



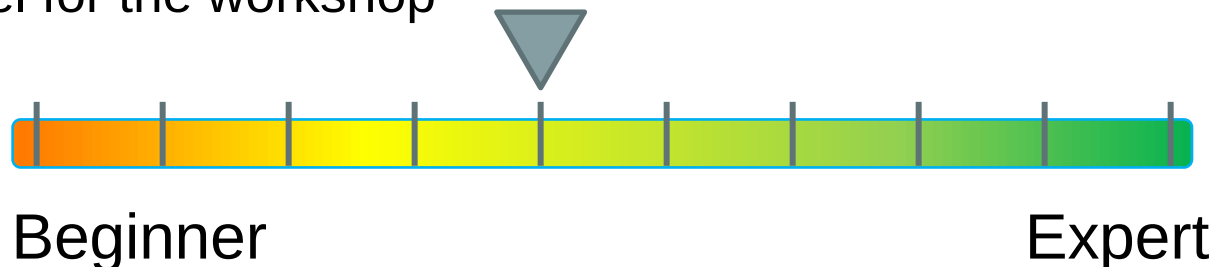
APROL Library Engineering Workshop



APROL Library Engineering Workshop

Time	Topic	Presenter
Day 1	Python Tutorial	Marböck Wolfgang
Day 2	Python API for DisplayCenter	Marböck Wolfgang
Day 3	Python API for DisplayCenter, LoginServer, UCB, ListViewWidget	Marböck Wolfgang

Aimed level for the workshop



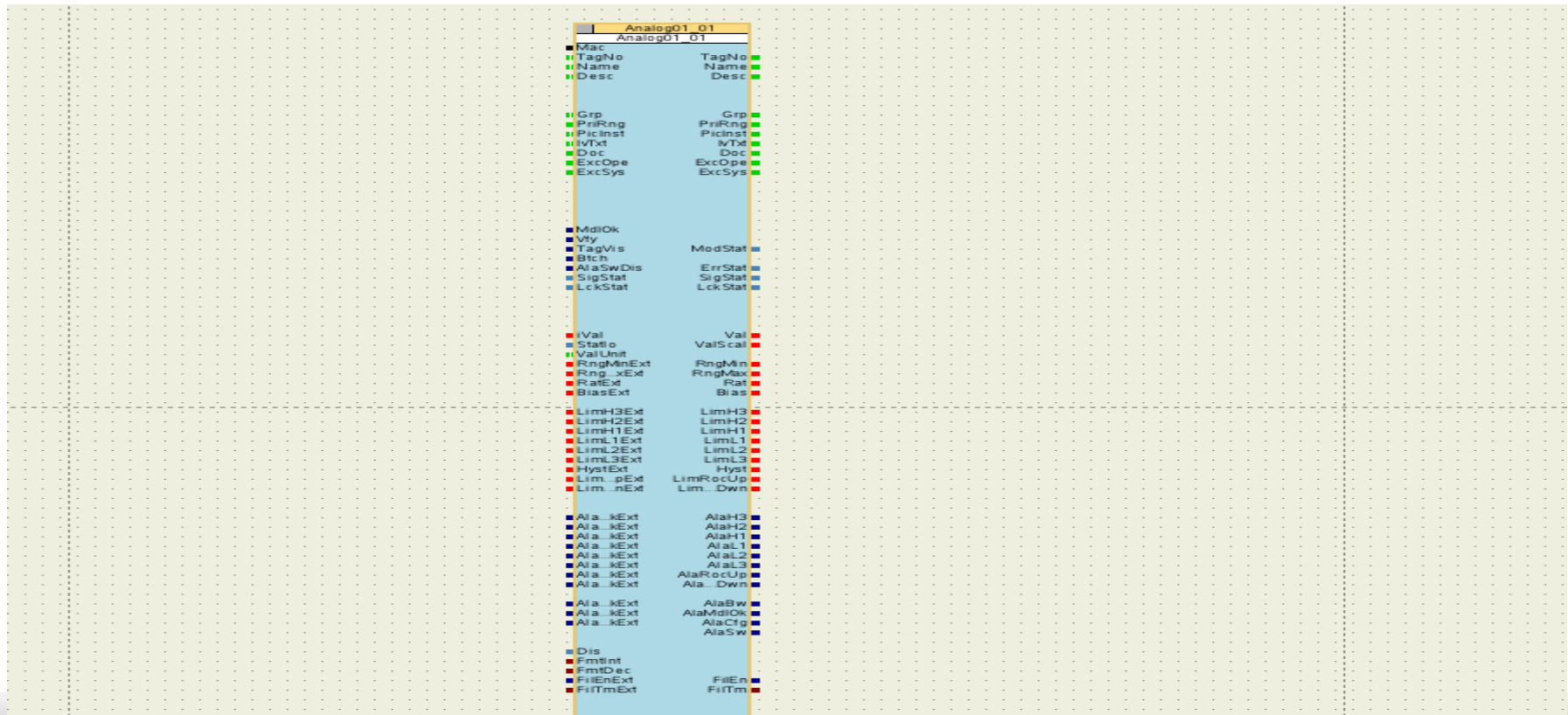


APROL Libraries



■ APROL Hypermacro concept

- Includes logic functionality (C- FUB)
- Includes visualization parts (graphic macros, faceplates)
- Includes control system blocks (alarm, trend)
- One block per typical, everything inside



