A new major release of ChimeraTK ApplicationCore and DeviceAccess.

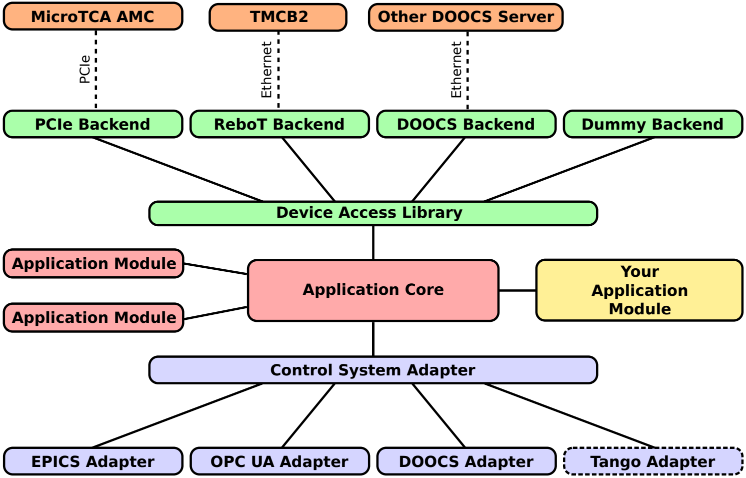
Martin Killenberg

Jens Georg, Martin Hierholzer, Christoph Kampmeyer, Tomasz Kozak,
Nadeem Shehzad, Jan Timm, Geogin Varghese

3rd December 2020

9th MicroTCA Workshop for Industry and Research
Virtual Workshop Hosted by DESY, Hamburg

ChimeraTK Overview



DeviceAccess

- Abstract access to different hardware
- Extensible backend interface

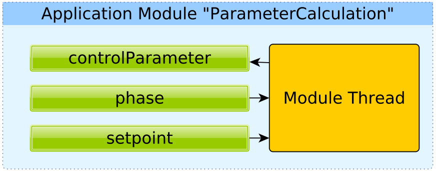
ApplicationCore

- ApplicationModules implement application logic
- Multi-threaded

ControlSystemAdapter

- Abstract interface to different control system middleware
- Integrate via configuration

