить сколько подустройств будет в модуле и разработать логику работы подустройств. В состав каждого подустройства может входить виджет, пульт управления, собственный набор действий, событий и состояний. Все составляющие подустройства необходимо указать при его создании. Содержимое подустройства не может меняться в ходе работы модуля. Для создания подустройства используется команда:

```
module.AddSubDevice(SystemName, [Device], [System], [Type], [Tags], [Name], [Callback]);
```

,где
**SystemName** - системное имя подустройства
**Device** - драйвер, к которому будет привязано подустройство
**System** - устаревший параметр, следует использовать false
**Type** - тип smart устройства. Параметр в разработке
**Tags** - Массив объектов виртуальных тегов
**Name** - имя подустройства, которое отображается пользователю
**Callback function(){do somethings}** - функция, которая выполняется до наступления события IR.EVENT_ADD_SUBDEVICE
В ответ функция вернет ссылку на подустройство, по которой к нему можно обратиться.
Пример создания подустройств для HDL Relay устройств. Скрипт проверяет количество устройств в шине и в цикле создает подустройства

```
for (var count = 1; count <= COUNT_CHANNEL; count++) {
      // Создаем подустройство
      l_oSubDevice = module.AddSubDevice("Relay " + count, HDL_SERVER, false,
IR.SUB_DEVICE_TYPE_THROUGH_RELAY);
      // Создаем команду StatusOnStart
      if (count == 1)
         l_oSubDevice.AddChannel("Relay:statusOnStart", [ChannelData]);
      var l_sChannelName = "Relay:channel" + count;
      // Добавляем каналы
      l_oSubDevice.AddChannel(l_sChannelName, [ChannelData]);
      l_oSubDevice.AddTag(l_sChannelName, [ChannelData]);
      // Добавляем виртуальный тег
      l_oSubDevice.AddVirtualTag("Power", , false, IR.SUB_DEVICE_TAG_POWER,
true);
   };
```

Таким способом можно добавить в подустройство все необходимые сущности. Также, при создании подустройства, активируется событие

```
module.AddListener(IR.EVENT_ADD_SUBDEVICE, , function, [pointer]);
```

В функцию передается ссылка на только что созданное подустройство. В данном событии можно добавлять к подустройству различные компоненты(действия, события, состояния, виджеты, каналы, теги). Например:

```
module.AddListener(IR.EVENT_ADD_SUBDEVICE, , function(in_oSubDevice){
      Создаем действия
          in_oSubDevice.AddAction("On", false, in_oSubDevice.SystemName +
"_Power", lightID + "_1", IR.SUB_DEVICE_COMMAND_POWER_ON);
      in_oSubDevice.AddAction("Off", false, in_oSubDevice.SystemName +
"_Power", lightID + "_0", IR.SUB_DEVICE_COMMAND_POWER_OFF);
      Создаем события
      in_oSubDevice.AddEvent("On", "Drivers.PhilipsHue." +
in_oSubDevice.SystemName + "_Power", false, "==", "1");
      in_oSubDevice.AddEvent("Off", "Drivers.PhilipsHue." +
in_oSubDevice.SystemName + "_Power", false, "==", "0");
      Создаем состояния
      in_oSubDevice.AddCondition("On", "Drivers.PhilipsHue." +
in_oSubDevice.SystemName + "_Power", false, "==", "1");
      in_oSubDevice.AddCondition("Off", "Drivers.PhilipsHue." +
in_oSubDevice.SystemName + "_Power", false, "==", "0");
   });
```

Таким образом разработчик может создавать нужное количество подустройств при установке модуля или ручном добавлении новых подустройств

# Добавление виджетов к подустройству

Полсе создания интерфейсов в редакторе и добавления функции добавления подустройств, вам потребуется создавать для каждого поустройства свой экземпляр виджета и привязывать его к подустройству. Для привязки виджета к подустройству используется метод

```
SubDevice.addWidget(in_Widget)
```

Входным параметром метода является: **in_Widget** - объект - виджет, заранее созданный в редакторе.
Выходным параметром метода является: успешно или нет (**True**, **False**).
К примеру у нас 3 лампочки и для того, чтобы не создавать для каждой лампочки отдельный виджет, мы воспользуемся уже готовым примером нашего окна и просто будем клонировать его. Напишем пример создания виджета с помощью кода по уже существующему образцу.

Метод Clone позволяет клонировать уже существующее всплывающее окно.

```
Module.ClonePopup (in_Popup, in_Name)
```

Входные параметры метода:

- **in_Name** - имя нового виджета.
- **in_Popup** - ссылка на всплывающее окно.

Выходным параметром метода является: успешно или нет (**True**, **False**).

```
module.AddListener(IR.EVENT_ADD_SUBDEVICE, , function(in_oSubDevice){
 var popup = module.GetPopup("Dimmer"); // Обращаемся к виджету который
собираемся скопировать
 // Создаем виджет методом клонирования
 var widget = in_oSubDevice.addWidget(module.ClonePopup(popup, "Dimmer"+
in_oSubDevice.Name));
 });
```

---

Имена новых виджетов не должны повторяться.

