ChimeraTK.
ChimeraTK ApplicationCore in a nutshell.

Martin Killenberg
Martin Hierholzer

20th June 2024

Accelerator Middle Layer Workshop
DESY, Hamburg

ChimeraTK — **C**ontrol system and **H**ardware **I**nterface 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 — **C**ontrol system and **H**ardware **I**nterface 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

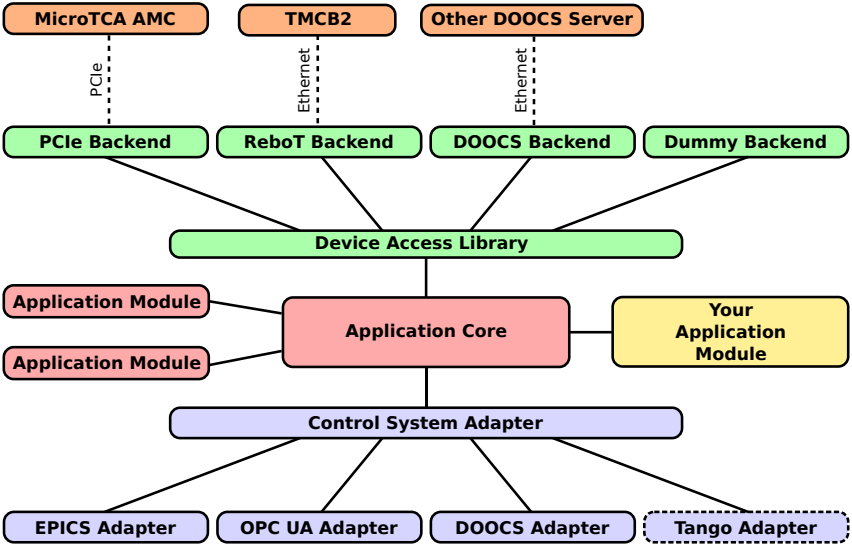
Task 2

- Run the business logic / algorithm
- ChimeraTK ApplicationCore

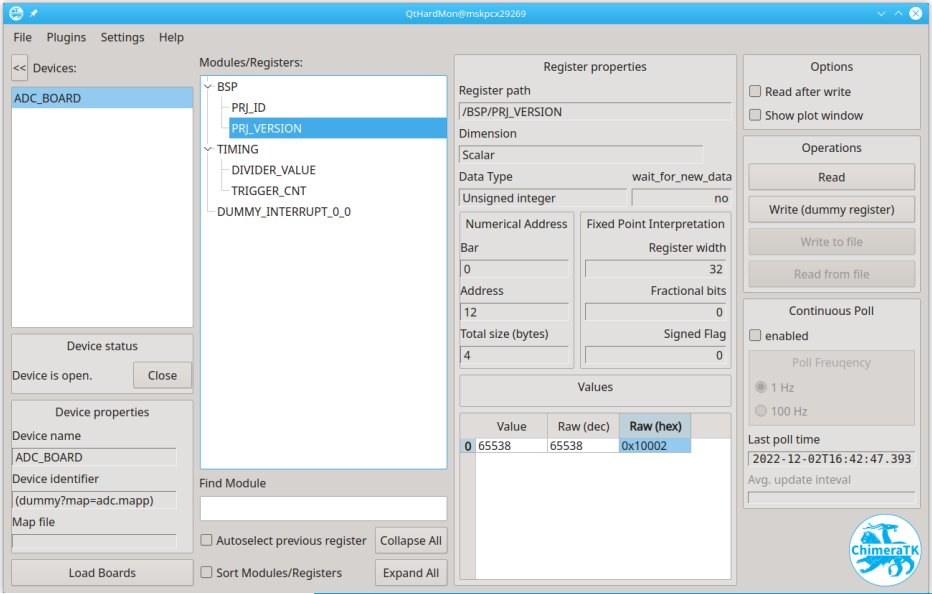
