How to create your own sensor elements (EEL files)

Digi-Key IoT Studio enables you to create and import your own elements to add support for third-party sensors not currently available in the platform.

Creating sensor elements is done using Digi-Key's Embedded Element Library (EEL) Utility tool. The EEL Utility tool is a command line interface used for generating elements to be imported into DK IoT. All of the information surrounding use of the tool is detailed on the DK IoT GitHub.

Completed EEL files ready to be added into DK IoT Studio for use can be imported through the Import button in the Add Element window. Once a file is imported, it's included in the list of available sensor elements that can be enabled in an DK IoT Studio project. Note that custom EEL files imported into DK IoT Studio are only available for the project they are added to.

This intent of this guide is to describe the step by step process of creating an EEL from scratch. This EEL will control the MikroElektronika Relay Click board.

Step-by-step guide

Step 1: Prerequisites

- Node.js 8.x.x (An older version _may_ work)
- npm 5.x.x (An older version _may_ work)

Step 2: Clone This Repository

Using your tool of choice, clone this repository to your local machine:

```
git clone git@github.com:Digi-Key/DK-IoT-Studio/eel-builder-master.git
```

Navigate to the newly created eel-builder directory and run

```
sudo npm install
```

to install dependencies that the eel-builder needs.

Step 3: Generate New EEL Directory

Let's create a new directory to hold all of our EEL source and metadata. To do this, navigate to the eel-builder directory and run

```
node eelbuilder.js --new --name="relayclick" --dir="../"
```

You should see that a new directory, `relayclick_EEL` was created one level up in the same directory as the eel-builder directory.

*_NOTE_*: The `--dir` argument can be anything you like. It's the directory in which the new source directory will be created. You could put it in `C:\Users\me\Documents` or `/home/me/Documents`, for example.

Step 4: Editing Metadata

Navigate to your relayclick_EEL directory. Open the file metadata.json. This file contains all of the high level metadata about your EEL that the studio needs to properly display and use it.

Step A: manufacturer

Change the manufacturer name to MikroElektronika.

Step B: description

Write a short description of your EEL here. Something like "Relay Click evaluation board" would suffice. You could leave this blank if you wanted.

Step C: requires

This EEL will require the GPIO driver, so add `gpio` to the list. You should have something like this:

```
"requires": [
  "embedded",
  "gpio"
],
```

This field allows the studio to filter EELs that are not compatible with certain platforms.

Step D: Element name

In the Elements object, set the name to "RelayClick". This is the name that would be displayed in the Element Library browser in the "Element" column.

Step E: Element type

In the Elements object, set the element type to "EmbeddedRelayClick". This type is used for the language object and is generally how the Studio refers to this element.

Step F: Icon

For the icon, just use the generic `EmbeddedFunction.svg`.

Step G: Abilities

Change `myability` to setup and add a field `hidden` set to true. Every element must have a setup ability, and it should be hidden. It should look like this:

```
 "abilities": [
  {
      "name": "setup",
      "hidden": true,
      "triggers": []
  }
  ],
```

Add a new ability called `setRelay1On` with a trigger `relay1SetOn`. Once you've done this, add a third ability called `setRelay1Off` with a trigger `relay1SetOff`. Finally, add an ability `toggleRelay1` with a trigger `relay1Toggled`. Your final abilities list should look like this:

Step H: Default Ability/Trigger

You must set the default ability and trigger selected when the element is initially created. Set `defaultAbility` to `setRelay1On` and `defaultTrigger` to `relay1SetOn`.

Step I: Properties

Delete the default `myproperty` property. Thinking about the relay click board, what does it need in order to function? The EEL needs to know the GPIO Driver Instance and the GPIO Pin connected to relay 1, so those are our two properties.