---

При открытии полноценного пульта управления из виджета, необходимо помнить о том что пользователь будет запускать модуль как на планшета, так и на телефоне. Из-за разницы в величине экранов, разработчик модуля должен разработать 2 вида окон. Вид под телефонную версию модуля и вид под планшетную версию. При этом надо добавить в скрипт метод, который будет спрашивать у системы вид устройства, на котором запущен модуль, и открывать соответствующее окно

```
 if (IR.DisplayType == IR.DISPLAY_TYPE_PHONE) { //Если модуль открыт на
телефоне
    var l_oPopupScanner = module.GetPopup("Phone"); //Показываем попап,
отрисованный для телефона
  }
 else {
   var l_oPopupScanner = module.GetPopup("Tablet");//Если планшет, то
открываем попап для планшета
 }
```

## Работа с каналами и тегами

После создания подустройства и добавления виджета, вам понадобится добавить к подустройству нужные для работы каналы и теги. Как и в версии i3 Pro, драйвер можно добавить в графическом интерфейсе редактора или в скрипте. Для каждого подустройства потребуется собственный набор каналов и тегов для выбранного драйвера. В ходе работы может потребоваться работать с тремя компонентами:

- Канал. Для создания канала следует использовать метод

```
SubDevice.AddChannel(Name, DataArray)
```

где,

**Name** - имя канала
**DataArray** - массив данных, специфичный для каждого драйвера

- Тег. Для создания тега следует использовать метод

```
SubDevice.AddTag(Name, DataArray)
```

где,

**Name** - имя тега
**DataArray** - массив данных, специфичный для каждого драйвера

- Виртуальный тег. Для создания виртуального следует использовать метод

```
SubDevice.AddVirtualTag(Name, Value, [Edit], [smartID], [Hidden])
```

где,

**Name**- имя виртуального тега подустройства
**Value** - Значение тега
**Edit** - Разрешение редактирования тега
**smartID** - Параметр в разработке! Необходимо устанавливать false
**Hidden** - Признак скрытости тега

### Пример создания каналов и тегов при создании подустройства:

```
l_oSubDevice = module.AddSubDevice("Light " + count, HDL_SERVER, false,
IR.SUB_DEVICE_TYPE_THROUGH_DIMMER);

// Adding channel StatusOnStart
if (count == 1)
 l_oSubDevice.AddChannel("Dimmer:statusOnStart", [ChannelData]);

// Assigned to the variable name of the channel
var l_sChannelName = "Dimmer:channel" + count;

// Adding channels
l_oSubDevice.AddChannel(l_sChannelName, [ChannelData]);
l_oSubDevice.AddTag(l_sChannelName, [TagData]);

// Adding smart virtual tag
l_oSubDevice.AddVirtualTag("Power", , false, IR.SUB_DEVICE_COMMAND_POWER,
true);
l_oSubDevice.AddVirtualTag("Power On", , false,
IR.SUB_DEVICE_COMMAND_POWER_ON, true);
l_oSubDevice.AddVirtualTag("Power Off", , false,
IR.SUB_DEVICE_COMMAND_POWER_OFF, true);
l_oSubDevice.AddVirtualTag("Level", , false, IR.SUB_DEVICE_COMMAND_LEVEL,
true);
```

