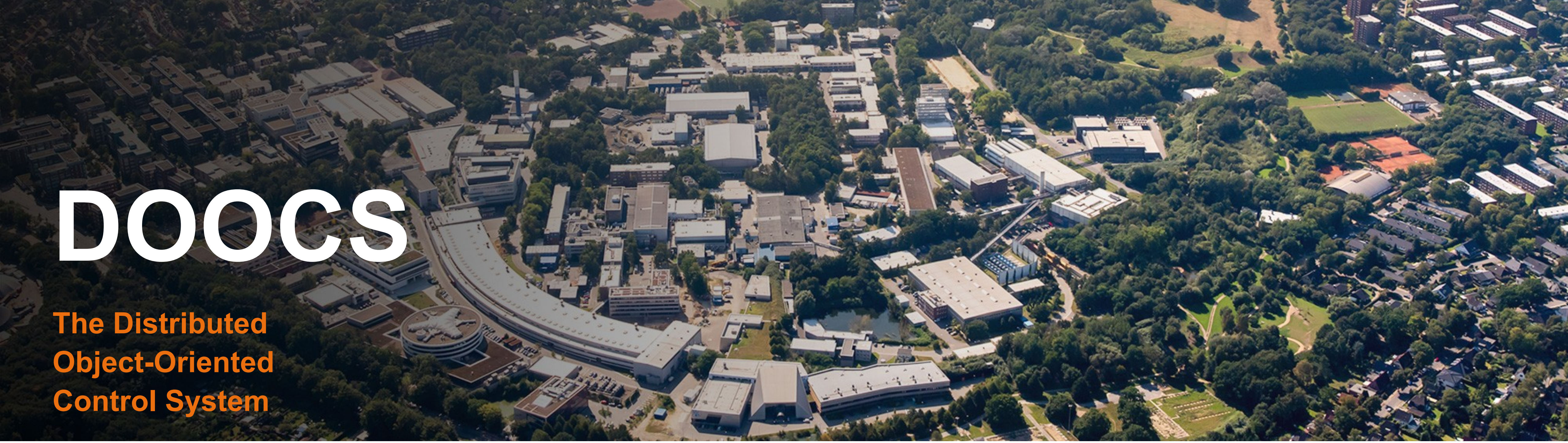


DOOCS

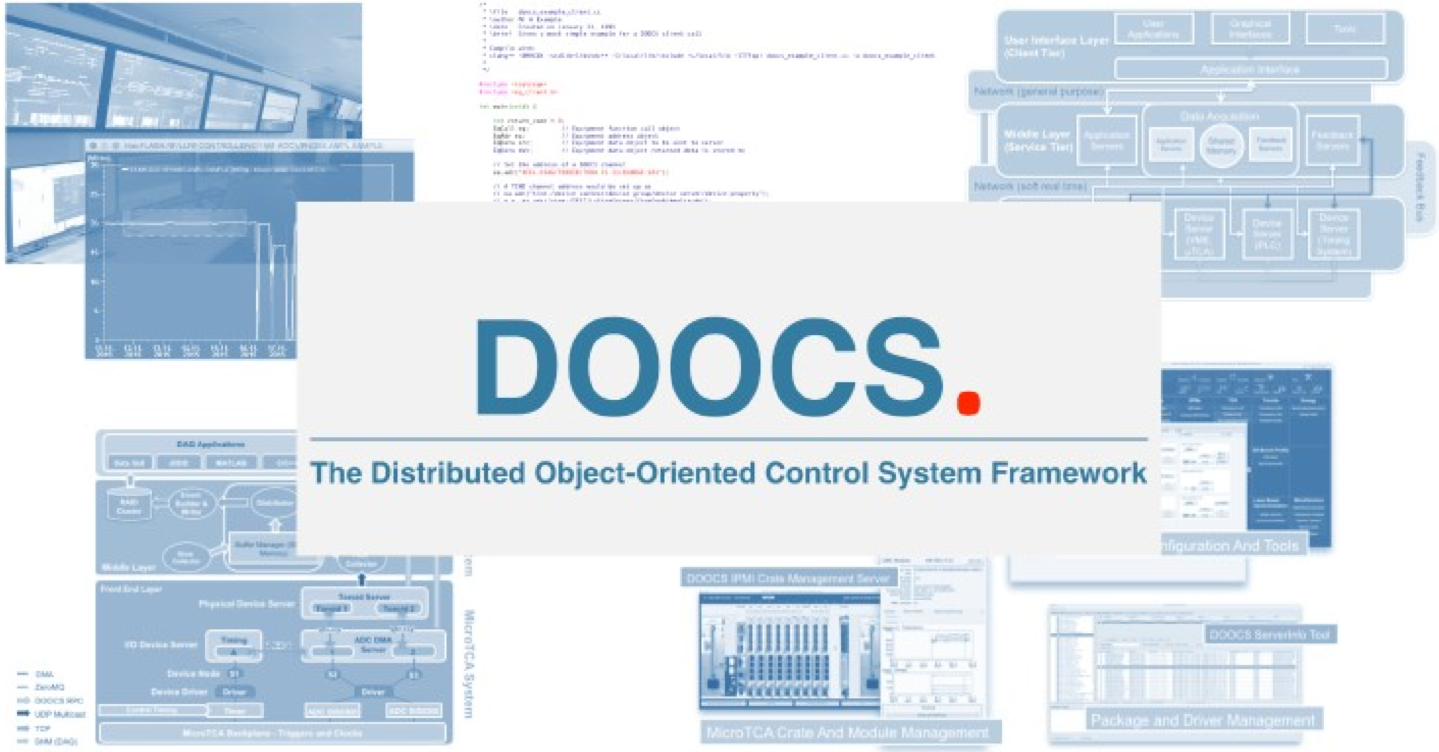
The Distributed Object-Oriented Control System

