

ChimeraTK.

ChimeraTK ApplicationCore in a nutshell.



Martin Killenberg
Martin Hierholzer

20th June 2024

Accelerator Middle Layer Workshop
DESY, Hamburg

ChimeraTK — Control system and **H**ardware Interface with **M**apped and **E**xtensible **R**egister-based device **A**bstraction **T**ool **K**it

What does a control application do?

Task 1

- Talk to the hardware
- ChimeraTK DeviceAccess

Task 2

- Run the business logic / algorithm
- ChimeraTK ApplicationCore

Task 3

- Interface to the control system
- ChimeraTK ControlSystemAdapter

ChimeraTK — Control system and **H**ardware **I**nterface with **M**apped and **E**xtensible **R**egister-based device **A**bstractio**n T**ool **K**it

What does a control application do?

Task 1

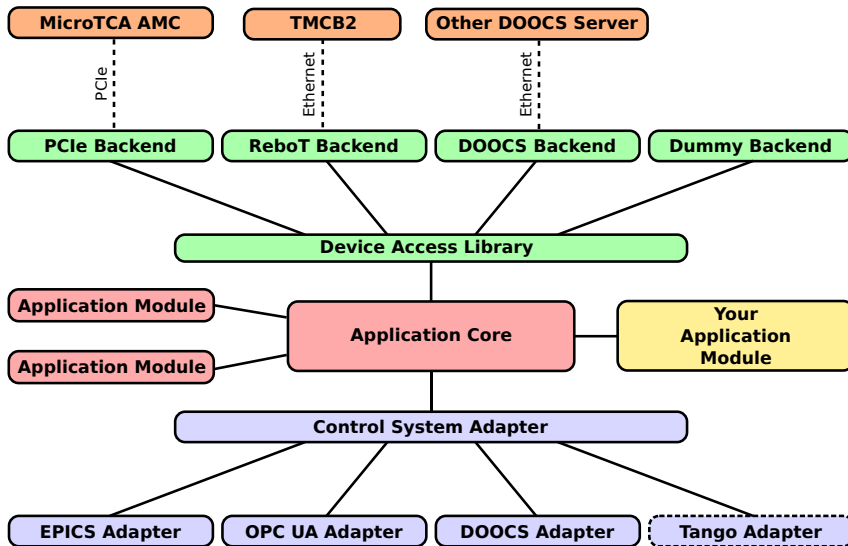
- Talk to the hardware
- ChimeraTK DeviceAccess