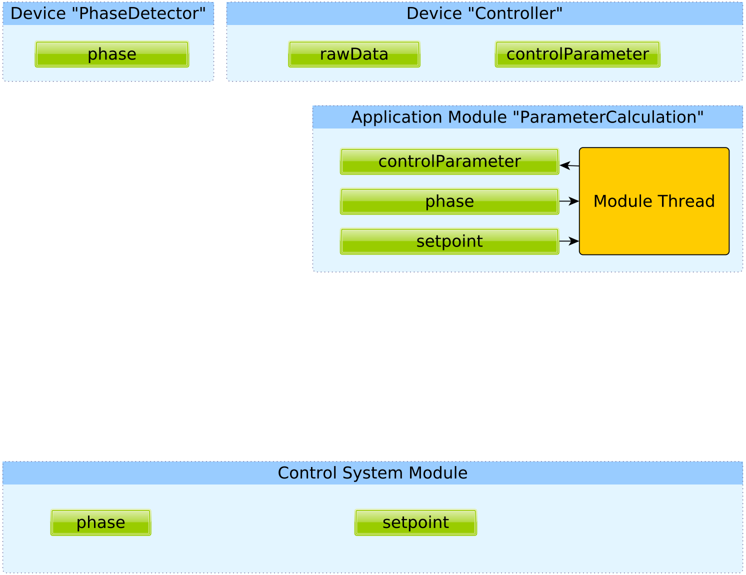
ApplicationCore



Modules

- Input/output variables
- Application Modules
  - One thread per module

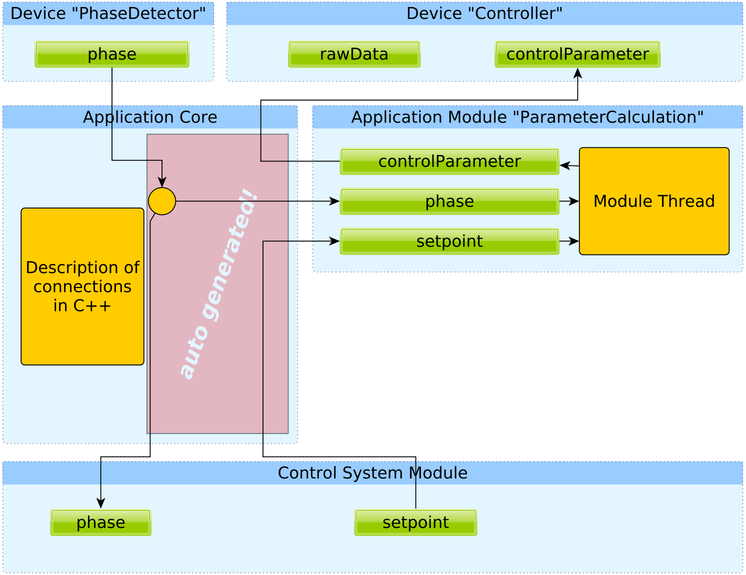
ApplicationCore



Modules

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

ApplicationCore



Modules

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

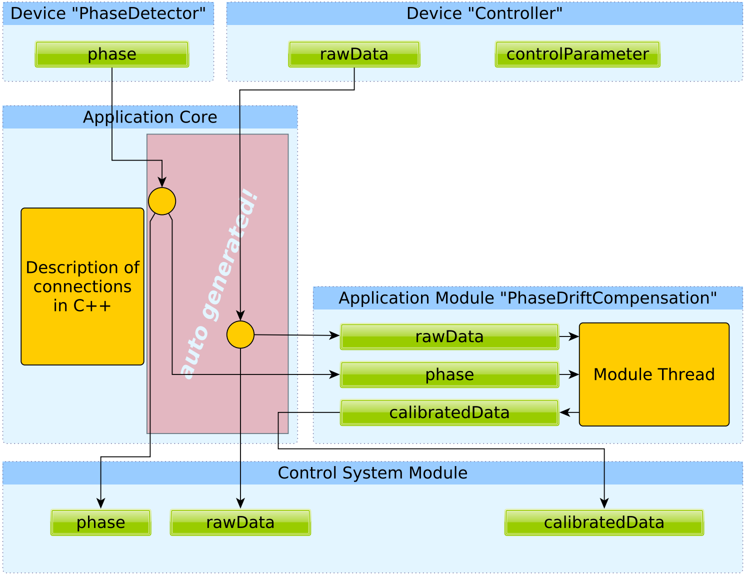
Connections

- Mostly auto-generated

High locality

- Algorithms don't need to know how variables are connected
- Perfect modularity, as modules are self-contained

ApplicationCore



Modules

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

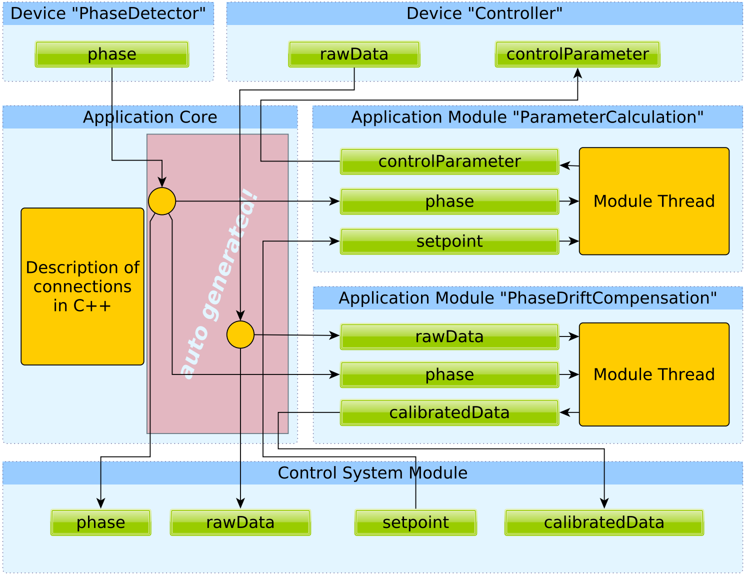
Connections

- Mostly auto-generated

High locality

- Algorithms don't need to know how variables are connected
- Perfect modularity, as modules are self-contained

ApplicationCore



Modules

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

Connections

- Mostly auto-generated

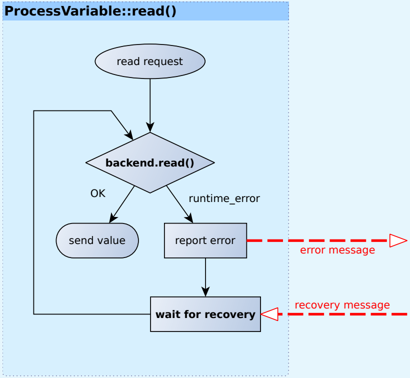
High locality

- Algorithms don't need to know how variables are connected
- Perfect modularity, as modules are self-contained