Jan Behrens (DESY-MCS)
Hamburg, 4.9.2025

HELMHOLTZ

DESY.

Outline



DOOCS.
The Distributed Object-Oriented Control System Framework

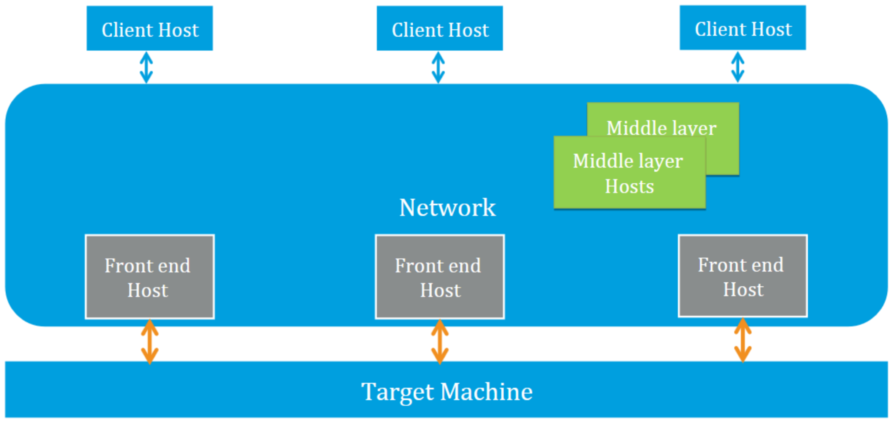
Basic concepts

Introduction

DOOCS - The Distributed Object-Oriented Control System

- Versatile **software framework** for creating accelerator-based **control system applications**
  - e.g. simple temperature monitoring, high-level controls, feedback of beam parameters
- Distributed **client-server architecture** combined with device-oriented view

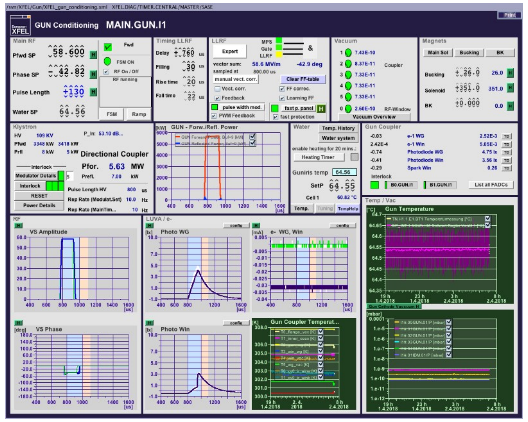
- **Control system parameters** are accessible via network calls
- Transportation layer based on **RPC/XDR** or **ZeroMQ** protocol
  - RPC: synchronous protocol, robust & efficient data transfer
  - ZMQ: independent layer, will replace RPC in the future

