

# Device Error Handling in ChimeraTK.

M. Killenberg<sup>a</sup>, J. Timm<sup>b</sup>, J. Georg, M. Hierholzer, C. Kampmeyer, T. Kozak, N. Shehzad, G. Varghese  
(Deutsches Elektronen-Synchrotron DESY, Hamburg)



<sup>a</sup>Primary author  
<sup>b</sup>Presenter

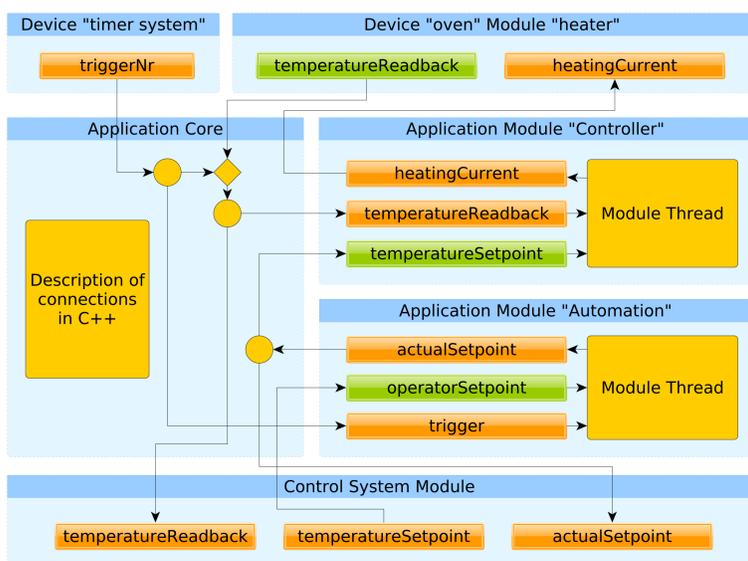
## Motivation

- Large fractions of code in control system applications are error handling
  - Many errors are device errors, often I/O errors
  - Error handling usually is similar
    - Report the error to the control system
    - Wait until the error condition is resolved
    - Resume normal operation
- ⇒ Lots of concepts and code are copied across applications

## Goal

- Handle device errors in the framework
  - Business logic should not have to deal with device errors (it can just read and write)
- ⇒ Shorter and cleaner application code  
⇒ Sophisticated error handling is available out of the box

## ApplicationCore



## Device Error Handling

### ProcessVariable::read()

Each process variable has its own exception detection and reporting.

### backend.read()

The backend provides the implementation which performs the read or write operations. In case of I/O errors or communication problems it will raise a `runtime_error` exception, which is caught in the process variable and reported to the device module thread.

### Waiting for recovery

After reporting the error, the `read()` or `write()` function will wait for a message from the device module that the backend is operational again. It will then retry the `read()/write()` such that the action eventually is performed correctly. This automatically will block the calling application module until the device is available again. No further handling is required in the business logic.

## Resources

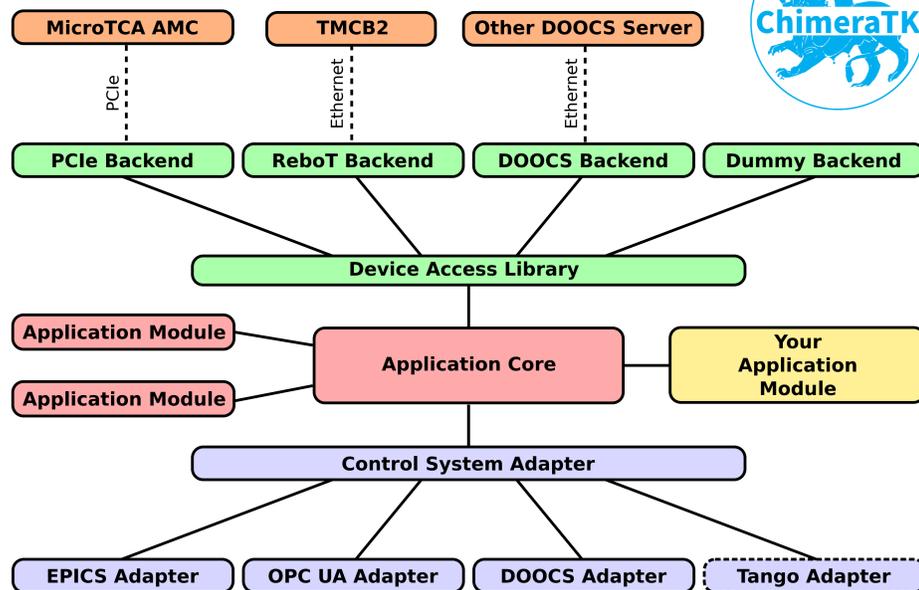
- ChimeraTK source code: <https://github.com/ChimeraTK/>
- Ubuntu 16.04 packages are available in the [DESY DOOCS repository](#)

All ChimeraTK components are published under the GNU GPL or the GNU LGPL.