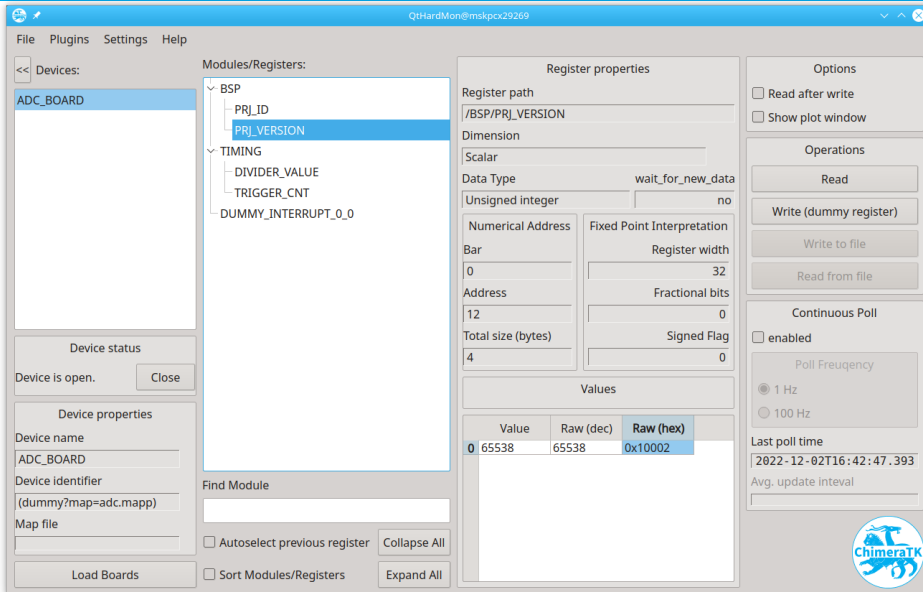
Task 2

- Run the business logic / algorithm
- ChimeraTK ApplicationCore

Task 3

- Interface to the control system
- ChimeraTK ControlSystemAdapter

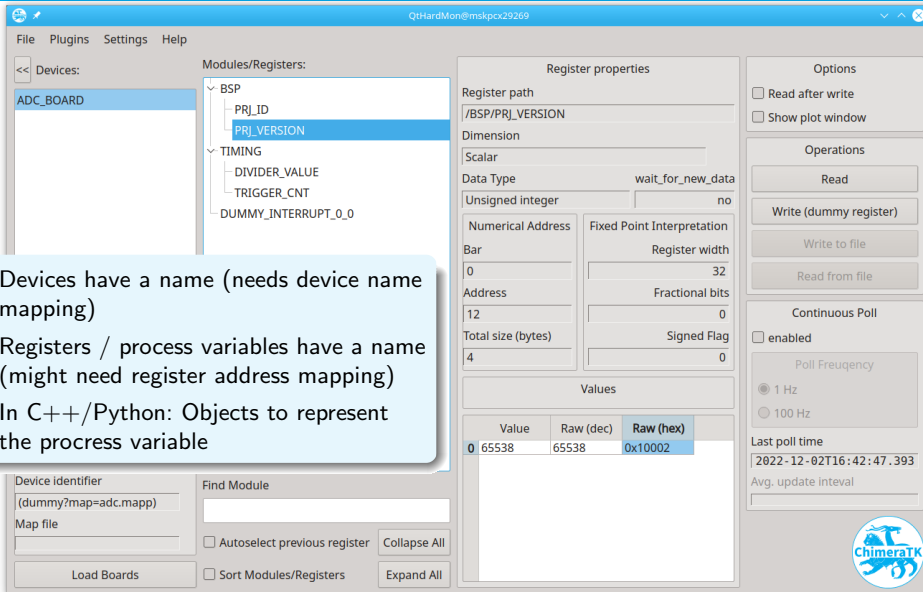




The screenshot displays the QtHardMon application window, titled "QtHardMon@mskpcx29269". The interface is organized into several panels:

- File Plugins Settings Help**: The top menu bar.
- Devices:** A list on the left showing "ADC_BOARD" selected.
- Modules/Registers:** A tree view in the center showing a hierarchy:
 - BSP
 - PRJ_ID
 - PRJ_VERSION** (selected)
 - TIMING
 - DIVIDER_VALUE
 - TRIGGER_CNT
 - DUMMY_INTERRUPT_0_0
- Register properties:** A panel on the right showing details for the selected register:
 - Register path: /BSP/PRJ_VERSION
 - Dimension: Scalar
 - Data Type: Unsigned integer (with a "wait_for_new_data" checkbox set to "no")
 - Numerical Address: Bar (0), Address (12), Total size (bytes) (4)
 - Fixed Point Interpretation: Register width (32), Fractional bits (0), Signed Flag (0)
 - Values table:

	Value	Raw (dec)	Raw (hex)
0	65538	65538	0x10002
- Options:** Checkboxes for "Read after write" and "Show plot window".
- Operations:** Buttons for "Read", "Write (dummy register)", "Write to file", and "Read from file".
- Continuous Poll:** A section with an "enabled" checkbox, "Poll Frequency" (radio buttons for 1 Hz and 100 Hz), "Last poll time" (2022-12-02T16:42:47.393), and "Avg. update interval".
- Device status:** A panel at the bottom left showing "Device is open." with a "Close" button.
- Device properties:** A panel at the bottom left showing fields for "Device name" (ADC_BOARD), "Device identifier" (dummy?map=adc.mapp), and "Map file".
- Find Module:** A search bar and checkboxes for "Autoselect previous register" and "Sort Modules/Registers".
- Buttons:** "Load Boards", "Collapse All", and "Expand All" buttons are located at the bottom.



The screenshot shows the QtHardMon application window with the following components:

- Menu Bar:** File, Plugins, Settings, Help.
- Left Panel:** A tree view showing the hierarchy of modules and registers. The 'ADC_BOARD' device is selected, and the 'PRJ_VERSION' register is highlighted.
- Register properties:** A panel displaying details for the selected register, including its path, dimension, data type, and numerical address.
- Options:** A panel with checkboxes for 'Read after write' and 'Show plot window'.
- Operations:** A panel with buttons for 'Read', 'Write (dummy register)', 'Write to file', and 'Read from file'.
- Continuous Poll:** A panel with a checkbox for 'enabled', a 'Poll Frequency' section with radio buttons for '1 Hz' and '100 Hz', and a 'Last poll time' field.
- Bottom Panel:** A section for device identification and module finding, including fields for 'Device identifier', 'Map file', and buttons for 'Load Boards', 'Find Module', 'Autoselect previous register', 'Sort Modules/Registers', 'Collapse All', and 'Expand All'.