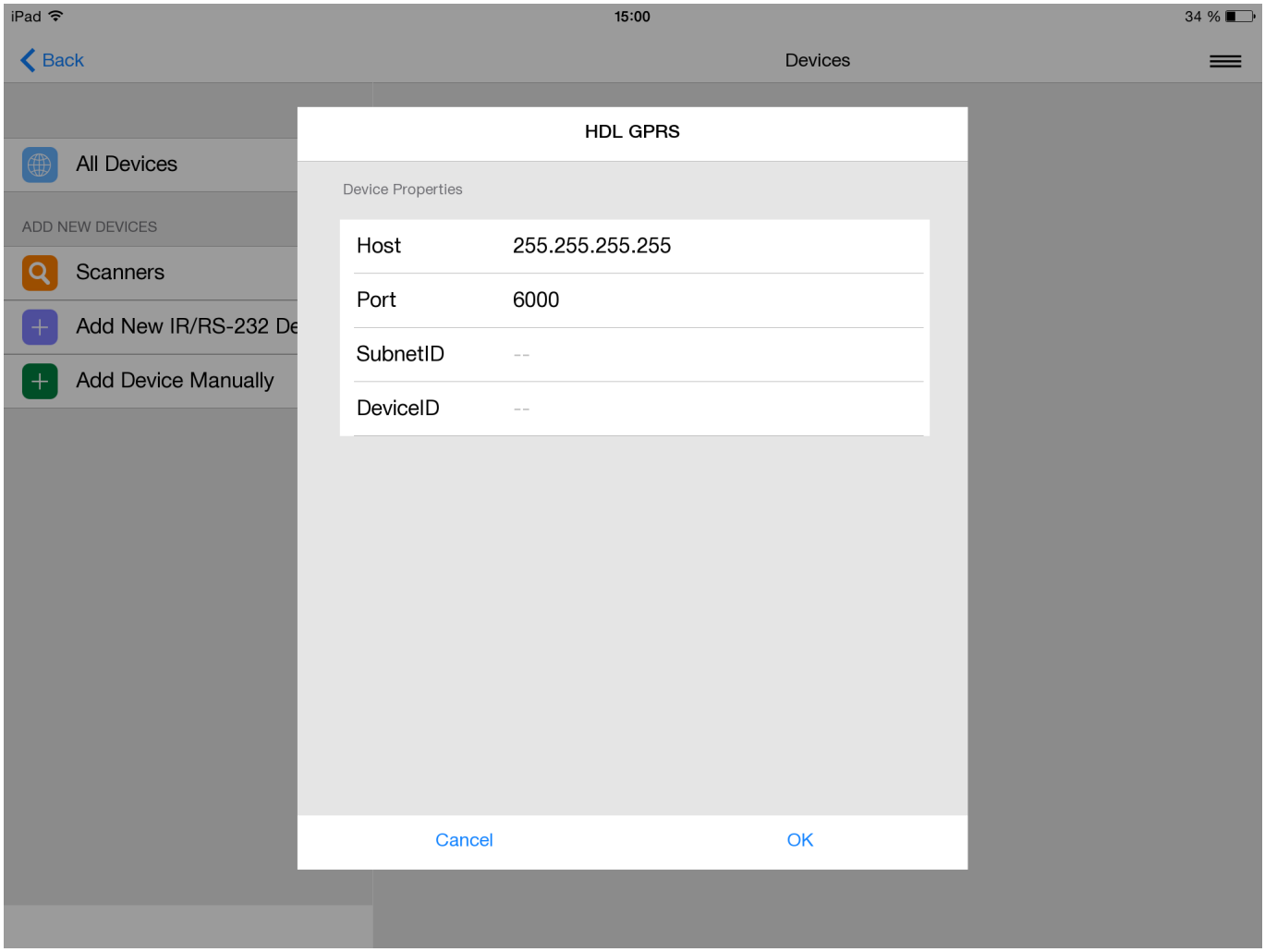
## Установка модуля в приложение

Модули хранятся в магазине модулей iRidium Store и могут устанавливаться в приложение двумя способами:

- С помощью сканера
- Вручную

### Разработка Setup

Для любого варианта необходимо разрабатывать файл Setup. Это специальный скриптовой файл, в котором прописаны поля, необходимые для работы модуля. Такими полями могут выступать ip адрес устройства, порт, название города итд. Для разработки Setup необходимо создать новый скриптовый файл и задать его тип "Setup". Каждое поле является тегом и к нему можно получить доступ из скрипта и получить введенное значение




Файл Setup состоит из 2-ух частей. Первая часть это настройки драйверов. В этих настройках следует запрашивать ip адреса, порты и логины, для подключения драйвера у устройству. Вторая часть файла это общие настройки модуля, такие как количество выходов, ключи доступа. Для установки поля в Setup необходимо настроить следующие поля в скрипте:

- Тип поля - надо указать какое поле надо показать пользователю(текстовое поле, массив значений итд. Подробнее описано в API)
- Имя поля - название поля, которое увидит пользователь при настройке модуля
- Значение по умолчанию - какое значение следует подставлять по умолчанию
- Функция проверки - функция валидации значения в поле. Здесь требуется написать функцию, которая будет проверять корректность введенных данных.

Например, скрипт "Setup" для модуля HDL выглядит следующим образом:

```
{
        // Data to connect drivers, created in Project Device Panel
        Drivers:
        [
      {
         Name: "HDL-BUS Pro Network (UDP)",
         Properties:
         [
            {
               Type: "textfield",
```

```
            Name: "Host",
            DefaultValue: "255.255.255.255",
            Validation: function(in_sValue) {
               var l_aValueHost = in_sValue.split(".");
               if (l_aValueHost.length != 4) {
                  return "Please input correct Host";
               } else
                  if (parseInt(l_aValueHost[], 10)>=
                     && parseInt(l_aValueHost[], 10)<=255
                     && parseInt(l_aValueHost[1], 10)>=
                     && parseInt(l_aValueHost[1], 10)<=255
                     && parseInt(l_aValueHost[2], 10)>=
                     && parseInt(l_aValueHost[2], 10)<=255
                     && parseInt(l_aValueHost[3], 10)>=
                     && parseInt(l_aValueHost[3], 10)<=255
                  )
                     return
                  else
                     return "Please input correct Host";
            }
         },
         {
            Type: "textfield",
            Name: "Port",
            DefaultValue: "6000",
            Validation: function(in_sValue) {
              if(parseInt(in_sValue, 10)>=)
                  return
               else
                  return "Please input correct Port";
            }
         }
      ]
             }
      ],

      // General information for the module (driver and generated by
script)
      Module: [{
              Name: "Channels Count",
              Validation: function (in_sValue) {
                      if (parseInt(in_sValue, 10)>=)
                              return ;
                      else
                              return "Please input Count";
      }
   },
              {
       Type: "textfield",
                      Name : "SubnetID",
                      Validation : function (in_sValue) {
```

```
                                if (parseInt(in_sValue, 10)>=)
                                        return ;
                                else
                                        return "Please input SubnetID";
                        }
                },
        {
           Type: "textfield",
                        Name : "DeviceID",
                        Validation : function (in_sValue) {
                                if (parseInt(in_sValue, 10)>=)
                                        return ;
                                else
                                        return "Please input DeviceID";
                        }
                }

          ]
}
```

Для получения данных из полей Setup следует использовать функцию

```
module.GetProperty("Имя поля")
```

Помимо стандартных параметров, можно запрашивать любые, необходимые для инициализации модуля, параметры. Параметры драйвера подставляются в драйвер автоматически. Остальные параметры следует запрашивать и использовать вручную
Пример работы с параметрами Setup

```
var COUNT_CHANNEL = parseInt(module.GetProperty("Channel count")); // Getting
count channel
var SubNetID = parseInt(module.GetProperty("SubnetID")); // Getting subnet id
var DeviceID = parseInt(module.GetProperty("DeviceID")); // Getting device id
```

## Установка модуля вручную

Для ручной установки модуля не требуется дополнительных скриптовых обработок. Пользователь должен зайти в магазин модулей из приложения, найти модуль, скачать его и заполнить поля, указанные в Setup. После этого приложение установит модуль в систему с указанными пользователем параметрами

## Установка модуля с помощью сканера

**Scanner (сканер)** - это модуль, который анализирует какую-либо шину на подключённые устройства либо ищет устройства в локальной сети. Найденные устройства формируются в список, для выбранного устройства качается модуль и становится доступен в i3 lite.
Схема работы сканера:

1. Сканер загружается из магазина;
2. Пользователь вводит параметры сканера (IP, Port и т.д) и эти параметры передаются в скрипт сканера;
3. Сканер, обработав введенные параметры, опрашивает шину на устройства или ищет устройства в локальной сети;
4. Из полученных устройств формируется список;
5. При выборе определенного устройства в списке, сканер передаёт параметры выбранного устройства в модуль.

