

Introduction to EPICS

from a user point of view

Fernando Henrique de Sá

fernando.sa@lnls.br

Accelerator Physics Group (FAC)

DAC/LNLS/CNPq

EPICS Basics



CNPq


```
(sirius) ~$ caget SI-Fam:PS-Q1:Current-Mon  
SI-Fam:PS-Q1:Current-Mon      91.33
```



```
(sirius) ~$ caget SI-Fam:PS-Q1:Current-Mon  
SI-Fam:PS-Q1:Current-Mon      91.33
```

The client: environment where
connections requests are issued.

EPICS Basics

CA stands for Channel
Access, the name of the
communication
protocol underlying all
EPICS 3 connections

```
(sirius) ~$ caget SI-Fam:PS-Q1:Current-Mon  
SI-Fam:PS-Q1:Current-Mon      91.33
```

The client: environment where
connections requests are issued.

EPICS Basics

CA stands for Channel Access, the name of the communication protocol underlying all EPICS 3 connections

Connects to a CA server (input-output controller, or IOC) and gets the current value of one record

```
(sirius) ~$ caget SI-Fam:PS-Q1:Current-Mon  
SI-Fam:PS-Q1:Current-Mon      91.33
```

The client: environment where connections requests are issued.

EPICS Basics



CNPEM

CA stands for Channel Access, the name of the communication protocol underlying all EPICS 3 connections

Connects to a CA server (input-output controller, or IOC) and gets the current value of one record

Name of the property we want to connect. **Represents some process variable (PV) of a given device.** These **names** are defined by a **string** (a-z A-Z 0-9 _ - : [] < > ;) and **must be unique** across the entire EPICS network. At SIRIUS, we use naming conventions to improve interpretability.

```
(sirius) ~$ caget SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon      91.33
```

The client: environment where connections requests are issued.

CA stands for Channel Access, the name of the communication protocol underlying all EPICS 3 connections

Connects to a CA server (input-output controller, or IOC) and gets the current value of one record

Name of the property we want to connect. **Represents some process variable (PV) of a given device.** These **names** are defined by a **string** (a-z A-Z 0-9 _ - : [] < > ;) and **must be unique** across the entire EPICS network. At SIRIUS, we use naming conventions to improve interpretability.

```
(sirius) ~$ caget SI-Fam:PS-Q1:Current-Mon  
SI-Fam:PS-Q1:Current-Mon 91.33
```

The client: environment where connections requests are issued.

Value of the process variable. In this case, the DCCT measurement of the electric current of a quadrupole power supply.

CA stands for Channel Access, the name of the communication protocol underlying all EPICS 3 connections

Connects to a CA server (input-output controller, or IOC) and gets the current value of one record

Name of the property we want to connect. **Represents some process variable (PV) of a given device.** These **names** are defined by a **string** (a-z A-Z 0-9 _ - : [] < > ;) and **must be unique** across the entire EPICS network. At SIRIUS, we use naming conventions to improve interpretability.

```
(sirius) ~$ caget SI-Fam:PS-Q1:Current-Mon  
SI-Fam:PS-Q1:Current-Mon 91.33
```

The client: environment where connections requests are issued.

Value of the process variable. In this case, the DCCT measurement of the electric current of a quadrupole power supply.

- Non-hierarchical, “flat”, control system. Based on process variables instead of objects:

CA stands for Channel Access, the name of the communication protocol underlying all EPICS 3 connections

Connects to a CA server (input-output controller, or IOC) and gets the current value of one record

Name of the property we want to connect. **Represents some process variable (PV) of a given device.** These **names** are defined by a **string** (a-z A-Z 0-9 _ - : [] < > ;) and **must be unique** across the entire EPICS network. At SIRIUS, we use naming conventions to improve interpretability.

```
(sirius) ~$ caget SI-Fam:PS-Q1:Current-Mon  
SI-Fam:PS-Q1:Current-Mon 91.33
```

The client: environment where connections requests are issued.

Value of the process variable. In this case, the DCCT measurement of the electric current of a quadrupole power supply.

- Non-hierarchical, “flat”, control system. Based on process variables instead of objects:
 - Modular, distributed and scalable: no bottlenecks, very robust and decentralized;

CA stands for Channel Access, the name of the communication protocol underlying all EPICS 3 connections

Connects to a CA server (input-output controller, or IOC) and gets the current value of one record

Name of the property we want to connect. **Represents some process variable (PV) of a given device.** These **names** are defined by a **string** (a-z A-Z 0-9 _ - : [] < > ;) and **must be unique** across the entire EPICS network. At SIRIUS, we use naming conventions to improve interpretability.

```
(sirius) ~$ caget SI-Fam:PS-Q1:Current-Mon  
SI-Fam:PS-Q1:Current-Mon 91.33
```

The client: environment where connections requests are issued.

Value of the process variable. In this case, the DCCT measurement of the electric current of a quadrupole power supply.

- Non-hierarchical, “flat”, control system. Based on process variables instead of objects:
 - Modular, distributed and scalable: no bottlenecks, very robust and decentralized;
 - Lack of abstraction and atomic actions (at least with version 3);

CA stands for Channel Access, the name of the communication protocol underlying all EPICS 3 connections

Connects to a CA server (input-output controller, or IOC) and gets the current value of one record

Name of the property we want to connect. **Represents some process variable (PV) of a given device.** These **names** are defined by a **string** (a-z A-Z 0-9 _ - : [] < > ;) and **must be unique** across the entire EPICS network. At SIRIUS, we use naming conventions to improve interpretability.

```
(sirius) ~$ caget SI-Fam:PS-Q1:Current-Mon  
SI-Fam:PS-Q1:Current-Mon 91.33
```

The client: environment where connections requests are issued.

Value of the process variable. In this case, the DCCT measurement of the electric current of a quadrupole power supply.

- Non-hierarchical, “flat”, control system. Based on process variables instead of objects:
 - Modular, distributed and scalable: no bottlenecks, very robust and decentralized;
 - Lack of abstraction and atomic actions (at least with version 3);
- Tool based: minimizes need for customer-specific coding (independent development);

CA stands for Channel Access, the name of the communication protocol underlying all EPICS 3 connections

Connects to a CA server (input-output controller, or IOC) and gets the current value of one record

Name of the property we want to connect. **Represents some process variable (PV) of a given device.** These **names** are defined by a **string** (a-z A-Z 0-9 _ - : [] < > ;) and **must be unique** across the entire EPICS network. At SIRIUS, we use naming conventions to improve interpretability.

```
(sirius) ~$ caget SI-Fam:PS-Q1:Current-Mon  
SI-Fam:PS-Q1:Current-Mon 91.33
```

The client: environment where connections requests are issued.

Value of the process variable. In this case, the DCCT measurement of the electric current of a quadrupole power supply.

- Non-hierarchical, “flat”, control system. Based on process variables instead of objects:
 - Modular, distributed and scalable: no bottlenecks, very robust and decentralized;
 - Lack of abstraction and atomic actions (at least with version 3);
- Tool based: minimizes need for customer-specific coding (independent development);
- Written in C++, with support for python and matlab, re-implemented in Java, etc.

Each PV has several additional properties:

```
(sirius) ~$ caget -d 34 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
  Native data type: DBF_DOUBLE
  Request type:     DBR_CTRL_DOUBLE
  Element count:    1
  Value:            91.33
  Status:           NO_ALARM
  Severity:         NO_ALARM
  Units:            A
  Precision:        4
  Lo disp limit:    0
  Hi disp limit:    120
  Lo alarm limit:   0
  Lo warn limit:    0
  Hi warn limit:    120
  Hi alarm limit:   120
  Lo ctrl limit:    0
  Hi ctrl limit:    120
```


Each PV has several additional properties:

If > 1 , PV is an array.

```
(sirius) ~$ caget -d 34 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
Native data type: DBF_DOUBLE
Request type: DBR_CTRL_DOUBLE
Element count: 1
Value: 91.33
Status: NO_ALARM
Severity: NO_ALARM
Units: A
Precision: 4
Lo disp limit: 0
Hi disp limit: 120
Lo alarm limit: 0
Lo warn limit: 0
Hi warn limit: 120
Hi alarm limit: 120
Lo ctrl limit: 0
Hi ctrl limit: 120
```


Each PV has several additional properties:

If > 1 , PV is an array.

Possible alarms or warnings related to this PV: including disconnection with hardware or other reliability problems.

```
(sirius) ~$ caget -d 34 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
Native data type: DBF_DOUBLE
Request type: DBR_CTRL_DOUBLE
Element count: 1
Value: 91.33
Status: NO_ALARM
Severity: NO_ALARM
Units: A
Precision: 4
Lo disp limit: 0
Hi disp limit: 120
Lo alarm limit: 0
Lo warn limit: 0
Hi warn limit: 120
Hi alarm limit: 120
Lo ctrl limit: 0
Hi ctrl limit: 120
```


Each PV has several additional properties:

If > 1 , PV is an array.

Possible alarms or warnings related to this PV: including disconnection with hardware or other reliability problems.

Physical interpretation of data

```
(sirius) ~$ caget -d 34 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
Native data type: DBF_DOUBLE
Request type: DBR_CTRL_DOUBLE
Element count: 1
Value: 91.33
Status: NO_ALARM
Severity: NO_ALARM
Units: A
Precision: 4
Lo disp limit: 0
Hi disp limit: 120
Lo alarm limit: 0
Lo warn limit: 0
Hi warn limit: 120
Hi alarm limit: 120
Lo ctrl limit: 0
Hi ctrl limit: 120
```


Each PV has several additional properties:

If > 1 , PV is an array.

Possible alarms or warnings related to this PV: including disconnection with hardware or other reliability problems.

Physical interpretation of data

Limits that raise alarms

```
(sirius) ~$ caget -d 34 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
Native data type: DBF_DOUBLE
Request type: DBR_CTRL_DOUBLE
Element count: 1
Value: 91.33
Status: NO_ALARM
Severity: NO_ALARM
Units: A
Precision: 4
Lo disp limit: 0
Hi disp limit: 120
Lo alarm limit: 0
Lo warn limit: 0
Hi warn limit: 120
Hi alarm limit: 120
Lo ctrl limit: 0
Hi ctrl limit: 120
```


Each PV has several additional properties:

If > 1 , PV is an array.

Possible alarms or warnings related to this PV: including disconnection with hardware or other reliability problems.

Physical interpretation of data

Limits that raise alarms

```
(sirius) ~$ caget -d 34 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
Native data type: DBF_DOUBLE
Request type:     DBR_CTRL_DOUBLE
Element count:    1
Value:            91.33
Status:           NO_ALARM
Severity:         NO_ALARM
Units:            A
Precision:        4
Lo disp limit:    0
Hi disp limit:    120
Lo alarm limit:   0
Lo warn limit:    0
Hi warn limit:    120
Hi alarm limit:   120
Lo ctrl limit:    0
Hi ctrl limit:    120
```

```
(sirius) ~$ caget -d 20 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
Native data type: DBF_DOUBLE
Request type:     DBR_TIME_DOUBLE
Element count:    1
Value:            91.3305
Timestamp:        2025-08-28 14:10:03.964393
Status:           NO_ALARM
Severity:         NO_ALARM
```


Each PV has several additional properties:

If > 1 , PV is an array.

Possible alarms or warnings related to this PV: including disconnection with hardware or other reliability problems.

Physical interpretation of data

Limits that raise alarms

Date and time of the last update of this PV

```
(sirius) ~$ caget -d 34 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
Native data type: DBF_DOUBLE
Request type: DBR_CTRL_DOUBLE
Element count: 1
Value: 91.33
Status: NO_ALARM
Severity: NO_ALARM
Units: A
Precision: 4
Lo disp limit: 0
Hi disp limit: 120
Lo alarm limit: 0
Lo warn limit: 0
Hi warn limit: 120
Hi alarm limit: 120
Lo ctrl limit: 0
Hi ctrl limit: 120

(sirius) ~$ caget -d 20 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
Native data type: DBF_DOUBLE
Request type: DBR_TIME_DOUBLE
Element count: 1
Value: 91.3305
Timestamp: 2025-08-28 14:10:03.964393
Status: NO_ALARM
Severity: NO_ALARM
```


Each PV has several additional properties:

If > 1 , PV is an array.

Possible alarms or warnings related to this PV: including disconnection with hardware or other reliability problems.