- API documentation: <https://chimeratk.github.io/>
- Ask us for tutorials and live demos
- e-mail support: [chimeratk-support@desy.de](mailto:chimeratk-support@desy.de)

## ChimeraTK

A tool kit for creating control applications



### DeviceAccess

- Access to hardware, control system servers and dummies
- Backends implement different protocols
- Plugin mechanism: Add new backends at run time

### ApplicationCore

- Library to write modular control applications

### ControlSystemAdapter

- Connect applications to various control system middlewares
  - EPICS
  - DOOCS
  - OPC UA
  - *Your control system*

## Code Examples

```
void initialiseOven(ChimeraTK::DeviceModule* oven) {
    // turn on the power of the oven in the initialisation
    oven->device.write<uint32_t>("/power", 1);
}

void Controller::mainLoop() {
    const double gain = 100.0;
    while(true) {
        readAll(); // waits until temperatureReadback has been updated,
                  // then reads temperatureSetpoint

        heatingCurrent = gain * (temperatureSetpoint - temperatureReadback);
        writeAll(); // writes all outputs
    }
};

struct ExampleApp : public ChimeraTK::Application {
    ExampleApp() : Application("demoApp2") {}
    ~ExampleApp() { shutdown(); }

    Controller controller(this, "Controller", "The Controller");

    ChimeraTK::PeriodicTrigger timer(this, "Timer",
                                     "Periodic timer for the controller", 1000);

    // Instantiate the oven with an initialisation function
    ChimeraTK::DeviceModule oven(this, "oven", &initialiseOven);
    ChimeraTK::ControlSystemModule cs;

    void defineConnections();
};
```

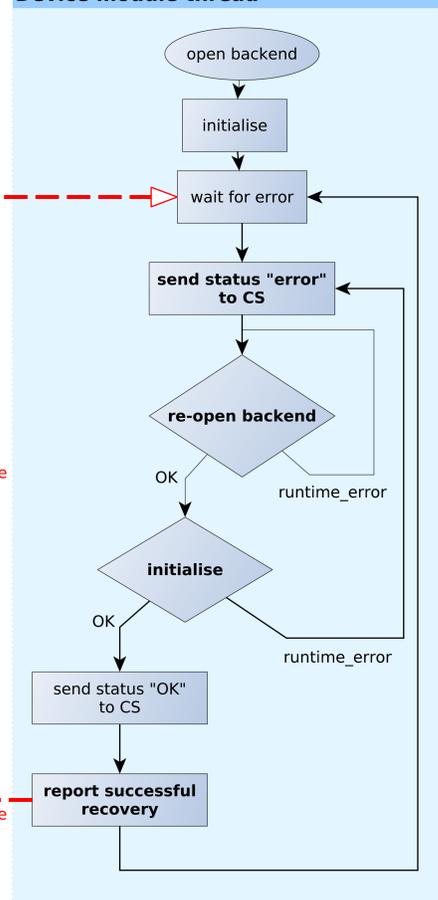
### Application modules

- Input/output variables
- One thread per module
- Special modules
  - Device module
  - Control system module

### Connection code

- Connect application modules
- Triggering
  - Read multiple variables synchronously
  - Synchronise application modules to HW trigger

### Device module thread



### Device module thread

The device module has a thread which first opens the backend and then waits for errors to be reported from the process variables.

### Send status "error" to CS

When an error is reported, the device module automatically updates the device status that is shown in the control system.

### Re-open and initialise

The device module tries to re-open the backend until it succeeds. The backend knows the exact actions that are needed. After the backend could be opened successfully, the device module tries to run the initialisation procedure again. The necessary initialisation sequences depend on the hardware/firmware and are registered to the device module by the application. In case the initialisation fails, the device module updates the error message and tries to re-open the backend again.

### Reporting success

After successful initialisation, the device module reports to the control system that the device is functional again, and then notifies all blocked process variables that they can resume operation.

