OSAL REFERENCE

DATE:
October 13, 2022

OSAL API VERSION:
1.0.0

s.m.s, smart microwave sensors GmbH
In den Waashainen 1
38108 Braunschweig
Germany

Phone: +49 531 39023-0
Fax: +49 531 39023-599
info@smartmicro.de
www.smartmicro.com

CONTENT

1 OPERATING SYSTEM ABSTRACTION LAYER(OSAL)

OSAL is a glue layer between the Smart Access library and the customers underlying operating system. It needs to support the customers CAN, RS485 and Ethernet drivers. The OSAL implementation should provide the following functionality:

1.1 OSAL INTERFACE

In this section is provided a platform independent interface description, which can be commonly implemented for Linux, Windows or other operating systems.

1.1.1 GETTING OSAL INSTANCE

The upper layer(Smart Access Library) shall be able to retrieve the OSAL instance by calling the function below:

```
static std::shared_ptr<OSALIface> GetOSAL();
```

It could be implemented in the following way:

```
static std::shared_ptr<OSALImpl> instance;

std::shared_ptr<OSALIface> OSALIface::GetOSAL()
{
    if(!instance)
    {
        instance.reset(new OSALImpl());
    }
    return instance;
}
```

1.1.2 INITIALIZATION

The upper layer(Smart Access Library) shall be able to initialize the OSAL layer:

```
virtual bool Init(IN const std::string& configPath) = 0;
```

Returns true, if the initialization was successful and false, if it failed. It gets a configuration path (please see configuration section of this manual) as argument.

1.1.3 GETTING LIST OF HW DEVICES

The upper layer(Smart Access Library) shall be able to retrieve from OSAL a list of all supported hardware devices :

```
virtual bool GetHwIfaces(OUT std::list<std::shared_ptr<HwIfaceDescriptor>>& ) = 0;
```

Where **HwIfaceDescriptor** is the base class, those descendants are:

1.1.3.1 HW CAN DESCRIPTOR

Class HwCanDescriptor has the following members:

```
private:
    CanDevId        _deviceId;
    std::string     _ifaceName;
    uint32_t        _baudrate;
```

- **_deviceId** – CAN device id. IMPORTANT: it must match with device id from the routing table.
- **_ifaceName** – CAN interface name.
- **_baudrate** – CAN baudrate .

For more information please see ExternalTypes.h file.

1.1.3.2 HW RS485 DESCRIPTOR

Class HwRS485Descriptor has the following members:

```
private:
    SerialDevId         _deviceId;
    std::string         _ifaceName;
    uint32_t            _baudrate;
```

- **_deviceId** – Serial Device Id. IMPORTANT: it must match with device id from the routing table.
- **_ifaceName** – Interface name e.g. “/dev/ttyS0” or “COM1”.
- **_baudrate** – RS485 baudrate.

For more information please see ExternalTypes.h file.

1.1.3.3 HW ETHERNET DESCRIPTOR

Class HwEthernetDescriptor has the following members:

```
private:
    uint16_t        _deviceId;
    std::string     _ifaceName;
    std::string     _ip;
    uint32_t        _port;
```

- **_deviceId** – Ethernet adapter device id.
- **_ifaceName** – Interface name e.g. “eth0”.
- **_ip** – IP Address.
- **_port** – UDP port.

For more information please see ExternalTypes.h file.

1.1.4 HW REGISTRY

The purpose of Hardware Registries is to describe a certain physical interface. As opposed to a hardware descriptor it describes both real physical hardware devices e.g. CAN and logical ones e.g. UDP sockets. **HwRegistry** , similar to **HwIfaceDescriptor** , is used in the interface as a base class and needs to be casted in OSAL implementation to its descendants:

1.1.4.1 CAN HW REGISTRY

Class CanHwRegistry has the following members:

```
private:
    std::set<CanId>             _ids;
    CanDevId                    _deviceId
```

- **_ids** – List of CAN Identifiers that shall be received by CAN device, the rest needs to be dropped.