При разработке сканера нужно учесть:

- Параметры сканера, при добавлении из магазина;
- Логику сканера;
- Передачу параметров выбранного устройства в модуль;
- Логику модуля, с учётом переданных от сканера параметров.

Процесс разработки сканера ничем не отличается от разработки модуля. Точно также сканеру необходимо нарисовать визуальную часть, создать драйвер и написать скрипт с логикой. Основным отличием является то, что сканер может работать только на панели управления и сканер никогда не будет запущен на сервере, поэтому, при разработке скриптов, нет необходимости разделять скрипты на драйверные и интерфейсные.
Также при разработке сканера появляется новая логика работы. Первым делом необходимо сделать модуль для управления устройством, далее надо загрузить модуль в магазин модулей. При загрузке модуля, вы увидите уникальный идентификатор вашего модуля. Когда вы будете разрабатывать сканер, вам необходимо написать скрипт со следующей логикой:

1. Создание драйвера
2. Поиск оборудования
3. Если оборудование найдено, надо определить что это за оборудование
4. При помощи команды ModuleSetupFinish (ID модуля, js объект с настройками setup данного модуля)

После этого приложение скачает модуль из магазина модулей и установит его.
Отличительной особенностью в разработке сканера является событие по которому драйвер начинает работать. При разработке сканера следует использовать событие

```
module.AddListener(IR.EVENT_OPEN_SCANNER, , function (){});
```

Сканер следует разрабатывать после окончания разработки модулей. Разработанные модули регистрируются в магазине модулей и им автоматически присваивается уникальный id. После того как сканер найдет устройства в локальной сети, скрипт должен скачать модуль для найденного оборудования по указанному id. Для скачивания модуля следует использовать функцию

```
IR.ModuleSetupFinish(StoreID, ModuleData, [Callback])
```

где,

- StoreID - id модуля в магазине модулей
- ModuleData - данные, необходимые для установки модуля(поля в Setup)
- Callback - необязательныя функция для обработки ошибок, происходящих в процессе установки

Пример разработанного сканера для модулей HDL. Для установки любого модуля требуются параметры: Тип, SubNet id, Device id, название модуля, IP адрес шины и порт

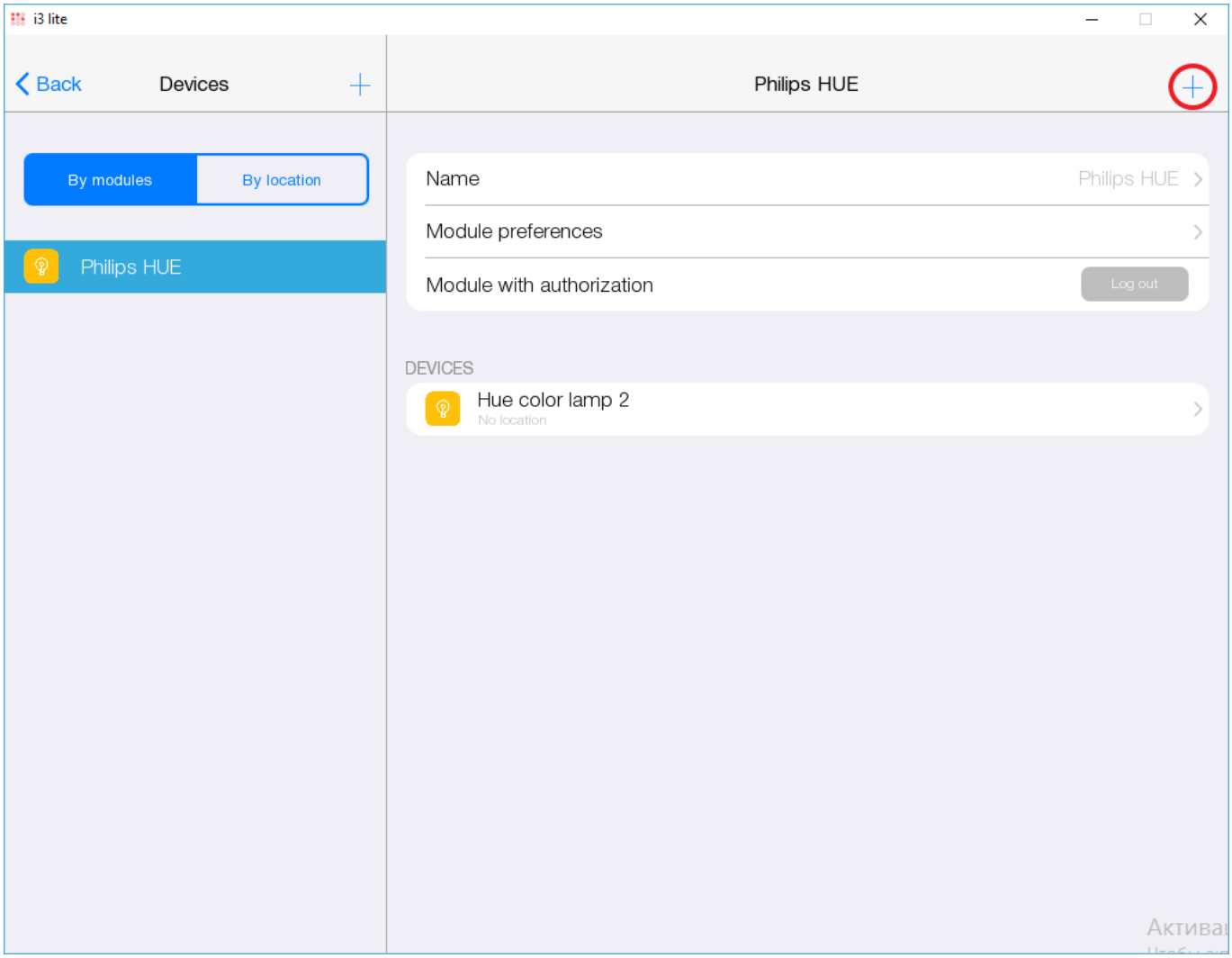
```
//Пример установки HDL Dimmer со сканнера
var l_nStoreID = 82;
var l_oModuleData = {
        Module: {
                //Данные необходимые для работы модуля
                Type: "Dimmer",
                SubnetID: 3,
                DeviceID: 6
                Name: "HDL Dimmer"
        }
        Drivers: {
                //Данные для драйвера
                "HDL-BUS Pro Network (UDP)": {
                   Host: "255.255.255.255", //установка хоста для драйвера
                   Port: "6000" //установка порта для драйвера
                },
        }
}


IR.ModuleSetupFinish(l_nStoreID, l_oModuleData, function(in_error){
        if (!in_error){
                ...
        } else {
                ...
        }
});
```