Task 3

- Interface to the control system
- ChimeraTK ControlSystemAdapter

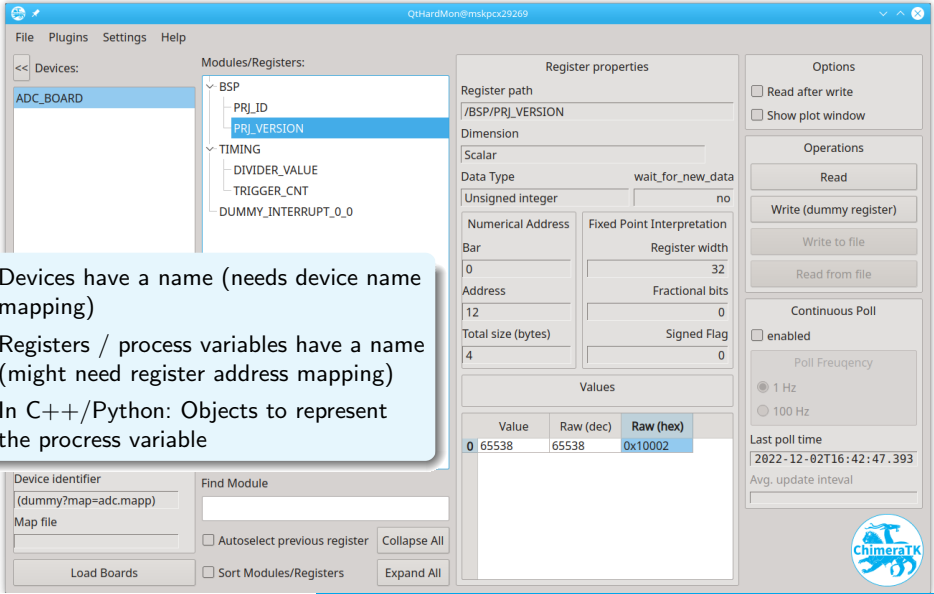
ChimeraTK Overview



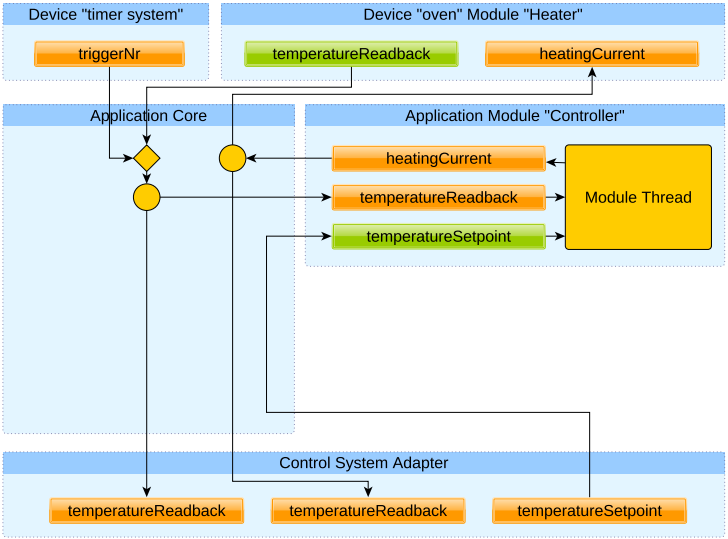
QtHardMon: A graphical user interface for DeviceAccess



QtHardMon: A graphical user interface for DeviceAccess



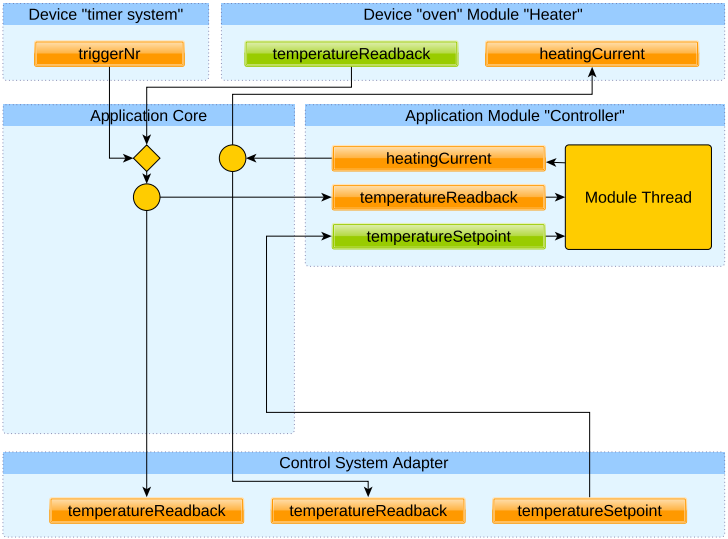
- Devices have a name (needs device name mapping)
- Registers / process variables have a name (might need register address mapping)
- In C++/Python: Objects to represent the procress variable



