eWON
MACHINES CAN TALK

Pierre RIPAK
R&D Firmware Engineer

Benoît COLLIGNON
R&D Firmware Engineer

Acquisition of custom industrial data in an eWON

In order of appearance

Pierre RIPAK
Benoit COLLIGNON
Simon DETOLLENAERE

Acquisition of custom data

Agenda

- What does exist now
  - TAGS & IOServers
  - BASIC
  - JAVA
- What's new
  - BASIC2
  - BASIC IDE
- Workshops

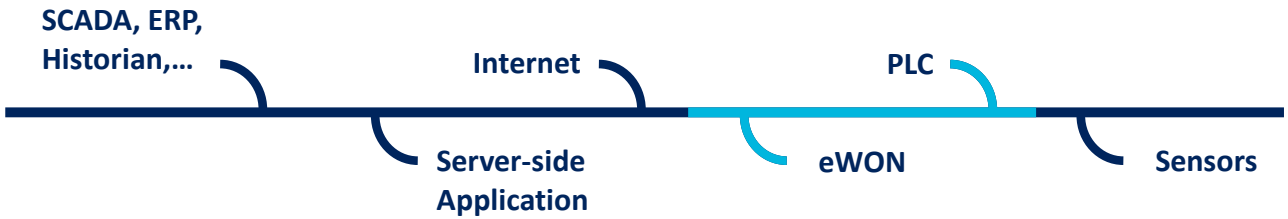
Where are we?

In this presentation, the topic we talk about is:

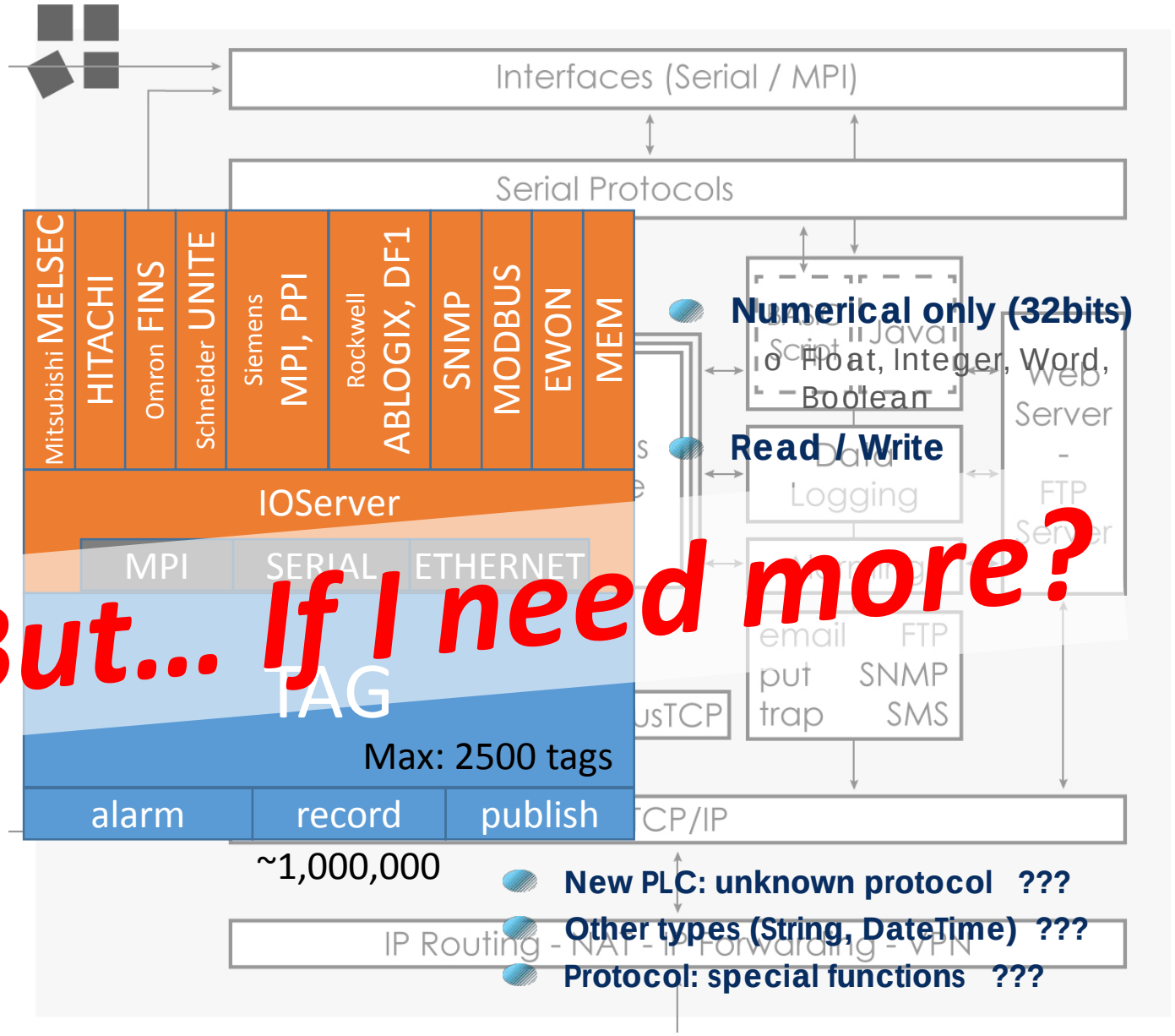
- Data Acquisition

This means we show how to…

- Retrieve data from PLC to eWON
- Build a custom I/O Server



Built-in



But… If I need more?