Physical interpretation of data

Limits that raise alarms

Date and time of the last update of this PV

PVs also have read and write access control. For instance, we cannot write on a PV that only maps the readout of some property:

```
(sirius) ~$ caput SI-Fam:PS-Q1:Current-Mon 92
Old : SI-Fam:PS-Q1:Current-Mon      91.3298
Error from put operation: Write access denied
```

```
(sirius) ~$ caget -d 34 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
Native data type: DBF_DOUBLE
Request type:     DBR_CTRL_DOUBLE
Element count:    1
Value:            91.33
Status:           NO_ALARM
Severity:         NO_ALARM
Units:            A
Precision:        4
Lo disp limit:    0
Hi disp limit:    120
Lo alarm limit:   0
Lo warn limit:    0
Hi warn limit:    120
Hi alarm limit:   120
Lo ctrl limit:    0
Hi ctrl limit:    120
```

```
(sirius) ~$ caget -d 20 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
Native data type: DBF_DOUBLE
Request type:     DBR_TIME_DOUBLE
Element count:    1
Value:            91.3305
Timestamp:        2025-08-28 14:10:03.964393
Status:           NO_ALARM
Severity:         NO_ALARM
```


Each PV has several additional properties:

If > 1 , PV is an array.

Possible alarms or warnings related to this PV: including disconnection with hardware or other reliability problems.

Physical interpretation of data

Limits that raise alarms

Date and time of the last update of this PV

PVs also have read and write access control. For instance, we cannot write on a PV that only maps the readout of some property:

Command to write on a PV

```
(sirius) ~$ caput SI-Fam:PS-Q1:Current-Mon 92
Old : SI-Fam:PS-Q1:Current-Mon      91.3298
Error from put operation: Write access denied
```

```
(sirius) ~$ caget -d 34 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
Native data type: DBF_DOUBLE
Request type:     DBR_CTRL_DOUBLE
Element count:    1
Value:            91.33
Status:           NO_ALARM
Severity:          NO_ALARM
Units:            A
Precision:        4
Lo disp limit:    0
Hi disp limit:    120
Lo alarm limit:    0
Lo warn limit:    0
Hi warn limit:    120
Hi alarm limit:    120
Lo ctrl limit:    0
Hi ctrl limit:    120
```

```
(sirius) ~$ caget -d 20 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
Native data type: DBF_DOUBLE
Request type:     DBR_TIME_DOUBLE
Element count:    1
Value:            91.3305
Timestamp:        2025-08-28 14:10:03.964393
Status:           NO_ALARM
Severity:          NO_ALARM
```


Each PV has several additional properties:

If > 1 , PV is an array.

Possible alarms or warnings related to this PV: including disconnection with hardware or other reliability problems.

Physical interpretation of data

Limits that raise alarms

Date and time of the last update of this PV

PVs also have read and write access control. For instance, we cannot write on a PV that only maps the readout of some property:

Command to write on a PV

Desired setpoint value

```
(sirius) ~$ caput SI-Fam:PS-Q1:Current-Mon 92
Old : SI-Fam:PS-Q1:Current-Mon      91.3298
Error from put operation: Write access denied
```

```
(sirius) ~$ caget -d 34 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
Native data type: DBF_DOUBLE
Request type:     DBR_CTRL_DOUBLE
Element count:    1
Value:            91.33
Status:           NO_ALARM
Severity:         NO_ALARM
Units:            A
Precision:        4
Lo disp limit:    0
Hi disp limit:    120
Lo alarm limit:    0
Lo warn limit:    0
Hi warn limit:    120
Hi alarm limit:    120
Lo ctrl limit:    0
Hi ctrl limit:    120
```

```
(sirius) ~$ caget -d 20 SI-Fam:PS-Q1:Current-Mon
SI-Fam:PS-Q1:Current-Mon
Native data type: DBF_DOUBLE
Request type:     DBR_TIME_DOUBLE
Element count:    1
Value:            91.3305
Timestamp:        2025-08-28 14:10:03.964393
Status:           NO_ALARM
Severity:         NO_ALARM
```


EPICS Servers



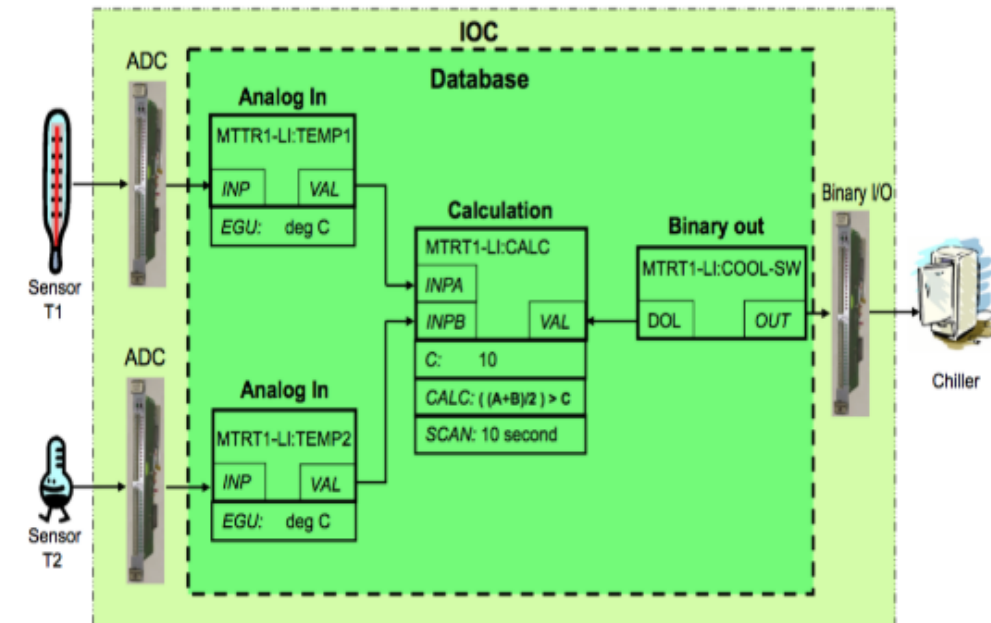
CNPq

- On the server side, the IOC building blocks are records that are mapped into CA PVs by the server layer;

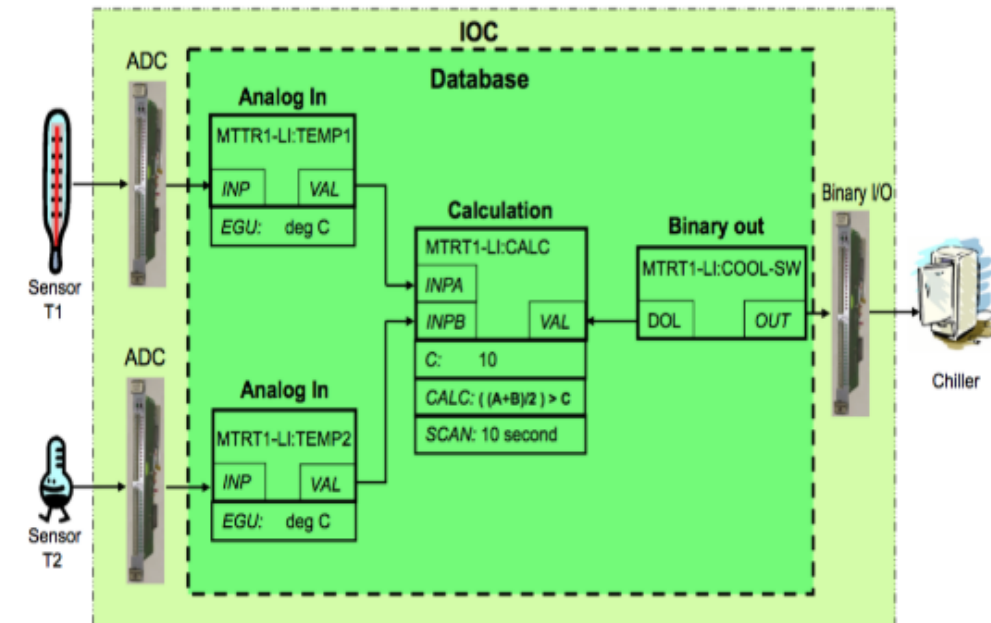
- On the server side, the IOC building blocks are records that are mapped into CA PVs by the server layer;
- There are several types of records, to perform different tasks:

EPICS Servers

- On the server side, the IOC building blocks are records that are mapped into CA PVs by the server layer;
- There are several types of records, to perform different tasks:
 - Analog/Long/Binary/String Input: read data from hardware (they differ on the data type that can be handled);
 - Analog/Long/Binary/String Output: send data to hardware;
 - Calculation Record (Output): value based on state of other records (and have conditional updates);

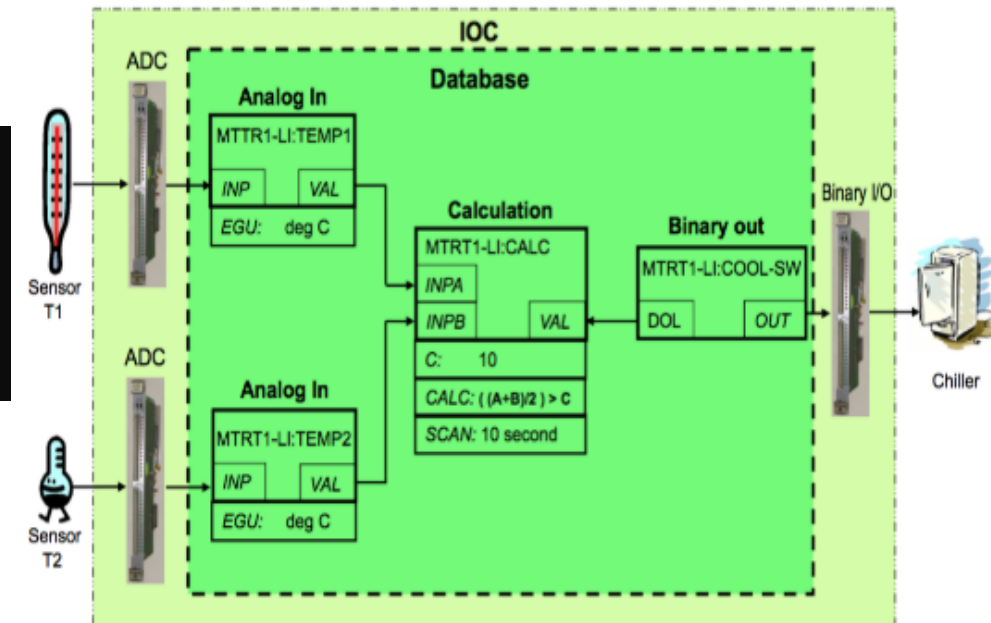


- On the server side, the IOC building blocks are records that are mapped into CA PVs by the server layer;
- There are several types of records, to perform different tasks:
 - Analog/Long/Binary/String Input: read data from hardware (they differ on the data type that can be handled);
 - Analog/Long/Binary/String Output: send data to hardware;
 - Calculation Record (Output): value based on state of other records (and have conditional updates);
- Combining them in an IOC allows **complex logic with very little (conventional) programming**