Your properties list should look like this:

```
"properties": [
   {
       "name": "gpioDriverInstance",
       "type": "driverInstance",
       "driverType": "gpio"
   },
   {
       "name": "relay1GpioPin",
       "type": "number",
       "value": 0
   }
   ],
```

Step J: Language

The last step in the metedata is the language object. This tells the studio how to print the element name, ability names, and trigger names. It should look something like this:

```
"language": {
    "en-US": {
        "EmbeddedRelayClick": "Relay Click",
        "setRelay1On": "Set Relay 1 On",
        "relay1SetOn": "Relay 1 Set On",
        "setRelay1Off": "Set Relay 1 Off",
        "relay1SetOff": "Relay 1 Set Off",
        "toggleRelay1": "Toggle Relay 1",
        "relay1Toggled": "Relay 1 Toggled"
    }
}
```

Your final metadata.json file should look like this:

```
{
  "libName": "relayclick",
  "manufacturer": "MikroElektronika",
  "description": "Dual Relay Evaluation Board",
  "version": "",
  "eelVersion": "3",
  "shoppingCarLinks": {},
  "requires": [
    "embedded",
    "gpio"
  ],
  "elements": [
    {
      "name": "RelayClick",
      "type": "EmbeddedRelayClick",
      "icon": "EmbeddedFunction.svg",
      "defaultAbility": "setRelay1On",
      "defaultTrigger": "relay1SetOn",
      "hidden": false,
      "abilities": [
        {
          "name": "setup",
          "hidden": true,
          "triggers": []
        },
        {
          "name": "setRelay1On",
          "triggers": ["relay1SetOn"]
        },
        {
          "name": "setRelay1Off",
          "triggers": ["relay1SetOff"]
        },
        {
          "name": "toggleRelay1",
          "triggers": ["relay1Toggled"]
        }
      ],
      "properties": [
        {
          "name": "gpioDriverInstance",
          "type": "driverInstance",
          "driverType": "gpio"
        },
        {
          "name": "relay1GpioPin",
          "type": "number",
          "value": 0
        }
      ],
      "triggers": [],
      "variables": [],
      "language": {
        "en-US": {
            "EmbeddedRelayClick": "Relay Click",
            "setRelay1On": "Set Relay 1 On",
            "relay1SetOn": "Relay 1 Set On",
            "setRelay1Off": "Set Relay 1 Off",
            "relay1SetOff": "Relay 1 Set Off",
            "toggleRelay1": "Toggle Relay 1",
            "relay1Toggled": "Relay 1 Toggled"
        }
                      }
    }
  ]
}
```

Step 5: Generating stub abilities

We can use the tool to generate stub .c files that we can fill in later for all of its abilities. To do this, navigate to the eel-builder directory and run

```
node eelbuilder.js --generate --dir="../relayclick_EEL" --dest="../"
```

You should see the following output:

```
Ability dir does not exist, creating it.
Unable to read ability source ../relayclick_EEL/elements/EmbeddedRelayClick/abilities/setup.c
Generating default stub ../relayclick_EEL/elements/EmbeddedRelayClick/abilities/setup.c
Unable to read ability source ../relayclick_EEL/elements/EmbeddedRelayClick/abilities/setRelay1On.c
Generating default stub ../relayclick_EEL/elements/EmbeddedRelayClick/abilities/setRelay1On.c
Unable to read ability source ../relayclick_EEL/elements/EmbeddedRelayClick/abilities/setRelay1Off.c
Generating default stub ../relayclick_EEL/elements/EmbeddedRelayClick/abilities/setRelay1Off.c
Unable to read ability source ../relayclick_EEL/elements/EmbeddedRelayClick/abilities/toggleRelay1.c
Generating default stub ../relayclick_EEL/elements/EmbeddedRelayClick/abilities/toggleRelay1.c
Writing EEL to ..\relayclick.eel
```

Step 6: C Source

This step is actually writing the C driver for the EEL. This will be located in `relayclick_EEL/files/common`.