Built-in
how to do more?

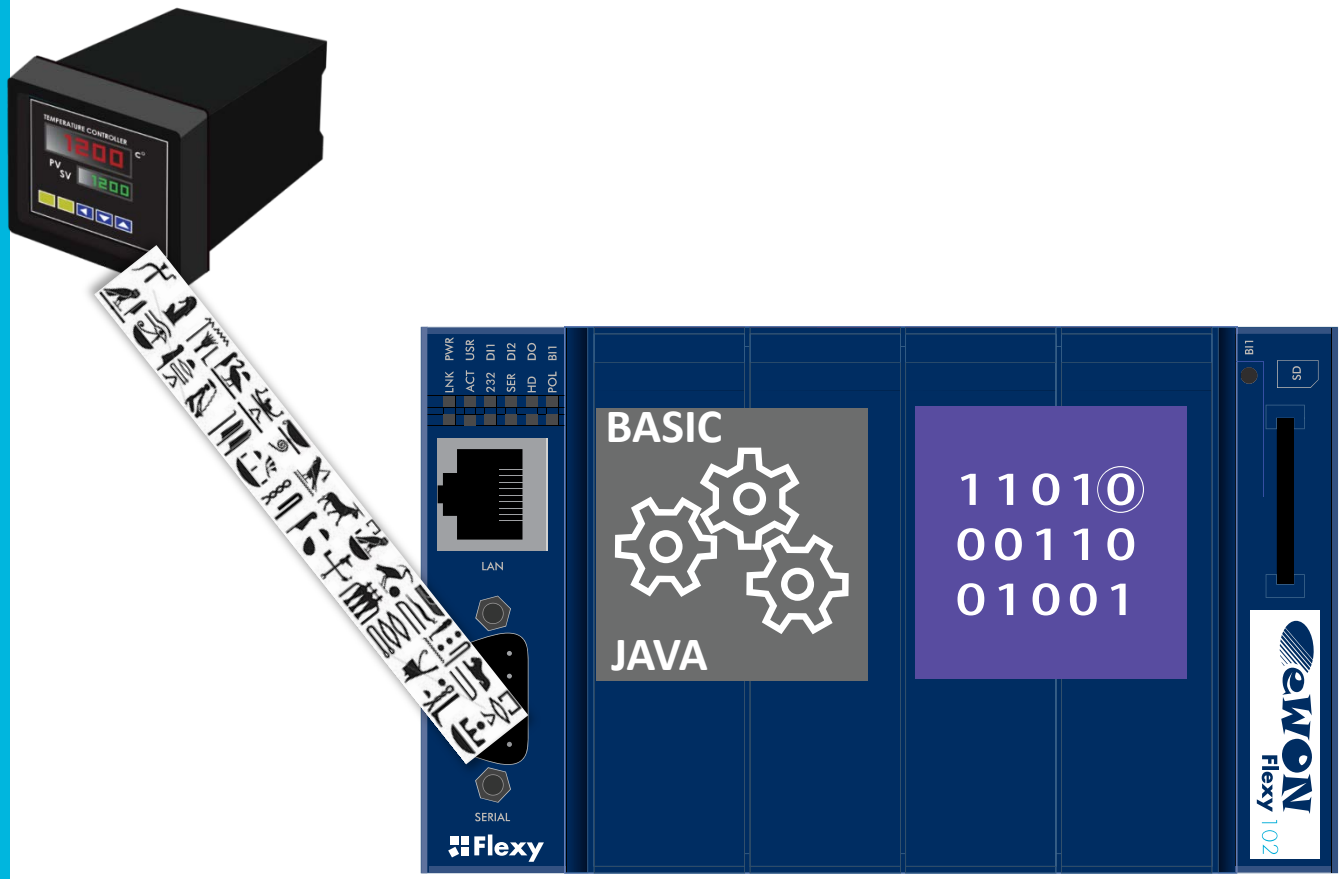
How to do more?