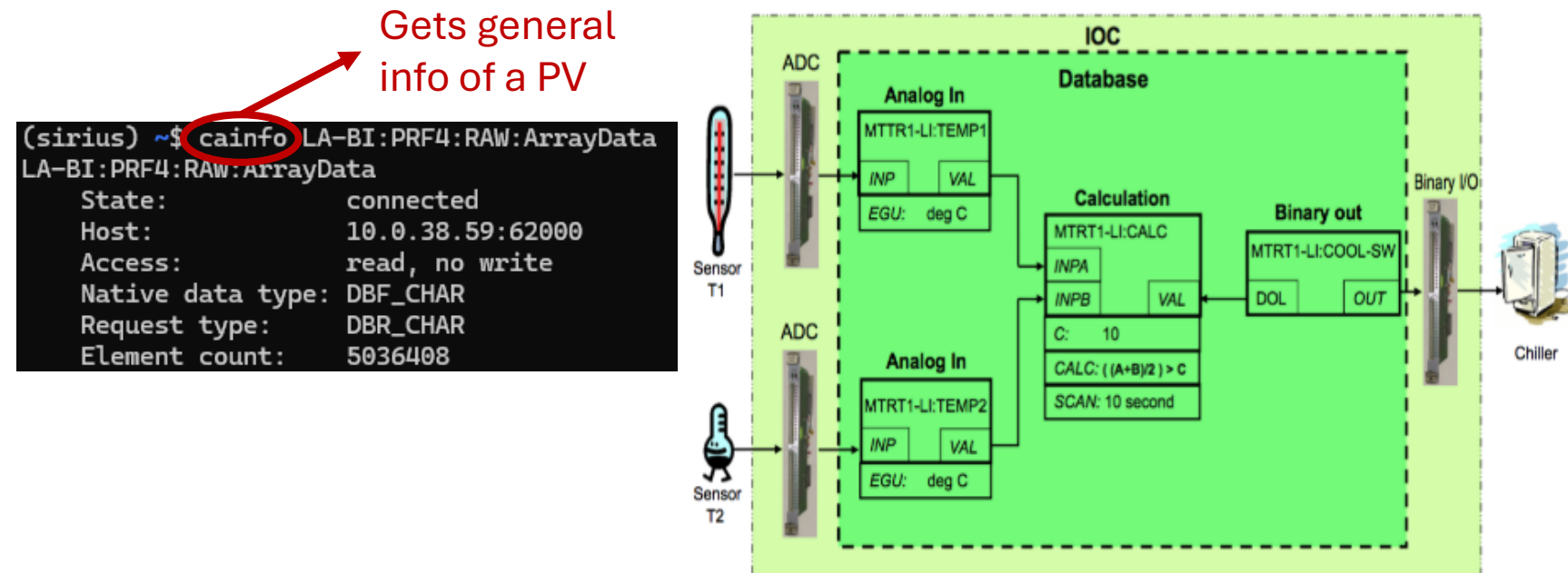


- On the server side, the IOC building blocks are records that are mapped into CA PVs by the server layer;
- There are several types of records, to perform different tasks:
 - Analog/Long/Binary/String Input: read data from hardware (they differ on the data type that can be handled);
 - Analog/Long/Binary/String Output: send data to hardware;
 - Calculation Record (Output): value based on state of other records (and have conditional updates);
 - Waveform: stores waveforms of data (only 1D arrays);
- Combining them in an IOC allows **complex logic with very little (conventional) programming**

```
(sirius) ~$ cainfo LA-BI:PRF4:RAW:ArrayData
LA-BI:PRF4:RAW:ArrayData
  State:      connected
  Host:       10.0.38.59:62000
  Access:     read, no write
  Native data type: DBF_CHAR
  Request type: DBR_CHAR
  Element count: 5036408
```



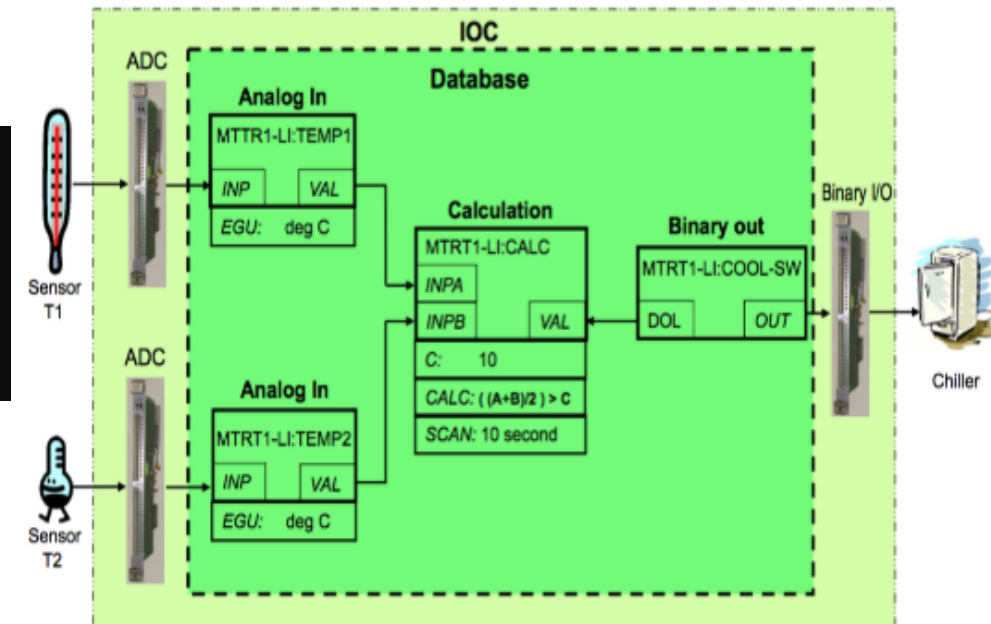
- On the server side, the IOC building blocks are records that are mapped into CA PVs by the server layer;
- There are several types of records, to perform different tasks:
 - Analog/Long/Binary/String Input: read data from hardware (they differ on the data type that can be handled);
 - Analog/Long/Binary/String Output: send data to hardware;
 - Calculation Record (Output): value based on state of other records (and have conditional updates);
 - Waveform: stores waveforms of data (only 1D arrays);
- Combining them in an IOC allows **complex logic with very little (conventional) programming**



- On the server side, the IOC building blocks are records that are mapped into CA PVs by the server layer;
- There are several types of records, to perform different tasks:
 - Analog/Long/Binary/String Input: read data from hardware (they differ on the data type that can be handled);
 - Analog/Long/Binary/String Output: send data to hardware;
 - Calculation Record (Output): value based on state of other records (and have conditional updates);
 - Waveform: stores waveforms of data (only 1D arrays);
- Combining them in an IOC allows **complex logic with very little (conventional) programming**

Server's IP Gets general info of a PV

```
(sirius) ~$ cainfo LA-BI:PRF4:RAW:ArrayData
LA-BI:PRF4:RAW:ArrayData
State:      connected
Host:      10.0.38.59:62000
Access:     read, no write
Native data type: DBF_CHAR
Request type: DBR_CHAR
Element count: 5036408
```



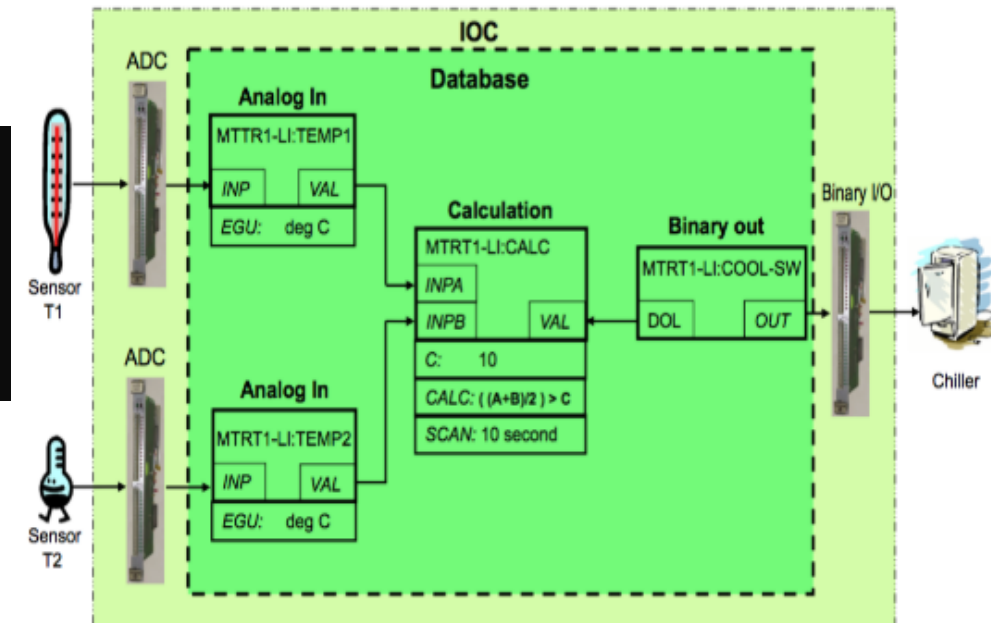
EPICS Servers

- On the server side, the IOC building blocks are records that are mapped into CA PVs by the server layer;
- There are several types of records, to perform different tasks:
 - Analog/Long/Binary/String Input: read data from hardware (they differ on the data type that can be handled);
 - Analog/Long/Binary/String Output: send data to hardware;
 - Calculation Record (Output): value based on state of other records (and have conditional updates);
 - Waveform: stores waveforms of data (only 1D arrays);
- Combining them in an IOC allows **complex logic with very little (conventional) programming**

Server's IP Gets general info of a PV

```
(sirius) ~$ cainfo LA-BI:PRF4:RAW:ArrayData
LA-BI:PRF4:RAW:ArrayData
State:      connected
Host:      10.0.38.59:62000
Access:     read, no write
Native data type: DBF_CHAR
Request type: DBF_CHAR
Element count: 5036408
```

This is an **image**, but the **PV is a 1D array**. Additional info must be provided by other record for image reconstruction (EPICS 3).



EPICS Servers

- On the server side, the IOC building blocks are records that are mapped into CA PVs by the server layer;
- There are several types of records, to perform different tasks:
 - Analog/Long/Binary/String Input: read data from hardware (they differ on the data type that can be handled);
 - Analog/Long/Binary/String Output: send data to hardware;
 - Calculation Record (Output): value based on state of other records (and have conditional updates);
 - Waveform: stores waveforms of data (only 1D arrays);
 - Multi-Bit Binary Input/Output: trigger tasks based on up to 16 choices (ENUMS)
- Combining them in an IOC allows **complex logic with very little (conventional) programming**

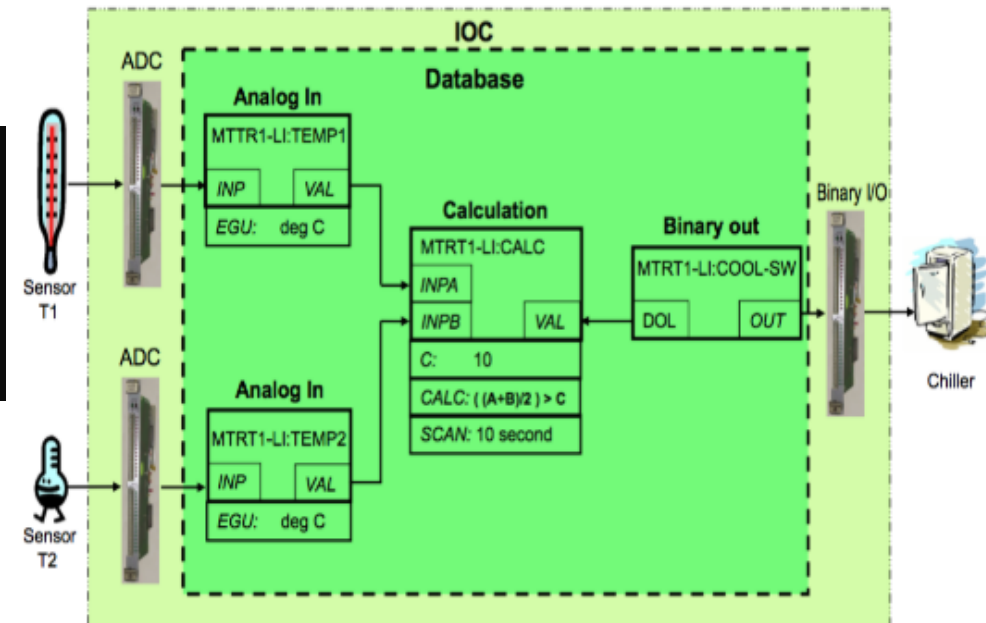
```
(sirius) ~$ caget -d 31 SI-Fam:TI-BPM:State-Sel
SI-Fam:TI-BPM:State-Sel
Native data type: DBF_ENUM
Request type: DBR_CTRL_ENUM
Element count: 1
Value: Enbl
Status: NO_ALARM
Severity: NO_ALARM
Enums: ( 2)
        [ 0] Dsbl
        [ 1] Enbl
```

Server's IP

Gets general info of a PV

```
(sirius) ~$ cainfo LA-BI:PRF4:RAW:ArrayData
LA-BI:PRF4:RAW:ArrayData
State: connected
Host: 10.0.38.59:62000
Access: read, no write
Native data type: DBF_CHAR
Request type: DBR_CHAR
Element count: 5036408
```

This is an **image**, but the **PV is a 1D array**. Additional info must be provided by other record for image reconstruction (EPICS 3).