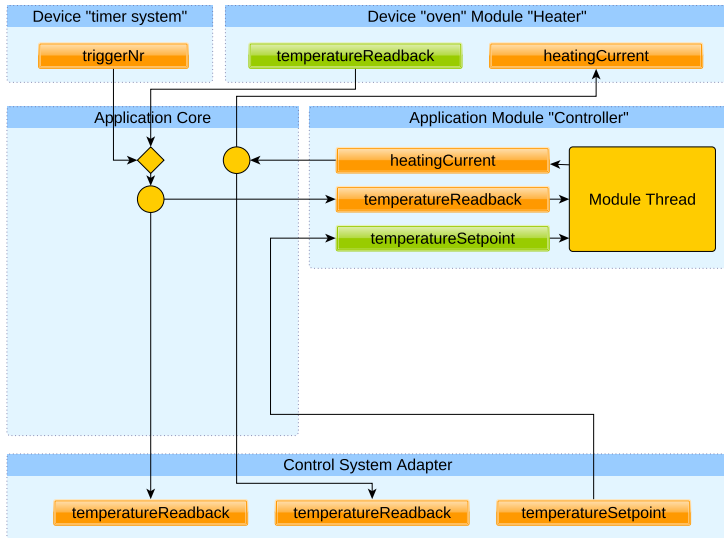
Register properties details:

- Register path: /BSP/PRJ_VERSION
- Dimension: Scalar
- Data Type: Unsigned integer
- wait_for_new_data: no
- Numerical Address: 0
- Fixed Point Interpretation: Register width 32, Fractional bits 0, Signed Flag 0

Values table:

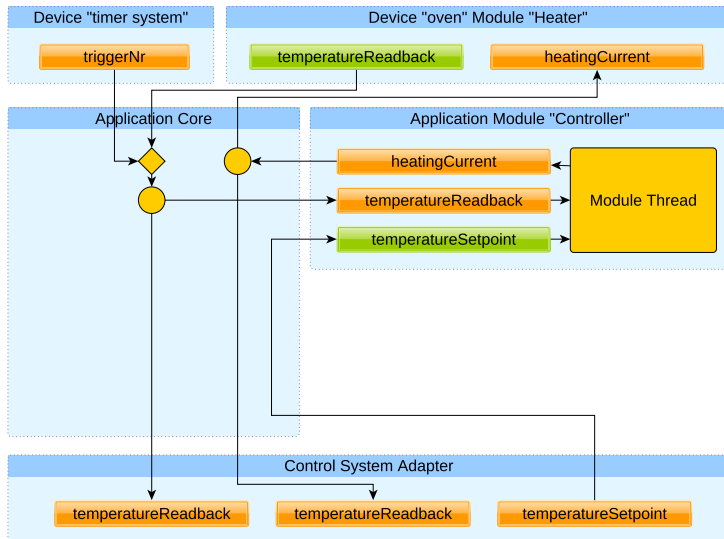
	Value	Raw (dec)	Raw (hex)
0	65538	65538	0x10002

- Devices have a name (needs device name mapping)
- Registers / process variables have a name (might need register address mapping)
- In C++/Python: Objects to represent the process variable



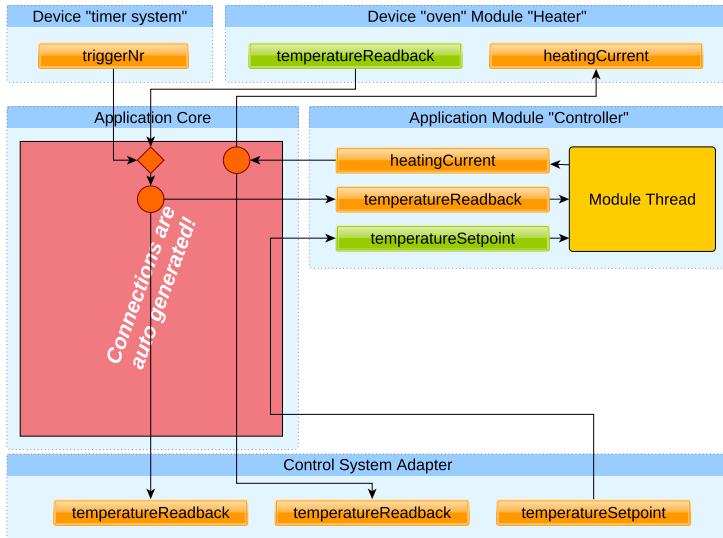
Modules

- Input/output variables
- Application Modules
 - One thread per module
- Special "modules"
 - Device module
 - Control system adapter