1. **Do it all yourself!!**
   - In Basic or Java
2. **Add IOServer functionalities**
   - in Basic
3. **Create a new IOServer**
   - In Java

New protocol implementation

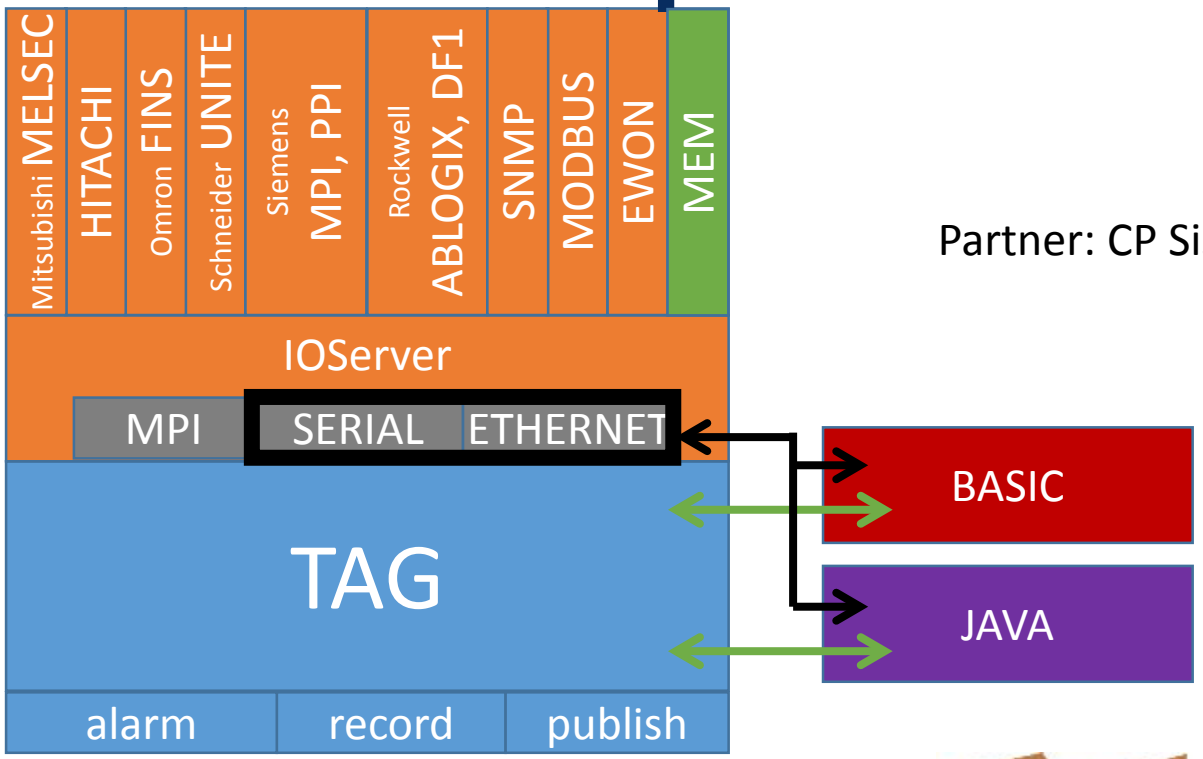
Built-in how to do more?

1. DIY



New protocol implementation

Built-in how to do more?
DIY



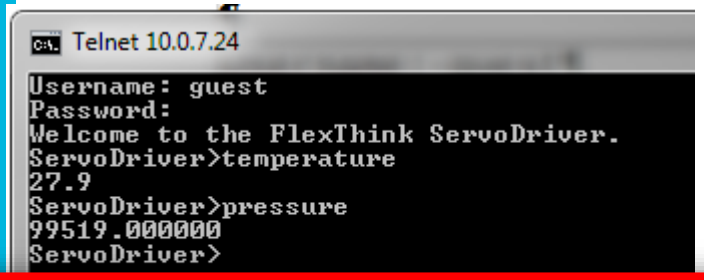
Partner: CP Sistemi

- **BASIC or JAVA**
- **Serial Port (rs232/rs422/rs485)**
- **Ethernet Port (TCP/UDP)**
- **Memory TAGS**
- **File (/usr)**

Protocol implementation

Built-in how to do more?

Basic DIY




```
SD_Addr$ = "10.0.7.24"
SD_Port$ = "8023"
handle% = @openServoDriver%( 1, SD_Addr$, SD_Port$, "guest", "guest" )

Servo_Current@ = @getCommandValue(handle%, "CURRENT")
Print "Current : ";value
Servo_Voltage@ = @getCommandValue(handle%, "VOLTAGE")
Print "Voltage : ";value
```