APROL Libraries basic concepts 2/9

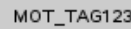
PERFECTION IN AUTOMATION
www.br-automation.com



Logic



Graphic
Macro



MOT_TAG123

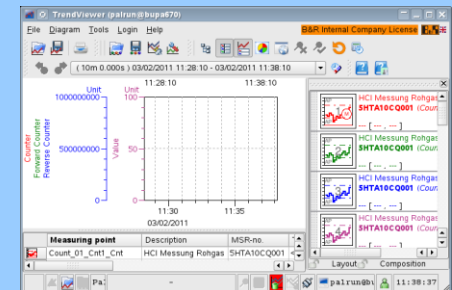
Faceplate



Alarm

3 - 03/02/2011 11:36:29 CET	0	FQ 1207	Operations counter interval 1	PV - OpCh_R4_S5201_AW1_C.ASCH
4 - 03/02/2011 11:36:33.788 CET	0	FQ 1208	Operations counter interval 1	PV - OpCh_R4_S5202_AW1_C.ASCH
5 - 03/02/2011 11:36:36.289 CET	0	FQ 1209	Operations counter interval 1	PV - OpCh_R4_S5203_AW1_C.ASCH

Trend





Logic function blocks

- Pins
- Code = C programming
- Documentation B3.2.7

The screenshot displays the CaeManager (engin@muehlbauer) <2> interface. The main window is titled 'PAL_V005/internal_C/CfgErrSel01'. The left pane shows the 'Block' tab with a diagram of the 'CfgErrSel01' block, which has 10 inputs (6 x 6) and 4 outputs: Err01, Err02, Err03, and Err04. The right pane shows the 'Program' tab with the C code for the block. The code includes comments for history and start, a declaration for a BOOL variable 'ErrChanged', and a program start section with a conditional statement for Err04.

```
1  /*=====*/
2  /*.....*/
3  /*History:.....*/
4  /*20.04.2010→Start.....*/
5  /*.....*/
6  /*=====*/
7
8  /*DECLARATION*/
9
10 BOOL ErrChanged;
11
12 /*local variables for value changed detection*/
13 BOOL Err01Changed := (Err01 != Err01Old)?1:0;
14 BOOL Err02Changed := (Err02 != Err02Old)?1:0;
15 BOOL Err03Changed := (Err03 != Err03Old)?1:0;
16 BOOL Err04Changed := (Err04 != Err04Old)?1:0;
17
18 /*INITIALIZATION*/
19 Ala := 0;
20 ErrChanged := 0;
21
22
23 /*PROGRAM START*/
24
25 if(Err04Changed) → → /*Err04 has changed it's value..Err04 LOWEST
26 {}
```