Modules

- Input/output variables
- Application Modules
 - One thread per module
- Special "modules"
 - Device module
 - Control system adapter
- Algorithms don't need to know how variables are connected
- Perfect modularity, as modules are self-contained
- Modern multithreading: lock-free communication between modules ("for free")



Modules

- Input/output variables
- Application Modules
 - One thread per module
- Special "modules"
 - Device module
 - Control system adapter
- Algorithms don't need to know how variables are connected
- Perfect modularity, as modules are self-contained
- Modern multithreading: lock-free communication between modules ("for free")
- Connections are auto-generated

Taken from the API dokumentation:

https://chimeratk.github.io/ApplicationCore/master/conceptual_overview.html

Application module

An application module represents a relatively small task of the total application, e.g. one particular computation. The task will be executed in its own thread, so it is well separated from the rest of the application. Ideally, each module should be somewhat self-contained and independent of other modules, and only ApplicationCore process variables should be used for communication with other parts of the application.

An application module is a class deriving from **ChimeraTK::ApplicationModule**, e.g.:

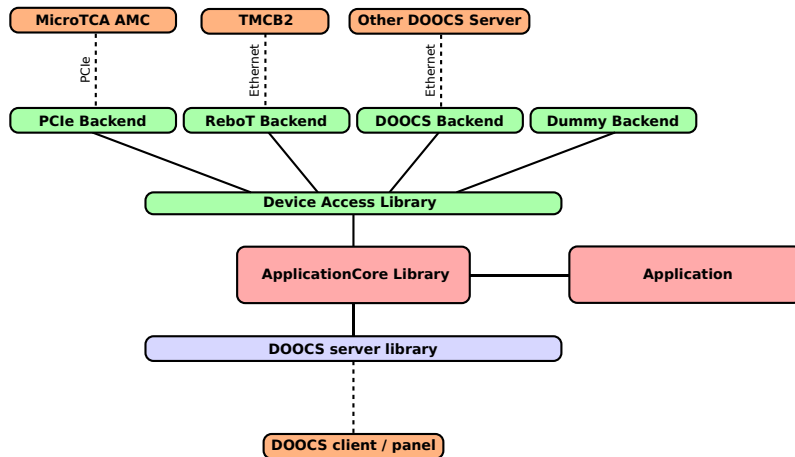
```
19 class Controller : public ctk::ApplicationModule {
20     using ctk::ApplicationModule::ApplicationModule;
21
22     ctk::ScalarPollInput<float> temperatureSetpoint{
23         this, "temperatureSetpoint", "degC", "Setpoint for the temperature controller"};
24     ctk::ScalarPushInput<float> temperatureReadback{
25         this, "temperatureReadback", "degC", "Actual temperature used as controller input"};
26     ctk::ScalarOutput<float> heatingCurrent{this, "heatingCurrent", "mA", "Actuator output of the controller"};
27
28     void mainLoop() override;
29 };
```

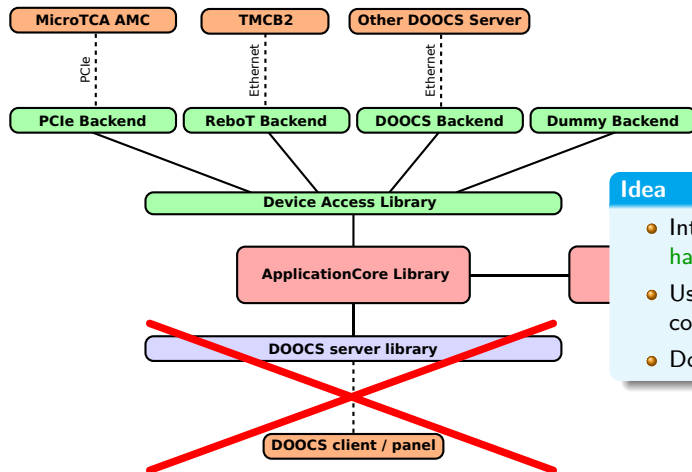
The code processing the data needs to go into the implementation of the `ApplicationModule::mainLoop()` function implementation. In our small example, we implement a simple, fixed-gain proportional controller like this:

```
18 void Controller::mainLoop() {  
19     const float gain = 100.0F;  
20     while(true) {  
21         heatingCurrent = gain * (temperatureSetpoint - temperatureReadback);  
22         writeAll(); // writes any outputs  
23         readAll(); // waits until temperatureReadback updated, then reads temperatureSetpoint  
24     }  
25 }  
26 }
```

The `ApplicationModule::mainLoop()` function literally needs to contain a loop, which runs for the lifetime of the application. Any preparations which need to be executed once at application start should go before the start of the loop.