Modules

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

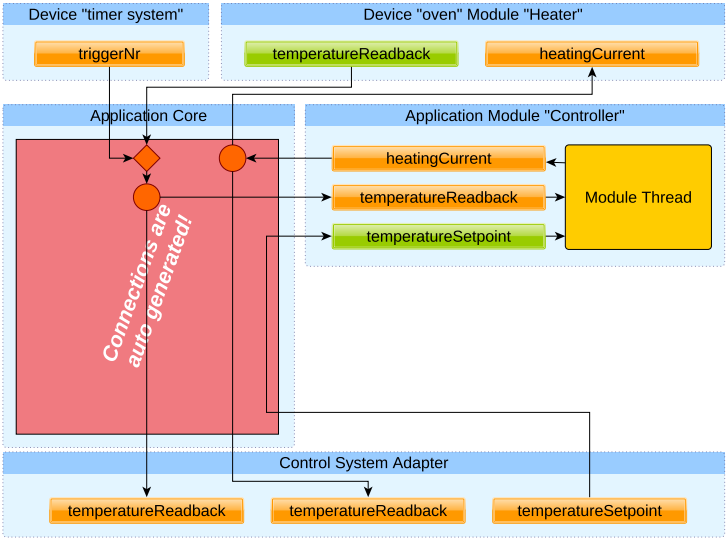
ApplicationCore



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")

ApplicationCore



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-genegated

Writing an ApplicationModule (Header Example)