Graphic block

- Pins
 - Graphical part
 - Python Code
 - Documentation
- B3.2.8

Master Data

Block

I/O

Preview

Python Code

Graphic Code

Documentation

Master Data

Block

I/O

Preview

Python Code

Graphic Code

Documentation

Master Data

Block

I/O

Preview

Python Code

Graphic Code

Documentation

Master Data

Block

I/O

Preview

Python Code

Graphic Code

Documentation

Master Data

Block

I/O

Preview

Python Code

Graphic Code

Documentation

Master Data

Block

I/O

Preview

Python Code

Graphic Code

Documentation

Value U
TagNo

I/O-variable						
P	Name	Remanent	IEC type	Default value	Unit	Description
Block inputs:						
1	TagNo	---	LSTRING	TagNo	---	Tag Number
2	Name	---	LSTRING		---	Name
3	Desc	---	LSTRING		---	Description
5	TagVis	---	BOOL	1	---	TagNo Visible
6	FaceEn	---	BOOL	1	---	Faceplate enal
7	Val	---	DINT	0	---	Value
8	FmtInt	---	USINT	0	---	Format Integer
9	FmtDec	---	USINT	0	---	Format Decim
10	Unit	---	LSTRING	*C	---	Unit
12	PriRng	---	SSTRING	0	---	Priority range
16	ModStat	---	DWORD	0	---	Mode state
17	ErrStat	---	DWORD	0	---	Error summary
18	LckStat	---	BYTE	0	---	Lock state
19	SigStat	---	BYTE	0	---	Signal Status
Block outputs:						
1	Lvl1Face	---	FACEPLATE		---	Level 1 facepla
Unconnected global graphic I/Os:						
Local graphic I/Os:						
BgCol						
Bli						

```

1 #####
2 #####
3 ## History:
4 ## -20.04.2010 -> Start
5 ## -09.07.2010 -> use adapted names for functions from paldef_addon
6 ##
7 #####
8
9 # -*- coding: utf-8 -*-
10 import paldef
11
12 def Alarm():
13     # Get textcolor, backgroundcolor, etc. for dynamization
14     paldef.GetAlaDyn(block.pin.ErrStat, block.pin.PriRng, block.local.
15
16
17 ##### Hooks
18 #####
19
20
21 ##### Function Collector for "None" detection
22 func_collector = paldef.class_func_collector( block, block.local.Nor
23
24
25 ##### Mode Image
26 DC.add_hook( block.pin.ModStat.changed_hook, ->func_collector( pal
27
28
29 DC.add_hook( block.pin.FmtInt.changed_hook, ->func_collector( pal
30 DC.add_hook( block.pin.FmtDec.changed_hook, ->func_collector( pal
31 DC.add_hook( block.pin.Unit.changed_hook, ->func_collector( pal
32
33
34 ##### Alarm
35 DC.add_hook( block.pin.ErrStat.changed_hook, ->func_collector(
36
37 ##### Lock
38 DC.add_hook( block.pin.LckStat.changed_hook, ->func_collector( pal
39
40 ##### Dynamization Signal Status of Input Signals
41 DC.add_hook( block.pin.SigStat.changed_hook, ->func_collector( pal
42
43 ##### Mac Tooltip
44 DC.add_hook( block.pin.TagVis.changed_hook, ->func_collector( pal
45

```



■ available animations:

The following are the basic functions available for animation:

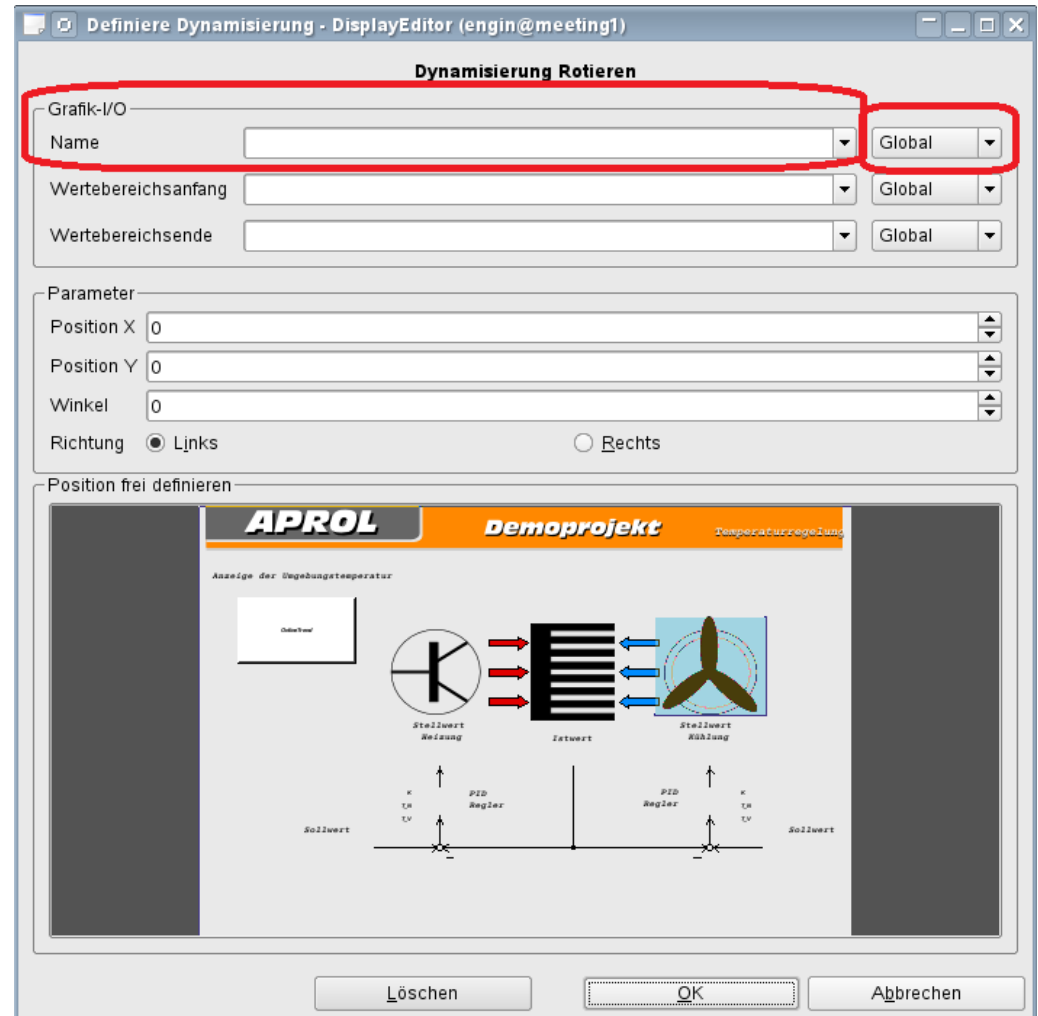
Animation type	Description
Blinking	Objects can be blink in many different ways depending on a process variable.
Colors	Objects can change colors depending on a process variable. Background and foreground colors, fill patterns, and line types can be changed.
Rotate	Objects rotate around a definable fixed point depending on a process variable.
Move	Makes it possible to move objects to a different location in the process graphic. Movement along the x-axis can be defined independently of that on the y-axis.
Scale	Objects can change their size depending on two process variables. The x and y axes can be defined independently of one another.
Percent	Objects are trimmed depending on a process variable and no longer displayed in full size.
Text	The actual value of a process variable is displayed. STRING variables can be displayed.
Text list	Defined texts can be displayed depending on a process variable.
Image list	Serves to dynamically display 1 from n images - depending on a process value. E.g. a significant increase in productivity can be reached during the creation of status displays from your system via image (icon).
Invisible	The object is switched invisible.