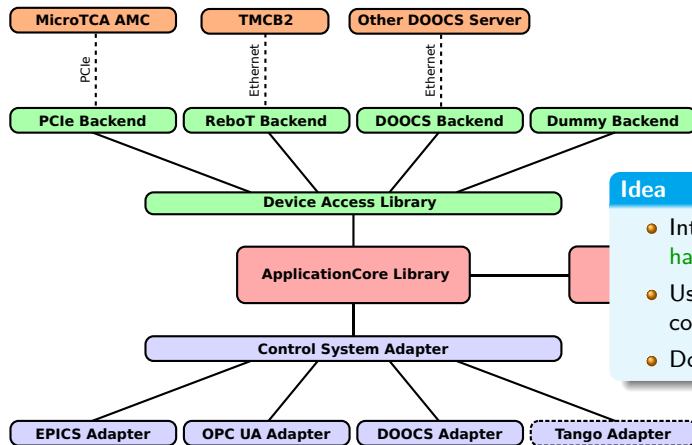
Fully functional code from a working example!





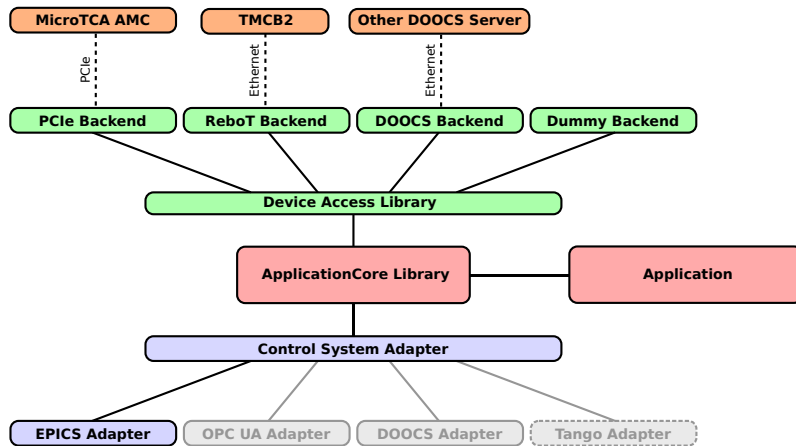
Idea

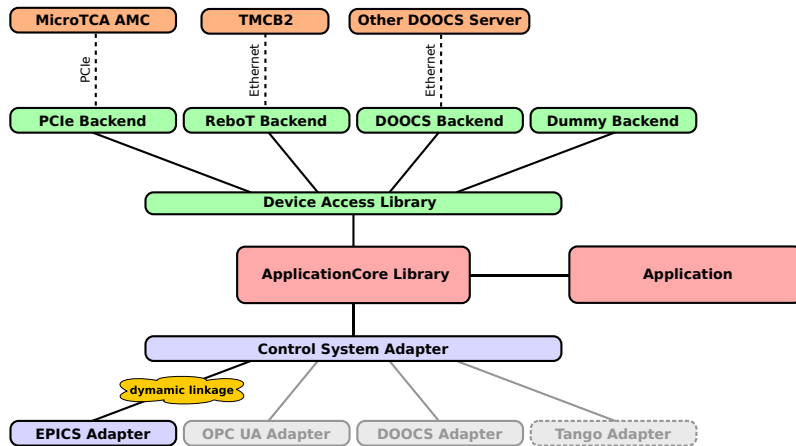
- Introduce similar abstraction as for the **hardware access**
- Use ChimeraTK applications with various control system middlewares
- Do **not** create a new control system!