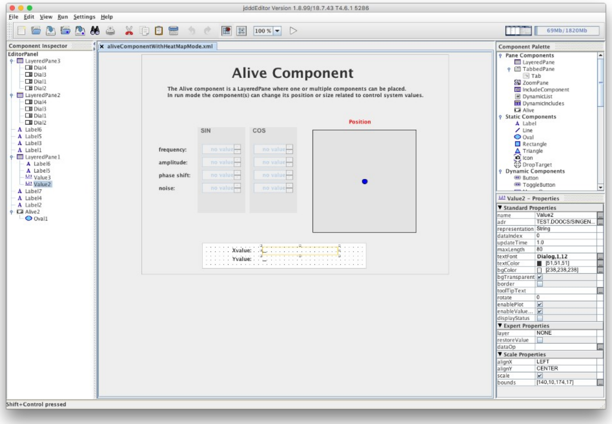


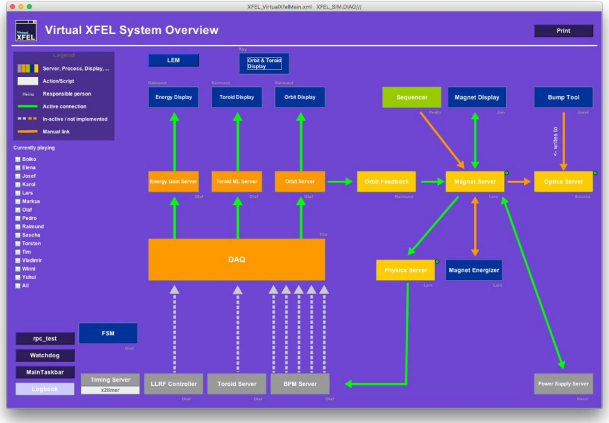
Introduction

DOOCS - The Distributed Object-Oriented Control System

- The DOOCS framework is written in **C++**
- Freely available under GNU GPL v2.1
- Libraries for client applications in **C++**, **Java**, **Python**, **MATLAB**, … exist
- **Graphical user interface** implemented as a lightweight Java application (JDDD)







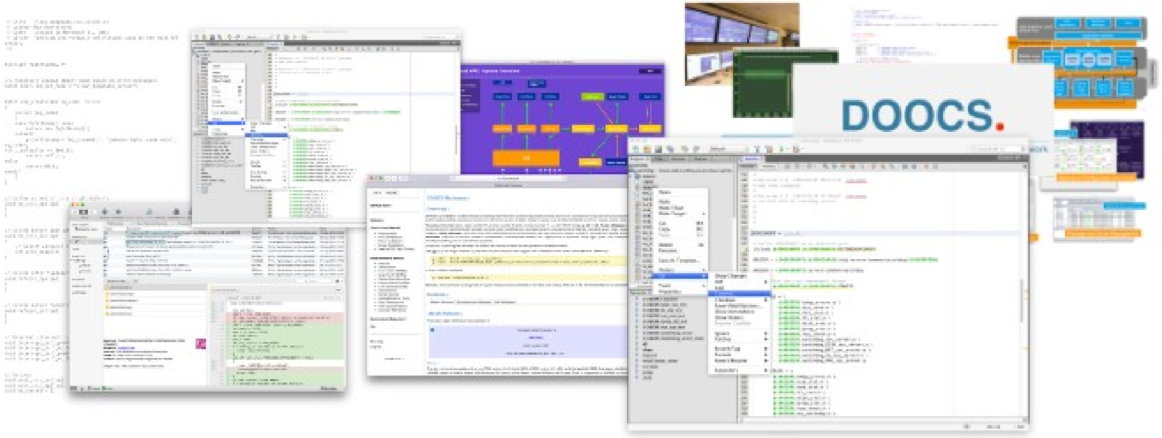
Development

- Stable DOOCS release about every **2 months**
  - Latest release version is v25.7
- **Nightly builds** for all platforms (DESY internal)
- Package repository for **Linux** (.deb) and **MacOS**