■ Apply animation to a graphical element

■ Local/Global graphic IO

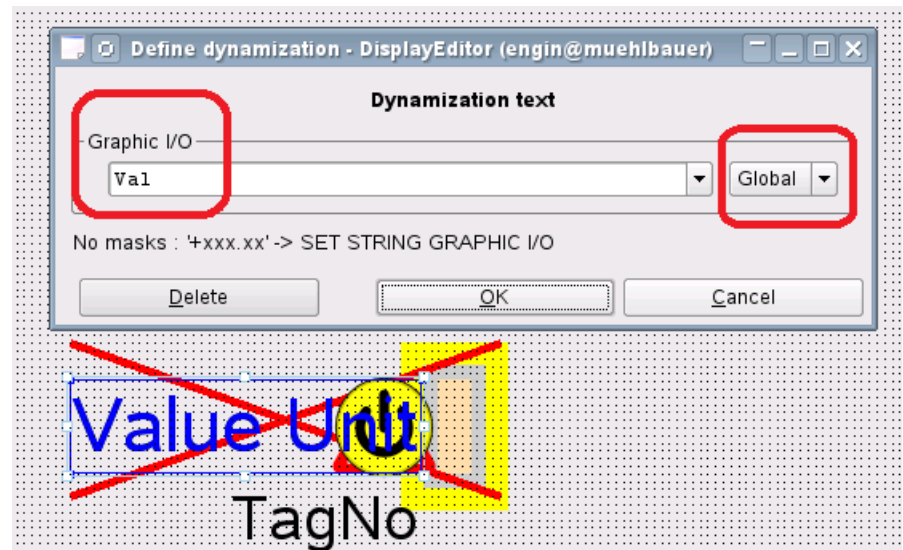
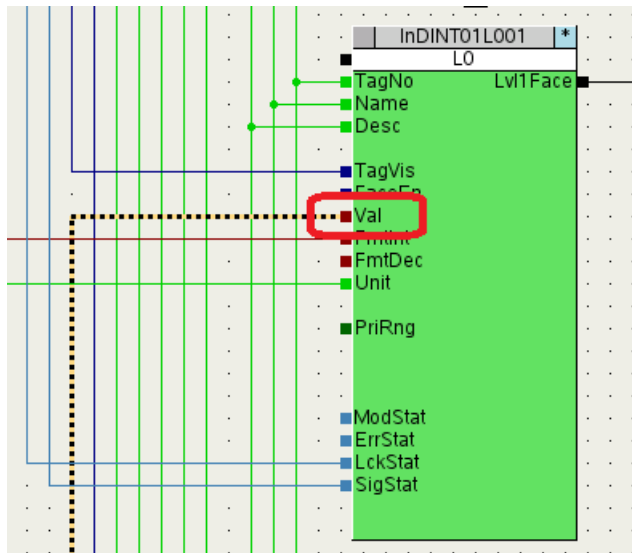
■ Documentation B3.2.8.2.3





■ Animation with global graphic IOs

- Use a pin of the graphic block as signal source
- The pin of the graphic block can be connected to a signal coming from a logic block, input pin, gateway io, hm- constant, ...

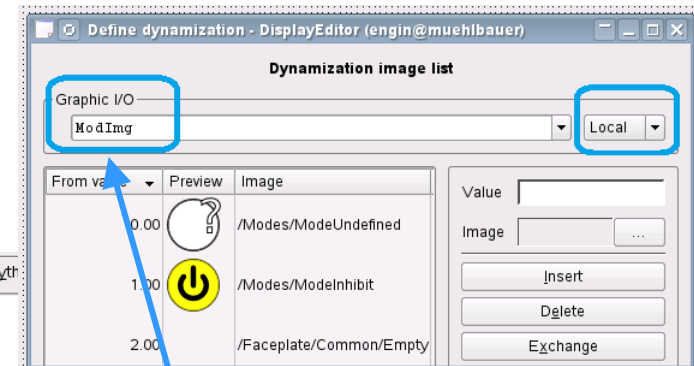




■ Animation with local graphic IOs (= Python variables)

■ Use pin(s) of the graphic block in Python Code.

