# Customized SILECS Deploy-unit C++ wrapper

**Introduction**

For each Silecs Deploy-Unit it is possible to generate C++ objects which will allow the user to access his data in an object oriented manner.

## Why should I to use the wrapper?

The big advantage of this solution with respect to directly using the library are:

1. The names of controllers, devices, blocks and registers are names of objects and methods so if the user mistypes one of this names he will get an error at compilation time; Using the library this names were handled as string and exceptions were raised at runtime.
2. The set-up of the communication is simplified and can be completely transparent to the user.

## Should I always use the wrapper?

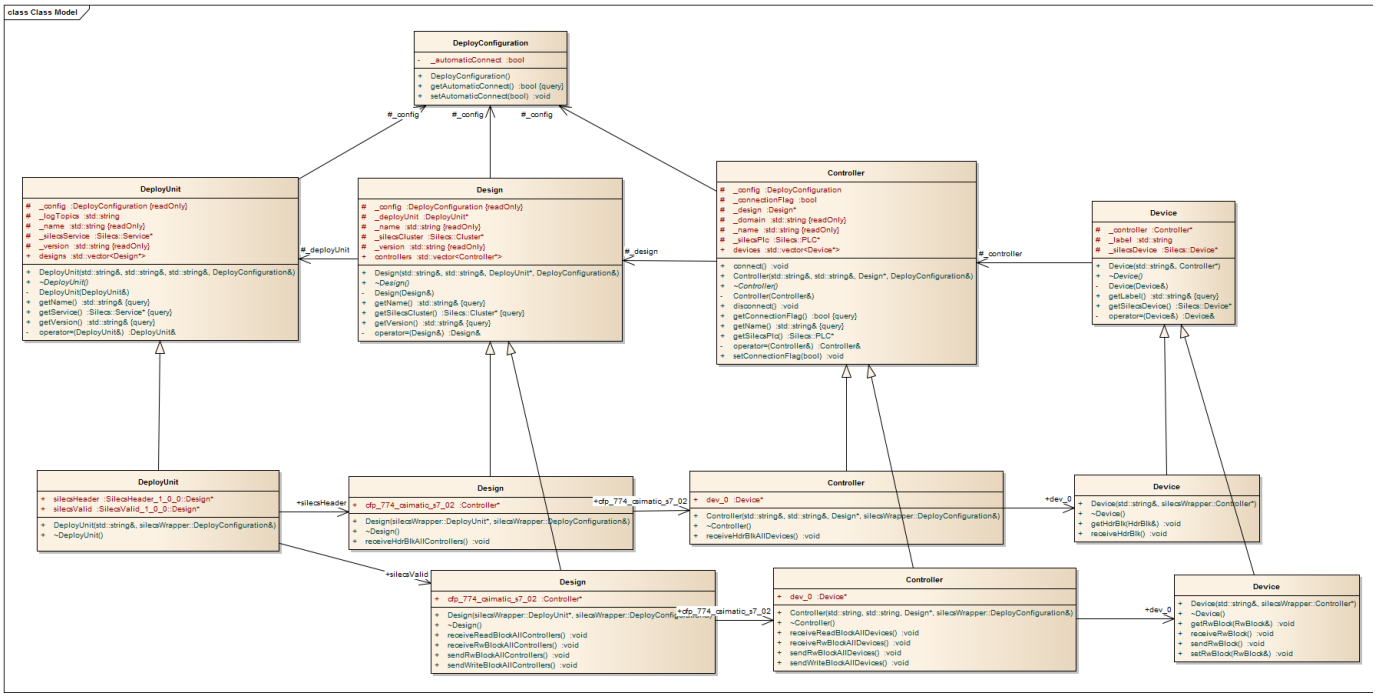
The wrapper simplifies the user code. It covers most of the cases uses by the user but the library is always supported and gives more flexibility to the user.