EPICS Servers

- On the server side, the IOC building blocks are records that are mapped into CA PVs by the server layer;
- There are several types of records, to perform different tasks:
 - Analog/Long/Binary/String Input: read data from hardware (they differ on the data type that can be handled);
 - Analog/Long/Binary/String Output: send data to hardware;
 - Calculation Record (Output): value based on state of other records (and have conditional updates);
 - Waveform: stores waveforms of data (only 1D arrays);
 - Multi-Bit Binary Input/Output: trigger tasks based on up to 16 choices (ENUMS)
- Combining them in an IOC allows **complex logic with very little (conventional) programming**

```
(sirius) ~$ caget -d 31 SI-Fam:TI-BPM:State-Sel
SI-Fam:TI-BPM:State-Sel
Native data type: DBF_ENUM
Request type: DBR_CTRL_ENUM
Element count: 1
Value: Enbl
Status: NO_ALARM
Severity: NO_ALARM
Enums: ( 2)
       [ 0] Dsbl
       [ 1] Enbl
```

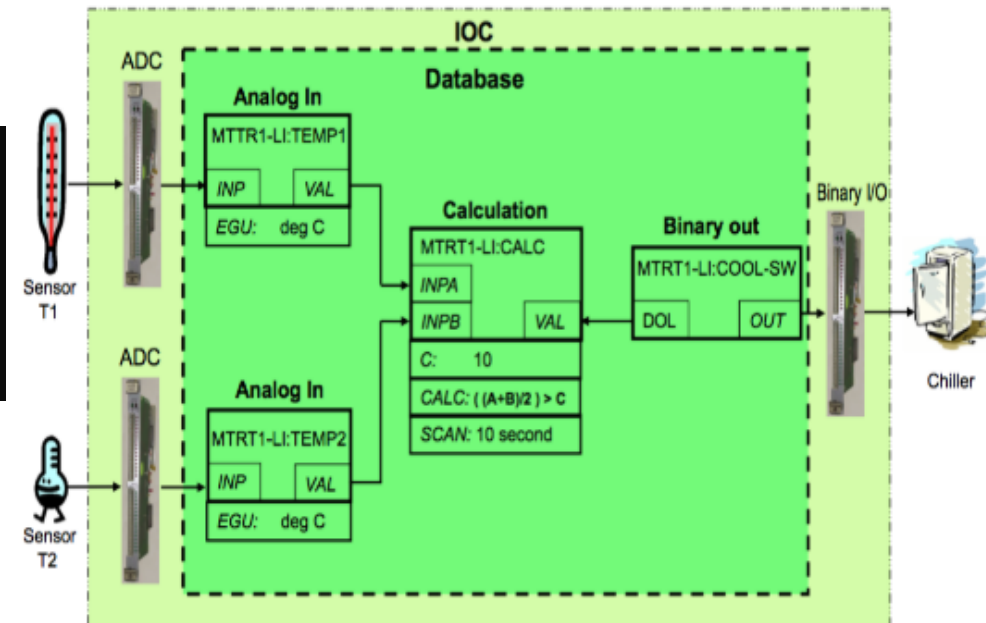
Server's IP

```
(sirius) ~$ cainfo LA-BI:PRF4:RAW:ArrayData
LA-BI:PRF4:RAW:ArrayData
State: connected
Host: 10.0.38.59:62000
Access: read, no write
Native data type: DBF_CHAR
Request type: DBR_CHAR
Element count: 5036408
```

Gets general
info of a PV

This is an **image**, but the **PV is a 1D array**.
Additional info must be provided by other
record for image reconstruction (EPICS 3).

Can be set either with the
integer or the string value



EPICS Servers

- On the server side, the IOC building blocks are records that are mapped into CA PVs by the server layer;
- There are several types of records, to perform different tasks:
 - Analog/Long/Binary/String Input: read data from hardware (they differ on the data type that can be handled);
 - Analog/Long/Binary/String Output: send data to hardware;
 - Calculation Record (Output): value based on state of other records (and have conditional updates);
 - Waveform: stores waveforms of data (only 1D arrays);
 - Multi-Bit Binary Input/Output: trigger tasks based on up to 16 choices (ENUMS)
 - ...
- Combining them in an IOC allows **complex logic with very little (conventional) programming**

```
(sirius) ~$ caget -d 31 SI-Fam:TI-BPM:State-Sel
SI-Fam:TI-BPM:State-Sel
Native data type: DBF_ENUM
Request type: DBR_CTRL_ENUM
Element count: 1
Value: Enbl
Status: NO_ALARM
Severity: NO_ALARM
Enums: ( 2)
       [ 0] Dsbl
       [ 1] Enbl
```

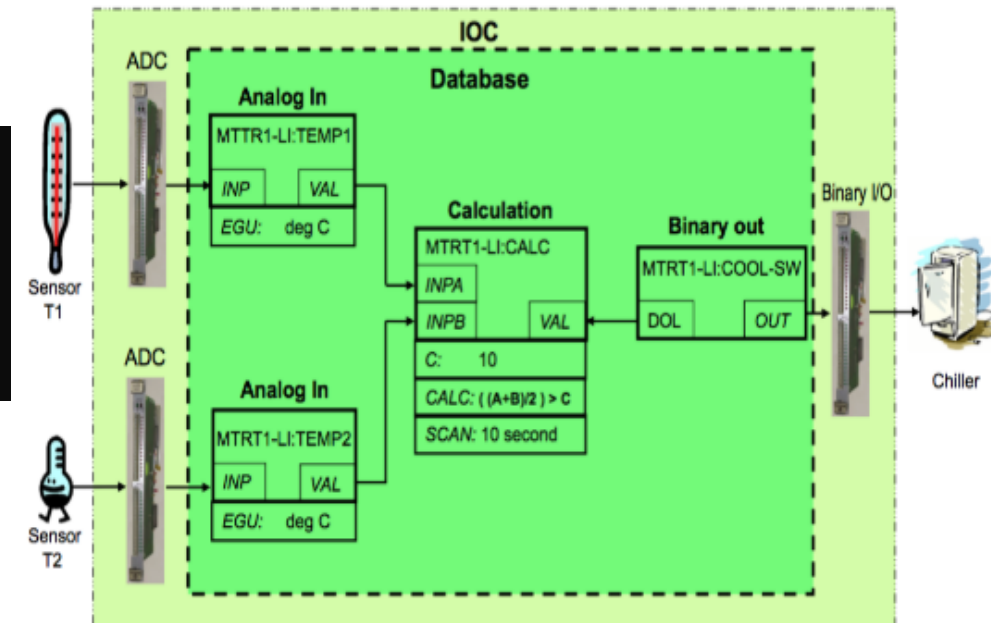
Server's IP

```
(sirius) ~$ cainfo LA-BI:PRF4:RAW:ArrayData
LA-BI:PRF4:RAW:ArrayData
State: connected
Host: 10.0.38.59:62000
Access: read, no write
Native data type: DBF_CHAR
Request type: DBR_CHAR
Element count: 5036408
```

Gets general
info of a PV

This is an **image**, but the **PV is a 1D array**.
Additional info must be provided by other
record for image reconstruction (EPICS 3).

Can be set either with the
integer or the string value



EPICS Servers

- On the server side, the IOC building blocks are records that are mapped into CA PVs by the server layer;
- There are several types of records, to perform different tasks:
 - Analog/Long/Binary/String Input: read data from hardware (they differ on the data type that can be handled);
 - Analog/Long/Binary/String Output: send data to hardware;
 - Calculation Record (Output): value based on state of other records (and have conditional updates);
 - Waveform: stores waveforms of data (only 1D arrays);
 - Multi-Bit Binary Input/Output: trigger tasks based on up to 16 choices (ENUMS)
 - ...
- Each **record type** have a different **set of fields**, and different **data types** (char/int/long, float/double, string)
- Combining them in an IOC allows **complex logic with very little (conventional) programming**

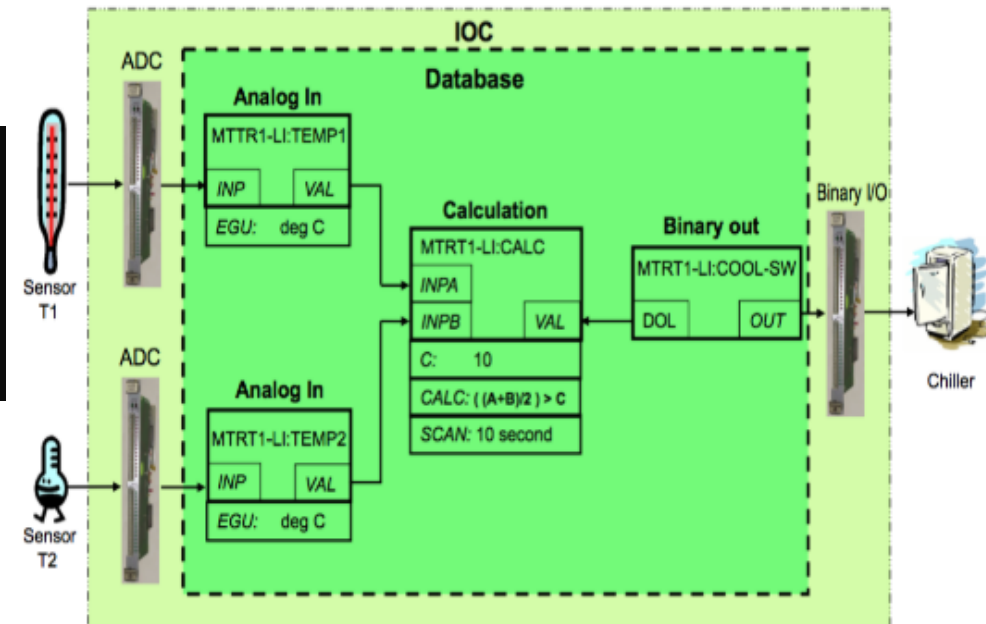
```
(sirius) ~$ caget -d 31 SI-Fam:TI-BPM:State-Sel
SI-Fam:TI-BPM:State-Sel
Native data type: DBF_ENUM
Request type: DBR_CTRL_ENUM
Element count: 1
Value: Enbl
Status: NO_ALARM
Severity: NO_ALARM
Enums: ( 2)
       [ 0] Dsbl
       [ 1] Enbl
```

Server's IP

Gets general
info of a PV

```
(sirius) ~$ cainfo LA-BI:PRF4:RAW:ArrayData
LA-BI:PRF4:RAW:ArrayData
State: connected
Host: 10.0.38.59:62000
Access: read, no write
Native data type: DBF_CHAR
Request type: DBR_CHAR
Element count: 5036408
```

This is an **image**, but the **PV is a 1D array**.
Additional info must be provided by other
record for image reconstruction (EPICS 3).



Can be set either with the
integer or the string value

PVs \neq Records



CNPq

PVs \neq Records

PVs	Records
Is the information unit the client receives via a CA (pvAccess) connection.	Are the IOC building blocks, elements of its database.
Has several properties. Different data types have a different set of properties.	Have a set of fields, which varies according to the record type.
Can be used to retrieve information or change state of the records, triggering different computations, or setting hardware.	Can be combined to perform complex logic. Can also be managed via a finite state machine controller (sequencer) to enhance their computational features.

