Porting Guide

SUSI IoT Module

Secured & Unified Smart Interface Software for Internet of Things Module

ADVANTECH EmbCore

Edition 1.0 Part. No.
March 19 2015 Printed in Taiwan

Contents

1. Introduction

This document will show you how to porting a module library, including what should module library provide, how to create the JSON package format, introducing the id system, API and so on

1.1 Environment Requirements

1.1.1 Operating Systems

Windows XP Embedded

Windows XP Pro or Home Edition 32-bit

Windows 7 (x86 / x64)

WES7 (x86 / x64)

Windows 8 Desktop (x86 / x64)

1.1.2 Integrated Development Environment

Visual Studio 2008 or newer

2. What to do

The main of propose of module is collecting information and encapsulating it to a JSON package, and then pass it to SUSI IoT kernel. For porting a module library, it is implement all module's APIs below. See the Appendix B. for more detailed about API.

2.1 SusiIoTInitialize / SusiIoTUninitialize

These two functions were called from SUSI IoT kernel when module being loaded and unloaded. `SusiIoTInitialize` function is to initialize the module library need, like initializing the real device's library. On the other hand, you should create the capability JSON package here (see the next chapter for capability of JSON package).

`SusiIoTUninitialize` function is to uninitialize the module library need, like uninitializing the real device's library and releasing the memory space of JSON package created by `SusiIoTInitialize`.

2.2 SusiIoTGetMDCapability

In this function, you just use the `json_object_update` function to pass the capability JSON package which had been created by `SusiIoTInitialize` to SUSI IoT kernel. That is why we recommend creating the capability JSON package in `SusiIoTInitialize.` You don't need to create the capability every time when `SusiIoTGetMDCapability` be called from SUSI IoT kernel.

2.3 SusiIoTGetMDValue / SusiIoTSetMDValue

These two functions provide the data operation to SUSI IoT kernel immediately. SUSI IoT kernel uses `SusiIoTGetMDValue` to get the newest data by unique ID (see the Appendix A), and uses `SusiIoTSetMDValue` to set the newest data by unique ID.

`SusiIoTGetMDValue` function is to get the data from real device's library, pack the data to a JSON package, and then passing it to SUSI IoT kernel.

`SusiIoTSetMDValue` function is to unpack the data of JSON package from SUSI IoT kernel to extract the data value, and then setting the data value to real device's library.

2.4 SusiIoTGetModuleID

For all module libraries, you have to specify a unique 2 byte ID for it. The SUSI IoT kernel uses this ID to identify each module, so you just return the ID in this function.

3. How to do

This section we focus how to use the IoT JSON Library to produce the final JSON package we wanted. See the Appendix C for more detailed about IoT JSON Library. If you want to learn more detailed about JSON format or IPSO criterion, you can reference their official website.

3.1 About JSON Packages

3.1.1 JSON Object

A collection of name/value pairs. In various languages, this is realized as a record, dictionary, hash table or keyed list.

Ex:

```
“age”: 5
```

3.1.2 JSON Array

An ordered list of values. In most languages, this is realized as an array, vector or sequence. Each array element is composed of json objects, and using braces to encase it. By our ID system, “Id” item is necessary in every element.

EX:

```
"e":[
   {"n":"tree", "v":5, "Id":12345XX}, -> an array element
   {"n":"flower", "v":20, "Id":12345XX},
   ]
```

3.1.3 IPSO JSON Package

An IPSO JSON Package has the json object items below:

- Root items: “e”, “bn”, “bu”, “bt”“ver”,
- Measurement or parameter entries: “n”, “u”, “v”, “sv”, “bv”, “s”, “t”, “ut”

In our definition, adding “Id” into this package is necessary.

EX:

```
"Voltage": {
    "e":[
        {"n":"Vcore", "v":1.0, "Id":12345XX},
        {"n":"V3.3", "v":3.28, "Id":12345XX},
        ...
    ]
    "bn": "Voltage",
    "bu": "V"
    "bt": 1276020076,
    "ver": 1,
    “Id”: 12345XX
}
```

3.1.4 Function JSON Package

This JSON package is our definition. It is a set of the related IPSO JSON packages and adding “bn” and “Id” items. The value of “bn” is same with key name of this package.