- Development on self-hosted **GitLab** instance
  - Heavy use of GitLab features like workflows, code reviews

- Uses **Meson** build system (only *doocs4py* uses CMake)
- Unit tests implemented with **catch2** framework

Basic Concepts



- Focus on operating particle **accelerator facilities**
- **Object-oriented** design paradigm → devices & data are objects

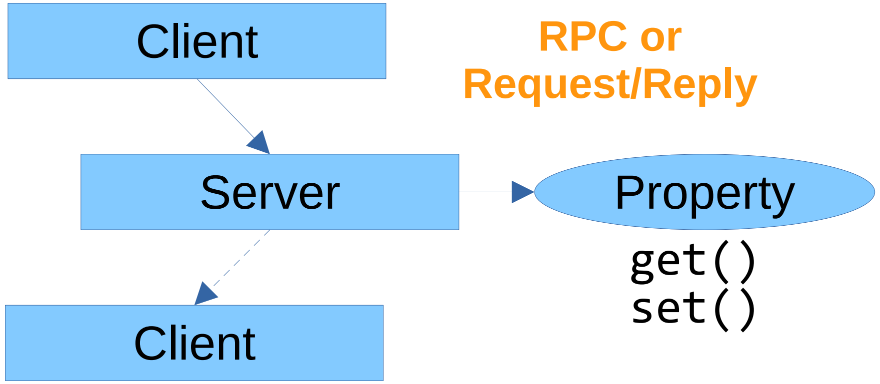
- Basic entity is a **device** → control system hardware or virtual logic
- Server application contains one or more **location** instances
- Each instance holds the **properties** of the (hardware/virtual) device
- Properties are access points to the communication network
  → represent **control system parameters**

- The **naming scheme** adheres to a four-string identifier: ***Facility / Device / Location / Property***

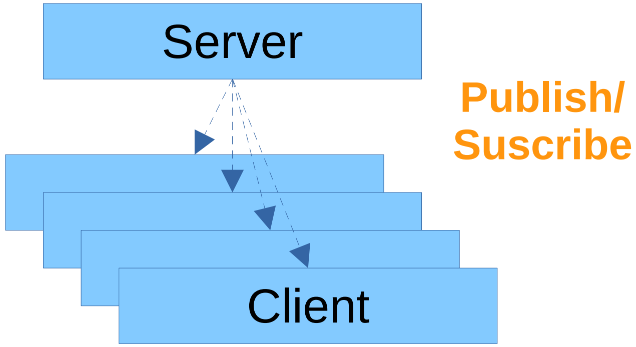
Server interfaces

Server interfaces

- Server core API implemented as **C++ library**
- Follows a **client-server** model
- **Multi-threaded** with update threads, interrupt handlers, …



- Network communication based on **RPC** with XDR data model
  - Additional interfaces (hardware, protocols) as separate libraries
- **ZeroMQ** protocol added in v24.1
  - Zero-broker message queue library
  - Request/Reply pattern → same functionality as RPC
  - Publish/Subscribe pattern → distribute data to subscribed clients

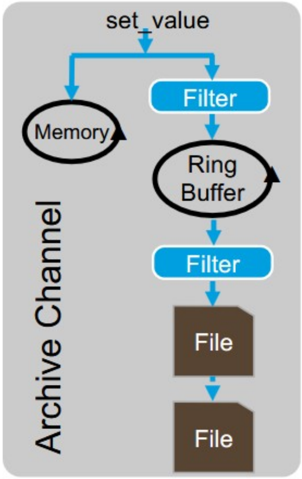


Server interfaces

- **Configuration** stored in local file, with **save & restore** functionality
- Locations & properties can be added/removed **on-the-fly**

- Extended **archiving** functionality
  - Histories stored locally on server, no network required
  - Access through regular server API