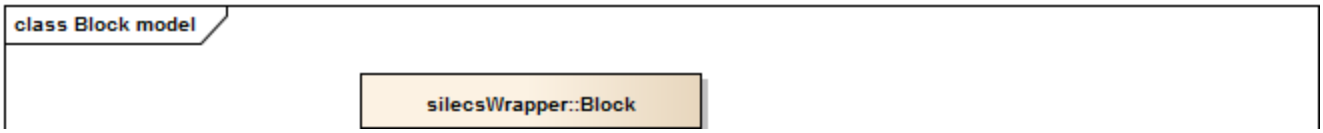
## Wrapper Structure

The wrapper is composed of 2 layers on top of the current C++ library. The first layer is a generic layer which namespace is called SilecsWrapper. This layer implements the common wrapper function and should be completely transparent to the user. The second layer is customized for the user Deploy Unit. This layer is composed by:

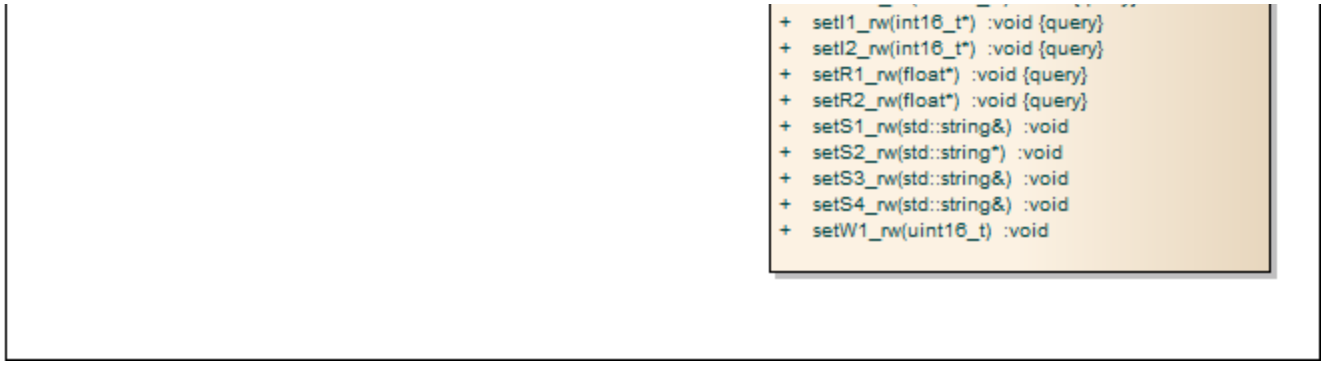
1. A single <DeployUnitName>::DeployUnit class.
2. One <DesignName>::Design class, for each design which is deployed by the deploy unit.
3. One <DesignName>::Controller class, for each design which is deployed by the deploy unit.
4. One <DesignName>::Device class, for each design which is deployed by the deploy unit.
5. One <DesignName>::<BlockName> class, for each block present in each design which is deployed by the deploy unit.

As a more concrete example the following class diagram presents the case where there is a deployUnit called Cfp774Simatic400_1_0_0 deploying 2 design called SilecsValid_1_0_0 and SilecsHeader_1_0_0.