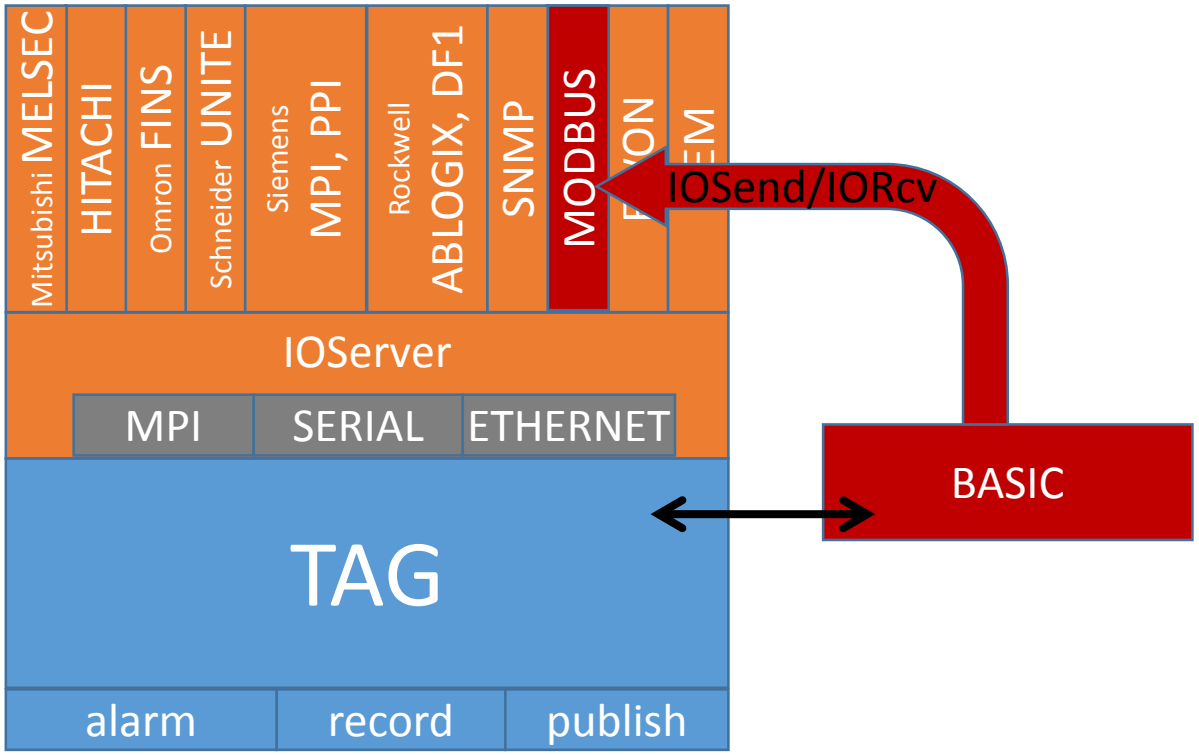
Workshop after the lunch

How to do more?

1. **Do it all yourself!!**
   - **In Basic or Java**
2. **Add IOServer functionalities**
   - **in Basic**

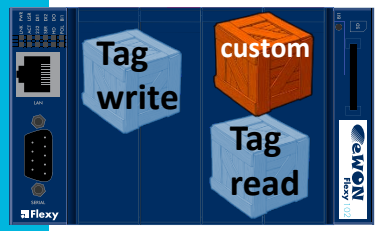
Built-in
how to do
more?
Basic DIY
Basic IOSrv+