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 };
```

Writing an ApplicationModule (Main Loop)

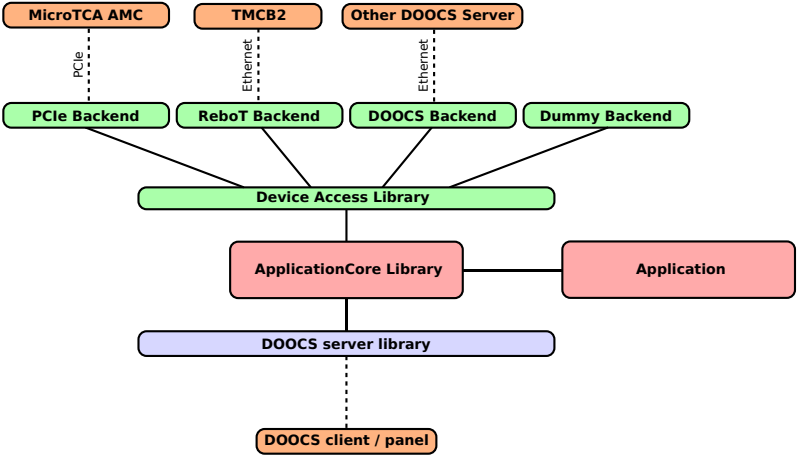
The code processing the data needs to go into the implementation of the ApplicationModule::mainLoop() function implementation. In our small example, we implement a simple, fixed-gain proportional controller like this:

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

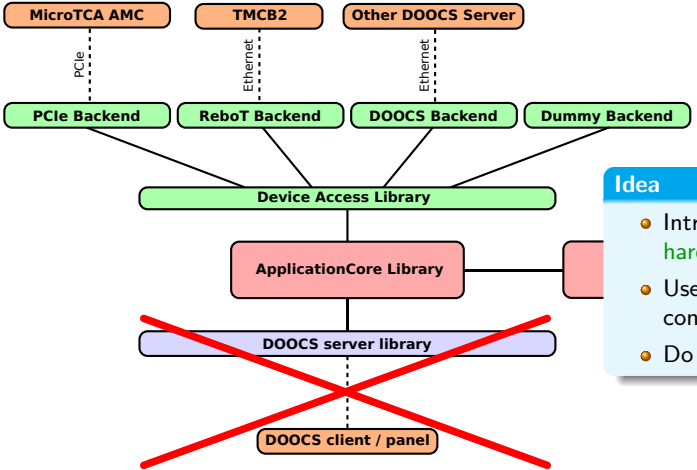
The ApplicationModule::mainLoop() function litteraly needs to contain a loop, which runs for the lifetime of the application. Any preparations which need to be executed once at application start should go before the start of the loop.

Fully functional code from a working example!

Control system integration



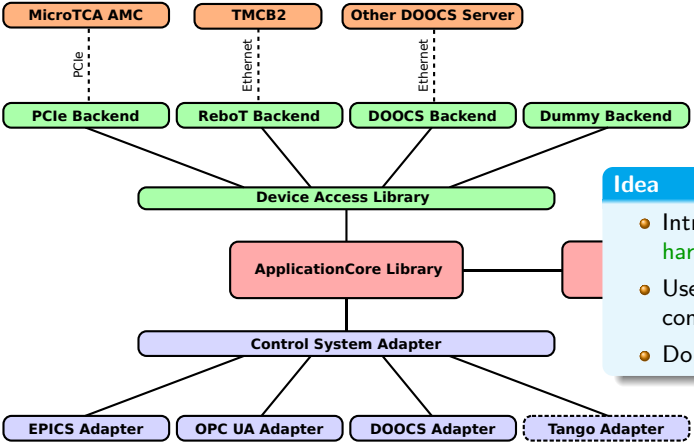
Control system integration



Idea

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

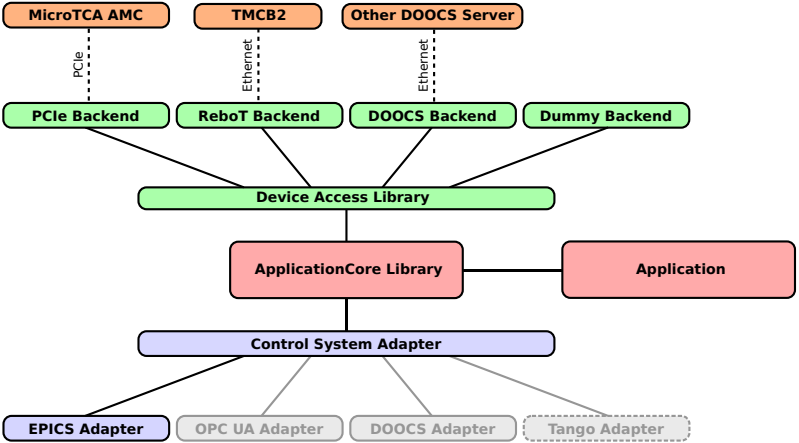
Control system integration



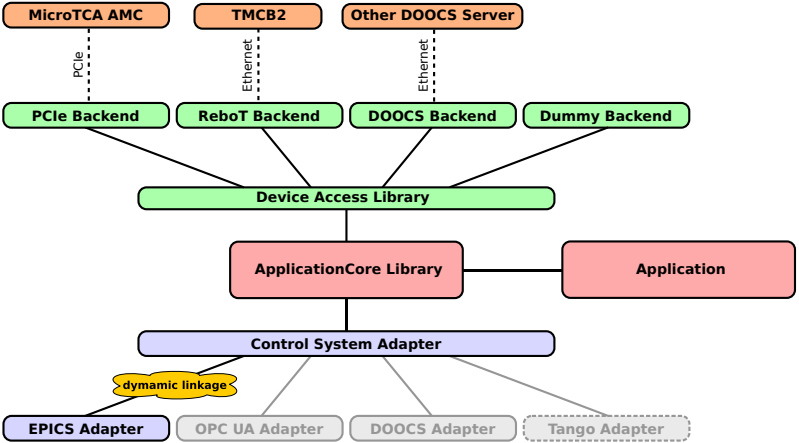
Idea

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

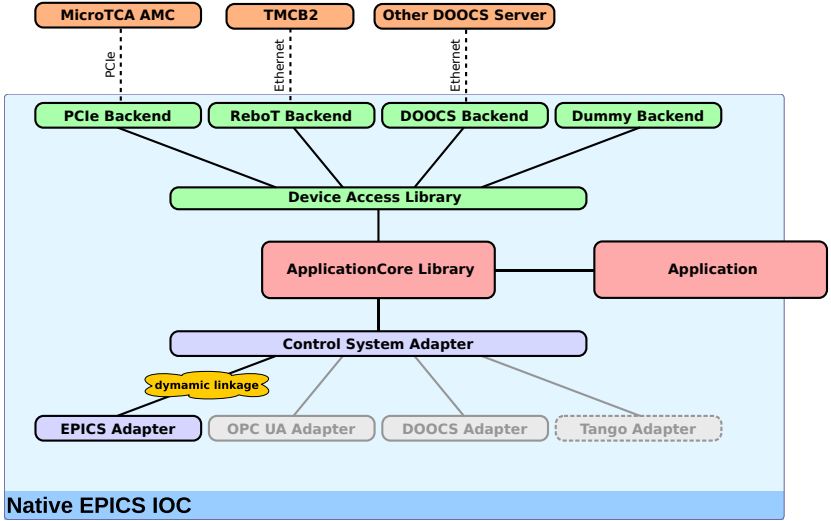
Control system integration



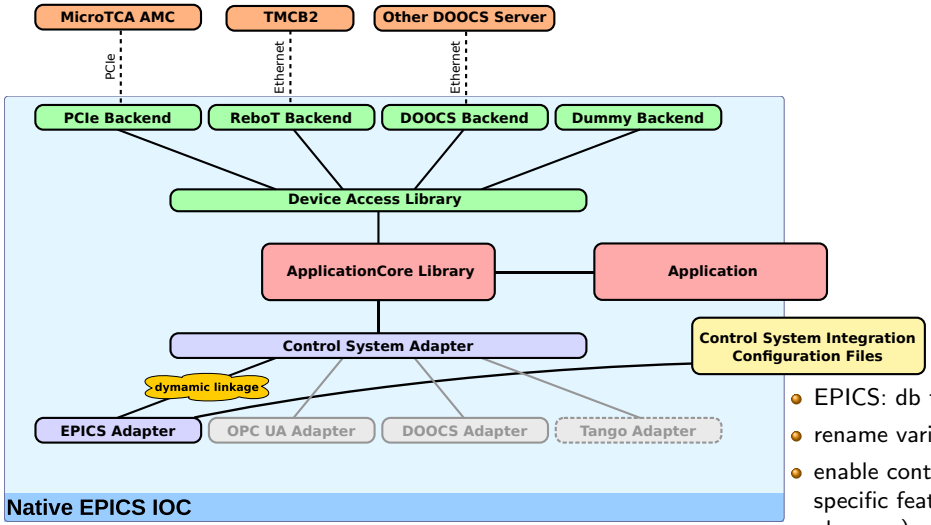
Control system integration



Control system integration



Control system integration