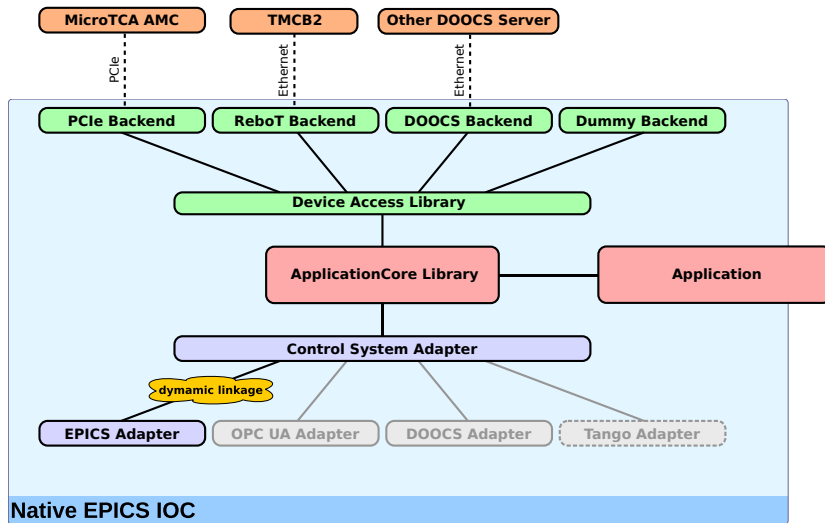


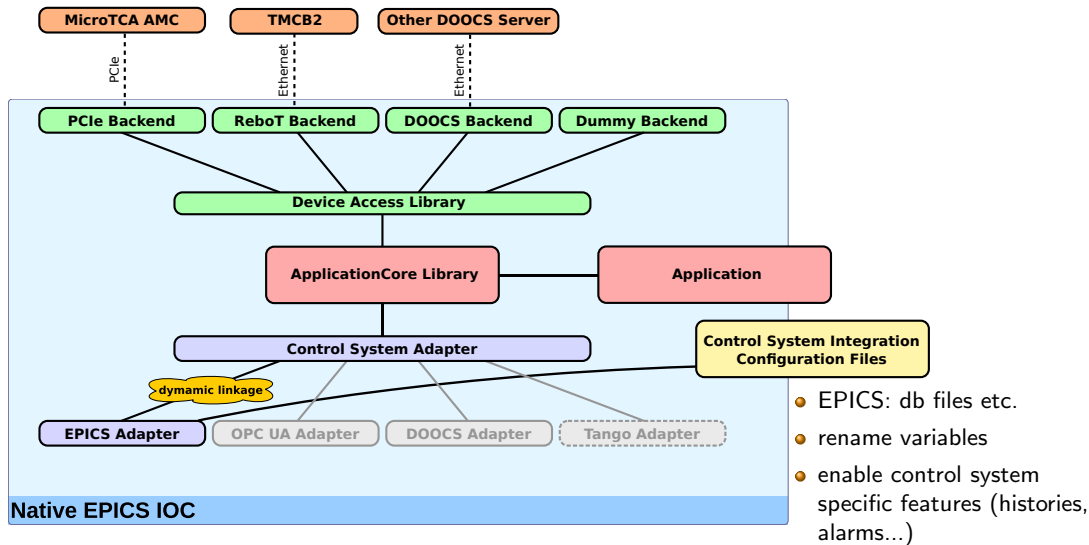
Idea

- Introduce similar abstraction as for the **hardware access**
- Use ChimeraTK applications with various control system middlewares
- Do **not** create a new control system!









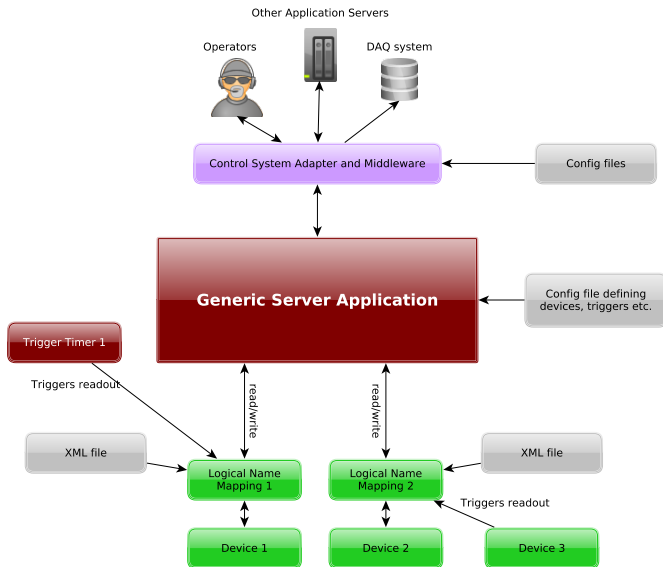
- Motivation: Simplify plain control system integration of devices
- No C++ (or Python) coding necessary, ready-to-use binary just needs configuration

- Motivation: Simplify plain control system integration of devices
- No C++ (or Python) coding necessary, ready-to-use binary just needs configuration
- Publish all device registers as control system variables
- Configurable: multiple devices, readout trigger sources etc.