EX:

```
“Hardware Monitor”: {
    “Id”: 12345XX,
    “bn”: “Hardware Monitor”
    "Voltage": {
        "e":[
            {"n":"Vcore", "v":1.0, "Id":12345XX },
            {"n":"V33", "v":3.28, "Id":12345XX },
            ...
        ],
        "bn": "Voltage",
        "bu": "V"
        "bt": 1276020076,
        "ver": 1,
        “Id”:
        }
    “Temperature”: {
        ...
    }
}
```

3.1.5 Capability JSON Package

This JSON package is our definition. It packs all Function JSON Packages to the Capability JSON Package. The Module API `SusiIoTGetMDCapability` will pass this Capability JSON package to SUSI IoT kernel directly.

EX:

```
{
    “Hardware Monitor”: {
        “id”: 12345XX,
        “bn”: “Hardware Monitor”
        "Voltage": {
            "e":[
                {"n":"Vcore", "v":1.0, "Id":12345XX },
                {"n":"V33", "v":3.28, "Id":12345XX },
                ...
            ],
            "bn": "Voltage",
            "bu": "V"
            "bt": 1276020076,
            "ver": 1,
            “id”:
            }
        “Temperature”: {
            ...
        }
    }
    “GPIO”: {
        ...
    }
}
```

3.2 Create Capability JSON Package using IoT JSON Library

3.2.1 createIoTArrayElementInt/Real/String

The array element is the lowest of structure of our Capability JSON package that encapsulating one data information, so we create an array element first.

These functions have three type for different data type like integer, real and string. You have to call corresponding function depend on your data type. This rule is applied for other APIs.

EX:

```
json_t* arrayElements[5] = {0};
arrayElements[0] = createIoTArrayElementInt("item 0", 100, 0xF01);

// output:
// {“n”:“item 0”, “v”:100, “Id”:0xF01}
```

3.2.1.1 insertItemInt/Real/String/JsonObject/Ex

In SUSI IoT, we define the other items we wanted (see the below). If you want to add your array elements into our IPSO JSON Packages, you have to insert them. You can use these functions to insert these items or other item your own definition.

```
insertItemEx(arrayElements[0], "{s:i, s:i, s:s, s:s}",
                                "max", 255,
                                "min", 0,
                                "asm", "rw");
insertItemInt(arrayElements[0], "other", 10);

// output:
// {“n”:“item 0”, “v”:100, “Id”:0xF01, “max”:255, “min”:0, “asm”:”rw”, “other”:10}
```

Our definition of variable

- “max”: the maximum of data value
- “min”: the minimum of data value
- “asm”: the read/write property of data. r: read only, w: write only, rw: both
- “u”: optional. the unit of data value
- “ui”: optional. What kind of UI applying to the data.
- “sui”: optional. What kind of SUSIAccess UI applying to the data.
- “uiop”: optional.

3.2.2 createIoTIPSOPackage/Ex

This function will pack array elements that you created before. If you want to add your array elements in our hardware monitor type like voltage, temperature, fan speed, current, case open and so on, you just call `createIoTIPSOPackage`.

If you want to create your own IPSO JSON Package, you should call `createIoTIPSOPackageEx` because you have to specify some items like “bn”, “bu” and “Id” by yourself.

To create the IPSO packages of GPIO and backlight has to use `createIoTIPSOPackageEx` because we have no ID reserved for them.

```
json_t* ipsoPackages[5] = {0};
...
ipsoPackages[0] =
  createIoTIPSOPackageEx(arrayElements, 5, "IPSO item 0", "V", 0xF10);

// output:
// “IPSO item 0”: {
//     “e”: [
//          {“n”:“item 0”, “v”:100, “id”:0xF01 ...}
//          ],
//     “bn”: “IPSO item 0”,
//     “bu”: “V”
//     “bt”: “1276020076”
//     “ver”: “1”
//     “Id”: “0xF10”
//}
```

3.2.3 createIoTFuncPackage/Ex

This function will pack IPSO JSON Packages that you created before. If you want to add yours IPSO JSON Packages in our class like platform information, hardware monitor, GPIO, backlight and so on, you just call `createIoTFuncPackage`.