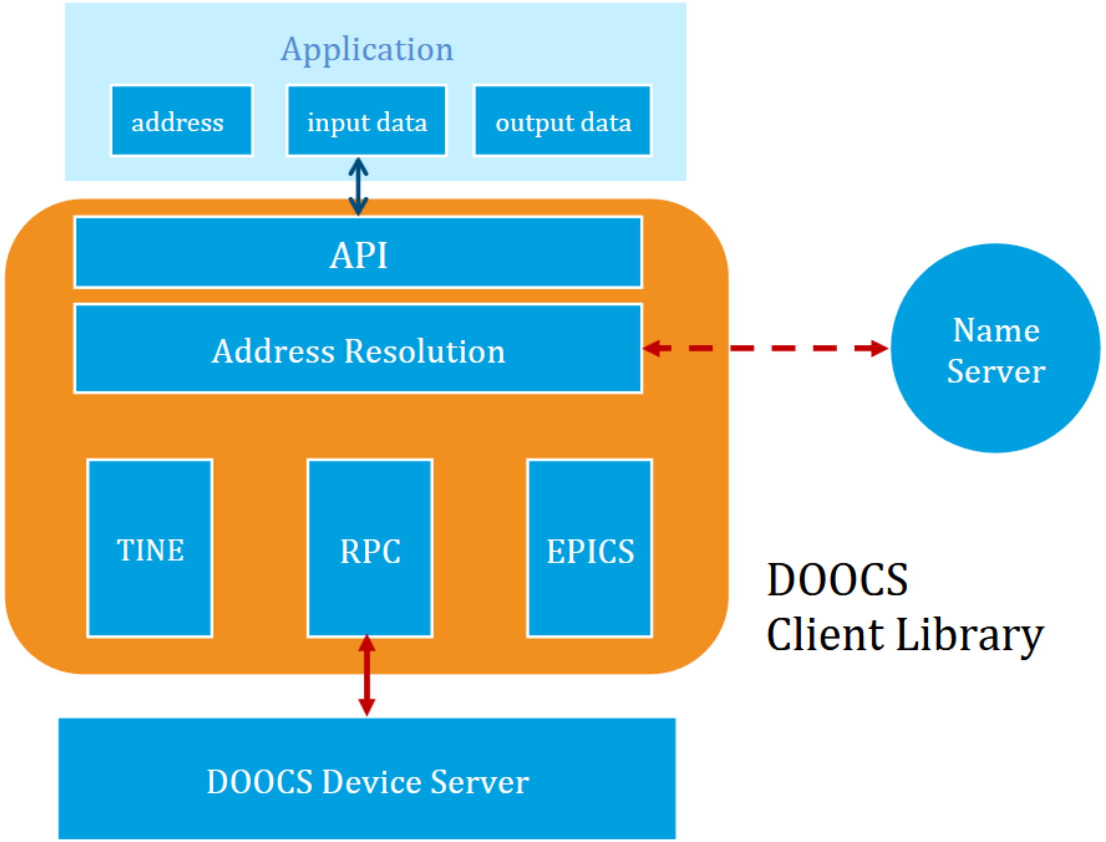
- Server **access authorization**
  - Basic access levels via UNIX user + group id lists (per server)
  - Fine-grained access control (on property level) via XML file



Client interfaces

Client interfaces

- Client API library written in **C++**
- Core features:
  - Unified addressing of **devices** & their **properties**
  - Universal **data** exchange independent of data type
  - Protocol-independent **access** to devices:
    RPC, ZMQ, TINE, EPICS
- DOOCS servers must be registered at dedicated **name server** for address resolution
- Some **high-level control** libraries exist in C++
  → provide commonly used features, similar approach as pyAML



Client interfaces – C++ API

- **EqCall** class
  - Represents client interface to DOOCS
  - **`get()`** + **`set()`** calls read/write device data on server
  - **`names()`** call to retrieve list of devices/locations/properties
  - Updates (reads) in background loop through **`get_monitor()`**
  - With ZeroMQ: asynchronous **`subscribe()`** / **`unsubscribe()`**
- **EqAdr** class
  - Parameter address, allows DOOCS/TINE/EPICS schemes
- **EqData** class
  - Main class to implement all DOOCS data types + getters/setters
  - Stores additional info like timestamps, error codes

```cpp
#include <iostream>
#include <eq_client.h>

int main(void) {

    int return_code = 0;
    EqCall eq;          // Equipment function call object
    EqAdr ea;           // Equipment address object
    EqData src;         // Equipment data object to be sent to server
    EqData dst;         // Equipment data object returned data is stored to

    // Set the address of a DOOCS channel
    ea.adr("XFEL.DIAG/TOROID/TORA.25.I1/CHARGE.SA1");

    // A TINE channel address would be set up as
    // ea.adr("tine:/device context/device group/device server/device property");
    // e.g. ea.adr("tine:/TEST/LxSineServer/SineGen0/Amplitude");

    // And an EPICS channel as
    // ea.adr("epics://host_ip/epics_channel_name")
    // e.g. ea.adr("epics://my_epics_ioc/test:sine");

    // Make the call
    return_code = eq.get(&ea, &src, &dst);

    if (return_code) {
        std::cout << "Error code: " << dst.error() << std::endl;
    } else {
        std::cout << "Data is: " << dst.get_string() << std::endl;
    }

    return 0;
}
```