PVs ≠ Records

PVs	Records
Is the information unit the client receives via a CA (pvAccess) connection.	Are the IOC building blocks, elements of its database.
Has several properties. Different data types have a different set of properties.	Have a set of fields, which varies according to the record type.
Can be used to retrieve information or change state of the records, triggering different computations, or setting hardware.	Can be combined to perform complex logic. Can also be managed via a finite state machine controller (sequencer) to enhance their computational features.

```
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP.VAL
SI-01M2:PS-FCH:Current-SP.VAL  0
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP
SI-01M2:PS-FCH:Current-SP      0
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP.DRVH
SI-01M2:PS-FCH:Current-SP.DRVH 0.95
```

```
record(ao,"SI-01M2:PS-FCH:Current-SP"){
    field(DESC,"set manual current control")
    field(EGU, "A")
    field(SCAN,"Passive")
    field(PINI,"YES")
    field(DRVH,"0.95")
    field(DRVL,"-0.95")
}
```


PVs ≠ Records

PVs	Records
Is the information unit the client receives via a CA (pvAccess) connection.	Are the IOC building blocks, elements of its database.
Has several properties. Different data types have a different set of properties.	Have a set of fields, which varies according to the record type.
Can be used to retrieve information or change state of the records, triggering different computations, or setting hardware.	Can be combined to perform complex logic. Can also be managed via a finite state machine controller (sequencer) to enhance their computational features.

record name

```
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP.VAL
SI-01M2:PS-FCH:Current-SP.VAL  0
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP
SI-01M2:PS-FCH:Current-SP      0
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP.DRVH
SI-01M2:PS-FCH:Current-SP.DRVH 0.95
```

```
record(ao,"SI-01M2:PS-FCH:Current-SP"){
    field(DESC,"set manual current control")
    field(EGU, "A")
    field(SCAN,"Passive")
    field(PINI,"YES")
    field(DRVH,"0.95")
    field(DRVL,"-0.95")
}
```


PVs ≠ Records



PVs	Records
Is the information unit the client receives via a CA (pvAccess) connection.	Are the IOC building blocks, elements of its database.
Has several properties. Different data types have a different set of properties.	Have a set of fields, which varies according to the record type.
Can be used to retrieve information or change state of the records, triggering different computations, or setting hardware.	Can be combined to perform complex logic. Can also be managed via a finite state machine controller (sequencer) to enhance their computational features.

record name

field name

```
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP.VAL
SI-01M2:PS-FCH:Current-SP.VAL 0
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP
SI-01M2:PS-FCH:Current-SP 0
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP.DRVH
SI-01M2:PS-FCH:Current-SP.DRVH 0.95
```

```
record(ao,"SI-01M2:PS-FCH:Current-SP"){
    field(DESC,"set manual current control")
    field(EGU, "A")
    field(SCAN,"Passive")
    field(PINI,"YES")
    field(DRVH,"0.95")
    field(DRVL,"-0.95")
}
```


PVs ≠ Records

PVs	Records
Is the information unit the client receives via a CA (pvAccess) connection.	Are the IOC building blocks, elements of its database.
Has several properties. Different data types have a different set of properties.	Have a set of fields, which varies according to the record type.
Can be used to retrieve information or change state of the records, triggering different computations, or setting hardware.	Can be combined to perform complex logic. Can also be managed via a finite state machine controller (sequencer) to enhance their computational features.

PV name

record name

field name

```
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP.VAL
SI-01M2:PS-FCH:Current-SP.VAL 0
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP
SI-01M2:PS-FCH:Current-SP 0
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP.DRVH
SI-01M2:PS-FCH:Current-SP.DRVH 0.95
```

```
record(ao,"SI-01M2:PS-FCH:Current-SP"){
    field(DESC,"set manual current control")
    field(EGU, "A")
    field(SCAN,"Passive")
    field(PINI,"YES")
    field(DRVH,"0.95")
    field(DRVL,"-0.95")
}
```


PVs ≠ Records

PVs	Records
Is the information unit the client receives via a CA (pvAccess) connection.	Are the IOC building blocks, elements of its database.
Has several properties. Different data types have a different set of properties.	Have a set of fields, which varies according to the record type.
Can be used to retrieve information or change state of the records, triggering different computations, or setting hardware.	Can be combined to perform complex logic. Can also be managed via a finite state machine controller (sequencer) to enhance their computational features.

PV name

record name

field name

```
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP.VAL
SI-01M2:PS-FCH:Current-SP.VAL 0
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP
SI-01M2:PS-FCH:Current-SP 0
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP.DRVH
SI-01M2:PS-FCH:Current-SP.DRVH 0.95
```

```
record(ao,"SI-01M2:PS-FCH:Current-SP"){
  field(DESC,"set manual current control")
  field(EGU, "A")
  field(SCAN,"Passive")
  field(PINI,"YES")
  field(DRVH,"0.95")
  field(DRVL,"-0.95")
}
```


PVs ≠ Records



PVs	Records
Is the information unit the client receives via a CA (pvAccess) connection.	Are the IOC building blocks, elements of its database.
Has several properties. Different data types have a different set of properties.	Have a set of fields, which varies according to the record type.
Can be used to retrieve information or change state of the records, triggering different computations, or setting hardware.	Can be combined to perform complex logic. Can also be managed via a finite state machine controller (sequencer) to enhance their computational features.

PV name

record name

field name

```
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP.VAL
SI-01M2:PS-FCH:Current-SP.VAL 0
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP
SI-01M2:PS-FCH:Current-SP 0
(sirius) ~$ caget SI-01M2:PS-FCH:Current-SP.DRVH
SI-01M2:PS-FCH:Current-SP.DRVH 0.95
```

```
record(ao,"SI-01M2:PS-FCH:Current-SP"){
  field(DESC,"set manual current control")
  field(EGU, "A")
  field(SCAN,"Passive")
  field(PINI,"YES")
  field(DRVH,"0.95")
  field(DRVL,"-0.95")
}
```

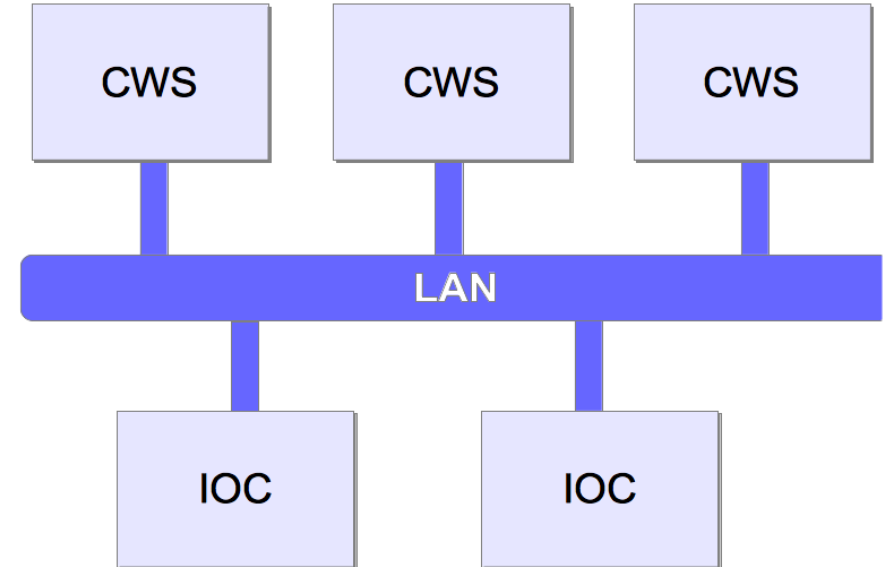
Each record field is a separate PV with its own properties and read/write access control. “VAL” is the default field.

CA (and pvAccess) Basics

CWS: Client workstation.

IOC: Input/Output Controller. PVs server.

LAN: Communication network.



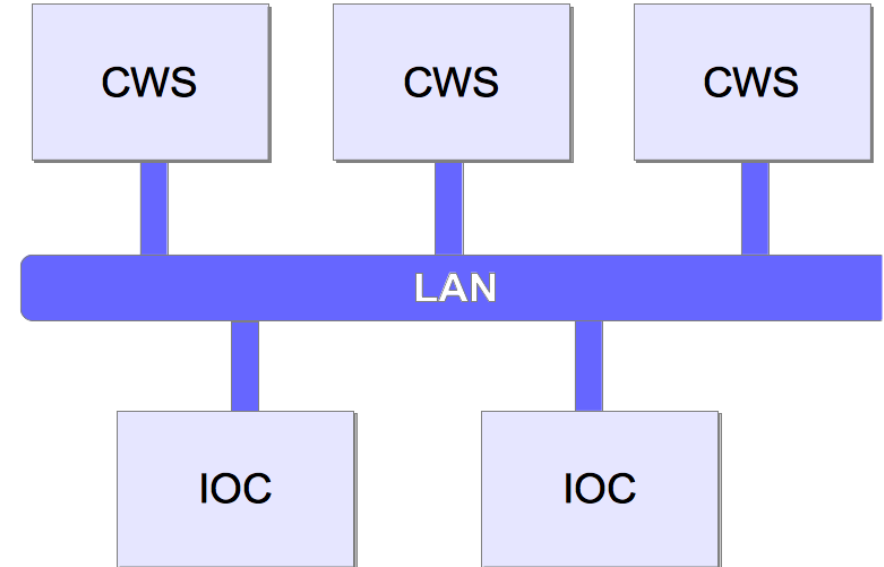
CA (and pvAccess) Basics

The connection process works this way:

CWS: Client workstation.

IOC: Input/Output Controller. PVs server.

LAN: Communication network.



CA (and pvAccess) Basics

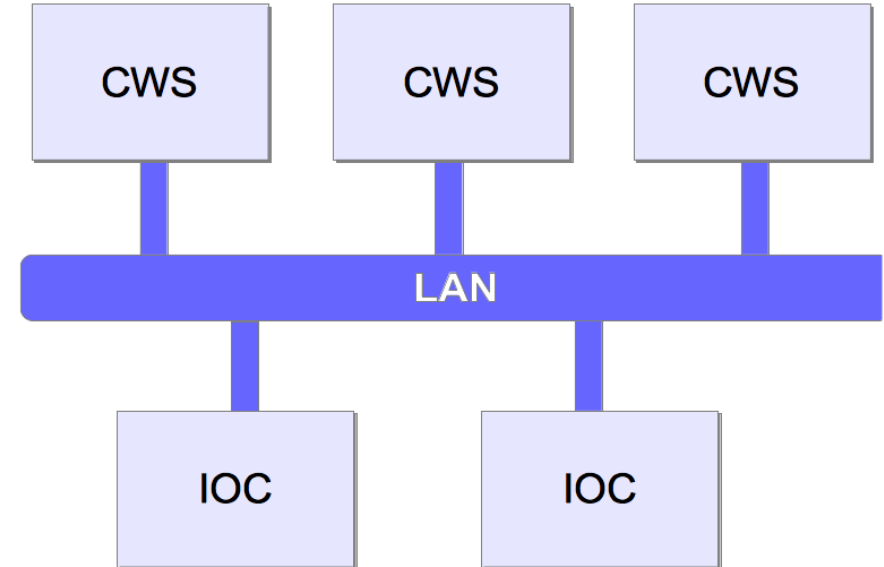
The connection process works this way:

- The client sends a broadcast in the network “asking which IOC serves a given PV name”;

CWS: Client workstation.

IOC: Input/Output Controller. PVs server.

LAN: Communication network.



CA (and pvAccess) Basics

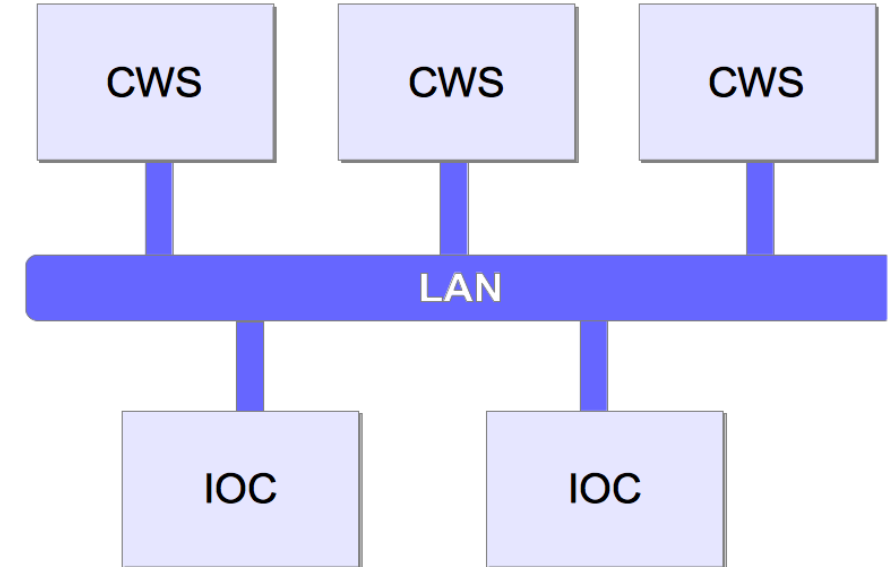
The connection process works this way:

- The client sends a broadcast in the network “asking which IOC serves a given PV name”;
- The IOC that serves that PV responds to the client “I do.”

CWS: Client workstation.

IOC: Input/Output Controller. PVs server.

LAN: Communication network.