If you want to create your own Function JSON Package, you have to call `createIoTIPSOPackageEx` because you have to specify some items like “bn” and “Id” by yourself.

```
json_t* funcPackages[5] = {0};
...
funcPackages[0] = createIoTFuncPackageEx(ipsoPackages, 5, "Function 0", 0xF00);

// output:
// “Function 0”: {
//     “bn”: “Funcion 0”,
//     “Id”: 0xF00,
//     “IPSO item 0”: {
//         ...
//     }
// }
```

3.2.4 createIoTJsonCapability

Finally, you just call `createIoTJsonCapability` to create the Ccapability JSON package. This function will pack all Function JSON Packages that you created before.

```
json_t* capability = createIoTJsonCapability(funcPackages, 5);

// output:
// {
//     “Function 0”: {
//         “bn”: “Funcion 0”,
//         “Id”: 0xF00,
//         “IPSO item 0”: {
//             ...
//         }
//     }
//     “Function 1”: {
//         ...
//     }
//     ...
// }
```

4. Example

Now, we will give you a demonstration of porting a module library from an empty project.

4.1 Including related project and files

First, create an empty module project, and you can add the ***jansson*** and ***IoTJsonLibrary*** projects in your solution of the project your created. Right click the new project->Project Dependencies, mark the ***jansson*** and ***IoTJsonLibrary*** item to link their symbol. By the way, your configuration type of project needs to set to Dynamic Library.

4.2 Header file

Creating an empty header file and including the ***OsDeclarations.h***, ***SusiIoT.h*** and ***IoTJsonLibrary.h*** files. If your compiler cannot find some header files when you compiling your source code, you have to add the path of header file into Additional Include Directories in the project or just change relative path to absolute path in your code directly.

```
#include "OsDeclarations.h"
#include "SusiIoT.h"
#include "IoTJsonLibrary.h"
```

Next, declaring the prototype of module's APIs (See the Appendix B). If your module library is using C++ compiler, you should use "extern C".

```
SusiIoTStatus_t SUSI_IOT_API SusiIoTInitialize(void);
SusiIoTStatus_t SUSI_IOT_API SusiIoTUninitialize(void);
SusiIoTStatus_t SUSI_IOT_API SusiIoTGetMDCapability(json_t *capability);
SusiIoTStatus_t SUSI_IOT_API SusiIoTGetMDValue(SusiIoTId_t id, json_t *value);
SusiIoTStatus_t SUSI_IOT_API SusiIoTSetMDValue(SusiIoTId_t id, json_t *value);
uint8_t SUSI_IOT_API SusiIoTGetModuleID ();
```

4.3 Source files

Creating an empty .c/.cpp file and implement the APIs. Don't forget to include the header file you create at 4.2.

4.3.1 SusiIoTInitialize

Declare a static global pointer variable point to json_t to store the Capability JSON Packages. Declare an array of point to json_t for Function JSON Packages in *SusiIoTInitialize*. Using *createOEM* to create Function JSON Packages and then using *createIoTJsonCapability* to create final Capability JSON Package

```
static json_t* jsonCapabilityPackage;

SusiIoTStatus_t SUSI_IOT_API SusiIoTInitialize(void)
{
  json_t* funcPacks[1] = {0};
  funcPacks[0] = createOEM();

  jsonCapabilityPackage = createIoTJsonCapability(funcPacks, 1);

  return SUSIIOT_STATUS_SUCCESS;
}
```

