

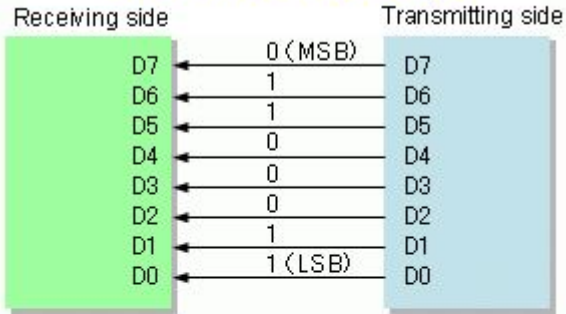
Cómo comunicarse con un instrumento con Python

Nicolás Nuñez Barreto
nnunez@df.uba.ar

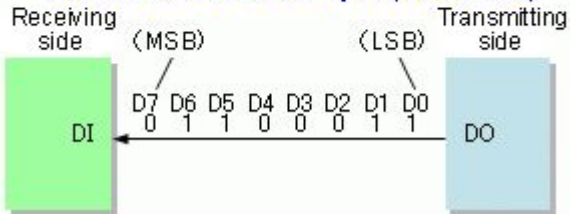






Tipos de comunicación

Parallel interface example

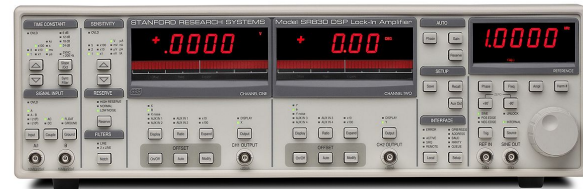


Serial interface example (MSB first)

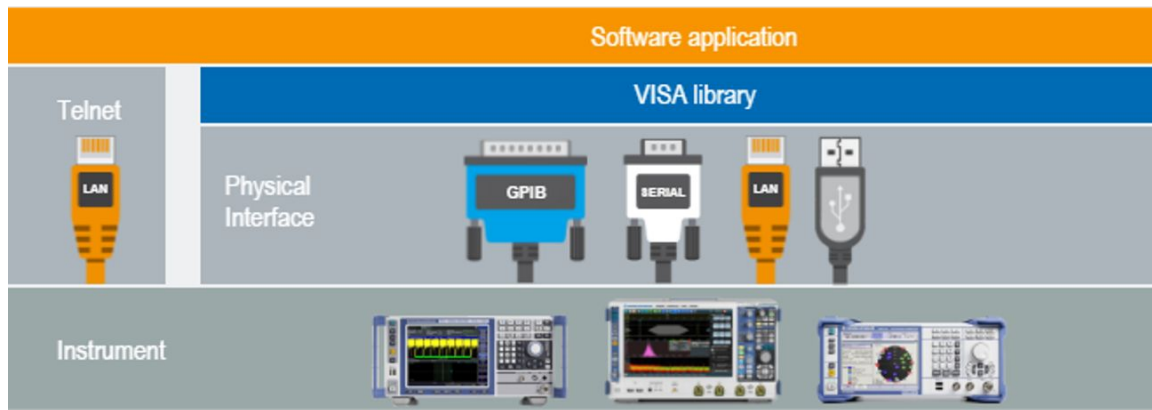


Interface	GPIB	LAN	USB	RS232
				
Bandwidth	1.8Mbit/s (488.1)	12.5 Mbit/s 125 Mbits/s (GB LAN)	60 Mbit/s (Hi-Speed) 120 Mbit/s (USB 3.0)	28.8 kB/s
Latency (informative) *Note1	300 μ s	250 μ s	1000 μ s (USB) 125 μ s (Hi-Speed)	~100 ms

Ej: Lock-In SR830: GPIB



Comunicándonos con Python




PyVISA



**Virtual Instrument
Software Architecture**

Usando el paquete **PyVisa** (<https://pyvisa.readthedocs.io/en/latest/>) podremos controlar casi todo lo que querramos

Otros paquetes más específicos: PySerial, PyUSB, linux-GPIB...

Ejemplo en MUY pocas líneas

```
In [143]: import visa
```

```
In [144]: rm = visa.ResourceManager()
```

```
In [145]: rm.list_resources()
```

```
Out[145]: ('ASRL10::INSTR', 'GPIB0::11::INSTR')
```

```
In [146]: inst = rm.open_resource('GPIB0::11::INSTR')
```

inst es un objeto que representa a la comunicación con el instrumento

¿Qué comandos entiende VISA?

write: le enviamos un mensaje al instrumento

```
In [148]: inst.write('SENS 22')  
Out[148]: (9, <StatusCode.success: 0>)
```

read: si el instrumento mandó un mensaje, lo leemos
(si no mandó nada, obtendremos un timeout)

```
In [149]: inst.read()
```

query: enviar mensaje y leer respuesta luego

```
In [150]: inst.query('*IDN?')  
Out[150]: 'Stanford_Research_Systems,SR830,s/n81296,ver1.07 \n'
```

¿Qué comandos entiende VISA?

Leyendo valores

- **query:**

```
In [154]: inst.query('SNAP? 1, 2')  
Out[154]: '-0.00323679,0.00092316\n'
```

0, dependiendo la configuración del instrumento, podemos especificar el encoding:

- **query_ascii_values**
- **query_binary_values**

¿Qué comandos entiende el instrumento?