Device Error Handling

- Modules can just use the input and output process variables
- Frameworks takes care of device opening and error handling

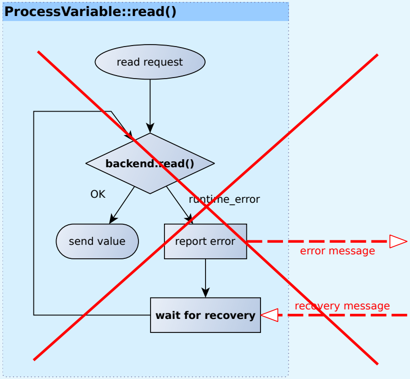
Concept presented on MTCA workshop 2019



Device Error Handling

- Modules can just use the input and output process variables
- Frameworks takes care of device opening and error handling

Concept presented on MTCA workshop 2019



Problems with the proposed solution

- You don't see which PVs are stale in the application
- The `read()` blocks where it should not

⇒ Back to the drawing board

  - Write extensive detailed specification first
  - Match details in DeviceAccess, ApplicationCore and ControlSystemAdapter

Device Error Handling 2.0

Typical example: RF phase

- Basically constant at one operation point
- Slowly drifts with time

Assumption

- When it's not updating I can keep the system running with the old value for a while

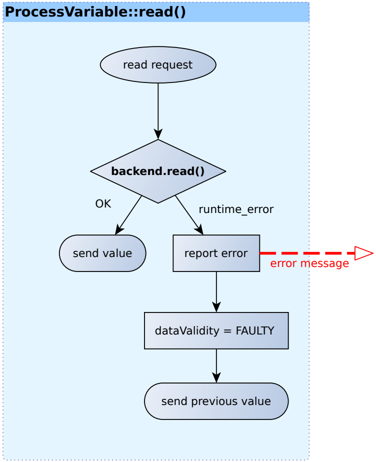
Requirements

- I want to know when it's not updating
- I want to keep the old value

Device Error Handling 2.0

New concept

- Send last good value with `DataValidity` set to `FAULTY`



Typical example: RF phase

- Basically constant at one operation point
- Slowly drifts with time

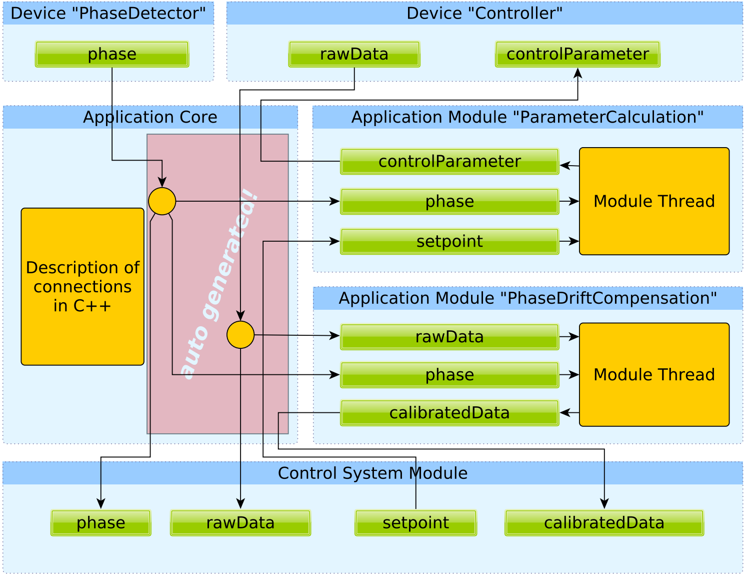
Assumption

- When it's not updating I can keep the system running with the old value for a while