■ The result of the Python Code is a local graphic IO



Master Data | **Block** | **I/O** | **Preview** | **Python Code** | **Graphic Code** | **Documentation**

InDINT01L001 (10 x 23)	
TagNo	Lv1Face
Name	
Desc	
TagVis	
FaceEn	
Val	
FmtInt	
FmtDec	
Unit	
PriRng	
ModStat	
ErrStat	
LckStat	
SigStat	

Python Code

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14 block.local.TxtCol,block.local.BgCol,block.local.El1,None)  
15  
16  
17 ####  
18 ####  
19  
20  
21  
22 .local.NoneIndicator..  
23  
24  
25  
26 ctor( paldef.GetModeImage, block.pin.ModStat, block.local.ModImg() )  
27  
28  
29 ctor( paldef.GetDynWidth, block.name(), block.pin.FmtInt, block.pin.  
30 ctor( paldef.GetDynWidth, block.name(), block.pin.FmtInt, block.pin.  
31 ctor( paldef.GetDynWidth, block.name(), block.pin.FmtInt, block.pin.  
32  
33  
34 collector( Alarm )  
35
```

Define dynamization - DisplayEditor (engin@muehlbauer)

Dynamization image list

Graphic I/O: ModImg Local

From va	Preview	Image	Value
0.00	?	/Modes/ModeUndefined	
1.00	⏻	/Modes/ModeInhibit	
2.00		/Faceplate/Common/Empty	

Value: Image: Insert Delete Exchange



- In general all animations are possible without Python (use global graphic IOs)
- Python is available for extended possibilities with animations
- Python-API gives you access to system events via HOOKS
- Python-API offers FUNCTIONS to perform specific actions

The screenshot shows the CaeManager software interface. The title bar reads 'CaeManager (engin@muehlbauer) <2>'. The menu bar includes 'File', 'Edit', 'Build', 'View', 'Reports', 'Extras', 'Login', and 'Help'. The toolbar contains various icons for file operations and editing. The main window displays a Python module editor for a file named 'paldef' located at 'PAL_V005/Internal_Common/paldef'. The description of the module is 'Python module, function definitions'. The code editor shows a function definition for 'CheckAccessRights' with the following code:

```
612 #####
613 ### Check Access Rights
614 #####
615 def CheckAccessRights(blockname, LockPin, RightPin, ExcOpePin, ExcSysPin, ExcOffStrPin):
616     Lock = Get( LockPin )
617     #Get( RightPin ) ## Only check for None, no need for the result
618     Right = RightPin.right()
619
620     if Lock:
621         LogicCheckText=DC.tr('Locked from Logic;')
622     else:
623         LogicCheckText=''
624
625     if (DC.login.hasRight( Right )):
626         LoginCheckText=''
627     else:
628         LoginCheckText=DC.tr('Operator has no right;')+Right+';'
629
630     ExcOpeCheckText, ExcSysCheckText, nouse1, nouse2 = CheckExclusiveAccessRights( blockname, ExcOpePin, ExcSysPin, ExcOffStrPin ) ## tmp not us
631
632     return ( LogicCheckText, LoginCheckText, ExcOpeCheckText, ExcSysCheckText ) ## returns '' if access granted
633
634
```



APROL Python API



■ Documentation:

- APROL Product documentation – D2 – System API

■ API available for following components

- Visualization (DisplayCenter)
- Login (LoginServer)
- Parameter (ParameterCenter)

■ Don't use python for creating cyclic programs!

- D2.1.2 – IMPORTANT!
- Code is executed by Python interpreter, DC/LS wait for termination
- DC/LS are blocked when code has long execution time or endless loop

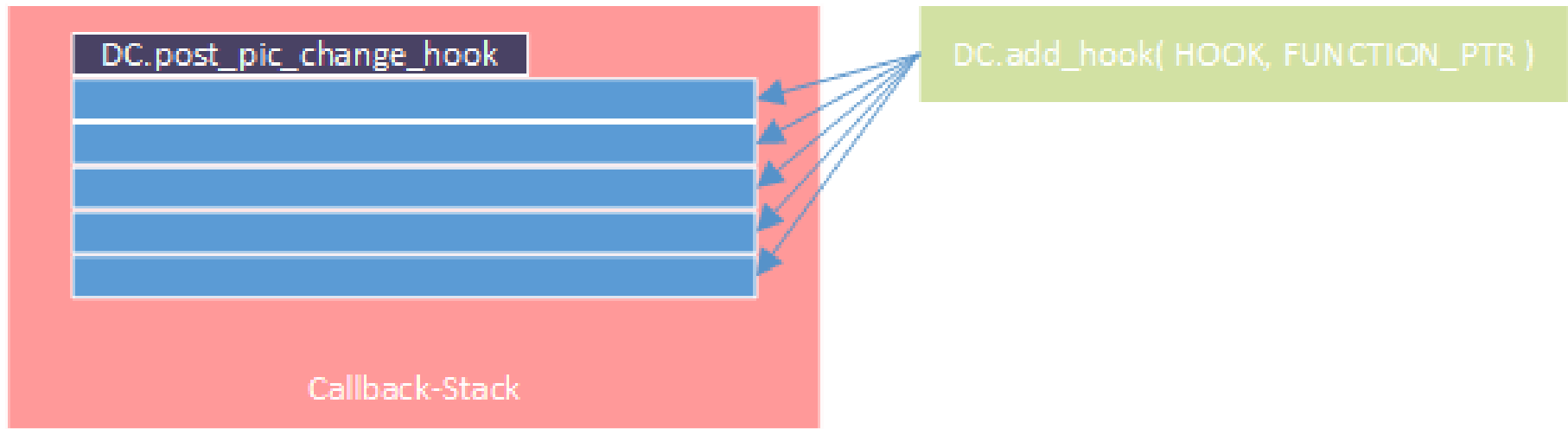


- Python code execution
 - Code has context (DisplayCenter, process graphic, graphic block, ...)
 - As long as the same context is active, the python code specific to this context is run in the Python interpreter.
 - All the “static” code (code which is not run within the context of a hook) is only run once, when this context becomes active.
 - You can use HOOKS to connect to system events. Therefore in the python code you just make a connection between hook and a user defined function. Always when the specific event occurs, your user defined function is run.



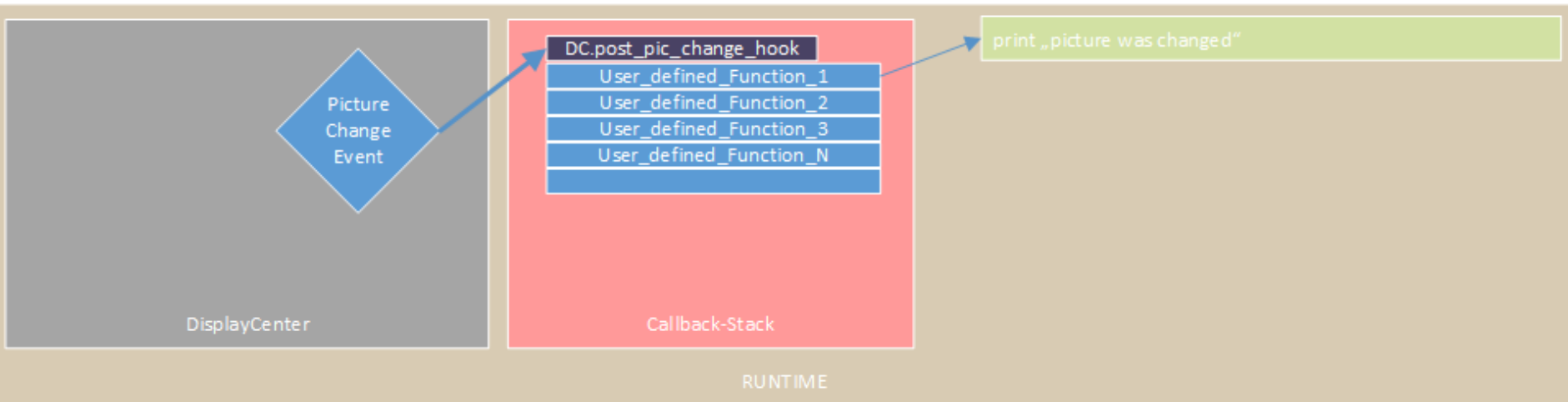
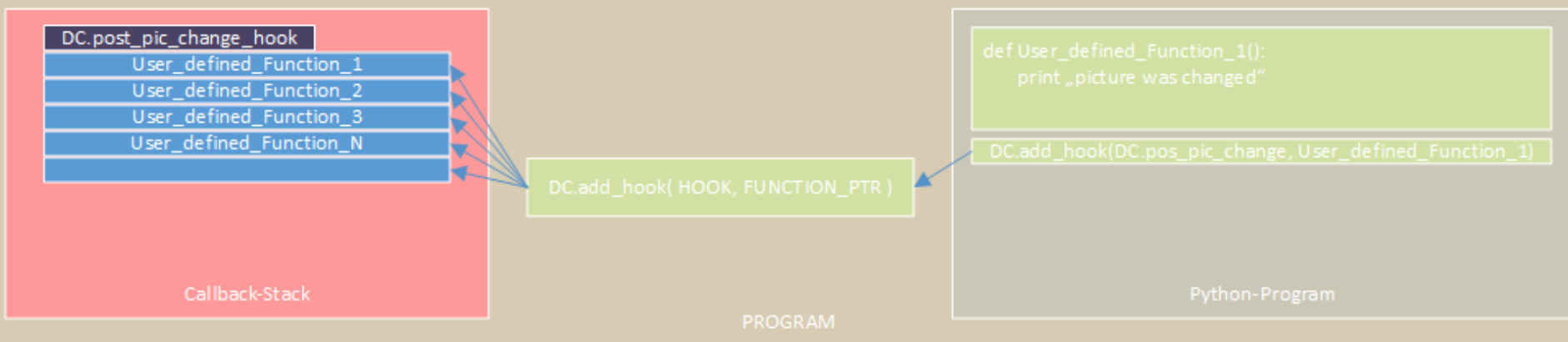
■ HOOK

- Event in the system (e.g. picture change)
- Possibility to react to this events – connect user functions to hooks with
`DC.add_hook( HOOKNAME, POINTER_TO_USERFUNCTION)`
- `DC.add_hook` creates a list of functions which need to be called on this event (callback-stack)
- Available Hooks see D2.1.2.1





HOOK in action





APROL Python API for DisplayCenter



■ What?

- Expansion of the possibilities for dynamizations in graphic macros
- Possibility to react to events in the visualization
- Usage of specific API-functions

■ Why?

- Offer extended functionality for the engineer

■ Where?

- In graphic blocks (graphic macros, faceplates, sub- faceplates)
- Own “tab” for python code
- Use of “local” variables (=python variables”) alternatively beside “global” variables (=block pins)
- Python code is executed event-triggered, certain events provide a **hook**, python code can be connected to those hooks



■ Debugging of python code

- For extended debugging the **python console** needs to be activated
RuntimeSystem->Monitoring->DisplayCenter->Prio3
-showPythonConsole ; (value = number of lines)
- Python console can be used for debug messages (e.g. code: “print var”)
- Errors in the python code are always shown in python console, independent of python console enabled or disabled
- For interactive mode also need to enable
RuntimeSystem->Monitoring->DisplayCenter->Prio3
-pythonDebugging
- See D2.1.2.4

■ Workshop

- Enable the python console in your project



■ Global python module for the DisplayCenter

- One python module from the project can be used as global python module
- For each runtime system or operator system a own global python module can be used
- This module is available at DisplayCenter start on the specified systems
RuntimeSystem->Monitoring->DisplayCenter->Prio2
-pythonModule = "Instancename"
- See D2.1.5.2
- Provides hooks for global actions
 - See hooks in D2.1.2.1
 - DC.init_hook, DC.user_changed_hook, DC.post_pic_change_hook

■ Workshop

- This example should write the instance name of the actual picture to the python console of the DisplayCenter