- Motivation: Simplify plain control system integration of devices
- No C++ (or Python) coding necessary, ready-to-use binary just needs configuration
- Publish all device registers as control system variables
- Configurable: multiple devices, readout trigger sources etc.
- Standard ApplicationCore functionality: good starting point for new applications, complex computations can be added later

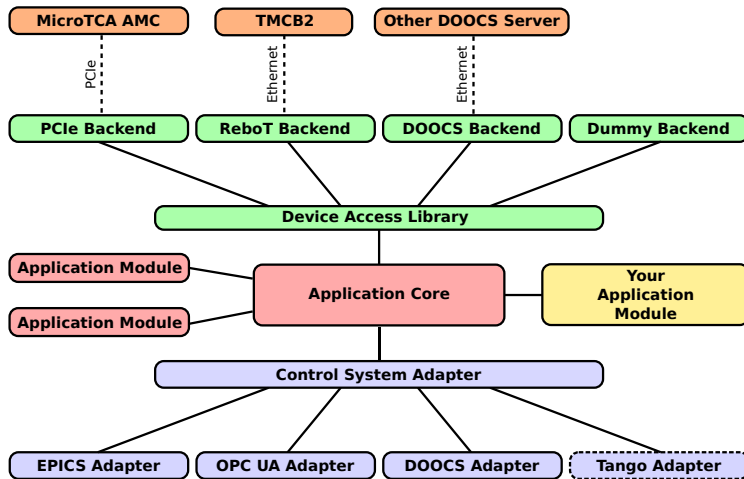
- Motivation: Simplify plain control system integration of devices
- No C++ (or Python) coding necessary, ready-to-use binary just needs configuration
- Publish all device registers as control system variables
- Configurable: multiple devices, readout trigger sources etc.
- Standard ApplicationCore functionality: good starting point for new applications, complex computations can be added later
- Typically used with “logical name mapping device”
 - Device-side mapping layer in XML format
 - Rename registers
 - Extract array elements, split bit fields etc.
 - Apply simple math e.g. to convert in physical units etc.

- Motivation: Simplify plain control system integration of devices
- No C++ (or Python) coding necessary, ready-to-use binary just needs configuration
- Publish all device registers as control system variables
- Configurable: multiple devices, readout trigger sources etc.
- Standard ApplicationCore functionality: good starting point for new applications, complex computations can be added later
- Typically used with “logical name mapping device”
 - Device-side mapping layer in XML format
 - Rename registers
 - Extract array elements, split bit fields etc.
 - Apply simple math e.g. to convert in physical units etc.
- Future plan: plug-in ApplicationModules written in Python



Idea:

- Write individual ApplicationModules in Python
 - Syntax similar to C++
 - Inputs and output
 - Run the Python ApplicationModules in addition to C++ modules
 - Let ApplicationCore do the connection to DeviceAccess, ControlSystem and other ApplicationModules
 - Run the Python ApplicationModules in the Generic Server
- Implementation is planned for next year
 - Should we increase the priority?





Software Repositories

All software is published under the GNU GPL or the GNU LGPL.

- ChimeraTK source code: <https://github.com/ChimeraTK>
- Ubuntu 20.04 and 24.04 packages are available in the [DESY DOOCS repository](#).

Documentation and Tutorials

- API documentation <https://chimeratk.github.io/>
- Tutorials on the [MicroTCA Workshop Indico page](#)
- e-mail support: chimeratk-support@desy.de

Backup.

```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```

```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```

```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```

Complete working example

- No device/protocol specific code (implementation details)

```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```

Complete working example

- No device/protocol specific code (implementation details)

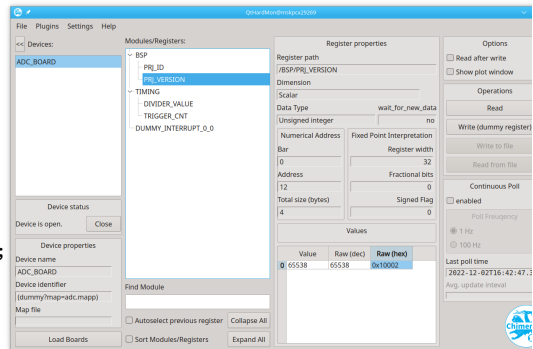
```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```



Complete working example

- No device/protocol specific code (implementation details)

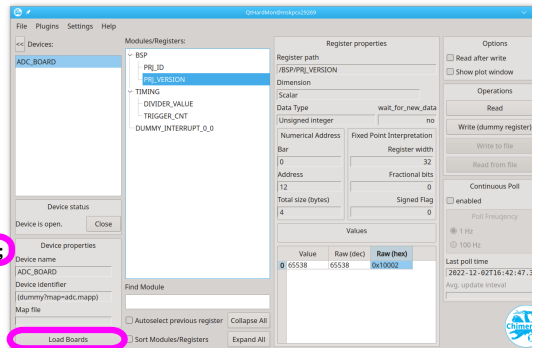
```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```