4.3.1.1 createOEM

Declare two arrays of pointer of point to json_t, one is set of array element for creating IPSO JSON Package, and another is set of IPSO JSON Package for creating Function JSON Package.
These five array element use *createIoTArrayElementInt* to create array element with integer type data, and insert different data type individually. After creating array elements, call the *createIoTIPSOPackageEx* to create the IPSO JSON Package. Finally, call the *createIoTFuncPackageEx* to get the Function JSON Package and return it.

```
static json_t* createOEM()
{
  json_t* arrayElems[5] = {0};
  json_t* ipsoPacks[1] = {0};

  arrayElems[0] = createIoTArrayElementInt("OEM item 0", 100, 0x81000F01);
  insertItemInt(arrayElems[0], "int", 1);
  insertItemString(arrayElems[0], "asm", "rw");

  arrayElems[1] = createIoTArrayElementReal("OEM item 1", 100.5, 0x81000F02);
  insertItemReal(arrayElems[1], "real", 1.0);

  arrayElems[2] = createIoTArrayElementString("OEM item 2", "hello", 0x81000F03);
  insertItemString(arrayElems[2], "string", "oem");

  arrayElems[3] = createIoTArrayElementInt("OEM item 3", 100, 0x81000F04);
  insertItemJsonObject(arrayElems[3], "object", json_integer(1));

  arrayElems[4] = createIoTArrayElementInt("OEM item 4", 100, 0x81000F05);
  insertItemEx(arrayElems[4], "{s:i, s:f, s:s, s:o}",
    "int", -100, "real", -100.0, "string", "oem", "object", arrayElems[0]);
  deleteItem(arrayElems[4], "int");

  ipsoPacks[0] = createIoTIPSOPackageEx(arrayElems, 5, "OEM IPSO 0", "",
                                        0x81000F10);

  return createIoTFuncPackageEx(ipsoPacks, 1, "OEMFunction 0", 0x81000F00);
}
```

4.3.2 SusiIoTUninitialize

Using *freeJsonObject* to release the memory space of Capability JSON Package

```
SusiIoTStatus_t SUSI_IOT_API SusiIoTUninitialize(void)
{
  freeJsonObject(&jsonCapabilityPackage);
  return SUSIIOT_STATUS_SUCCESS;
}
```

4.3.3 SusiIoTGetMDCapability

Use *json_object_update* to pass the Capability JSON Package to SUSI IoT kernel.

```
SusiIoTStatus_t SUSI_IOT_API SusiIoTGetMDCapability(json_t *capability)
{
  if (json_object_update(capability, jsonCapabilityPackage))
  {
    return SUSIIOT_STATUS_ERROR;
  }
  return SUSIIOT_STATUS_SUCCESS;
}
```

4.3.4 SusiIoTGetMDValue

You can use the id you specify before to know which data you want to get it. You can use your data structure to record the relationship between id and data, and then you can find the data easier. Next, you call the OEM function to get the value, and then you can use *json_xxx_set* function to set the value to parameter data.

```
SusiIoTStatus_t SUSI_IOT_API SusiIoTGetMDValue(SusiIoTId_t id, json_t *data)
{
  if (id == 0x81000F02) {
    double value = OEMAPItoGetValueR();
    json_real_set(data, value);
    return SUSIIOT_STATUS_SUCCESS;
  }
  else if (id == 0x81000F03) {
    char* value = OEMAPItoGetValueS();
    json_string_set(data, value);
    return SUSIIOT_STATUS_SUCCESS;
  }
  else {
    int value = OEMAPItoGetValueI();
    json_integer_set(data, value);
    return SUSIIOT_STATUS_SUCCESS;
  }
  return SUSIIOT_STATUS_UNSUPPORTED;
}
```

4.3.5 SusiIoTSetMDValue

You can use the id you specify before to know which data you want to set it. You can use your data structure to record the relationship between id and data, and then you can find the data easier. Next, you can use *json_xxx_value* function to get the value of parameter data, and then you call the OEM function to set the value.

```
SusiIoTStatus_t SUSI_IOT_API SusiIoTSetMDValue(SusiIoTId_t id, json_t *data)
{
  if (id == 0x81000F02) {
    double value = json_real_value(data);
    OEMAPItoSetValueR(value);
    return SUSIIOT_STATUS_SUCCESS;
  }
  else if (id == 0x81000F03) {
    char* value = json_string_value(data);
    OEMAPItoSetValueS(value);
    return SUSIIOT_STATUS_SUCCESS;
  }
  else {
    int value = (int)json_integer_value(data);
    OEMAPItoSetValueI(value);
    return SUSIIOT_STATUS_SUCCESS;
  }
}
```

4.3.6 SusiIoTGetModuleID

```
uint8_t SUSI_IOT_API SusiIoTGetModuleID()
{
  return 0x81;
}
```

4.4 Module-Definition file

Finally, you need to export your symbols of module's APIs. You can use Module-Definition file (.def) to list all symbols.

```
LIBRARY "SampleLib"
EXPORTS
    SusiIoTInitialize
    SusiIoTUninitialize
    SusiIoTGetMDCapability
    SusiIoTGetMDValue
    SusiIoTSetMDValue
    SusiIoTGetModuleID
```

4.5 Path of Binaries

You have to put your module library file to specific path.