 - Create a global module for Display Center in your project
 - Import module DC
 - Write a function (name "MyFunc1") which prints the name of the actual picture instance
 - Connect the function with DC.add_hook() using the DC.post_pic_change_hook
 - Assign the global python module for DisplayCenter in the RuntimeSystem



■ Global python module for process graphics

- Can be created in the project
- Can be imported from a process graphic
 - See D2.1.5.1

■ Workshop

- This example should write the name of the logged in user to the python console of the DisplayCenter
 - Create a global python module for process graphics in your project
 - Import module DC
 - Write a function (name “myFunc2”) which prints the name of the actual logged in user
 - Create a new picture in the visualization
 - Add a hook with `DC.add_hook()`; use the `DC.login.user_changed_hook` and connect the function (“MyFunc2”) from your global python module



■ Global python modules for CAE- Libraries

- Can be created in the library
- Global library module for collecting all functions in your library
- Can be imported from a graphic block

■ Workshop

- This example should write the actual system time to the python console of the DisplayCenter
 - Create a test library
 - Create a global python module for graphic blocks in your library
 - Import module DC
 - Write a function (name “myFunc4”) which prints the actual system time. Use the python module time ; function ctime().
 - Create a new picture block
 - Import the library global python module you created before
 - Add a hook with DC.add_hook(); use the block.show_hook and connect the function (“MyFunc4”) from your global python module
 - Create a new picture in the visualization, use an instance of your graphic block



■ Workshop

- Use the `DC.save_snapshot` function when a block is shown
 - Create a new picture block
 - Write a python function to save a snapshot (use `DC.save_snapshot`)
 - Add a hook with `DC.add_hook()`; use the `block.show_hook` and connect to your function
 - Create a new CFC, use an instance of your block
 - Create a new picture in the visualization, use the instance which you placed in the CFC
- `Block.show_hook` provides no argument, the hooked function must take no arguments !



■ Workshop

- Use the `DC.save_snapshot` function when a button is pressed
 - Create a new picture block
 - Use a python button; define local variable (“localIO”)
 - Write a python function to save a snapshot (`DC.save_snapshot`)
 - Add a hook with `DC.add_hook()`; use the `block.local.localIO.button_press_hook` and connect to your function
 - Create a new CFC, use an instance of your block
 - Create a new picture in the visualization, use the instance which you placed in the CFC
- `button_press_hook` provides 1 argument, the hooked function must take this argument !



■ Using lambda

- Doc D2.1.1.7
- Anonymous function (=function without function name, only function pointer)
- Google for “python lambda”
- Why
 - Modularity 1: When you want to use the same function with different kinds of hooks (different hooks deliver different amount of arguments)
 - Modularity 2: When you want to pass an attribute of the block (e.g. block.pin.XXX) to the connected function
- Alternative
 - Write own functions for each case
 - ->result: many similar functions, more effort for code development and maintenance
- Facts
 - The add_hook always wants to have a function object (=function name, =pointer to a function)
 - The add_hook doesn't accept a function call (which means also a function call including parameters is not accepted)
 - With lambda you make an anonymous function (which is a function object), allowing to pass parameters
 - With lambda it's possible to pass over the block object to the function



■ Workshop(1)

- Use a lambda function in python interactive mode
- Write a lambda equivalent to following function – addition of 3 parameters
 - `Func1(): return(a+b+c)`

■ Workshop(2)

- Use the `DC.save_snapshot` function when a button is pressed
 - Create a new picture block, with 2 input pins (Pin01, Pin02)