#  _name  :std::string {readOnly}
+  Block(std::string&)
+  ~Block()
+  getName()  :std::string& {query}
SilecsHeader_1_0_0::HdrBlk
-  _checksum  :uint32_t
-  _date  :double
-  _user  :std::string
-  _version  :std::string
+  get_checksum()  :uint32_t {query}
+  get_date()  :double {query}
+  get_user()  :std::string {query}
+  get_version()  :std::string {query}
+  HdrBlk()
+  ~HdrBlk()
+  set_checksum(uint32_t)  :void
+  set_date(double)  :void
+  set_user(std::string&)  :void
+  set_version(std::string&)  :void
SilecsValid_1_0_0::RwBlock
-  b1_rw  :uint8_t
-  b2_rw  :uint8_t
-  b3_rw  :uint8_t
-  b4_rw  :uint8_t ([b4_rwDim1])
+  b4_rwDim1  :std::size_t = 3 {readOnly}
-  b5_rw  :uint8_t ([b5_rwDim1])
+  b5_rwDim1  :std::size_t = 3 {readOnly}
-  b6_rw  :uint8_t
-  c1_rw  :int8_t
-  c2_rw  :int8_t
-  c3_rw  :int8_t
-  di1_rw  :int32_t
-  dw1_rw  :uint32_t
-  dw2_rw  :uint32_t ([dw2_rwDim1][dw2_rwDim2])
+  dw2_rwDim1  :std::size_t = 1 {readOnly}
+  dw2_rwDim2  :std::size_t = 3 {readOnly}
-  i1_rw  :int16_t ([i1_rwDim1])
+  i1_rwDim1  :std::size_t = 2 {readOnly}
-  i2_rw  :int16_t ([i2_rwDim1][i2_rwDim2])
+  i2_rwDim1  :std::size_t = 2 {readOnly}
+  i2_rwDim2  :std::size_t = 5 {readOnly}
-  r1_rw  :float ([r1_rwDim1])
+  r1_rwDim1  :std::size_t = 3 {readOnly}
-  r2_rw  :float ([r2_rwDim1][r2_rwDim2])
+  r2_rwDim1  :std::size_t = 1 {readOnly}
+  r2_rwDim2  :std::size_t = 2 {readOnly}
-  s1_rw  :std::string
-  s2_rw  :std::string ([s2_rwDim1])
+  s2_rwDim1  :std::size_t = 3 {readOnly}
-  s3_rw  :std::string
-  s4_rw  :std::string
-  w1_rw  :uint16_t
+  getB1_rw()  :uint8_t
+  getB2_rw()  :uint8_t {query}
+  getB3_rw()  :uint8_t {query}
+  getB4_rw(uint8_t*)  :void {query}
+  getB5_rw(uint8_t*)  :void {query}
+  getB6_rw()  :uint8_t {query}
+  getC1_rw()  :int8_t {query}
+  getC2_rw()  :int8_t {query}
+  getC3_rw()  :int8_t {query}
+  getDi1_rw()  :int32_t {query}
+  getDw1_rw()  :uint32_t {query}
+  getDw2_rw(uint32_t*)  :void {query}
+  getI1_rw(int16_t*)  :void {query}
+  getI2_rw(int16_t*)  :void {query}
+  getR1_rw(float*)  :void {query}
+  getR2_rw(float*)  :void {query}
+  getS1_rw()  :std::string& {query}
+  getS2_rw(std::string*)  :void {query}
+  getS3_rw()  :std::string& {query}
+  getS4_rw()  :std::string& {query}
+  getW1_rw()  :uint16_t {query}
+  RwBlock()
+  ~RwBlock()
+  setB1_rw(uint8_t)  :void
+  setB2_rw(uint8_t)  :void
+  setB3_rw(uint8_t)  :void
+  setB4_rw(uint8_t*)  :void {query}
+  setB5_rw(uint8_t*)  :void {query}
+  setB6_rw(uint8_t)  :void
+  setC1_rw(int8_t)  :void
+  setC2_rw(int8_t)  :void
+  setC3_rw(int8_t)  :void
+  setDi1_rw(int32_t)  :void
+  setDw1_rw(uint32_t)  :void
+  setDw2_rw(uint32_t*)  :void {query}




# How To Use The Wrapper

## Introduction

The sample code below is based on the following SILECS configuration (in blue the ones we use):

- Deploy-Unit (DU) name/version : Cfp774Simatic400 / 1.0.0
  - Design class name/version : SilecsValid / 1.0.0
    - Design class blocks : rwBlock, readBlock, writeBlock
      - rwBlock registers : b1_rw, b2_rw, w1_rw, ...
- Controller host-name : cfp-774-csimatic-s7-02 *('-' is replaced by '_' in generated C++ code)*
  - SilecsValid instances (devices) : 0, 1, 2 *(we do not use explicite labels in this example)*

## Include the SILECS service in your Makefile

In your application Makefile (or FESA Make.specific for instance), it's enough to add the following include (refering to the current **MAJOR release 1**):

```
SILECS_VERSION?=1.m.p
include /acc/local/$(CPU)/silecs/$(SILECS_VERSION)/library/MakeSILECS.include
```

## Include the DU resources in your C++ code

```
// Include each DU Wrapper header file. '...' should refer to the appropriate root
folder in case of FESA implementation
#include <.../Common/Cfp774Simatic400_1_0_0.h>
```

**N.B:** in this example, the generated Wrapper header file (Cfp774Simatic400_1_0_0.h) has been copied in the Common folder of the FESA class repository (...)