- EPICS: db files etc.
- rename variables
- enable control system specific features (histories, alarms...)

Generic Server Application

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

Generic Server Application

- 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.

Generic Server Application

- 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

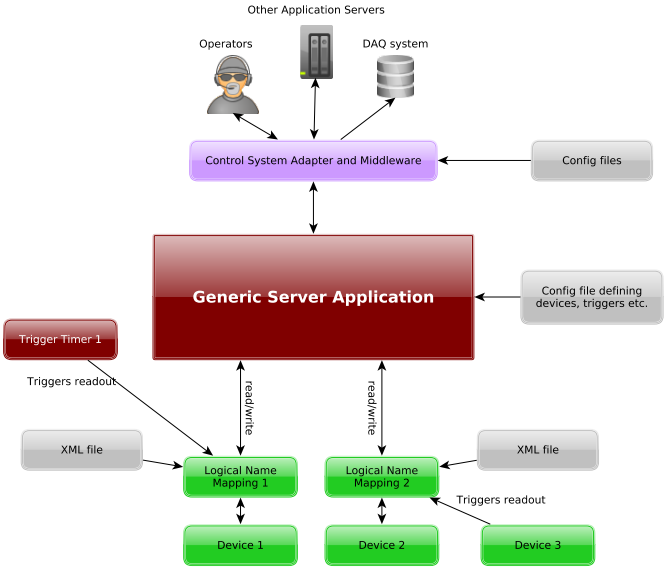
Generic Server Application

- 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.

Generic Server Application

- 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

Generic Server Application



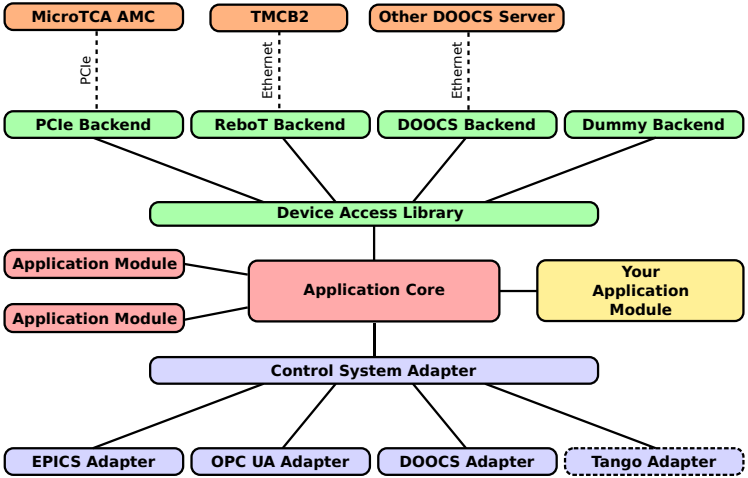
Planned: Python Bindings for ApplicationCore

Idea:

- Write individual ApplicationModules in Python
  - Syntax similar to C++
  - Inputs and output
- Run the Python ApplicationModules in addidion 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?

Summary


MicroTCA AMC
TMCB2
Other DOOCS Server
PCIe
Ethernet
Ethernet
PCIe Backend
ReboT Backend
DOOCS Backend
Dummy Backend
Device Access Library
Application Module
Application Module
Application Core
Your Application Module
Control System Adapter
EPICS Adapter
OPC UA Adapter
DOOCS Adapter
Tango Adapter

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.

Example C++ code