¿Qué comandos entiende el instrumento?

Leer manual!

¿Qué comandos entiende el instrumento?

SENS (?) {i}


The SENS command sets or queries the sensitivity. The parameter i selects a sensitivity below.

<u>i</u>	<u>sensitivity</u>	<u>i</u>	<u>sensitivity</u>
0	2 nV/fA	13	50 μ V/pA
1	5 nV/fA	14	100 μ V/pA
2	10 nV/fA	15	200 μ V/pA
3	20 nV/fA	16	500 μ V/pA
4	50 nV/fA	17	1 mV/nA
5	100 nV/fA	18	2 mV/nA
6	200 nV/fA	19	5 mV/nA
7	500 nV/fA	20	10 mV/nA
8	1 μ V/pA	21	20 mV/nA
9	2 μ V/pA	22	50 mV/nA
10	5 μ V/pA	23	100 mV/nA
11	10 μ V/pA	24	200 mV/nA
12	20 μ V/pA	25	500 mV/nA
		26	1 V/ μ A

¿Qué comandos entiende el instrumento?

SENS (?) {i}

The SENS command sets or queries the sensitivity. The parameter i selects a sensitivity below.



i	<u>sensitivity</u>
0	2 nV/fA
1	5 nV/fA
2	10 nV/fA
3	20 nV/fA
4	50 nV/fA
5	100 nV/fA
6	200 nV/fA
7	500 nV/fA
8	1 μV/pA
9	2 μV/pA
10	5 μV/pA
11	10 μV/pA
12	20 μV/pA

permite
preguntarle

i	<u>sensitivity</u>
13	50 μV/pA
14	100 μV/pA
15	200 μV/pA
16	500 μV/pA
17	1 mV/nA
18	2 mV/nA
19	5 mV/nA
20	10 mV/nA
21	20 mV/nA
22	50 mV/nA
23	100 mV/nA
24	200 mV/nA
25	500 mV/nA
26	1 V/μA

- query('SENS?')


pregunta la sensibilidad
y espera la respuesta

¿Qué comandos entiende el instrumento?

SENS (?) {i}

The SENS command sets or queries the sensitivity. The parameter i selects a sensitivity below.

permite
seteo



i	sensitivity
0	2 nV/fA
1	5 nV/fA
2	10 nV/fA
3	20 nV/fA
4	50 nV/fA
5	100 nV/fA
6	200 nV/fA
7	500 nV/fA
8	1 μ V/pA
9	2 μ V/pA
10	5 μ V/pA
11	10 μ V/pA
12	20 μ V/pA

i	sensitivity
13	50 μ V/pA
14	100 μ V/pA
15	200 μ V/pA
16	500 μ V/pA
17	1 mV/nA
18	2 mV/nA
19	5 mV/nA
20	10 mV/nA
21	20 mV/nA
22	50 mV/nA
23	100 mV/nA
24	200 mV/nA
25	500 mV/nA
26	1 V/ μ A

- `query('SENS?')`

pregunta la sensibilidad
y espera la respuesta

- `write('SENS 15')`

setea la sensibilidad
según lo que dice el
manual

Códigos eficientes de comunicación

Opción 1: escribir líneas individuales

`lockin_simple.py`

Códigos eficientes de comunicación

Opción 1: escribir líneas individuales

```
lockin_simple.py
```

Opción 2: armar estructuras de **clases**

```
class SR830:
```

Clases y objetos

```
class Auto:
```

Clase

Auto

```
In [40]: type(Auto)
Out[40]: type
```

Objeto/instancia
de la clase

AutoNico

```
In [41]: AutoNico = Auto(4, 'Azul')
In [42]: type(AutoNico)
Out[42]: __main__.Auto
```

Atributos

- Color
- Ruedas
- Marca
- Precio

```
In [43]: AutoNico.ruedas
Out[43]: 4
```

Métodos/funciones

- Decir la hora
- Pintar el auto
- Agregar una rueda
- Hablarle a otro auto
- Prender/apagar

```
In [44]: AutoNico.SetRuedas(5)
In [45]: AutoNico.GetRuedas()
Out[45]: 5
```

Una clase es un nuevo tipo de objetos (int, float, bool) cuyas propiedades las definimos nosotros

Ejemplo (¿muy?) abstracto

Definiendo una clase

```
10 class ClasePrueba:
11
12     def __init__(self, input1, input2):
13         self.atributo1 = input1
14         self.atributo2 = input2
15
16     def metodo1(self, inputmetodo1):
17         self.atributo1 = self.atributo1 + inputmetodo1
18         return self.atributo1
19
20     def metodo2(self):
21         print('Hola')
```

Obligatorio:

método `__init__(self, argumentos)`
para inicializar el objeto de la clase y
asignarle propiedades o ejecutar funciones

Instanciando (creando un objeto de) una clase

```
In [52]: objetoprueba = ClasePrueba(1, 2)

In [53]: objetoprueba.atributo1
Out[53]: 1

In [54]: objetoprueba.metodo1(3)
Out[54]: 4

In [55]: objetoprueba.metodo2()
Hola
```