Step A: Header

Create a new file: `relayclick_EEL/files/common/headers/relayclick.h`

Create a new structure to hold your configuration data (properties). You should have something like this, at this point:

```
#ifndef ATMO_RELAYCLICK_H_
#define ATMO_RELAYCLICK_H_

#include "../atmo/core.h"

typedef struct {
    ATMO_DriverInstanceHandle_t gpioDriverInstance;
    ATMO_GPIO_Device_Pin_t relay1Pin;
} ATMO_RelayClick_Config_t;

#endif
```

Note that these two struct items refer to the properties you defined in your metadata. Also note the `#include "../atmo/core.h"`. This should be done in every EEL, as it will import any datatypes you may need.

The next step is to create the function prototypes that we will need. You should end up with something like this:

```
#ifndef ATMO_RELAYCLICK_H_
#define ATMO_RELAYCLICK_H_

#include "../atmo/core.h"

typedef struct {
    ATMO_DriverInstanceHandle_t gpioDriverInstance;
    ATMO_GPIO_Device_Pin_t relay1Pin;
} ATMO_RelayClick_Config_t;

bool ATMO_RelayClick_Init(ATMO_RelayClick_Config_t *config);
bool ATMO_RelayClick_SetRelay1State(bool isOn);
bool ATMO_Relayclick_ToggleRelay1();

#endif
```

Step B: Source

Create a new file: `relayclick_EEL/files/common/objects/relayclick.c`

The first step is to include the header:

```
#include "relayclick.h"
```

NOTE Although the header and source are currently in different directories, they will be in the same directory when Atmosphere Studio generates the embedded source for your project.

The next step is to implement the initialization function:

```
#include "relayclick.h"
#include "../gpio/gpio.h"

// Private internal storage of driver configuration
static ATMO_RelayClick_Config_t _ATMO_RelayClick_PrivConfig;

bool ATMO_RelayClick_Init(ATMO_RelayClick_Config_t *config)
{
    // Copy the configuration structure and store internally
    memcpy(&_ATMO_RelayClick_PrivConfig, config, sizeof(ATMO_RelayClick_Config_t));

    // Set the default configuration and state of Relay1
    ATMO_GPIO_Config_t gpioConfig;
    gpioConfig.pinMode = ATMO_GPIO_PinMode_Output_PushPull;
    gpioConfig.pinState = ATMO_GPIO_PinState_Low;

    return ATMO_GPIO_SetPinConfiguration(config->gpioDriverInstance, config->relay1Pin, &gpioConfig) ==
ATMO_GPIO_Status_Success;
}
```

The documentation for the GPIO driver can be found here. The include path for any driver will be

```
 "../{drivername}/{drivername}.h"
```

The final step is to implement the two Relay1 control functions:

```
bool ATMO_RelayClick_SetRelay1State(bool isOn)
{
    ATMO_GPIO_PinState_t desiredPinState = isOn ? ATMO_GPIO_PinState_High : ATMO_GPIO_PinState_Low;


    return (ATMO_GPIO_SetPinState(_ATMO_RelayClick_PrivConfig.gpioDriverInstance, _ATMO_RelayClick_PrivConfig.
relay1Pin, desiredPinState) == ATMO_GPIO_Status_Success);
}

bool ATMO_Relayclick_ToggleRelay1()
{
    return (ATMO_GPIO_Toggle(_ATMO_RelayClick_PrivConfig.gpioDriverInstance, _ATMO_RelayClick_PrivConfig.
relay1Pin) == ATMO_GPIO_Status_Success);
}
```

