Эта страница является переводом страницы TestLite. Перевод выполнен на 100%.

Содержание

SmartAPI support

SmartAPI - API for managing all devices of a certain type (type - light, curtains, etc.) in the project. SmartAPI is used in modules with the help of a script.

What is it for?

As you know, the modules allow the i3 lite application to control certain equipment, be it a Philips HUE lamp, an HDL switch or some other device. Without SmartAPI, the modules could only manage the devices for which they were designed. SmartAPI also allows you to develop modules that work with specific types of devices. Using this API, you can, for example, make a module that turns on / off all the lights in the house.

Рис. Управление с помощью SmartAPI

The principle of SmartAPI operation is as follows: SmartAPI searches for the devices of the required type in the project and causes corresponding action by all these devices. In the image above, SmartAPI will cause the "Enable" action for all devices of the "Light" type in the project. Thus, if all the light bulbs are added to the project, then the light will turn on throughout the house.

SmartAPI for the module developer can be divided into 2 parts: typing the module (the module we manage) and using SmartAPI in the module (the module that manages).

Module typification

In the module, you need to type subdeals, actions and virtual tags. Subdeivays need to be typed so that SmartAPI understands what kind of device (for example light or air conditioning) and could manage it. Modules in which devices are not specified for the type will not be managed using SmartAPI. Example of type indication in the module:

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

Now the application knows that the created subwave is a device for controlling the light (constants for all existing types are in the second tab tables types) .

Each type of device has its own set of actions and virtual tags. For example, for light there are the following set of actions: turn on, off, switch, set brightness, set heat, set color. Also for light there are the following virtual tags: power (on / off), brightness, heat and color. Thus, you also need to specify a type for actions and virtual tags.

It is not necessary to use all the listed actions and virtual tags, the device has a set of actions and virtual tags based on the functionality (if the device for controlling the light does not control the brightness and heat, the actions for specifying heat and color are simply absent, and there is no sense in the corresponding tags).

An example of typing actions:

```
in_oSubDevice.AddAction("On", true, turnOn, {SubDevice: in_oSubDevice,
Channel: in_sChannelName}, null, IR.SUB_DEVICE_COMMAND_POWER_ON, null,false);
in_oSubDevice.AddAction("Off", true, turnOff, {SubDevice: in_oSubDevice,
Channel: in_sChannelName}, null, IR.SUB_DEVICE_COMMAND_POWER_OFF, null,
false);
in_oSubDevice.AddAction("Brightness", true, setBrightness, {SubDevice:
in_oSubDevice, Channel: in_sChannelName}, null, IR.SUB_DEVICE_COMMAND_LEVEL,
[{
   Name: "Value",
   Type: IR.ADVANCED_NUMBER,
   Min: ,
   Max: 100
}], false);
in_oSubDevice.AddAction("Power toggle", true, powerToggle, {SubDevice:
in_oSubDevice, Channel: in_sChannelName}, null,
IR.SUB_DEVICE_COMMAND_POWER_TOGGLE, null, true);
in_oSubDevice.AddAction("Power", true, powerOnOff, {SubDevice: in_oSubDevice,
Channel: in_sChannelName}, null, IR.SUB_DEVICE_COMMAND_POWER, null, true);
in_oSubDevice.AddAction("Brightness up", true, brughtnessUp, {SubDevice:
in_oSubDevice, Channel: in_sChannelName}, null,
IR.SUB_DEVICE_COMMAND_LEVEL_UP, null, true);
in_oSubDevice.AddAction("Brightness down", true, brughtnessDown, {SubDevice:
in_oSubDevice, Channel: in_sChannelName}, null,
IR.SUB_DEVICE_COMMAND_LEVEL_DOWN, null, true);
```

Example of virtual tagging:

```
l_oSubDevice.AddVirtualTag("Power", , false, IR.SUB_DEVICE_TAG_POWER, true);
l_oSubDevice.AddVirtualTag("Level", , false, IR.SUB_DEVICE_TAG_LEVEL, true);
```

When you set the type for the device, its actions and virtual tags - it means your module is fully

ready for management using SmartAPI.

Using SmartAPI in the module

Device management using SmartAPI is also implemented in the module. Examples of modules based on SmartAPI: a module for managing all the lights in the project, a module for managing all the curtains in the project, a module for managing the entire project using a third-party service (for example Amazon echo), and so on.