### Ручное добавление подустройств

После добавления подустройств с помощью сканера или Setup, у пользователя может возникнуть потребность добавить дополнительные подустройства. Для обеспечения такой возможности, разработчик модуля должен привязать использовать системную переменную

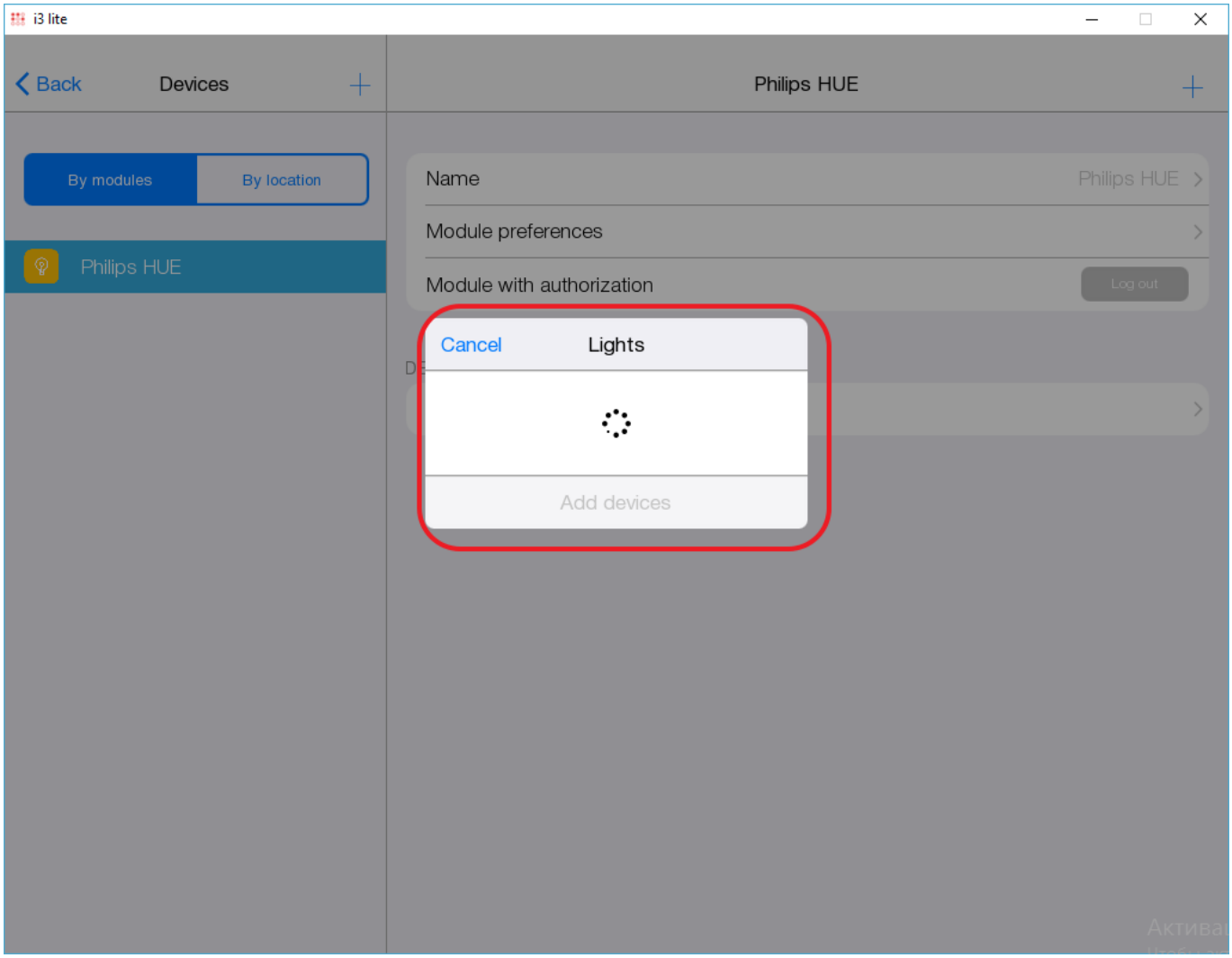
```
SettingsPopupName = module.GetPopup('AddSubDevicePopup');
```

К этой системной переменной надо привязать окно, с результатами работы сканера или попап, в котором пользователь может указать настройки нового подустройства и добавить это подустройство. Такой попап разработчик модуля должен разработать самостоятельно.
Если к указанной системной переменной привязан попап, то в разделе настроек модуля пользователь увидит кнопку "+", которая открывает указанный попап