- SampleLib.dll, otherModule.dll…
  - C:\windows\SUSI\DLL\
- jansson.dll, SusiIoT.dll:
  - C:\windows\system32\

Now, you can use SusiIoTDemo.exe tool to test your module!

Appendix A. Definition

1. Status Codes

All module API functions immediately return a status code from a common list of possible errors. Any function may return any of the defined status codes.

#define SUSIIOT_STATUS_NOT_INITIALIZED 0xFFFFFFFF

Description

The SUSIIOT API library is not yet or unsuccessfully initialized. SusiIoTInitialize needs to be called prior to the first access of any other SUSIIOT API functions.

Actions

Call SusiIoTInitialize.

#define SUSIIOT_STATUS_INITIALIZED 0xFFFFFFFE

Description

Library is initialized.

Actions

None.

#define SUSIIOT_STATUS_ALLOC_ERROR 0xFFFFFFFD

Description

Memory Allocation Error.

Actions

Free memory and try again.

#define SUSIIOT_STATUS_DRIVER_TIMEOUT 0xFFFFFFFC

Description

Time out in driver. This is Normally caused by hardware/software semaphore timeout.

Actions

Retry.

#define SUSIIOT_STATUS_INVALID_PARAMETER 0xFFFFFEFF

Description

One or more of the SUSIIOT API functions call parameters are out of defined range.

Actions

Verify Function Parameters.

#define SUSIIOT_STATUS_INVALID_BLOCK_ALIGNMENT 0xFFFFFEFE

Description

The Block Alignment is incorrect.

Actions

Use Inputs and Outputs to correctly select input and outputs.

#define SUSIIOT_STATUS_INVALID_BLOCK_LENGTH 0xFFFFFEFD

Description

This means that the Block length is too long.

Actions

Use Alignment Capabilities information to correctly align write access.

#define SUSIIOT_STATUS_INVALID_DIRECTION 0xFFFFFEFC

Description

The current Direction Argument attempts to set GPIOs to a unsupported direction. I.E. Setting GPI to Output.

Actions

Use Inputs and Outputs to correctly select input and outputs.

#define SUSIIOT_STATUS_INVALID_BITMASK 0xFFFFFEFB

Description

The Bitmask Selects bits/GPIOs which are not supported for the current ID.

Actions

Use Inputs and Outputs to probe supported bits.

#define SUSIIOT_STATUS_RUNNING 0xFFFFFEFA

Description

Watchdog timer already started.

Actions

Call SusiWDogStop before retrying.

#define SUSIIOT_STATUS_UNSUPPORTED 0xFFFFFCFF

Description

This function or ID is not supported at the actual hardware environment.

Actions

None.

#define SUSIIOT_STATUS_NOT_FOUND 0xFFFFFBFF

Description

Selected device was not found

Actions

None.

#define SUSIIOT_STATUS_TIMEOUT 0xFFFFFBFE

Description

Device has no response.

Actions

None.

#define SUSIIOT_STATUS_BUSY_COLLISION 0xFFFFFBFD

Description

The selected device or ID is busy or a data collision is detected.

Actions

Retry.

#define SUSIIOT_STATUS_READ_ERROR 0xFFFFFAFF

Description

An error is detected during a read operation.

Actions

Retry.

#define SUSIIOT_STATUS_WRITE_ERROR 0xFFFFFAFE

Description

An error is detected during a write operation.

Actions

Retry.

#define SUSIIOT_STATUS_MORE_DATA 0xFFFFF9FF

Description

The amount of available data exceeds the buffer size. Storage buffer overflow was prevented. Read count was larger than the defined buffer length.