In the end, you should end up with a final file looking like this:

```
#include "relayclick.h"
#include "../gpio/gpio.h"

// Private internal storage of driver configuration
static ATMO_RelayClick_Config_t _ATMO_RelayClick_PrivConfig;

bool ATMO_RelayClick_Init(ATMO_RelayClick_Config_t *config)
{
    // Copy the configuration structure and store internally
    memcpy(&_ATMO_RelayClick_PrivConfig, config, sizeof(ATMO_RelayClick_Config_t));

    // Set the default configuration and state of Relay1
    ATMO_GPIO_Config_t gpioConfig;
    gpioConfig.pinMode = ATMO_GPIO_PinMode_Output_PushPull;
    gpioConfig.pinState = ATMO_GPIO_PinState_Low;

    return ATMO_GPIO_SetPinConfiguration(config->gpioDriverInstance, config->relay1Pin, &gpioConfig) ==
ATMO_GPIO_Status_Success;
}

bool ATMO_RelayClick_SetRelay1State(bool isOn)
{
    ATMO_GPIO_PinState_t desiredPinState = isOn ? ATMO_GPIO_PinState_High : ATMO_GPIO_PinState_Low;

    return (ATMO_GPIO_SetPinState(_ATMO_RelayClick_PrivConfig.gpioDriverInstance, _ATMO_RelayClick_PrivConfig.
relay1Pin, desiredPinState) == ATMO_GPIO_Status_Success);
}

bool ATMO_Relayclick_ToggleRelay1()
{
    return (ATMO_GPIO_Toggle(_ATMO_RelayClick_PrivConfig.gpioDriverInstance, _ATMO_RelayClick_PrivConfig.
relay1Pin) == ATMO_GPIO_Status_Success);
}
```

Step 7: Ability Source

The next step is to write the source for the abilities. This is the code that appears in the code view in Studio. This code will be _using_ the driver we just wrote in Step 6.

Navigate to `relayclick_EEL/elements/EmbeddedRelayClick/abilities/` and open `setup.c`.

Using the ATMO_PROPERTY macro, we can retrieve the properties from the element toolbox and set them in the configuration struct we created:

```
ATMO_RelayClick_Config_t config;
config.gpioDriverInstance = ATMO_PROPERTY(undefined, gpioDriverInstance);
config.relay1Pin = ATMO_PROPERTY(undefined, relay1GpioPin);
return ATMO_RelayClick_Init(&config) ? ATMO_Status_Success : ATMO_Status_Fail;
```

NOTE: The ATMO_PROPERTY macro accepts two arguments: element name and property name. Using undefined as the element name tells the studio to automatically fill in the name of the element instance before the code is compiled. For example, if you create Relay Click element and name it "NicksRelayClick", this code will end up looking as follows before compilation:

```
ATMO_RelayClick_Config_t config;
config.gpioDriverInstance = ATMO_PROPERTY(NicksRelayClick, gpioDriverInstance);
config.relay1Pin = ATMO_PROPERTY(NicksRelayClick, relay1GpioPin);
return ATMO_RelayClick_Init(&config) ? ATMO_Status_Success : ATMO_Status_Fail;
```

The next step is to fill in the other three abilities. Your `setRelay1Off.c` should look like this:

```
return ATMO_RelayClick_SetRelay1State(false) ? ATMO_Status_Success : ATMO_Status_Fail;
```

Your `setRelay1On.c` should look like this:

```
return ATMO_RelayClick_SetRelay1State(true) ? ATMO_Status_Success : ATMO_Status_Fail;
```

Your `toggleRelay1.c` should look like this:

```
return ATMO_RelayClick_ToggleRelay1() ? ATMO_Status_Success : ATMO_Status_Fail;
```

Step 8: Generate Final EEL

Navigate to the eel-builder directory and run:

```
node eelbuilder.js --generate --dir="../relayclick_EEL" --dest="../"
```

Step 9: Celebrate

Congratulations, you've created your first EEL. You can see a more complex version of this EEL in the examples directory, here.

Related articles

- How to create an EEL file using source files from GitHub in Linux.
- How to create an EEL file using source files from GitHub in Windows.
- How to create your own sensor elements (EEL files)
- AVR®-IoT WG Development Board AC164160-ND