Requirements

- I want to know when it's not updating
- I want to keep the old value

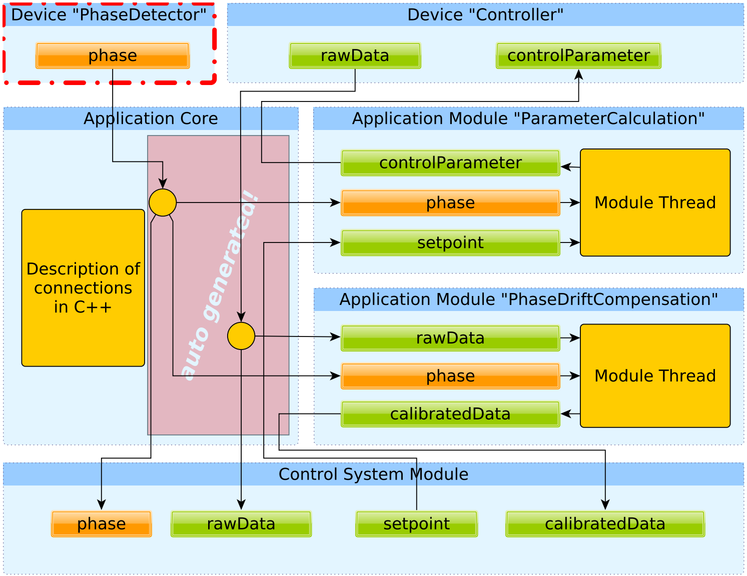
Propagation of the Data Validity Flag



Agenda

- data validity faulty
- data validity OK

Propagation of the Data Validity Flag



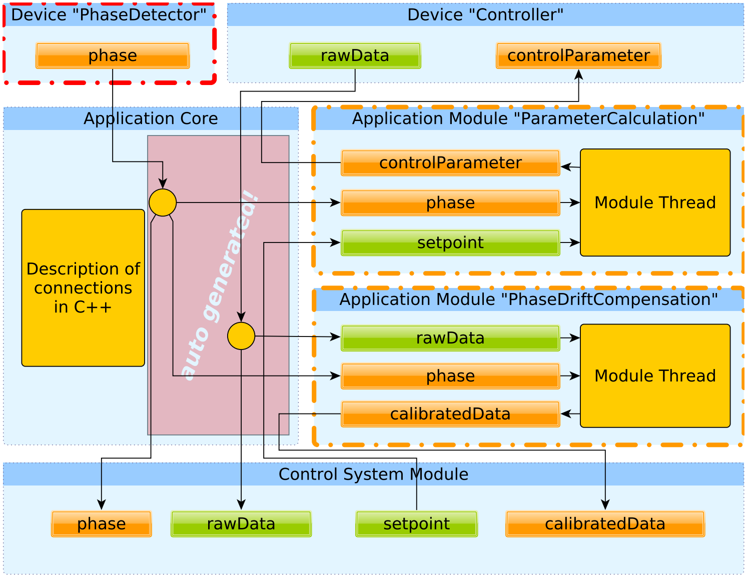
“PhaseDetector” has an error

- All outputs marked as faulty
- Framework tries to re-open

Agenda

- data validity faulty
- data validity OK

Propagation of the Data Validity Flag



"PhaseDetector" has an error

- All outputs marked as faulty
- Framework tries to re-open

Application Modules

- Small with correlated inputs and outputs
- If one input is faulty, all outputs are faulty
- Module stays active

Device Modules

- Uncorrelated inputs and outputs
- `rawData` stays valid

Agenda

🟧 data validity faulty

🟩 data validity OK

Initial Values

Application start

- I don't know the current operation point
- The RF phase can be anything between -180° and +180°
- There is no previous good value

Initial Values

Application start

- I don't know the current operation point
- The RF phase can be anything between -180° and +180°
- There is no previous good value

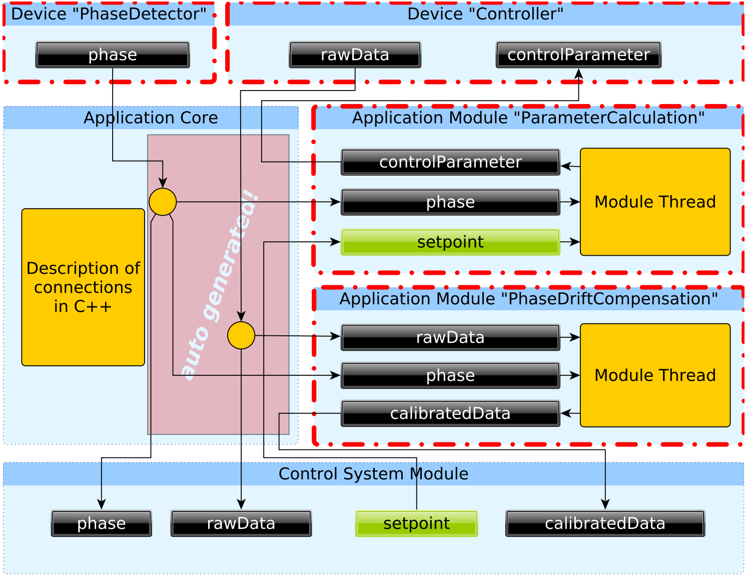
New concept: Initial values

- When constructed, process variables know they have not seen any data yet
- When reading, it blocks until the *initial value* has been received