– **_deviceId** – CAN device ID. The same id as in the hardware descriptor.

For more information please see ExternalTypes.h file.

1.1.4.2 RS485 HW REGISTRY

Class Rs485HwRegistry has the following members:

```
private:
    SerialDevId                 _deviceId
```

– **_deviceId** – Serial device ID. The same ID as in the hardware descriptor.

For more information please see ExternalTypes.h file.

1.1.4.3 ETHERNET HW REGISTRY

Class EthernetRegistry has the following members:

```
private:
    std::string                 _ip;
    uint32_t                    _port;
    EthTransportType            _transType;
```

– **_ip** – IP address.
– **_port** – UDP/TCP port.
– **_transType** – Transport type TCP/UDP/UDP MULTICAST (TCP is currently not supported).

For more information please see ExternalTypes.h file.

1.1.5 REGISTER RECEIVER

In order to receive incoming packets , the upper layer need to be able to register receiver callbacks, these callbacks shall be invoked upon receiving data on the described in hardware registry interface.

```
virtual ErrorCode RegisterReceiver(IN const HwRegistry& registry,
                                   IN ReceiverCallback callback) = 0;
```

The receiver callback has the following format:

```
typedef std::function<void(BufferDescriptor&)> ReceiverCallback;
```

Where the buffer descriptor is built as follows:

```
class BufferDescriptor
{
public:
    BufferDescriptor(uint8_t* bufferPtr, size_t size);
    BufferDescriptor();

    ~BufferDescriptor();

    uint8_t* GetBufferPtr() const;
    size_t GetSize() const;
    void SetBufferPtr(IN uint8_t* bufferPtr);
    void SetSize(IN size_t size);


private:
    uint8_t*            _bufferPtr;
    size_t              _size;
```

```
};
```

- **_bufferPtr** – Pointer to the allocated buffer. Please notice that the OSAL is responsible to allocate and to free this memory.
- **_size** – Size of the allocated buffer.

1.1.6 UNREGISTER RECEIVER

In order to remove a previously registered callback, the interface below will be called.

```
virtual ErrorCode UnRegisterReceiver(IN const HwRegistry& registry) = 0;
```

1.1.7 SEND DATA

The upper layer(Smart Access Library) shall be able to send data to a certain hardware registry.

```
virtual ErrorCode SendData(IN const HwRegistry& registry,
                           IN const BufferDescriptor& buffer) = 0;
```

Please do not free this memory it will be freed by the upper layer.

For more information about OSAL interface please see the OSALIface.h file.

1.2 CONFIGURATION

The Osal implementation is configured by the "hw_inventory". The configurations are described in the subsection Linux 1.2.1 and Windows 1.2.2.

1.2.1 Linux

Exemplary CAN configuration:

```
"hwItems" :
[
    {
        "type"       : "can",
        "dev_id"     : 1,
        "iface_name" : "can0",
        "baudrate"   : 500000
    },
    ......
]
```

- **type** - Device type, set to "can".
- **dev_id** - ID of the CAN device required by the OSAL layer for identifying the adapter to be used.
- **iface_name** - Interface name of the used can socket.
- **baudrate** - CAN bus baudrate

Exemplary RS485 configuration:

```
"hwItems" :
[
    {
        "type"    : "rs485",
        "dev_id"      : 1,
        "iface_name" : "/dev/ttyS0",
        "baudrate" : 115200
    },
    ......
]
```

- **type** - Device type, set to "rs485".
- **dev_id** - ID of the CAN device required by the OSAL layer for identifying the adapter to be used.
- **iface_name** - Interface name of the used rs485 socket.
- **baudrate** - Uart baudrate.