Complete working example

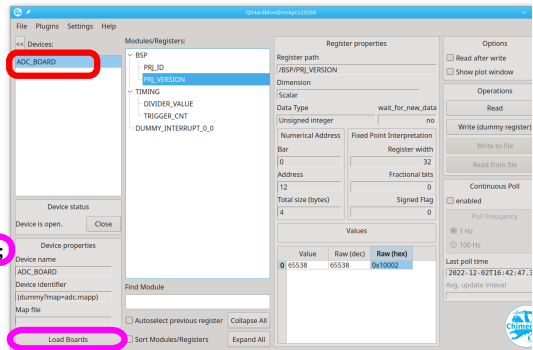
- No device/protocol specific code (implementation details)

```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");
    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```



Complete working example

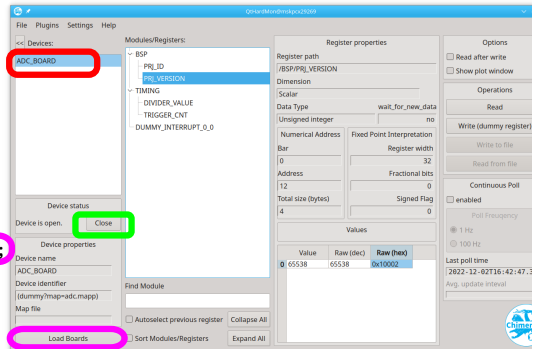
- No device/protocol specific code (implementation details)

```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");
    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```



Complete working example

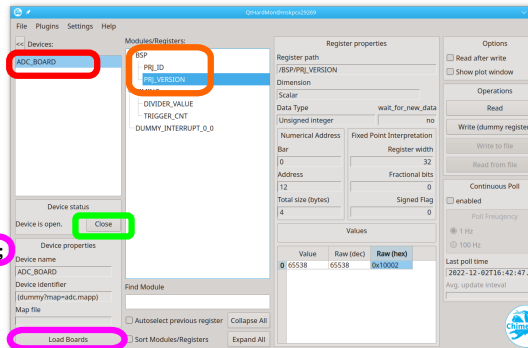
- No device/protocol specific code (implementation details)

```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");
    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```



Complete working example

- No device/protocol specific code (implementation details)

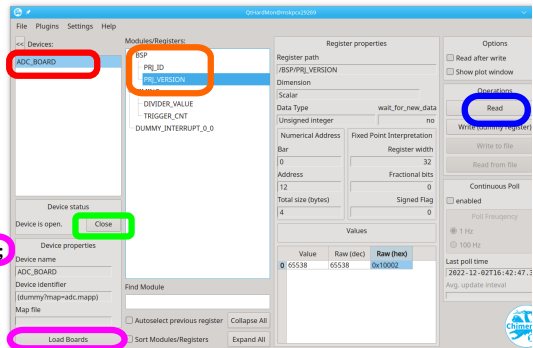
```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```



Complete working example

- No device/protocol specific code (implementation details)

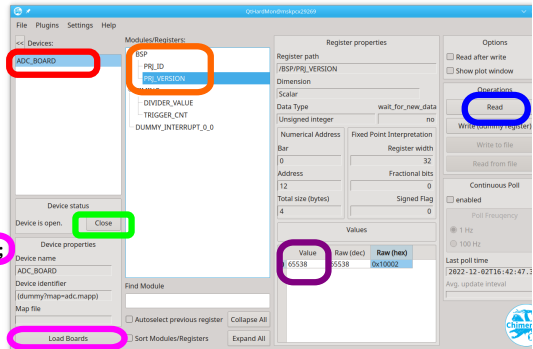
```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```



NEW! Python bindings have a syntax similar to C++

```
import deviceaccess as da
import numpy as np

# Setup section
da.setDMapFilePath('devices.dmap')
d = da.Device('ADC_BOARD')
d.open ()

# For technical reasons ScalarRegisterAccessor inherits from np.ndarray
projectVersion = d.getScalarRegisterAccessor(np.uint32, 'BSP/PRJ_VERSION')

#Business logic
projectVersion.read ();
if projectVersion[0] > 0x010000 :
    print('Compatible version ' + hex(projectVersion[0]))
```

Arrange the register content to logically match your application

- Rename registers
- Add constant registers or dummy registers
- Extract channels from 2D registers and give it a name
- Extract scalars from 1D registers and give it a name
- Extract bits from a scalar register

Math plugin

- Apply conversion formulas, e.g. ADC counts to physical units