Пример указания окна добавления подустройств. При разработке, надо создавать окно добавления в разрешении планшета и отдельно окно в разрешении смартфона

```
   module.SettingsPopupName = IR.DisplayType != IR.DISPLAY_TYPE_TABLET?
module.GetPopup("phone:NoAuthorize").Name:
module.GetPopup("NoAuthorize").Name;
```

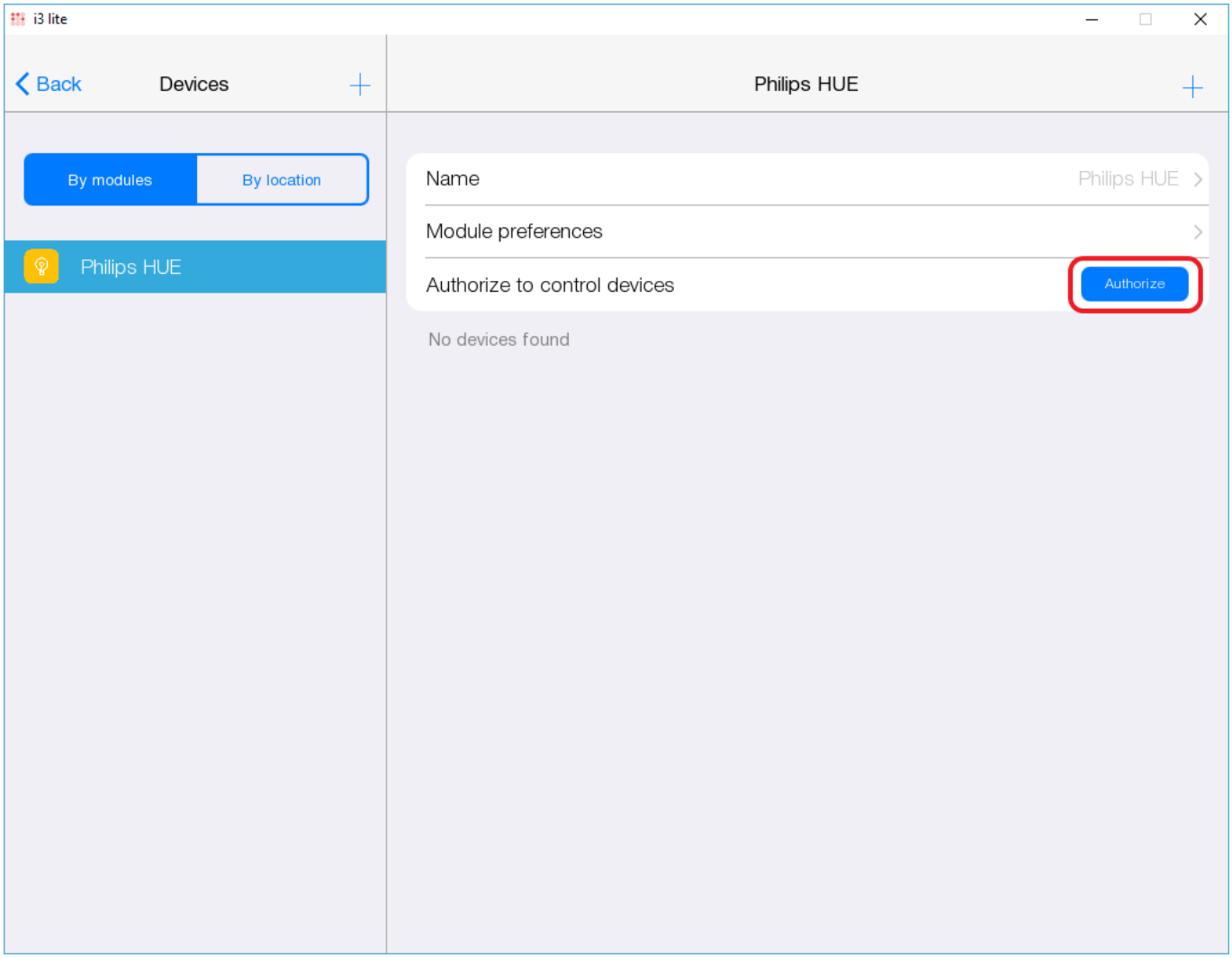



## Авторизация

Для работы с некоторыми устройствами необходима авторизация пользователя в сторонней системе. Для этих целей в приложении существует специальный механизм авторизации. Для прохождения авторизации пользователю необходимо ввести свои данные доступа в спецаильное окно. Это окно уникально для каждого устройства и разработчик модуля должен это окно создать. Окно ввода данных это попап с полями ввода. Данное окно создается стандартными средствами редактора и может быть не привязанно ни к одному подустройству При вводе этих данных, скрипт должен атворизоваться в сторонней системе и закрыть окно авторизации. После создания попапа авторизации, разработчик должен в скрипте вызвать системную переменную авторизации и привязать в ней созданный попап

```
AuthorizationPopupName = module.GetPopup("AutorisationPopup");
```