## Basic Setup:

The default way to construct a DeployUnit and automatically establish the connection with all the controllers deployed by the DU:

```
// Construct deploy unit with default configuration (adding 'COMM' as diagnostic
topic)
Cfp774Simatic400_1_0_0::DeployUnit* du =
Cfp774Simatic400_1_0_0::DeployUnit::getInstance("ERROR,COMM");
```

**N.B:** logTopics ("ERROR, ...) are global parameters of the Silecs library. If your client application uses multiple wrappers with different log topics the ones applied to the latest constructed object will overwrite all others.
Possible diagnostic topics are described from this page: Client software implementation

## Receiving a block

A block is a data structure which is specific to a certain design and represent the minimal amount of information that can be sent/received to/from a particular device. You can construct a block as follow:

```
// Declare custom rwBlock
SilecsValid_1_0_0::RwBlock rwBlock;
```

In order to retrieve the block “rwBlock” from instance “0”, the following code can be used:

```
// receive rwBlock of SilecsValid for cfp_774_csimatic_s7_02/ device "0"
du->getSilecsValid()->getCfp_774_csimatic_s7_02()->getDevice("0")->receiveRwBlock();
// get the block
du->getSilecsValid()->getCfp_774_csimatic_s7_02()->getDevice("0")->getRwBlock(rwBlock)
;
// get b1_rw register value from the block
uint8_t b1 = rwBlock.getB1_rw();
```

If you want to receive the data blocks for all devices of cfp_774_csimatic_s7_02 at the same time you can use:

```
// receive rwBlock of SilecsValid for all devices of cfp_774_csimatic_s7_02
du->getSilecsValid()->getCfp_774_csimatic_s7_02()->receiveRwBlockAllDevices();
```

If you want to receive the data blocks for all devices of all the controllers of the DU at the same time you can use:

```
// receive rwBlock for all devices of all controllers
du->getSilecsValid()->receiveRwBlockAllControllers();
```

## Sending a block

In order to send a block to a specific device the following code can be used:

```
// set B1_rw register value
rwBlock.setB1_rw(5);
// send rwBlock of SilecsValid for cfp_774_csimatic_s7_02/ device "0"
du->getSilecsValid()->getCfp_774_csimatic_s7_02()->getDevice("0")->sendRwBlock(rwBlock
);
```

To be noticed that once a block is created all register are initialized to 0 and the strings are initialized as empty string. Since it is not possible to send a single register all the value of the block shall be initialized by the user before sending.

As for receiving sending can also be triggered at controllers and design level. Remember that all block shall be set first!

## Accessing a specific device

If the device were defined by label also a method get<DeviceLabel>() will be generated at the within the controller class. If the user has defined a number of devices then the devices can be access directly throught the map or via the method getDevice("label"); In this case the default generated label are 0,1,2...

## Delete SILECS resources

One method call is enough to close all connection and delete resources properly:

```
// Release SILECS resources
du->deleteInstance();
```

---

## Optional settings:

If the user needs to specify different connection flags to different controllers or he does not want that by default all controllers are connected this can be done as follow:

```
// Create configuration structure
silecsWrapper::DeployConfiguration config;
config.setAutomaticConnect(false);                    //do not connect all controllers
by default
// Construct deploy unit with custom configuration
Cfp774Simatic400_1_0_0::DeployUnit* du = new
Cfp774Simatic400_1_0_0::DeployUnit("ERROR",config);
// Set controller connection flags
du->getSilecsValid()->getCfp_774_csimatic_s7_02()->setConnectionFlag(false);
// Manually connect one particular controller
du->getSilecsValid()->getCfp_774_csimatic_s7_02()->connect();
```