Advantages

- Allows clean application start
- ApplicationModules only start when they have all the data
  - No special exception handling

Initial Value Propagation



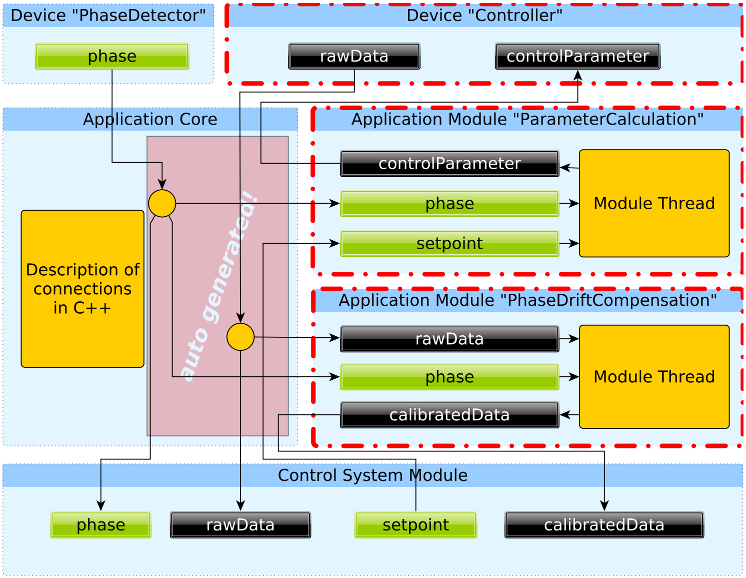
Application start

- Both devices still closed
- Application modules waiting for initial values
- Control system has initial values from persistency layer

Agenda

- no initial value
- data validity faulty
- data validity OK

Initial Value Propagation



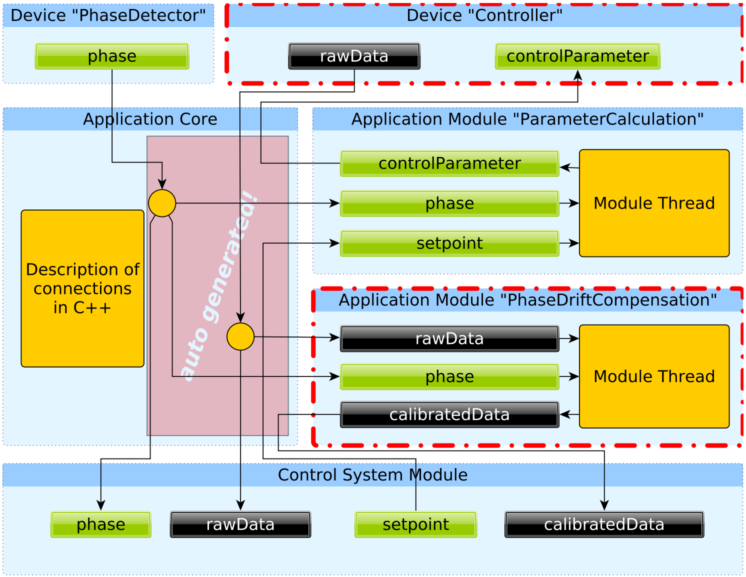
Devices are opening

- “PhaseDetector” opens successfully
- “Controller” still waiting for initial values
- “ParameterCalculation” has all initial values
- “PhaseDriftCompensation” still waiting for `rawData`