Actions

Either increase the buffer size or reduce the block length.

#define SUSIIOT_STATUS_ERROR 0xFFFFF0FF

Description

Generic error message. No further error details are available.

Actions

None.

#define SUSIIOT_STATUS_SUCCESS 0

Description

The operation is successful.

Actions

None.

#define SUSIIOT_STATUS_JSON_TYPE_ERROR 0XFFFFF8FF

Description

The json type is not correctly.

Actions

None.

#define SUSIIOT_STATUS_JSON_OBJECT_EMPTY 0XFFFFF8FE

Description

The json type is empty.

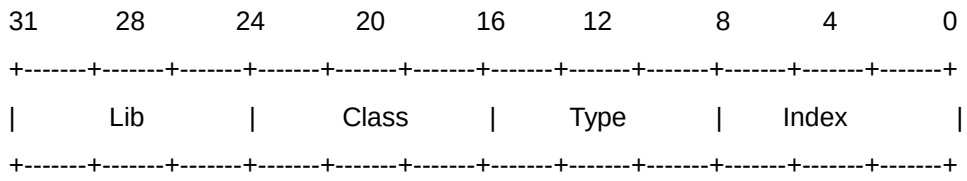
Actions

None.

2. ID System

Every data and JSON package of module need to specify a unique ID for SUSI IoT kernel to identify which data be operate. This ID must be unique at each modules.

2.1 ID format



2.1.1 Lib

Module library class code, ex: SUSI, SAB2000 and so on.

0x00~0x7F is reserved for Advantech device. 0x80~0xFE is reserved for OEM device.

2.1.2 Class

Main Class code, ex: HWM, WDT, SMBus...

0x00~0x7F is reserved for Advantech device. 0x80~0xFE is reserved for OEM device.

If you want to add your JSON package into Advantech Class, you can follow the ID below:

0x01: Platform Information

0x02: Hardware Monitor

0x04: GPIO

0x05: Backlight:

2.1.3 Type

Type of class code, ex: Temperature, Voltage...

0x00~0x7F is reserved for Advantech device. 0x80~0xFE is reserved for OEM device.

If you want to add your JSON package into Advantech Type, you can follow the ID below:

0x01: Temperature

0x02: Voltage

0x03: Fan Speed

0x04: Current:

0x05: CaseOpen

2.1.4 Index

The index of type items

Appendix B. Module APIs

3. Definition

The Module APIs provide the SUSI IoT kernel necessary information of module device. The SUSI_IOT_API modifier is used to define the calling convention.

3.1 Initialization Functions

3.1.1 SusiIoTInitialize

uint32_t SUSI_IOT_API SusiIoTInitialize(void)

Description:

To initialize the module. This function will be called first from SUSI IoT kernel if this module be loaded successfully.

Parameters:

None

Return Status Code:

Condition	Return Value
Success	SUSIIOT_STATUS_SUCCESS
Others	User defined

3.1.2 SusiIoTUninitialize

uint32_t SUSI_IOT_API SusiIoTUninitialize(void)

Description:

To uninitialize the module. This function will be called from SUSI IoT kernel when this module is unloaded.

Parameters:

None

Return Status Code:

Condition	Return Value
Success	SUSIIOT_STATUS_SUCCESS

Others	User defined

3.2 Information Functions

3.2.1 SusiIoTGetMDCapability

uint32_t SUSI_IOT_API SusiIoTGetMDCapability(json_t *capability)

Description:

To provide all information and data about module device.