CA (and pvAccess) Basics

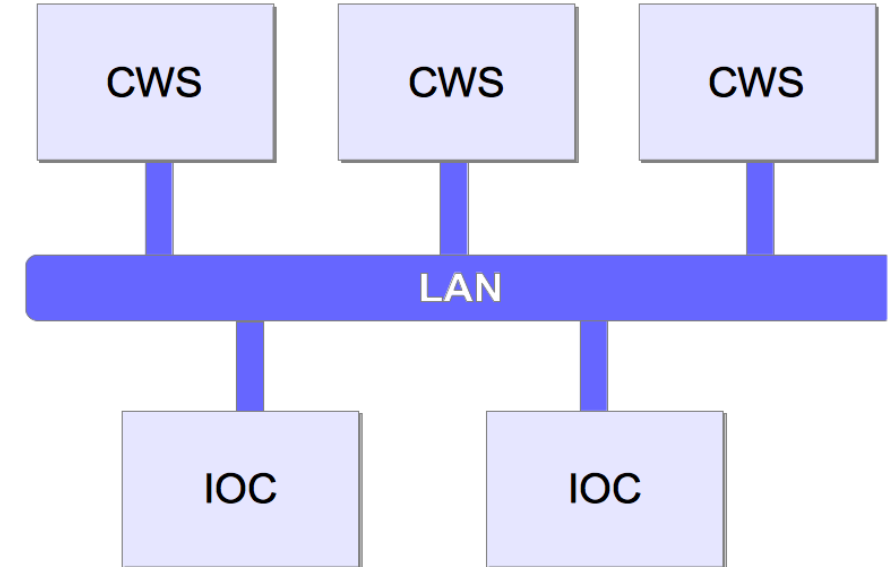
The connection process works this way:

- The client sends a broadcast in the network “asking which IOC serves a given PV name”;
- The IOC that serves that PV responds to the client “I do.”
- A virtual circuit (TCP connection) is established between them, and a channel of communication is created for that PV;

CWS: Client workstation.

IOC: Input/Output Controller. PVs server.

LAN: Communication network.



CA (and pvAccess) Basics

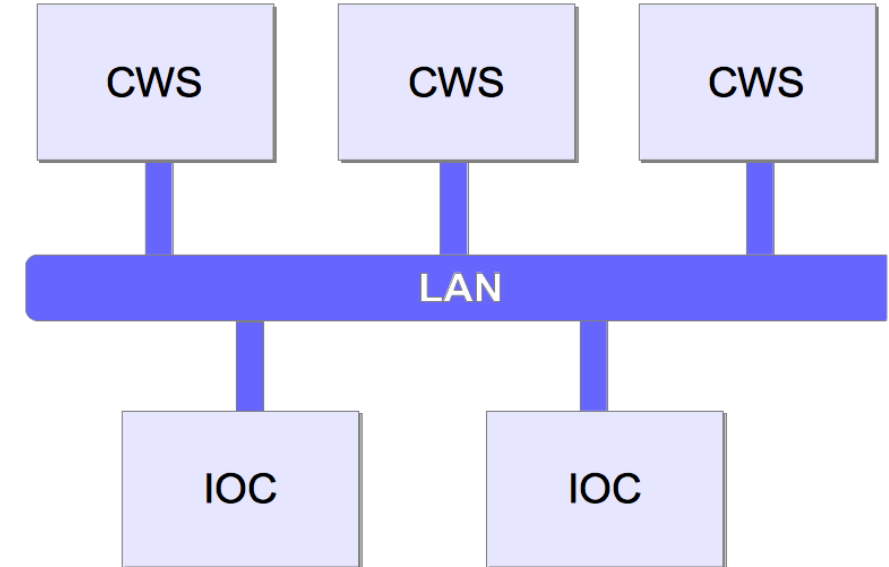
The connection process works this way:

- The client sends a broadcast in the network “asking which IOC serves a given PV name”;
- The IOC that serves that PV responds to the client “I do.”
- A virtual circuit (TCP connection) is established between them, and a channel of communication is created for that PV;
- Existing virtual circuits are reused to minimize the number of TCP connections. New channels are added to these circuits.

CWS: Client workstation.

IOC: Input/Output Controller. PVs server.

LAN: Communication network.



CA (and pvAccess) Basics

The connection process works this way:

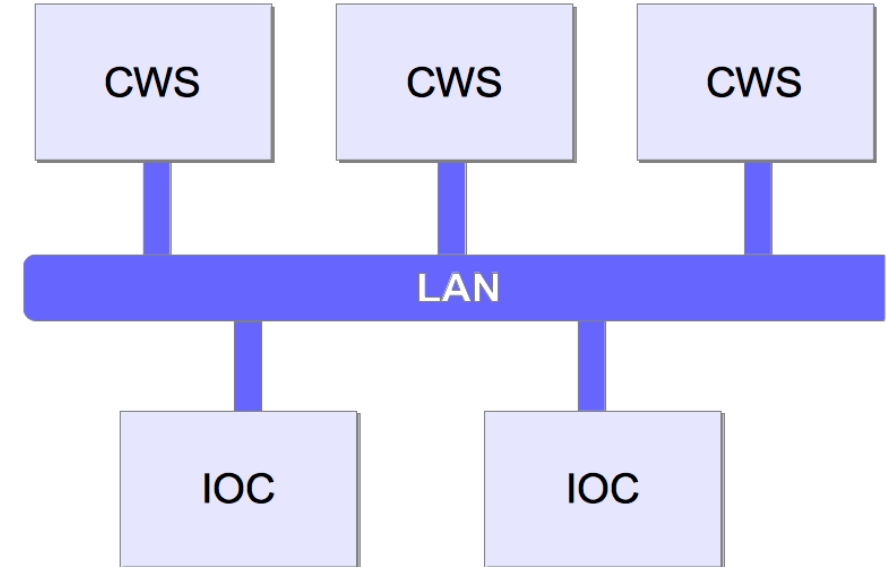
- The client sends a broadcast in the network “asking which IOC serves a given PV name”;
- The IOC that serves that PV responds to the client “I do.”
- A virtual circuit (TCP connection) is established between them, and a channel of communication is created for that PV;
- Existing virtual circuits are reused to minimize the number of TCP connections. New channels are added to these circuits.

After the connection is established, **get** and **put** requests can be issued. The client can also register that PV **to receive asynchronous updates** of new values or alarm states.

CWS: Client workstation.

IOC: Input/Output Controller. PVs server.

LAN: Communication network.



CA (and pvAccess) Basics

The connection process works this way:

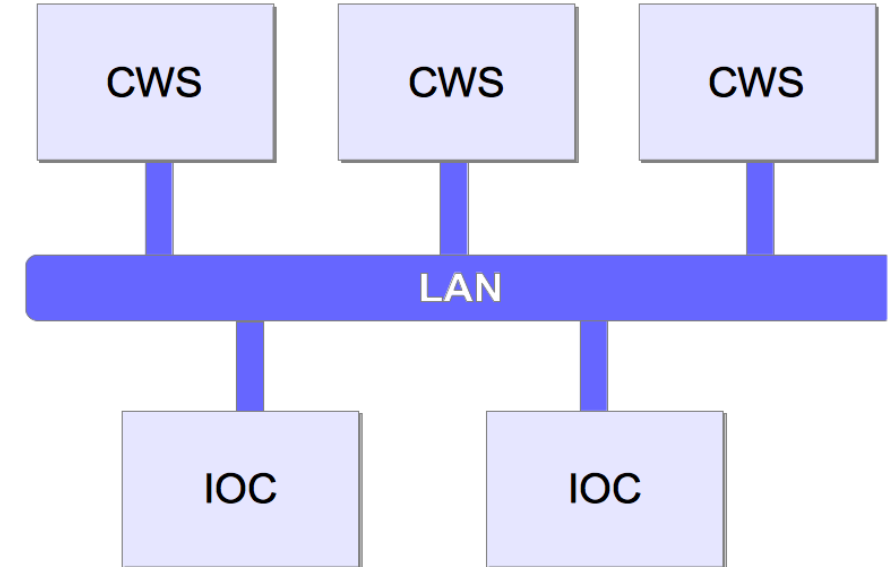
- The client sends a broadcast in the network “asking which IOC serves a given PV name”;
- The IOC that serves that PV responds to the client “I do.”
- A virtual circuit (TCP connection) is established between them, and a channel of communication is created for that PV;
- Existing virtual circuits are reused to minimize the number of TCP connections. New channels are added to these circuits.

After the connection is established, **get** and **put** requests can be issued. The client can also register that PV **to receive asynchronous updates** of new values or alarm states.

CWS: Client workstation.

IOC: Input/Output Controller. PVs server.

LAN: Communication network.



```
(sirius) ~$ camonitor SI-01M2:PS-CH:Current-Mon
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:41.606582 2.09092
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:41.807166 2.08214
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:42.006755 2.0932
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:42.207250 2.08853
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:42.406955 2.08996
```


CA (and pvAccess) Basics

The connection process works this way:

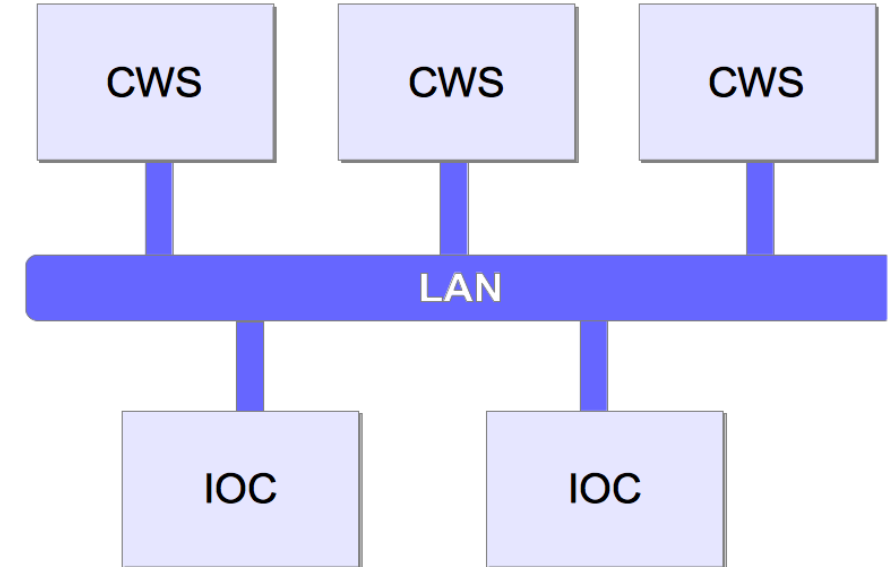
- The client sends a broadcast in the network “asking which IOC serves a given PV name”;
- The IOC that serves that PV responds to the client “I do.”
- A virtual circuit (TCP connection) is established between them, and a channel of communication is created for that PV;
- Existing virtual circuits are reused to minimize the number of TCP connections. New channels are added to these circuits.

After the connection is established, **get** and **put** requests can be issued. The client can also register that PV **to receive asynchronous updates** of new values or alarm states.

CWS: Client workstation.

IOC: Input/Output Controller. PVs server.

LAN: Communication network.



Creates a connection and register the PV for value updates

```
(sirius) ~$ camonitor SI-01M2:PS-CH:Current-Mon
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:41.606582 2.09092
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:41.807166 2.08214
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:42.006755 2.0932
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:42.207250 2.08853
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:42.406955 2.08996
```


CA (and pvAccess) Basics

The connection process works this way:

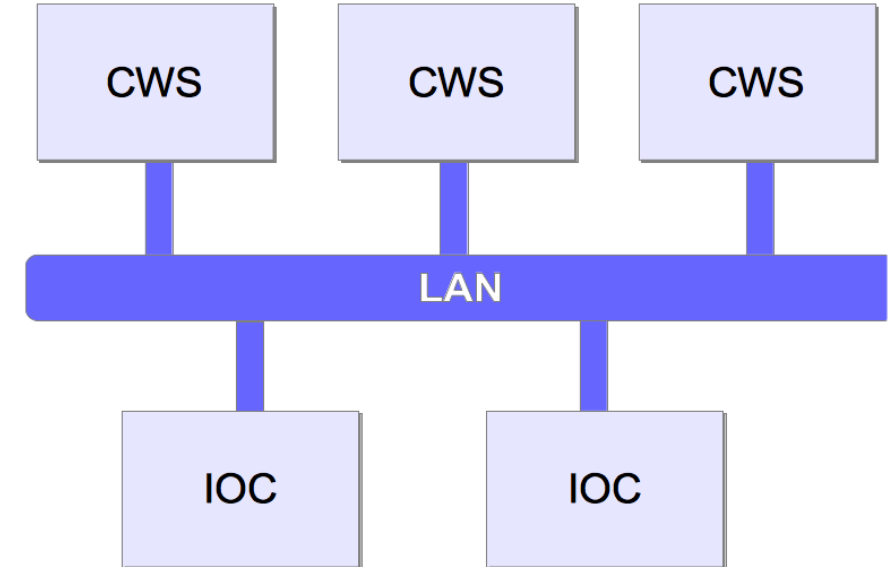
- The client sends a broadcast in the network “asking which IOC serves a given PV name”;
- The IOC that serves that PV responds to the client “I do.”
- A virtual circuit (TCP connection) is established between them, and a channel of communication is created for that PV;
- Existing virtual circuits are reused to minimize the number of TCP connections. New channels are added to these circuits.