Agenda

- no initial value
- data validity faulty
- data validity OK

Initial Value Propagation



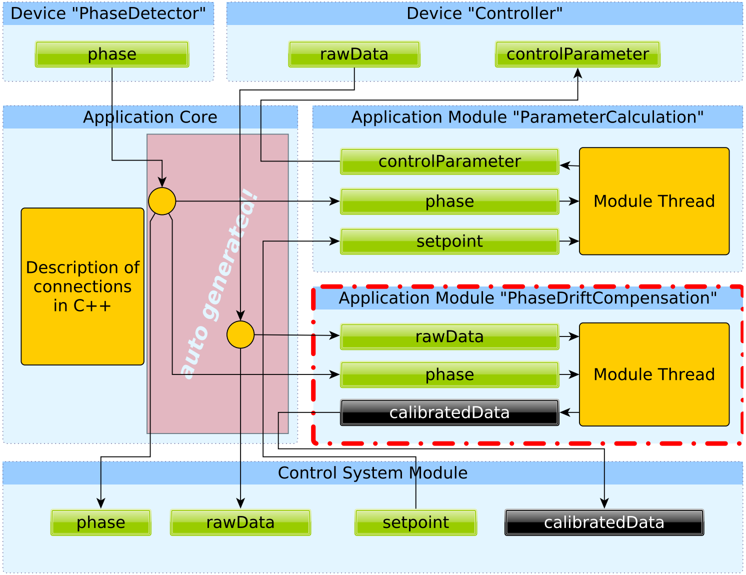
"ParameterCalculation" is starting

- sends `controlParameter` to "Controller"
- "PhaseDriftCompensation" still waiting for `rawData`

Agenda

- no initial value
- data validity faulty
- data validity OK

Initial Value Propagation



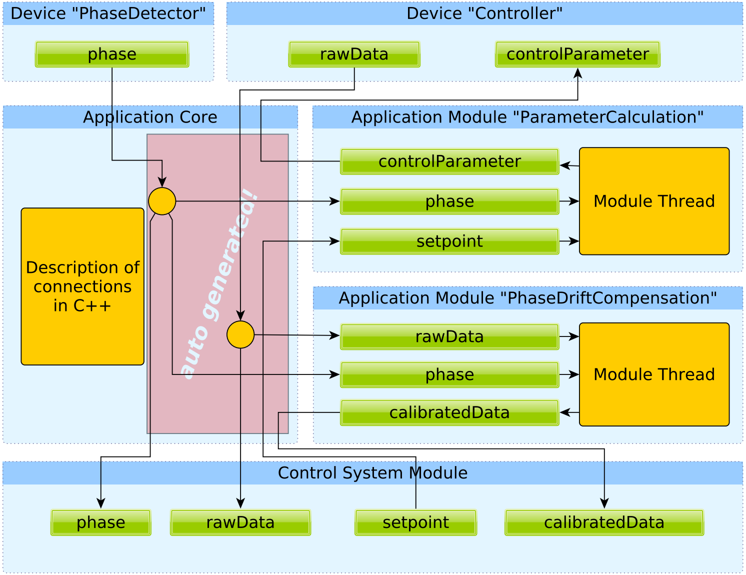
Device “Controller” is fully initialised

- Sends `rawData`
- “PhaseDriftCompensation” has all initial parameters

Agenda

- no initial value
- data validity faulty
- data validity OK

Initial Value Propagation



“PhaseDriftCompensation” is starting

- Sends `calibratedData`
- The application is up and running!

Agenda

- no initial value
- data validity faulty
- data validity OK

Summary

ChimeraTK

- Design modular, multi-threaded applications
- Talk to hardware
- Interface with the control system infrastructure

DeviceAccess 02.02 and ApplicationCore 02.00

- Consistent device exception handling
- Data validity propagation
- Initial value propagation

LLRF software at the European XFEL and FLASH are currently being updated!

Software Repositories

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

- ChimeraTK source code: https://github.com/ChimeraTK
- Ubuntu 20.04 packages are available in the DESY DOOCS repository.

Documentation and Tutorials

- API documentation https://chimeratk.github.io/
- Tutorials on the MicroTCA Workshop 2019 Indico page
- e-mail support: chimeratk-support@desy.de

Backup

Backup

Push Type and Poll Type Variables

Push Type Variables

- Producer/device is actively sending the data
- `read()` is blocking until new data is received

Exception handling

- In case of an error, exactly one exception is send per variable (DeviceAccess layer)
- The exception is caught in ApplicationCore, and the last value is send with dataValidity=Faulty

⇒ `read()` returns once with dataValidity=Faulty

⇒ The next `read()` blocks until the next value after device recovery has been received

Poll Type Variables

- Passive producer
- `read()` is polling the latest value
- `read()` is not blocking

Exception handling

- The exception in the synchronous device access is caught in ApplicationCore
- Each `read()` returns immediately with the last value and dataValidity=Faulty until the device has recovered