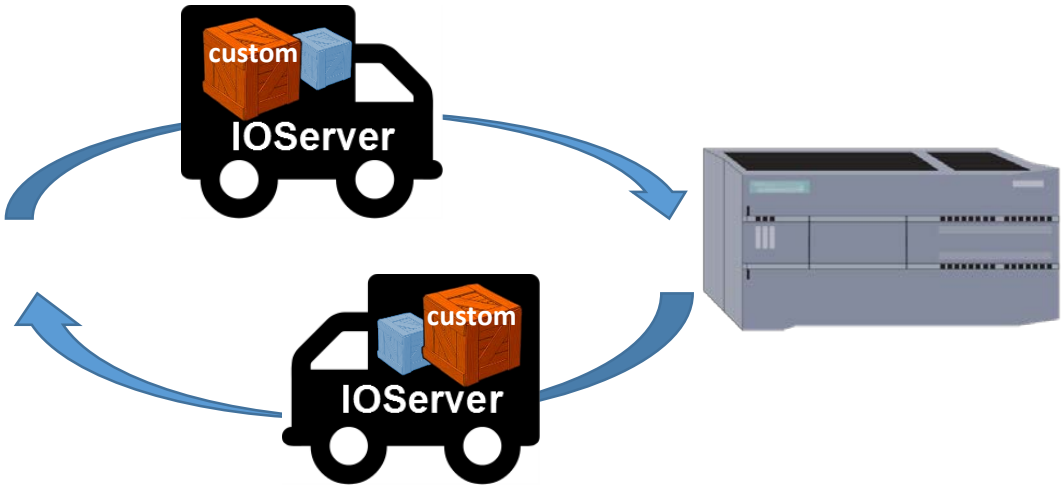
Add IOServer functionalities



Built-in
how to do more?
Basic DIY
Basic IOSrv+

Add IOServer functionalities



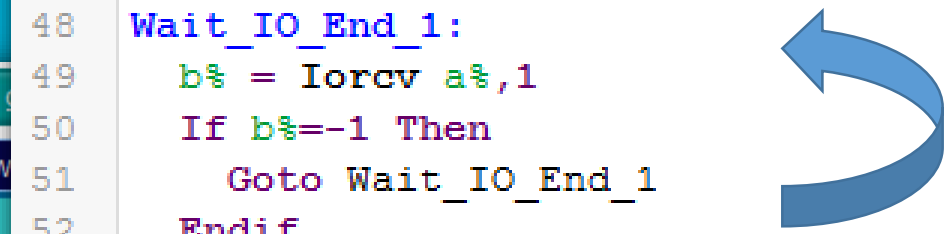


- **Send a custom request to the IOServer and read the answer**
  - To handle other data type than numeric
  - To use other protocol operations than “ReadData” or “WriteData”

Basic script
IOSend / IORcv

Built-in
how to do
more?
Basic DIY
Basic IOSrv+

```
41  ReadModbusRegister:
42
43    Addr$ = "100,10.2.100.147"
44    Frame$ = Chr$(3)+Chr$(3)+Chr$(231)+Chr$(0)+Chr$(7)
45
46    a% = Iosend "MODBUS",Addr$,Frame$
47
48  Wait_IO_End_1:
49    b% = Iorcv a%,1
50    If b%=-1 Then
51      Goto Wait_IO_End_1
52    Endif
53
54    B$ = Iorcv a%
55
56    Z$ = B$( 3 To )
57    Print "Read phone number"
58    Print "";Z$
```



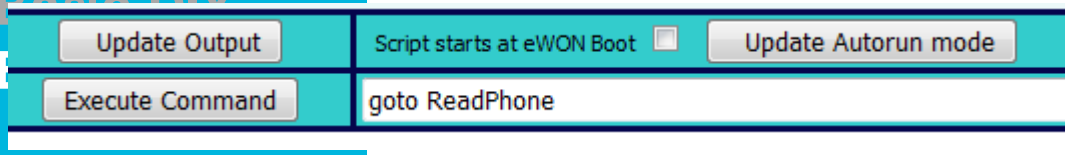
Basic script
IOSend / IORcv

Built-in how to do more?

- **Read a phone number from a Modbus PLC**

14 chars at address 1000

“+32 67 895 800”



Workshop after the lunch

	Tag Name	Value	New Value		Description
	phone_01	11059	11059	Update	[+3]
	phone_02	12832	12832	Update	[2_]
	phone_03	13879	13879	Update	[67]
	phone_04	8248	8248	Update	[_8]
	phone_05	14645	14645	Update	[95]
	phone_06	8248	8248	Update	[_8]
	phone_07	12336	12336	Update	[00]

Basic script IOSend / IORcv

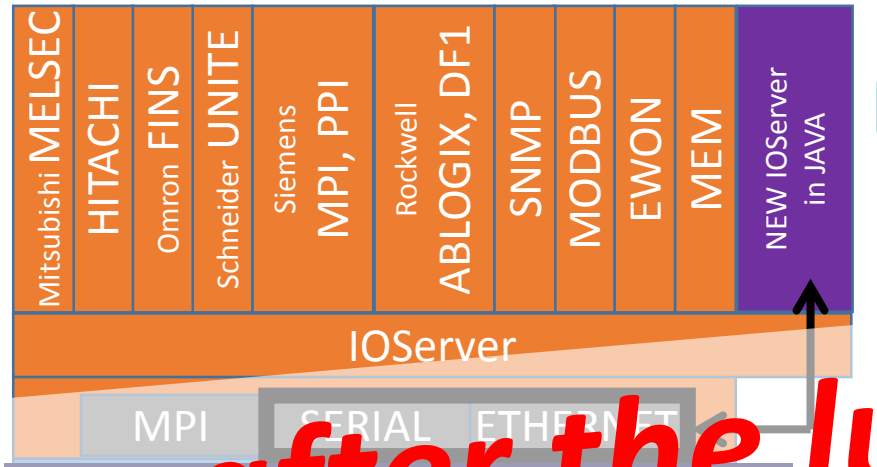
+ communication is handled by the IOServer
  (ie: CRC calculation)
+ you need to script only the required function
  ... and not the whole protocol
+ Tags polling and Basic IOSend/IORcv run together

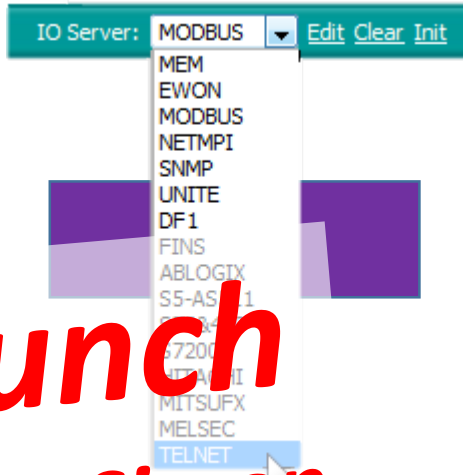
How to do more?