Client interfaces – C++ API

- Supported data types:
  - Simple **scalars**:
    BOOL, SHORT, USHORT, INT, UINT, LONG, ULONG, FLOAT, DOUBLE
  - Compound scalars:
    IIII (= 4x INT), *IFFF* (= INT + 3x FLOAT), *TTII, USTR, XY, XYZS,*
    IMAGE, SPECTRUM, GSPECTRUM
  - Scalar **arrays**:
    A_BOOL, A_SHORT, A_USHORT, A_INT, A_UINT, A_LONG, A_ULONG,
    A_FLOAT, A_DOUBLE, STRING, TEXT, XML
  - Compound arrays:
    A_USTR, A_XYZS, A_BYTE, A_XY
  - Multi-dimensional arrays:
    MDA_FLOAT, MDA_DOUBLE
  - **Structured** data

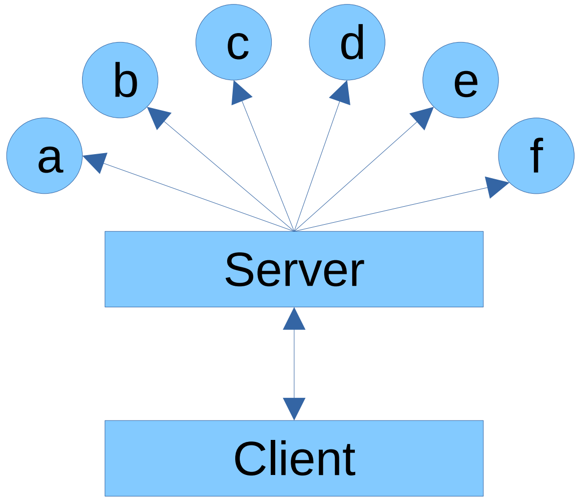
Client interfaces – Python API (doocs4py)

DOOCS4PY.

- **doocs4py** is a new approach to Python 3 bindings
  - Older implementation exists with PyDOOCS → *not covered here*
- API to write **clients** and **servers** with a single library
- Attempts to *completely* wrap the client & server API
- Direct interface to EqCall, EqAdr, EqData classes
- Supports all calls as **wrappers** to C++ methods:
  `get()` `set()` `names()` `get_monitor()` `subscribe()`
- Allows to write & run purely-Python server classes

```python
from doocs4py import set, get, names
from doocs4py.types import IIII

# --- names --- request names from doocs domain

location_list=names("TEST.DOOCS/UNIT_TEST_SUPPORT/*") # Make a list of locations
property_list=names("TEST.DOOCS/UNIT_TEST_SUPPORT/DOOCS4PY/*") # Make a list of properties

# --- set --- sets data to a Property from the server

set("TEST.DOOCS/UNIT_TEST_SUPPORT/DOOCS4PY/BOOL", False) # Set the value

#For scalar data types, the value can be set directly. For more complex data types, the type object must be created first.
#There is no need to create the data object first, it is created automatically when the value is set.

my_iiii = IIII(1, 2, 3, 4) # Create an object
set("TEST.DOOCS/UNIT_TEST_SUPPORT/DOOCS4PY/IIII", my_iiii) # Set the value in the property

# --- get --- Gets the data object from a Property from the server

my_bool = get("TEST.DOOCS/UNIT_TEST_SUPPORT/DOOCS4PY/BOOL") # Get the Data object
back_iiii = get("TEST.DOOCS/UNIT_TEST_SUPPORT/DOOCS4PY/IIII") # Get the Data object
print(f"my_iiii: {back_iiii.value.get_data()}") # Print the data stored in the object
print(f"my_bool: {my_bool.value}") # Print the value stored in the object
```

Special feature: Wildcard in addresses



- DOOCS does not support full **async/await** mechanism for calls
- However, one can use **star operations** (wildcards) in addresses:

  `PETRA/BPM/*/ORBIT`
- In this case, all locations/proprties are evaluated on the **server side**
- Aggregated results are returned to client in a **single data transfer**

- This only works for `get()` calls
- Not possible to `set()` variables at different addresses simultaneously

Single device server

Facility / Device / Location / Property

- Asynchonuous call feature is planned for a future release!