Exemplary Ethernet configuration:

```
"clients" :
[
    {
        "type"      : "eth",
        "dev_id"        : 1,
        "iface_name"    : "eth0",
        "port" : 55555
    },
    ......
]
```

- **type** - Device type, set to "eth".
- **dev_id** - ID of the CAN device required by the OSAL layer for identifying the adapter to be used.
- **iface_name** - Interface name of the used ethernet socket.
- **port** - Receive udp port.

1.2.2 Windows

There is no configuration for an CAN device, because the reference OSAL does not support CAN on Windows.

Exemplary RS485 configuration:

```
"clients" :
[
    {
        "type"       : "rs485",
        "deviceId"   : "2",
        "usedPort"   : "COM1",
        "linkSpeed" : "115200"
    },
    ......
]
```

- **type** - Device type, set to "rs485".
- **deviceId** - ID of the CAN device required by the OSAL layer for identifying the adapter to be used.
- **usedPort** - It is the name of the COM port.
- **linkSpeed** - Uart baudrate.

Exemplary Ethernet configuration:

```
"clients" :
[
```

```
        {
            "type"       : "eth",
            "deviceId"   : "1",
            "usedPort"   : "65000",
            "ipAddress"  : "192.168.11.1"
        },
        ......
    ]
```

- **type** - Device type, set to "eth".
- **deviceId** - ID of the CAN device required by the OSAL layer for identifying the adapter to be used.
- **usedPort** - Receive udp port.
- **ipAddress** - Used pc ip address.

2 LEGAL DISCLAIMER NOTICE

All products, product specifications and data in this document may be subject to change without notice to improve reliability, function or otherwise. Not all products and/or product features may be available in all countries and regions. For legal reasons features may be deleted from products or smartmicro may refuse to offer products. Statements, technical information and recommendations contained herein are believed to be accurate as of the stated date. smartmicro disclaims any and all liability for any errors, inaccuracies or incompleteness contained in this document or in any other disclosure relating to the product.

To the extent permitted by applicable law, smartmicro disclaims (i) any and all liability arising out of the application or use of the product or the data contained herein, (ii) any and all liability of damages exceeding direct damages, including - without limitation - indirect, consequential or incidental damages, and (iii) any and all implied warranties, including warranties of the suitability of the product for particular purposes.

Statements regarding the suitability of products for certain types of applications are based on smartmicro's knowledge of typical requirements that are often placed on smartmicro products in generic/general applications. Statements about the suitability of products for a particular/specific application, however, are not binding. It is the customer's/user's responsibility to validate that the product with the specifications described is suitable for use in the particular/specific application. Parameters and the performance of products may deviate from statements made herein due to particular/specific applications and/or surroundings. Therefore, it is important that the customer/user has thoroughly tested the products and has understood the performance and limitations of the products before installing them for final applications or before their commercialization. Although products are well optimized to be used for the intended applications stated, it must also be understood by the customer/user that the detection probability may not be 100% and that the false alarm rate may not be zero.

The information provided, relates only to the specifically designated product and may not be applicable when the product is used in combination with other materials or in any process not defined herein. All operating parameters, including typical parameters, must be validated for each application by the customer's/user's technical experts. Customers using or selling smartmicro products for use in an application which is not expressly indicated do so at their own risk. This document does not expand or otherwise modify smartmicro's terms and conditions of purchase, including but not being limited to the warranty. Except as expressly indicated in writing by smartmicro, the products are not designed for use in medical, lifesaving or life-sustaining applications or for any other application in which the failure of the product could result in personal injury or death.

No license expressed or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document or by any conduct of smartmicro. Product names and markings noted herein may be trademarks of their respective owners.

Please note that the application of the product may be subject to standards or other regulations that may vary from country to country. smartmicro does not guarantee that the use of products in the applications described herein will comply with such regulations in any country. It is the customer's/user's responsibility to ensure that the use and incorporation of products comply with regulatory requirements of their markets.

If any provision of this disclaimer is, or is found to be, void or unenforceable under applicable law, it will not affect the validity or enforceability of the other provisions of this disclaimer.