1. **Do it all yourself!!**
   - In Basic or Java
2. **Add IOServer functionalities**
   - in Basic
3. **Create a new IOServer**
   - In Java

JAVA IOServer

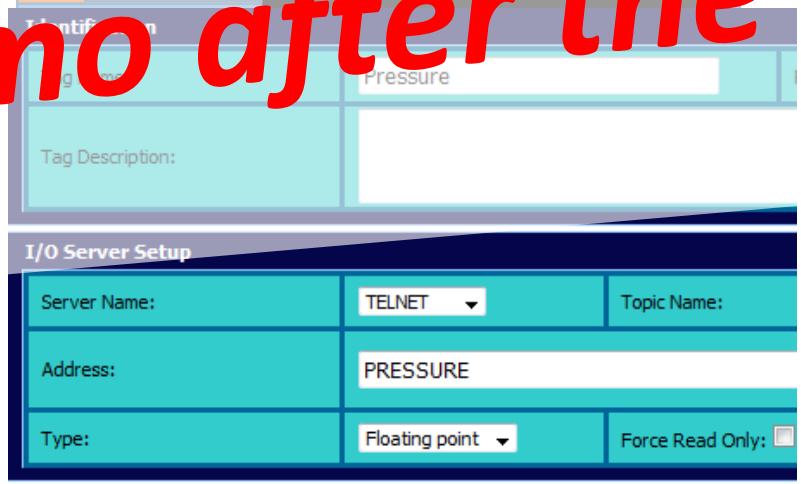
Built-in
how to do
more?
Basic DIY
Basic IOSrv+
JAVA IOServer





Demo after the lunch

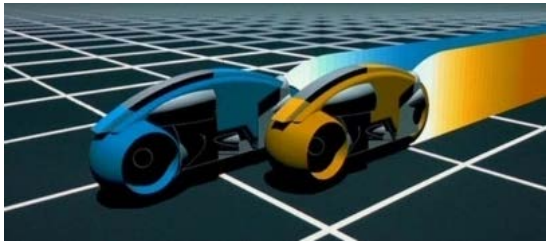
by Simon



BASIC2

BASIC2
BASIC IDE

- BASIC Script exists since first eWON inception… in 2001
  - based on 1980 Basic



- It was time to improve it with:
  - free variable naming
  - more data space
  - function declaration
  - integrated IDE
  - debugger

Long variable name

varname

- Float variable
  - **MyFloat PowerRatio Result**
- String variable
  - Old: A$ to Z$ (only 26)
  - New: **MyString$ IpAddr$ Message$ aaaa$**
- Integer variable
  - Old: A% to Z% (only26)
  - New: **MyInt% Counter% Status% ErrorNum%**
- Float Array
  - Old: A(x,y,z) to Z(x,y,z)
  - New: **ProcessStatus(x,y,z) LinesCounter(x)**
  - Max 16 dimensions
- Char Array
  - Old: A$(x,y,z) to Z$(x,y,z)
  - New: **ErrorMsg$(x,y,z) UserName$(x,y,z)**
  - Max 16 dimensions

Data memory

- BASIC (on eWON-CD)
  - Basic memory (prog + data) = 256 kB
- Now with BASIC2 (on Flexy only)
  - Basic memory (prog + stack) = **256 kB**
  - **NEW: VAR memory = 500 kB**
- New basic function "**MEMORY()**" return the free memory in bytes

  Print memory("tot") → **767115 bytes**

  Print memory("prog") → 261559 bytes

  Print memory("var") → 505552 bytes

Function

- Function syntax declaration
  - Begin with "**Function**"
    - + FuncName()
  - End with "**EndFn**"

```
19  Function basVarMemory
20    Print MEMORY("var")
21  EndFn
22
23  Function basProgMemory()
24    Print MEMORY("prog")
25  EndFn
```

- Function call
  - **Prefix** function name with '@'

```
28  info:
29    @basVarMemory
30    @basProgMemory()
31
```