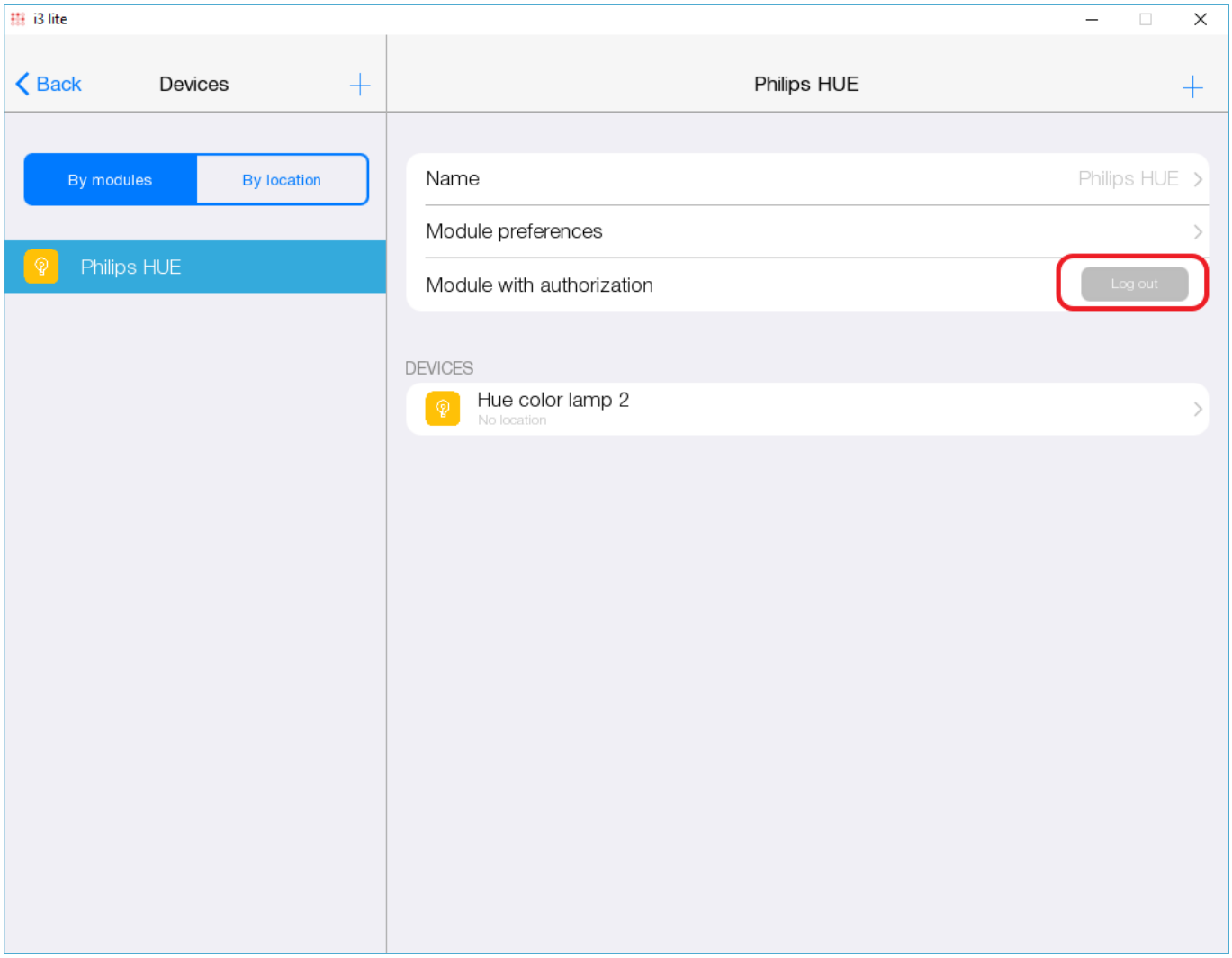
После привязки попапа авторизации к переменной, в настройках модуля появится кнопка "Авторизоваться"




При нажатии на данную кнопку, приложение откроет попап авторизации, привязанный в скрипте. Второй задачей разработчика модулей является отображение статуса пользователя(Авторизован или нет). Для решения данной задачи есть системная переменная

```
AuthorizationStatus = true;
```

Если пользователь авторизовался в системе, то следует присвоить значение true этой переменной и приложение изменить статус кнопки авторизации на:




Также разработчику модуля следует продумать механизм деавторизации пользователя. Это делается на попапе авторизации
Пример скрипта авторизации

```
if (module.AuthorizationStatus  == false) { //Если пользователь не
авторизован
  module.AuthorizationPopupName = WindowSettings.Window.Name;//Привязываем
созданный ранее попап авторизации
} else if (module.AuthorizationStatus) //Если пользователь авторизован
  {
        module.AuthorizationPopupName = IR.DisplayType !=
IR.DISPLAY_TYPE_TABLET? module.GetPopup("phone:deAuth").Name:
module.GetPopup("deAuth").Name;//Показываем попап деавторизации
  }
```