```cpp
#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;
  }
}
```

Example C++ code

```cpp
#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;
  }
}
```

Example C++ code

```cpp
#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;
  }
}
```

Example C++ code

Complete working example

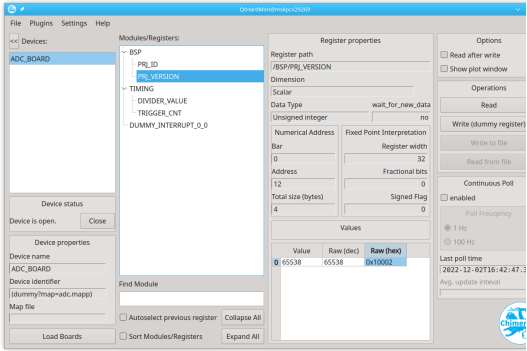
- No device/protocol specific code (implementation details)

```
#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;
  }
}
```

Example C++ code

Complete working example

- No device/protocol specific code (implementation details)

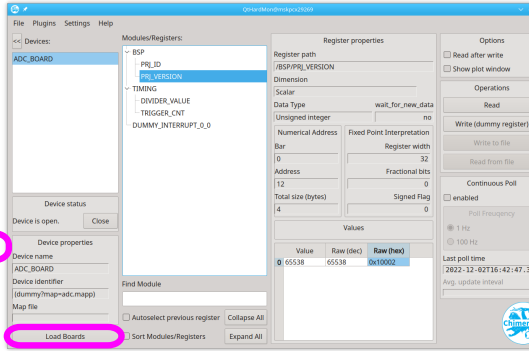