Return value

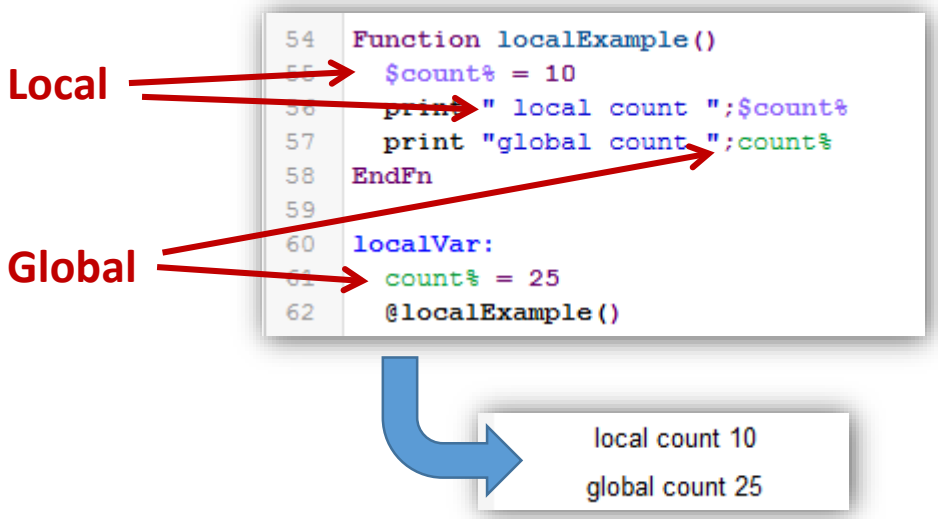
Varname
Memory
Function

```
33  Function inch()
34    $inch = 2.54
35  EndFn
36
37  Function nauticalMile%()
38    $nauticalMile% = 1852
39  EndFn
40
41  Function appName$()
42    $appName$ = "Demonstration"
43  EndFn
```

- Could be assigned several time into the function
  The last assignment will be returned

Local variable

- Local variable prefixed with $
  - Use the same Type convention as variable
    - String postfixed with $
    - Integer postfixed with %
    - Float postfixed with *nothing*



Parameters

- Function parameters prefixed with $
  - Use the same Type convention as variable
    - String postfixed with $
    - Integer postfixed with %
    - Float postfixed with *nothing*

```
45  Function paramExample( $AString$, $AFloat, $AnInteger% )
46    print $AString$
47    $paramExample = $AFloat * $AnInteger%
48  EndFn
```

- **Passed by value**

- Numeric autoconversion perfomed during function call

```
51  MyInt% = 256
52  MyFloat = 3.14159
53  Print @paramExample("hello", MyInt%, MyFloat)
```

Parameters by reference

- Parameters passed by reference are prefixed with @
  - Type convention
    - String postfixed with $
    - Numeric (Integer | Float) has no postfix. *Type is passed implicitly with the reference*

- New Basic function **TYPE$()**
  - Return the var type : “string” | “float” | “integer”

```
Function refExample( @$AString$, @$Numeric1, @$Numeric2 )
  Print "AString: "; $AString$
  Print "Numeric1: "; $Numeric1 ;"  type: "; TYPE$($Numeric1)
  Print "Numeric2: "; $Numeric2 ;"  type: "; TYPE$($Numeric2)

  $AString$ = "ref_" + $AString$
  $Numeric1 = $Numeric1 * 2.5
  $Numeric2 = $Numeric2 * 3.14

  Print "Numeric1: "; $Numeric1
  Print "Numeric2: "; $Numeric2
EndFn
```

Parameters by reference

```
77  Function refExample( @$AString$, @$Numeric1, @$Numeric2 )
78    Print "AString: "; $AString$
79    Print "Numeric1: "; $Numeric1 ;"  type: "; TYPE$($Numeric1)
80    Print "Numeric2: "; $Numeric2 ;"  type: "; TYPE$($Numeric2)
81
82    $AString$ = "ref_" + $AString$
83    $Numeric1 = $Numeric1 * 2.5
84    $Numeric2 = $Numeric2 * 3.14
85
86    Print "Numeric1: "; $Numeric1
87    Print "Numeric2: "; $Numeric2
88  EndFn
89
90  refExample:
91    GlobalString$ = "Message"
92    GlobalFloat = 10.0
93    GlobalInt% = 5
94
95    @refExample( GlobalString$, GlobalFloat, GlobalInt% )
96    //@refExample( GlobalString$, GlobalInt%, GlobalFloat )
97
98    Print GlobalString$
99    Print GlobalFloat
100   Print GlobalInt%
101
102 END
```

```
AString: Message
Numeric1: 10.00 type: float
Numeric2: 5 type: integer
Numeric1: 25.00
Numeric2: 15
ref_Message
25.00
15
```

```
//@refExample( GlobalString$, GlobalFloat, GlobalInt% )
@refExample( GlobalString$, GlobalInt%, GlobalFloat )
```

```
AString: Message
Numeric1: 5 type: integer
Numeric2: 10.00 type: float
Numeric1: 12
Numeric2: 31.40
ref_Message
31.40
12
```