In order for the module to be able to work with SmartAPI, you need to check the box when you load the module into the iRidium store and edit its parameters:

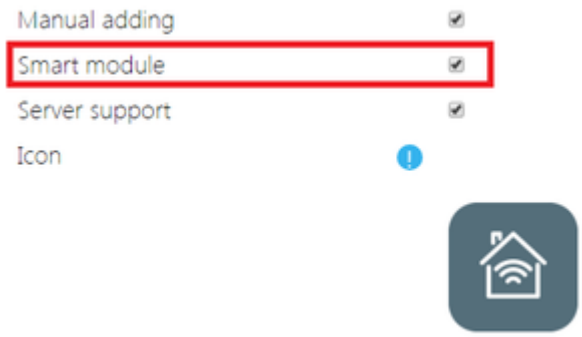

Рис. Модуль для управления через SmartAPI

Implementation example:

```
function turnOn () {
        var l_aDevices = module.GetSubDevices();
        for (var i = ; i < l_aDevices.length; i++)
                if (l_aDevices[i].SmartID == IR.SUB_DEVICE_TYPE_LIGHT)
                        module.Set(l_aDevices[i],
IR.SUB_DEVICE_COMMAND_POWER_ON);
};

function getVirtualTags () {
        var l_aVirtualTags = [];
        var l_aDevices = module.GetSubDevices();
        for (var i = ; i < l_aDevices.length; i++)
                if (l_aDevices[i].SmartID == IR.SUB_DEVICE_TYPE_LIGHT)
l_aVirtualTags.push(l_aDevices[i].GetVirtualTagBySmartID(IR.SUB_DEVICE_TAG_LE
VEL).Value);
};
```

The turnOn function in the example will include all devices of type Light. The getVirtualTags function fills the array with brightness values from all devices of type Light. All methods for using SmartAPI are described in i3 lite API.

Example of a ready-made module with SmartAPI support (HDL Dimmer): download link

SmartAPI

Methods and properties

This section lists the methods and properties available for SmartAPI.

module.Set

Send action to a sub-device

Синтаксис

```
module.Set(Subdevice, Command, Value)
```

Название	Пример	Описание
`Subdevice`	My_subdevice	type: object Subdevice object
`Command`	POWER in the example method	type: const Command Constant
`Value`	1	type: Integer Command value (Not necessary for all commands)

На выходе

-

Пример

```
subdevice = module.GetSubDevice ("MySubDevice");
module.Set(subdevice, IR.SUB_DEVICE_COMMAND_POWER, 1);
```

The entire list of constants can be found in the SmartAPI types section.

SubDevice.SmartID

Property. Get a type of sub-device

Синтаксис

```
SubDevice.SmartID
```

Название	Пример	Описание

-

На выходе

Constant of type of a device

Пример

```
var l_aDevices = module.GetSubDevices();
        for (var i = ; i < l_aDevices.length; i++) // The cycle will turn on
all devices of the light type in the project
                if (l_aDevices[i].SmartID == IR.SUB_DEVICE_TYPE_LIGHT)
                        module.Set(l_aDevices[i],
IR.SUB_DEVICE_COMMAND_POWER_ON);
```

The entire list of constants can be found in the SmartAPI types section.

SubDevice.GetVirtualTagBySmartID

Get the value of the virtual sub-tag tag

Синтаксис

```
SubDevice.GetVirtualTagBySmartID(Tag const)
```

Название	Пример	Описание
`Tag const`	IR.SUB_DEVICE_TAG_LEVEL	type: const Tag Type

На выходе

The value of the specified type tag

Пример

```
        var l_aVirtualTags = [];
        var l_aDevices = module.GetSubDevices();
        for (var i = ; i < l_aDevices.length; i++) // The cycle will record
the values of the brightness tags of all devices of the light type in the
project in an array
                if (l_aDevices[i].SmartID == IR.SUB_DEVICE_TYPE_LIGHT)
l_aVirtualTags.push(l_aDevices[i].GetVirtualTagBySmartID(IR.SUB_DEVICE_TAG_LE
VEL).Value);
```

The entire list of constants can be found in the SmartAPI types section.

Types

The supported SmartAPI types are listed in this table: link