After the connection is established, **get** and **put** requests can be issued. The client can also register that PV **to receive asynchronous updates** of new values or alarm states.

CWS: Client workstation.

IOC: Input/Output Controller. PVs server.

LAN: Communication network.



Creates a connection and register the PV for value updates

```
(sirius) ~$ camonitor SI-01M2:PS-CH:Current-Mon
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:41.606582 2.09092
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:41.807166 2.08214
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:42.006755 2.0932
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:42.207250 2.08853
SI-01M2:PS-CH:Current-Mon      2025-09-02 12:24:42.406955 2.08996
```

New values are sent by the IOC asynchronously as soon as they change, without the need for the client to request update: latency and network traffic optimization.

EPICS Clients: Archiver

Allows storage of hundreds of thousands of PV with custom sampling rate and storage policies. The web management interface permits controlling archive parameters of each PV and provides general reports.

To check the status of or to archive some PV's, please type in some PV names here.

SI-Fam:PS-QFA:Curr*

Check Status

Archive

Archive (specify sampling period)




Lookup

Pause

Resume

25

Page 1 of 1

PV Name	Status	Appliance	Connected?	Monitored?	Sampling period	Last event	Details	Quick chart
SI-Fam:PS-QFA:Current-Mon	Being archived	Inls_control_appliance_1	true	false	0.1	Sep/02/2025 13:59:04 -03:00		Option #1
SI-Fam:PS-QFA:Current-RB	Being archived	Inls_control_appliance_1	true	false	0.1	Sep/02/2025 03:56:22 -03:00		Option #1
SI-Fam:PS-QFA:Current-SP	Being archived	Inls_control_appliance_1	true	false	0.1	Sep/02/2025 03:56:22 -03:00		Option #1

25

Page 1 of 1

Instance Name	Status	PV Count	Event Rate	Data Rate (GB/day)
Inls_control_appliance_1	Working	168512	50,679.6	139.9

Here are the some detailed storage metrics of the appliance Inls_control_appliance_1.

Name	Total space (GB)	Available space (GB)	Available space (%)
STS	251.81	243.32	96.63
MTS	59,596.32	30,153.17	50.6
LTS	59,596.32	30,153.17	50.6

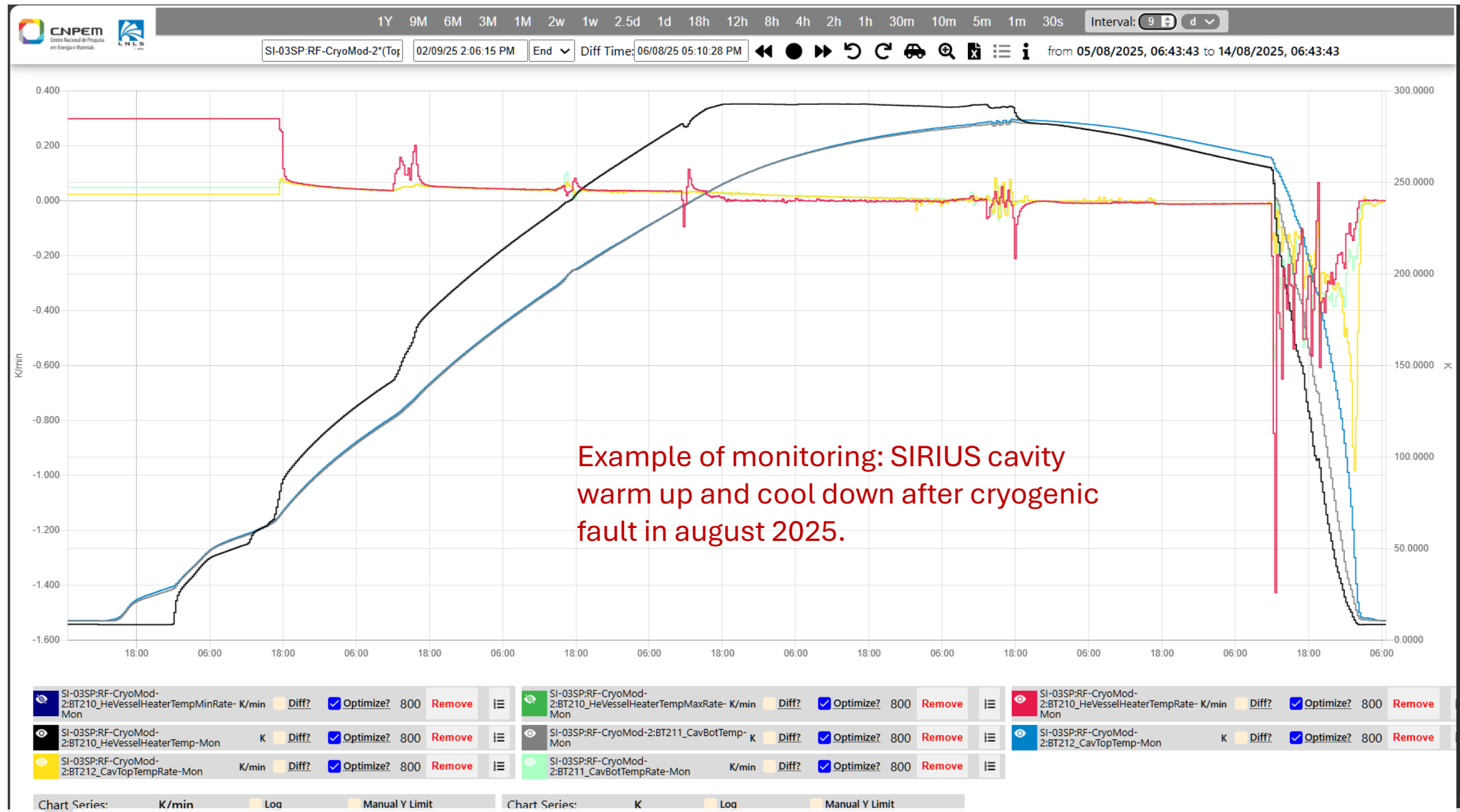
The web server has a REST API, so it is possible to control the archived PVs directly via *urls*. For instance, we can retrieve stored data:

<https://ais-eng-srv-ta.cnpem.br/retrieval/data/getData.json?pv=SI-01M1:PS-CH:Current-Mon&from=2025-08-10T00%3A00%3A00-03%3A00&to=2025-08-10T05%3A00%3A00-03%3A00>

This type of API for *url* construction can easily be wrapped with some python code:

```
In [1]: from siriuspy.clientarch import PVDataSet, Time
...: pvds = PVDataSet([
...:     'SI-01M1:PS-CH:Current-Mon',
...:     'SI-01M2:PS-CH:Current-Mon',
...:     'SI-01M1:DI-BPM:PosX-Mon',
...:     'SI-01M2:DI-BPM:PosX-Mon',
...: ])
...: pvds.time_start = Time(2025, 8, 10, 0, 0, 0)
...: pvds.time_stop = pvds.time_start + 5 * 60 * 60 # five hours later
...: pvds.update()
```


EPICS Clients: Archiver Viewer



EPICS Clients: PyEpics, caproto, Pvapy...



CNPEM

```
In [40]: from epics import PV

In [41]: pv = PV('SI-01M2:PS-CH:Current-Mon')

In [42]: pv.connected
Out[42]: True

In [43]: pv.value
Out[43]: 2.079660415649414

In [44]: pv.pvname
Out[44]: 'SI-01M2:PS-CH:Current-Mon'

In [45]: pv.timestamp
Out[45]: 1756841365.165316

In [46]: pv.auto_monitor
Out[46]: True

In [47]: def print_value(pvname, value, timestamp, **kwargs):
...:     print(pvname, value)
...:

In [48]: pv.add_callback(print_value)
Out[48]: 1

SI-01M2:PS-CH:Current-Mon 2.0806140899658203
SI-01M2:PS-CH:Current-Mon 2.0884361267089844
SI-01M2:PS-CH:Current-Mon 2.085383415222168
SI-01M2:PS-CH:Current-Mon 2.083285331726074
SI-01M2:PS-CH:Current-Mon 2.087386131286621
SI-01M2:PS-CH:Current-Mon 2.0803279876708984
SI-01M2:PS-CH:Current-Mon 2.0884361267089844
In [49]: pv.clear_callbacks()
```

PyEpics provides full control and monitoring of PVs properties, including the asynchronous update functionality.

```
In [37]: atts = [f'{att:25s}' for att in dir(pv) if not att.startswith('_')]

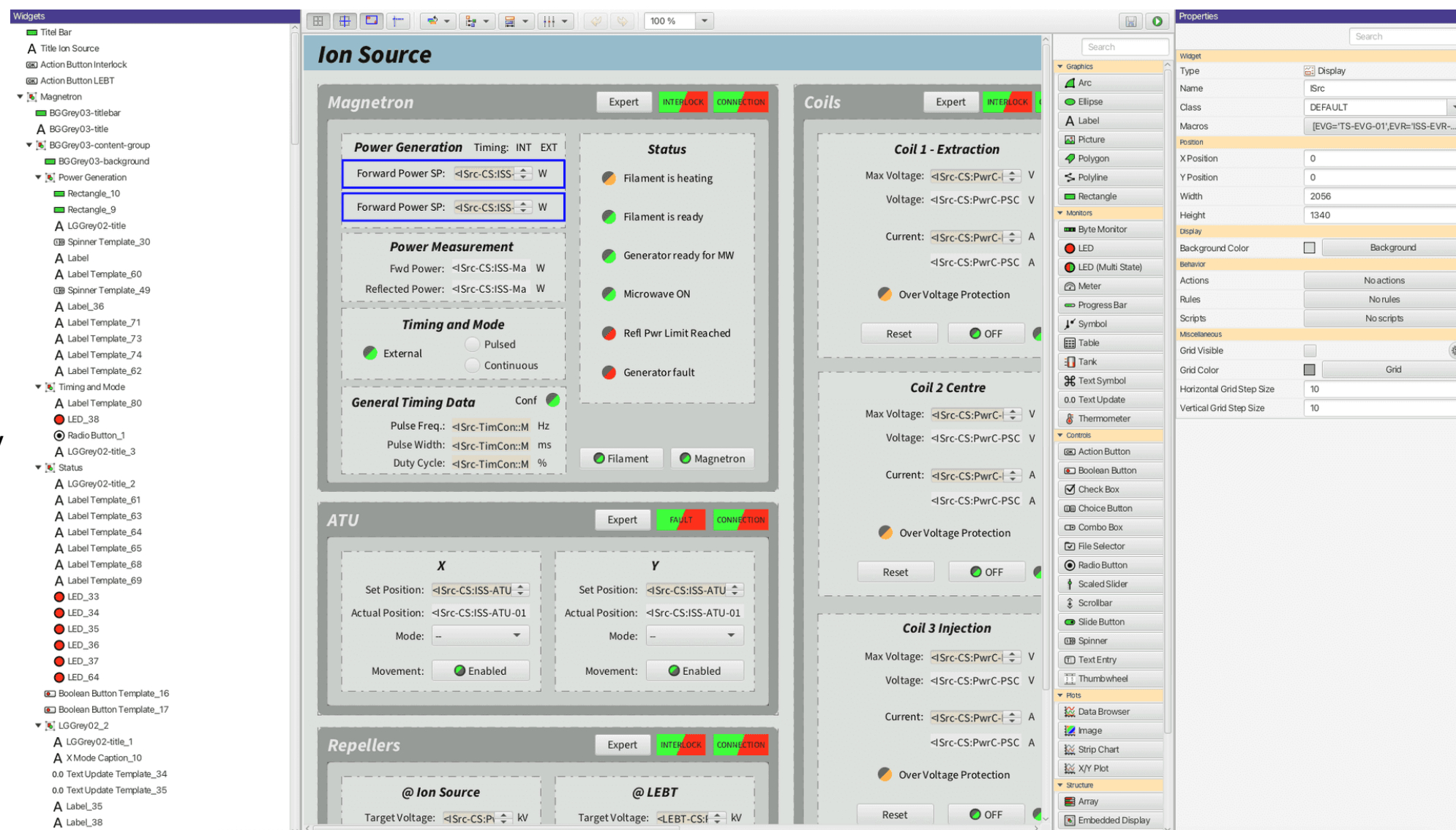
In [38]: _ = [print(''.join(atts[i:i+3])) for i in range(0, len(atts), 3)]
access                access_callbacks      add_callback
auto_monitor          auto_monitor_mask      callbacks
char_severity         char_status            char_value
chid                  clear_auto_monitor     clear_callbacks
connect               connected              connection_callbacks
connection_timeout    context                count
disconnect            enum_strs              force_connect
force_read_access_rights form                    ftype
get                   get_ctrlvars           get_timevars
get_with_metadata     host                   info
lower_alarm_limit     lower_ctrl_limit       lower_disp_limit
lower_warning_limit   nanoseconds            nelm
poll                  posixseconds           precision
put                   put_complete           pvname
read_access           reconnect              remove_callback
run_callback          run_callbacks          severity
status                timestamp              type
typefull              units                  upper_alarm_limit
upper_ctrl_limit      upper_disp_limit       upper_warning_limit
value                 verbose                wait_for_connection
write_access
```


EPICS Clients: Graphical User Interfaces



CS-studio:

- Largely used in the EPICS community;
- Implemented in Java;
- Extensive list of widgets;
- Heavy use of threading;
- Drag and drop display builder;
- Alarms table and integration with Logbooks and Archiver;
- Save and Restore feature.