Local label

Varname
Memory
Function

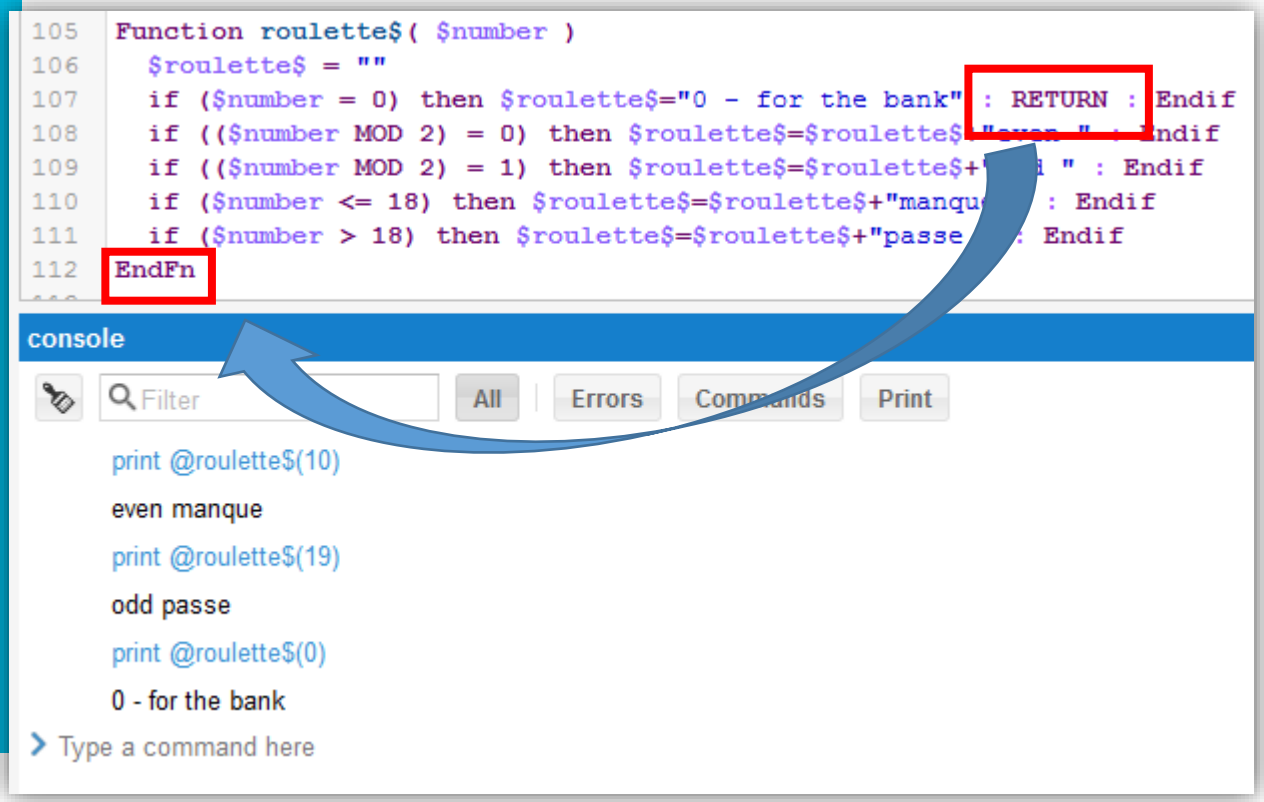
- You may define **Local Label** reachable only from the function itself
  - Prefix the label with $

```
114  Function roulette$( $number )
115    $roulette$ = ""
116    if ($number = 0) then $roulette$="0 - for the bank" : GOTO $RoulEnd : Endif
117    if (($number MOD 2) = 0) then $roulette$=$roulette$+"even " : Endif
118    if (($number MOD 2) = 1) then $roulette$=$roulette$+"odd " : Endif
119    if ($number <= 18) then $roulette$=$roulette$+"manque " : Endif
120    if ($number > 18) then $roulette$=$roulette$+"passe " : Endif
121  $RoulEnd:
122  EndFn
```

RETURN keyword

Varname
Memory
Function

- You can exit the execution of the function with the “RETURN” keyword



Recursive call

- Yes, you can call function recursively

```
114  Function fibonacci%( $number% )
115    cnt = cnt+1
116    if ( $number% = 0 ) then $fibonacci% = 0 : return : Endif
117    if ( $number% = 1 ) then $fibonacci% = 1 : return : Endif
118    $fibonacci% = @fibonacci%($number% - 1) + @fibonacci%($number% - 2)
119  EndFn
```

```
cnt=0                                    15:35:16.573
print @fibonacci%(20)                    15:35:30.321
6765                                     15:35:52.010
print cnt                                15:36:07.113
21891.00                                 15:36:07.152
```

For..Next optimization

- "FOR NEXT" loop structure was optimized to work fast with integer variables (a% → z%)
- To keep this optimization, in BASIC2, "FOR NEXT" does not accept long name integer variable
  - FOR counter%=1 TO 100 → interpreter error
  - FOR i%=1 to 100 → OK
  - Always only 26 variables available !
- "FOR NEXT" inside function are also optimized with local integer variables with VarName of 1 char long!
  - o $a% → $z%

```
170  Function delay( $delay% )
171    print time$
172    For $i%=1 to $delay%
173    next $i%
174    print time$
175  EndFn
```

Thank You

Next ...

BASIC IDE