Comparison to TANGO and EPICS

- DOOCS implements **device servers**
- Most DOOCS calls are **synchronous**
- **LDAP** name service to locate servers by DOOCS address
- Similar n**aming scheme** to TANGO: Facility / Device / Server / Property
- **TANGO** uses broker system as name/communication service
- **EPICS** traditionally uses internal database (IOC)
  - Modern Process Variable Access similar to DOOCS
- DOOCS supports direct communication with EPICS servers
- Structured data in DOOCS is compatible with EPICS

Summary

- DOOCS is a **control system framework** for accelerators
- Written in **C++**, interfaces to **Python** / Java / MATLAB
- **Client-server** model with **devices** as main entities
- Device **properties** reflect control system **parameters**

- Network communication over **RPC**+XDR or **ZeroMQ**
- Most calls are **synchronous**
- ZeroMQ adds publish/subscribe mechanism

- **doocs4py** is the new interface for Python 3

Thank you.

References

- **Current and general overview of the accelerator control system as implemented at the European XFEL:**
  T. Wilksen et. al., “The control system for the linear accelerator at the European XFEL - Status and first experiences”, in Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control System (ICALEPCS’17), Barcelona, Spain, Oct. 2017, pp. 1-5, paper MOAPL01
  http://icalepcs2017.vrws.de/papers/moapl01.pdf

- **Paper on the accelerator data acquisition system as used at the European XFEL:**
  T. Wilksen et. al., "A bunch-synchronized data acquisition system for the European XFEL accelerator”, in Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control System (ICALEPCS’17), Barcelona, Spain, Oct. 2017, pp. 958-961, paper TUPHA210
  http://accelconf.web.cern.ch/AccelConf/icalepcs2017/papers/tupha210.pdf

- **Paper on high-level controls as implemented at the European XFEL:**
  L. Fröhlich et. al., “High level controls for the European XFEL”, in Proc. 15th Int. Conf. on Accelerator and Large Experimental Physics Control System (ICALEPCS’15), Melbourne, Australia, Oct. 2015, paper MOPGF101
  https://accelconf.web.cern.ch/ICALEPCS2015/papers/mopgf101.pdf

- **Paper on virtual XFEL based on the DOOCS DAQ:**
  W. Decking et al., “The Virtual European XFEL Accelerator”, in Proc. 15th Int. Conf. on Accelerator and Large Experimental Physics Control System (ICALEPCS’15), Melbourne, Australia, Oct. 2015, paper TUD3O04.
  https://accelconf.web.cern.ch/AccelConf/ICALEPCS2015/papers/tud3o04.pdf

- **Short overview on tools for DAQ data retrieval (slightly out-of-date):**
  V.Rybnikov et al.,“FLASH DAQ data management and access tools”, PCaPAC’10, Saskatoon, Canada (2010), pp. 195–197.
  https://accelconf.web.cern.ch/pcapac2010/papers/frcoma02.pdf

- **Paper on the implementation of the inner workings of the DOOCS DAQ (slightly out-of-date):**
  V. Rybnikov et al., “A Buffer Manager Implementation for the FLASH Data Acquisition System”, PCaPAC 2008, Ljubljana, Slovenia, October 2008
  https://accelconf.web.cern.ch/pc08/papers/tup010.pdf

- **Original paper of the overall DAQ concept (slightly out-of-date):**
  A. Aghababyan et al., “Multi-Processor Based Fast Data Acquisition for a Free Electron Laser and Experiments”, in IEEE Transactions on Nuclear Science, vol. 55, No. 1, pp. 256-260, February 2008.
  https://ieeexplore.ieee.org/document/4382853