EPICS Clients: Graphical User Interfaces



PyDM:

- Implemented in Python, making use of PyQt;
- Easy integration with python libraries, advanced scripting and complex interfaces;
- Extensive list of widgets;
- Drag and drop display builder integrated with QtDesigner;
- EPICS and Archiver plugins;
- Most of SIRIUS GUIs were made with PyDM.

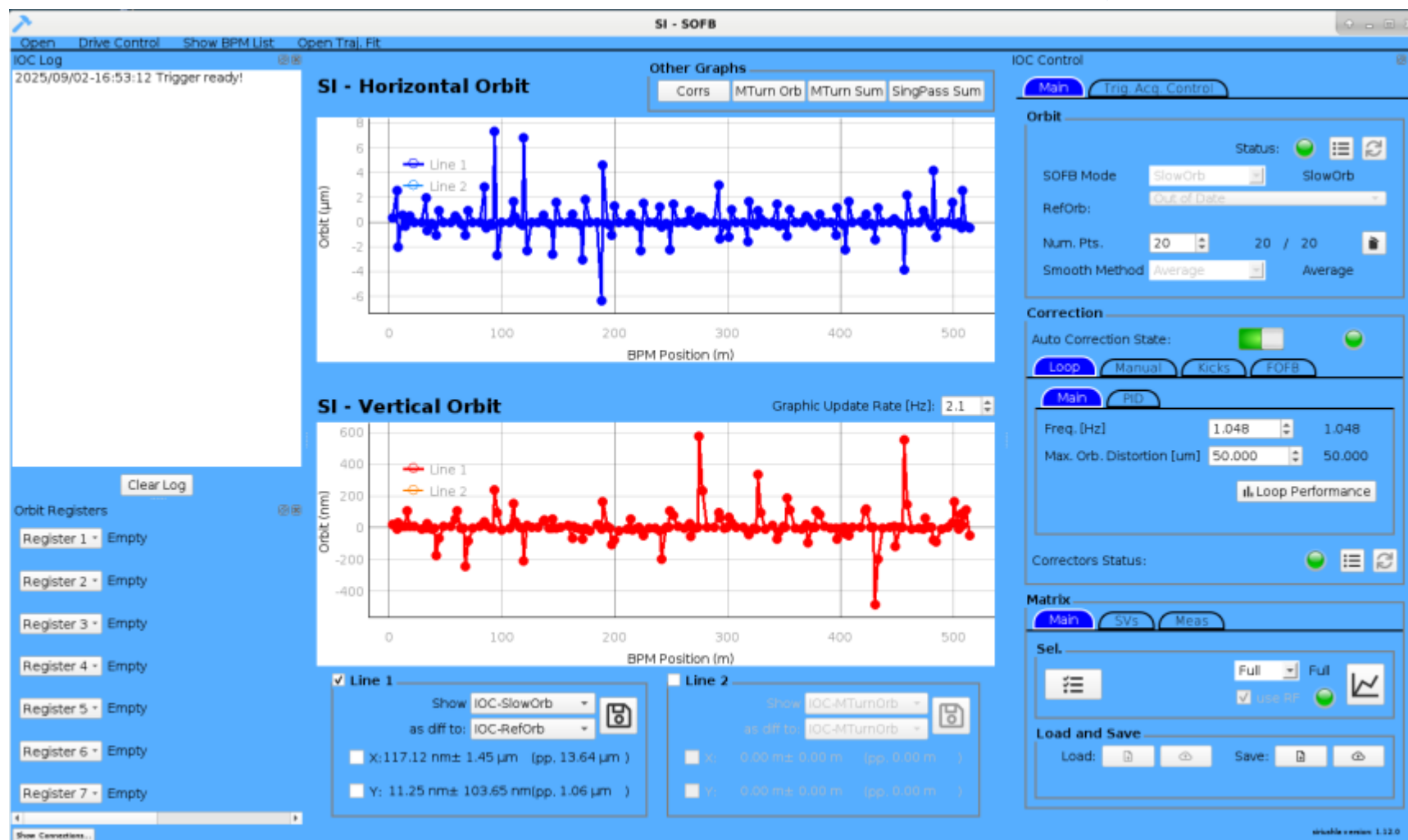
EPICS Clients: Graphical User Interfaces



CNPEM

PyDM:

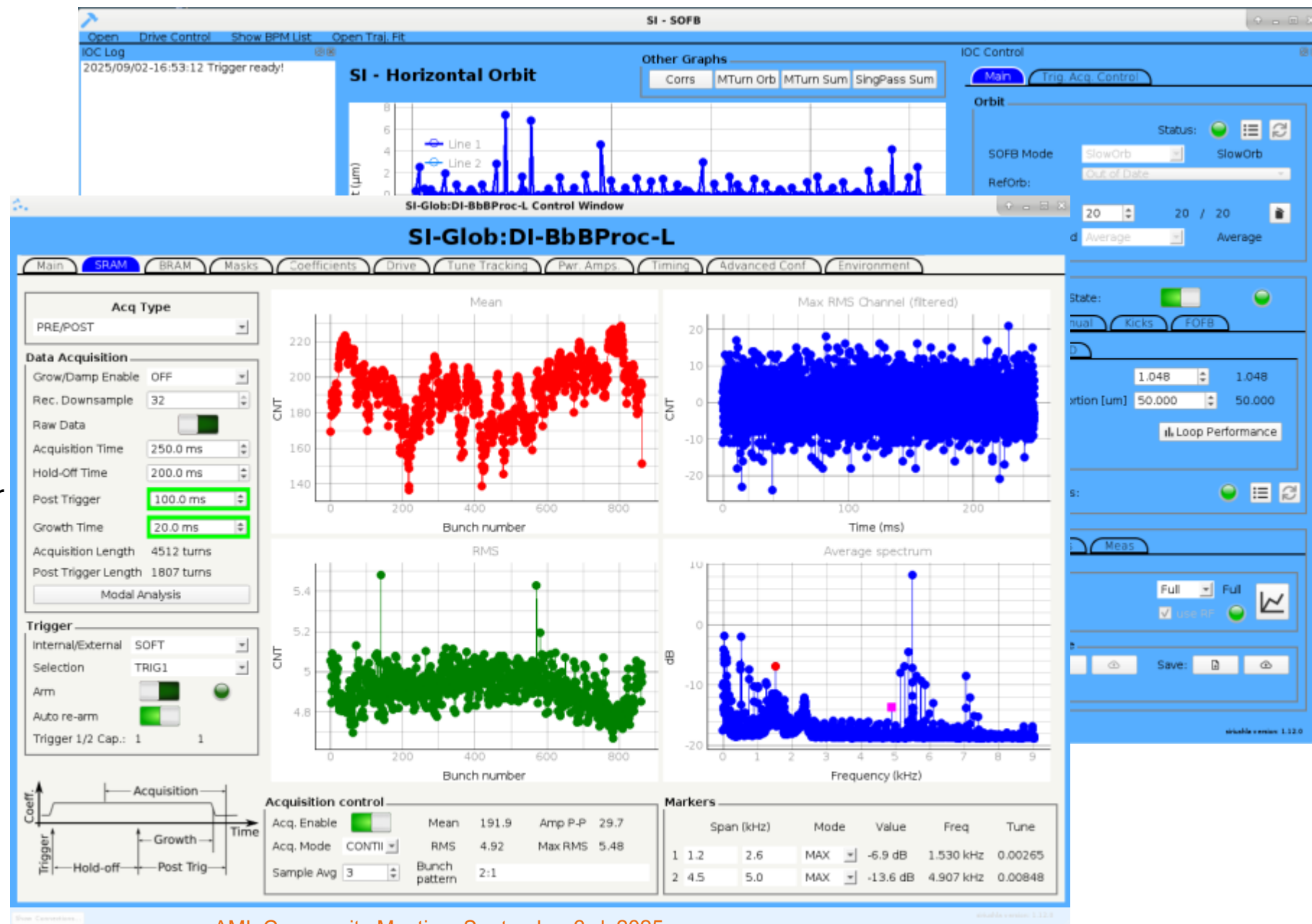
- Implemented in Python, making use of PyQt;
- Easy integration with python libraries, advanced scripting and complex interfaces;
- Extensive list of widgets;
- Drag and drop display builder integrated with QtDesigner;
- EPICS and Archiver plugins;
- Most of SIRIUS GUIs were made with PyDM.



EPICS Clients: Graphical User Interfaces

PyDM:

- Implemented in Python, making use of PyQt;
- Easy integration with python libraries, advanced scripting and complex interfaces;
- Extensive list of widgets;
- Drag and drop display builder integrated with QtDesigner;
- EPICS and Archiver plugins;
- Most of SIRIUS GUIs were made with PyDM.



- PCASpy allows writing EPICS IOCs in python;
- Easy to use. Even users with little knowledge of EPICS (such as myself), can write their own IOCs;
- Support for most of the PV data types;
- IOC programming must be done in python: no records to embed logic;
- Good option to create IOCs to provide abstraction layers to the control system (soft IOCs), such as:
 - Configurations data;
 - Accelerators model data;
 - Implementation of measurement scripts;
 - Slow orbit correction (performance bottlenecks in traditional IOC);
 - PVs translations and unit conversions;
 - Hardware abstraction IOCs;
 - ...
- Most of SIRIUS soft IOCs and all power supplies IOCs were implemented with PCASpy.

```
1  #!/usr/bin/env python
2
3  from pcaspy import Driver, SimpleServer
4
5  prefix = 'MTEST:'
6  ▾ pvdb = {
7      'RAND' : {
8          'prec' : 3,
9      },
10 }
11
12 class myDriver(Driver):
13     def __init__(self):
14         super(myDriver, self).__init__()
15
16 if __name__ == '__main__':
17     server = SimpleServer()
18     server.createPV(prefix, pvdb)
19     driver = myDriver()
20
21     # process CA transactions
22     while True:
23         server.process(0.1)
```


[Getting started with EPICS — EPICS Documentation](#)

[EPICS R3.15 Channel Access Reference Manual](#)

[Channel Access Protocol Specification](#)

[Channel Access](#)

[EPICS Archiver Appliance — archiverdocs 0.1 documentation](#)

[Epics Channel Access for Python — Epics Channel Access for Python](#)

[caproto: a pure-Python Channel Access protocol library — caproto 1.2.0 documentation](#)

[Control System Studio](#)

[PyDM - Python Display Manager — PyDM 1.27.2 documentation](#)

[pythonSoftIOc — pythonSoftIOc 4.5.0+22.g8c4b516 documentation](#)

[PCASpy Documentation — pcaspy 0.8.1 documentation](#)

Thank you for your attention!

Many thanks to Érico Rolim for his valuable feedback on this presentation and for helping me understand key concepts related to EPICS!