 - Use two python buttons; define local variables (“localIO1, localIO2”)
 - Write python function to save a snapshot (`DC.save_snapshot`)
 - The function should take an argument; the argument should be a pin from a graphic block. The pins should contain the filenames for the snapshots.
 - Add a hook with `DC.add_hook()`; use the `block.local.localIO1.button_press_hook` and connect to your function using lambda and pass over Pin01(do the same for localIO2, Pin02)
 - Create a new CFC, use an instance of your block
 - Create a new picture in the visualization, use the instance which you placed in the CFC
- `button_press_hook` provides 1 argument, the hooked function must take this argument !



- What can I do with the DC-Python-API in APROL ?
 - See D2.1.3 – The DisplayCenter API classes
 - See D2.1.4 – The DisplayCenter features



■ Practical use of python in the PAL

- Adapt system functions to specific needs (default values, ...)
 - LoadProcessPicture, StartDocumenation, ShowOperatorNote, ShowLogbook, StartAlarmReport, StartTrendViewer,...
- Function to define output format (integer / decimal places of values)
- Access functions: use different images to display if a button is enabled or disabled, give the user information why a button is disabled,...
- Image dynamization with images from image container (e.g. for the motor you can choose if you want to have a motor or a pump or a fan as symbol)
- Alarm-Faceplate with ListView-Widget
- Alarm color dynamization according PriRng
- Open faceplates via instance name
- Concatenate tooltips out of a few text snippets
- Error List Dynamization
- Scaling functions for bargraph
- ...



APROL Python API for LoginServer



■ Global python modules for LoginServer

- Can be created in the project
- Global library module for functions modifying the login process
 - See D2.1.6.4
- The global module needs to be assigned to the RuntimeSystem
RuntimeSystem->System Services->LoginServer->Prio3
-pythonModule (instance of the module)
- Has an own python console that can be activated, open console via context menu of runtime-login-symbol in systray



APROL Python for UCB



■ UCB...User configuration block

- Interface to operating system functionalities
- Automatic execution of scripts via trigger signal
- Use of any script language
- APROL System service UCB-Server is responsible for execution
- The first input pin (“UcbTrigger”) is mandatory and automatically generated
- The first output pin (“UcbStatus”) is mandatory and automatically generated
- Communication from block inputs to script with environment variables
- Communication from script to block outputs with stdout
- Doc B3.2.12
- Very detailed description of UCB block creation in Doc B3.2.12.2

■ Workshop

- Create an UCB-block
- Block reads system time with python (module time; function ctime())
- Block provides this time on an output
- Test this block in an CFC; block “AprFbTrig” can be used as trigger



Extended Workshop



■ APROL Widgets in Visualization

- Widgets can be used in a graphic block (context library)
- Documentation B2.6.4 (Widgets in Visualization)
- Documentation D2.1.3.12 (BlockInstWidgetMap)
- Can be used with pins (global variables)
- Extended functionality with python

■ Workshop(1)

- Use one widget using global variables (InputOutputBox)
 - Graphic block with one output pin
 - Entered value in the InputOutputBox should be written to the output pin



■ Workshop(2)

- Use ListView widget, fill up the content with Python-API
 - Need to assign a control variable in widget properties
 - Accessible via “object name” from widget properties
 - `block.widget.YOUROBJECTNAME.set_columns()`
 - See D2.1.3.12.3
 - `.set_columns()` to set the header of the ListView
 - `.set_items()` to set the content of the ListView
 - `.selected_item().value` delivers the value of the selected entry
 - `.selected_item().columns` delivers the contents from selected row

■ Workshop(3)

- Use `DC.Timer`
- Object oriented programming necessary
- Count up a value periodically as basis for a dynamization (e.g. rotate a rectangle)



YOUR GLOBAL PARTNER FOR AUTOMATION EXCELLENCE