Parameters:

capability

a buffer point to the value of capability JSON package

Return Status Code:

Condition	Return Value
Success	SUSIIOT_STATUS_SUCCESS
Others	User defined

3.3 Data Functions

3.3.1 SusiIoTGetMDValue

uint32_t SUSI_IOT_API SusiIoTGetMDValue(uint32_t id, json_t *data)

Description:

To provide the data of module device

Parameters:

Id

A unique 4 bytes value at all module devices

data

a buffer point to the value of data JSON package

Return Status Code:

Condition	Return Value
Success	SUSIIOT_STATUS_SUCCESS
Others	User defined

3.3.2 SusiIoTSetMDValue

uint32_t SUSI_IOT_API SusiIoTSetMDValue(uint32_t id, json_t *data)

Description:

Setting the data to module device

Parameters:

id

A unique 4 bytes value at all module devices

data

a buffer point to the value of data JSON package

Return Status Code:

Condition	Return Value
Success	SUSIIOT_STATUS_SUCCESS
Others	User defined

3.4 ID Functions

3.4.1 SusiIoTGetModuleID

uint8_t SUSI_IOT_API SusiIoTGetModuleID(void)

Description:

To provide the module ID of this module. It is a unique 2 byte value at all modules. The SUSI IoT kernel use this ID to identify each module.

Parameters:

None

Return Status Code:

Condition	Return Value
Module unique ID	2 byte value

Appendix C. IoT JSON Library APIs

1. Define

The IoT JSON Library APIs assist user to create the format of JSON object. The SUSI_IOT_API modifier is used to define the calling convention.

1.1 Create JSON Array Element

1.1.1 createIoTArrayElementInt

json_t* SUSI_IOT_API createIoTArrayElementInt(const char* n, const int v, const int id)

Description:

To create a JSON object with integer type data of array element.

Parameters:

n

the item name

v

the integer data

id

A unique 4 bytes value at all modules

Return Status Code:

Condition	Return Value
Success	A point of JSON object
Failed	NULL pointer

1.1.2 createIoTArrayElementReal

json_t* SUSI_IOT_API createIoTArrayElementReal(const char* n, const double v, const int id)

Description:

To create a JSON object with double type data of array element.

Parameters:

n

the item name

v

the double type data

id

A unique 4 bytes value at all modules

Return Status Code:

Condition	Return Value
Success	A point of JSON object
Failed	NULL pointer

1.1.3 createIoTArrayElementString

json_t* SUSI_IOT_API createIoTArrayElementString(const char* n, const char* v, const int id)

Description:

To create a JSON object with string type data of array element.

Parameters:

n

the item name

v

the string type data

id

A unique 4 bytes value at all modules

Return Status Code:

Condition	Return Value
Success	A point of JSON object
Failed	NULL pointer

1.2 Create JSON IPSO Package

1.2.1 createIoTIPSOPackage

json_t* SUSI_IOT_API createIoTIPSOPackage(json_t* jsonArrayObjects[], int size, IPSO_T type)

Description:

To create a JSON object which following IPSO format.

Parameters:

jsonArrayObjects

a set of JSON array element

size

the size of jsonArrayObjects

type

see xxx

Return Status Code:

Condition	Return Value
Success	A point of JSON object
Failed	NULL pointer

1.2.2 createIoTIPSOPackageEx

json_t* SUSI_IOT_API createIoTIPSOPackageEx(json_t* jsonArrayObjects[], int size, const char* bn, const char* bu, const int id)

Description:

To create a JSON object which following IPSO format.

Parameters:

jsonArrayObjects

a set of JSON array element

size

the size of jsonArrayObjects

bn

the base item name

bu

the base unit

id

A unique 4 bytes value at all modules

Return Status Code:

Condition	Return Value

Success	A point of JSON object
Failed	NULL pointer

1.3 Create JSON Function Package

1.3.1 createIoTFuncPackage

json_t* SUSI_IOT_API createIoTFuncPackage(json_t* jsonIPSOObjects[], int size, FUNC_T type)

Description:

To create a JSON object which pack IPSO JSON packages.

Parameters:

jsonIPSOObjects

a set of IPSO JSON packages

size

the size of jsonIPSOObjects

type

see xxx

Return Status Code:

Condition	Return Value
Success	A point of JSON object
Failed	NULL pointer

1.3.2 createIoTFuncPackageEx

json_t* SUSI_IOT_API createIoTFuncPackageEx(json_t* jsonIPSOObjects[], int size, const char* bn, const int id)

Description:

To create a JSON object which pack IPSO JSON packages

Parameters:

jsonIPSOObjects

a set of IPSO JSON packages

size

the size of jsonIPSOObjects

bn

the item name

id

A unique 4 bytes value at all modules

Return Status Code:

Condition	Return Value
Success	A point of JSON object
Failed	NULL pointer

1.4 Create JSON Capability Package

1.4.1 createIoTJsonCapability

json_t* SUSI_IOT_API createIoTJsonCapability(json_t* jsonObjects[], int size)

Description:

To create a capability of JSON object which pack function JSON packages

Parameters:

jsonObjects

a set of function JSON packages

size

the size of jsonObjects

Return Status Code:

Condition	Return Value
Success	A point of JSON object
Failed	NULL pointer

1.5 Inset Item Functions

1.5.1 insertItemInt

int SUSI_IOT_API insertItemInt(json_t* object, const char* key, const int value)

Description:

Inset an integer type item to a JSON object

Parameters:

objects

the object which be inserted.

key

the item name

value

the integer type data

Return Status Code:

Condition	Return Value
Success	0
Failed	-1

1.5.2 insertItemReal

int SUSI_IOT_API insertItemReal(json_t* object, const char* key, const double value)

Description:

Inset an double type item to a JSON object

Parameters:

objects

the object which be inserted.

key

the item name

value

the double type data

Return Status Code:

Condition	Return Value
Success	0
Failed	-1

1.5.3 insertItemString

int SUSI_IOT_API insertItemString(json_t* object, const char* key, const char* value)

Description:

Inset an string type item to a JSON object

Parameters:

objects

the object which be inserted.

key

the item name

value

the string type data

Return Status Code:

Condition	Return Value
Success	0
Failed	-1

1.5.4 insertItemJsonObject

int SUSI_IOT_API insertItemJsonObject(json_t* object, const char* key, json_t other)

Description:

Inset an JSON object type item to a JSON object

Parameters:

objects

the object which be inserted.

key

the item name

other

the JSON object type data

Return Status Code:

Condition	Return Value
Success	0
Failed	-1

1.5.5 insertItemEx

int SUSI_IOT_API insertItemEx(json_t* object, const char *fmt, ...)

Description:

Inset more item to a JSON object

Parameters:

object

the object which be inserted.

fmt

the format string. Please see the last section 1.7

…

one or more arguments are consumed and used to build the corresponding value

Return Status Code:

Condition	Return Value
Success	0
Failed	-1

1.5.6 deleteItem

int SUSI_IOT_API deleteItem(json_t* object, const char* key)

Description:

Delete an item of a JSON object by item name

Parameters:

objects

the object which be inserted.

key

the item name

Return Status Code:

Condition	Return Value
Success	0
Failed	-1

1.6 Release Resource Functions

1.6.1 freeJsonObject

void SUSI_IOT_API freeJsonObject(json_t **jsonObject)

Description:

Release the memory space of JSON object

Parameters:

jsonObject

a pointer to pointer of JSON object which be released

1.7 String Format

This section describes the format of parameter fmt of InsertItemEx API.

Here's the list of format specifiers. The type in brackets (if any) denotes the C type that is expected as the corresponding argument or arguments.

- s [const char *]
  - Convert a JSON string to a pointer to a null terminated UTF-8 string. The resulting string is extracted by using json_string_value() internally, so it exists as long as there are still references to the corresponding JSON string
- s% [const char *, size_t *]
  - Convert a JSON string to a pointer to a null terminated UTF-8 string and its length
- b [int]
  - Convert a JSON boolean value to a C int, so that true is converted to 1 and false to 0.
- i [int]
  - Convert a JSON integer to C int.
- f [double]
  - Convert a JSON real to C double
- o [json_t *]
  - Store a JSON value with no conversion to a json_t pointer

Whitespace, Colon(:) and Comma(,) are ignored.

Duo to json representation is composed of string key name and some type value pair, so you can use colon express the key and value pair. Two or more pair use comma to separate it.

For example,

```
"a": "item"   // s:s
"b": 0x123    // s:i
"c": "item", "d": 2.1, "e": 10  // s:s, s:f, s:i
```

So you can use the InsertItemEx to insert two or more variable and its value.

EX:

```
InsertItemEx(jsonObject, "{s:s, s:f, s:i}", "c", "item", "d", 2.1, "e", 10);
```