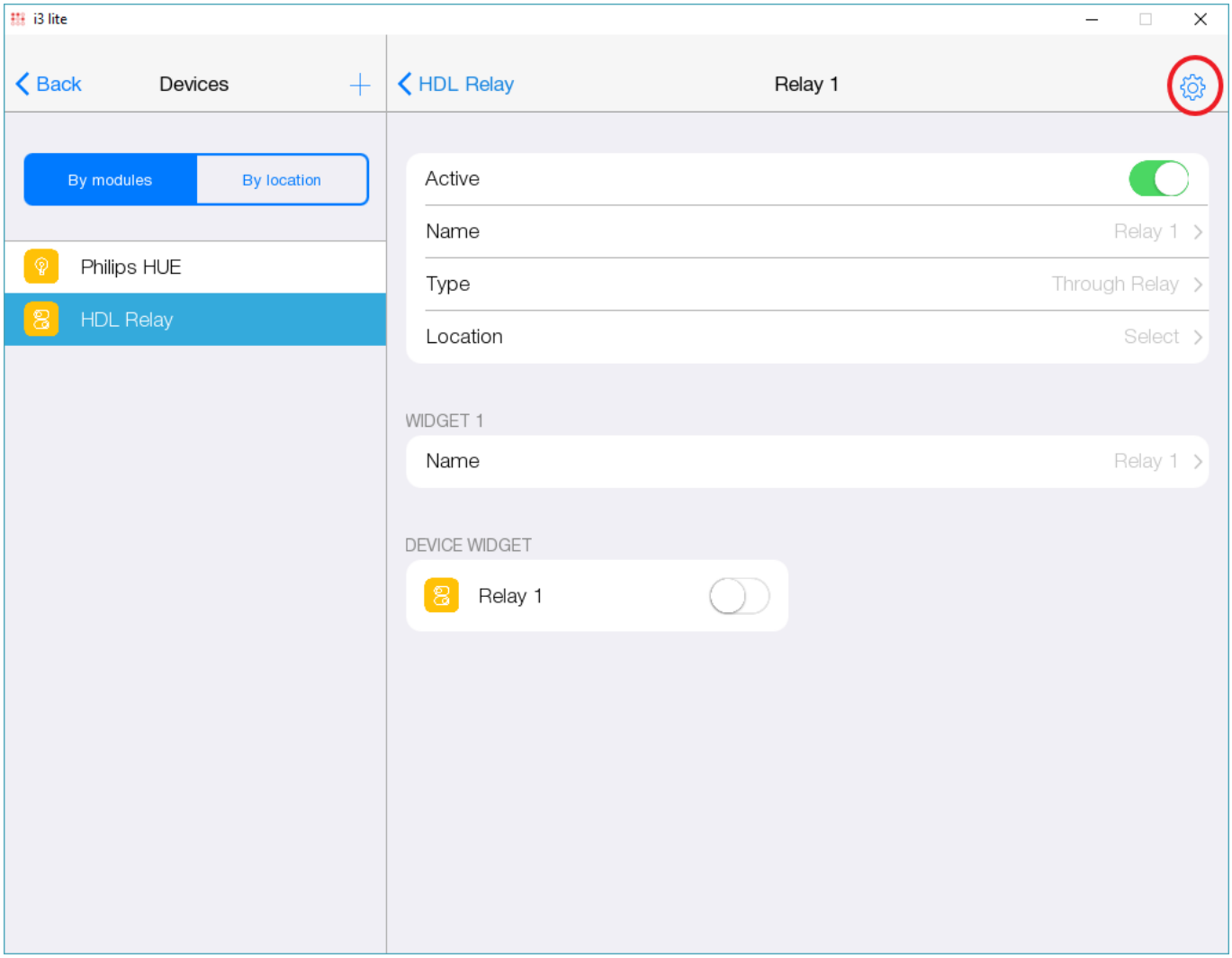
Примечание. Не забывайте создавать попап авторизации и деавторизации для планшета и отдельно попап авторизации для смартфона.

## Смена настроек

В ходе работы с приложением, может возникнуть потребность изменить настройки модуля. Для обеспечения такой возможностью, разработчик модуля должен создать функцию изменения настроек модуля и присвоить ее к системному методу

```
module.ChangeProperties = function (in_oSetup)
{
        обработка новых параметров модуля
}
```

При объявлении такой функции, приложение добавит в модуль кнопку редактирования настроек модуля.




Когда пользователь нажмет на данную кнопку, приложение откроет окно Setup, которое использовалось при установке модуля. В этом окне пользователь введет новые настройки и нажмет кнопку "готово". После этого приложение запустить указанную выше функцию. Настройки драйвера приложение сможет изменить самостоятельно, остальные же настройки разработчик модуля дулжен изменить вручную. Новые настройки записываются во входной параметр in_oSetup в виде JSON строки.
Также в данной функции необходимо обработать успешность изменения настроек. За это отвечает флаг

```
  module.FinishUpdateSettings = true;
```

Если установка новых параметров прошла успешно, то надо выставить значение флага true, а обратном случае флаг должен принять значение false. Это позволяет приложению понять что смена параметров не удалась(если статус переменной будет false) и приложение отменит изменение настроек драйвера, вернув старые настройки
Пример обработки новых настроек модуля

```
module.AddListener(IR.EVENT_MODULE_START, , function(){
   module.ChangeProperties = function (in_oSetup, in_oModule) {
      if (typeof (in_oSetup) == "string")
         in_oSetup = JSON.Parse(in_oSetup);
            var l_nChannel = in_oSetup.Module["Channel count"];
      if (l_nChannel >= COUNT_CHANNEL) {
         module.FinishUpdateSettings = true;
         // Recreating element on cycle
         for (var count = 1; count <= in_nChannel; count++) {

            // Creating subDevice
            l_oSubDevice = module.AddSubDevice("Relay " + count, HDL_SERVER,
false, IR.SUB_DEVICE_TYPE_THROUGH_RELAY);

            // Create revers
            if (!l_oSubDevice.GetProperty("Reverse"))
               l_oSubDevice.SetProperty("Reverse", );

            // Assigned to the variable name of the channel
            var l_sChannelName = "Relay:channel" + count;

            // Adding channels
            l_oSubDevice.AddChannel(l_sChannelName, Data);
            l_oSubDevice.AddTag(l_sChannelName, Data);

            // Adding smart virtual tag
            l_oSubDevice.AddVirtualTag("Power", , false,
IR.SUB_DEVICE_TAG_POWER, true);
         };
      }
      else {
         for (var count = 1; count <= COUNT_CHANNEL; count++)
            if (l_nChannel < count) {
               var l_oDeleteSubDevice = module.GetSubDevice("Relay " +
count);
               module.DeleteSubDevice (l_oDeleteSubDevice.ID);
            };
      };
   };
});
```

# Типирование модуля с помощью Smart API

Как известно модули дают возможность приложению i3 lite управлять определенным оборудованием, будь то лампочка Philips HUE, HDL переключатель или какое-то другое устройство. Без SmartAPI модули могли управлять только устройствами для которых их разрабатывали. SmartAPI же позволяет разрабатывать модули, которые работают с конкретными типами устройств. При использовании этого API можно, например, сделать модуль который включает / выключает весь свет в доме.

Типирование модуля