```
#include <iostream>
#include <ChimeraTK/Device.h>

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;
  }
}
```

Example C++ code

Complete working example

- No device/protocol specific code (implementation details)

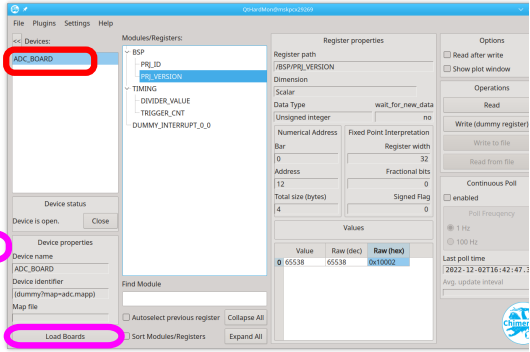



```
#include <iostream>
#include <ChimeraTK/Device.h>

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;
  }
}
```

Example C++ code

Complete working example

- No device/protocol specific code (implementation details)

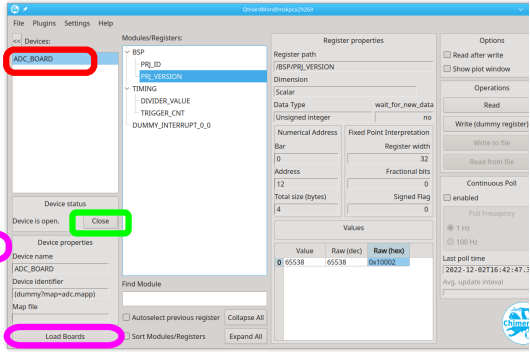



```
#include <iostream>
#include <ChimeraTK/Device.h>

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;
  }
}
```

Example C++ code

Complete working example

- No device/protocol specific code (implementation details)

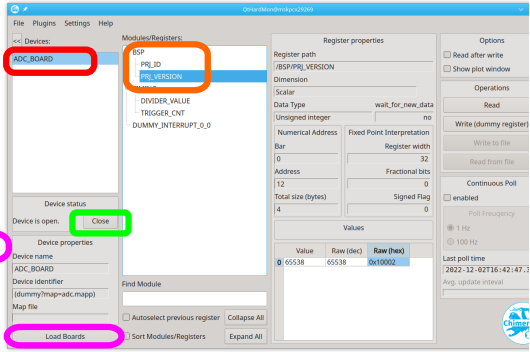



```
#include <iostream>
#include <ChimeraTK/Device.h>

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;
  }
}
```

Example C++ code

Complete working example

- No device/protocol specific code (implementation details)



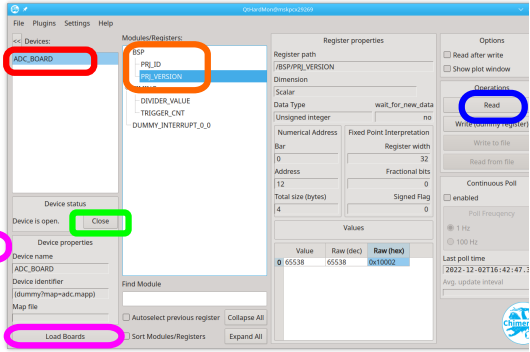

```
#include <iostream>
#include <ChimeraTK/Device.h>

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;
  }
}
```

Example C++ code

Complete working example

- No device/protocol specific code (implementation details)

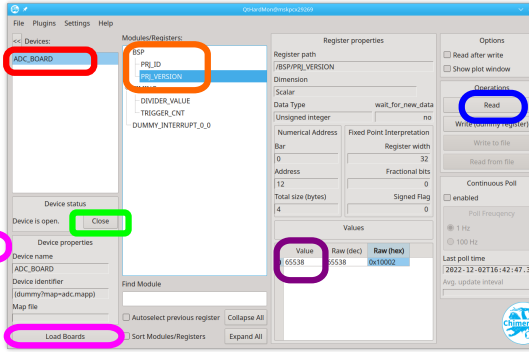
```
#include <iostream>
#include <ChimeraTK/Device.h>

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;
  }
}
```



Example C++ code

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]))
```

What can I do with Logical Name Mapping?

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