INTRODUCTION

In this document, you will find a description of each class and function used by the FPD software for the pot motion. The 3 types of files are :

- '.py' files (code source);
- '.dbm' files (local databases);
- '.gif' files (images).

The list of code source files is the following:

- FPDGui.py;
- FPDConstantVariables.py;
- FPDVariables.py;
- FPDDatabase.py;
- FPDExpertGui.py;
- FPDHelp.py;
- FPDOptionGui.py;
- FPDRun.py;
- MyD0Gui.py;
- MyPmw.py;
- MyTkinter.py.

Two local database files (FPDConfig.dbm and Potdbase.dbm) allow to save some local parameters.

Twenty Six image files are using for the small running frog. This animation prevents the users when the software become stuck (Frog freezed = sofware stucked).

Part I

FPDGui.py

Chapter 1

Introduction

This file (FPDGui.py) is the main program.

By typing the command `./FPDGui.py`, the grapihcal interface will be popped up.

1.1 Comment

This comment will be print on xterm window when the sofware will start.

```
"""
  FFFFFFFFFFFF   PPPPPPPPPP        DDDDDDDDDD
  FFFFFFFFFFFF   PPPPP   PPPPP     DDDDD   DDDDD
  FFFFF          PPPPP     PPPPP   DDDDD      DDDDD
  FFFFF          PPPPP     PPPPP   DDDDD      DDDDD
  FFFFFFFFFF     PPPPP   PPPPP     DDDDD      DDDDD
  FFFFFFFFFF     PPPPPPPPPP        DDDDD      DDDDD
  FFFFF          PPPPP             DDDDD      DDDDD
  FFFFF          PPPPP             DDDDD      DDDDD
  FFFFF          PPPPP             DDDDD      DDDDD
  FFFFF          PPPPP             DDDDD   DDDDD
  FFFFF          PPPPP             DDDDDDDDDD
"""
```

1.2 Libraries

3 types of libraries are used:

- python : Tkinter, Dialog, sys, tempfile;
- D0 : helpD0Dialog;
- FPD : FPDExpertGui, FPDRun, FPDDatabase, FPDVariables, FPDHelp, FPDOptionGui and helpD0Dialog.

```
from Tkinter import *
from Dialog import *
import sys
import tempfile
from FPDExpertGui import *
from FPDRun import *
```

Chapter 2

FPDVariable class

2.1 Introduction

This class define all the variables. They are first created in the `__init__` class and then updated in the `update` class. The class `__del__` is called when the Exit command is used. This class kills all variables created.

2.2 Definition

The superclass is `Database` defined in FPDDatabase.py file.

```
class FPDVariable(Database):
```

2.3 Comments

These comments will be print on xterm window by calling the command `print FPDVariable.__doc__`.

```
    """
##############################################################################

   INITIALIZATION VARIABLES :
       - POT
       - Rack Monitor
       - Power Box Electronic
       - Power Box Driver
       - Smoke Detector
       - Emergency Line
       - Control Line
       - Power PC
       - Beam

    """
```

2.4 Function __init__

This function prints the previous comments and call all the classes where the variables are defined. The double call of the `update` function allows to set the correct value of each variable before creating the main window.

All these classes are defined in the FPDVariables.py file.

```
    def __init__(self, parent):

        print FPDVariable.__doc__

        self.Files_Directories = Files_DirectoriesVariables(parent, 'FILES')

        self.Pot = PotVariables(parent, 'POT')
        self.RM = RMVariables(parent, 'RM')
        self.PB_Electronic = PB_ElectronicVariables(parent, 'PB_ELECTRONIC')
        self.PB_Driver = PB_DriverVariables(parent, 'PB_DRIVER')
        self.Smoke = SmokeVariables(parent, 'SMOKE_DETECTOR')
        self.Emergency_Line = Emergency_LineVariables(parent, 'EMERGENCY_LINE')
        self.Control_Line = Control_LineVariables(parent, 'CONTROL_LINE')
        self.Power_PC = Power_PCVariables(parent, 'POWERPC')
        self.Beam = Beam_Variables(parent, 'BEAM')

        self.update()
        self.update()

        return
```

2.5 Function __del__

Erase all the variables. The order is important. The Rack Monitor variables(`self.RM.__del__()`) must be the last variables deleted. Also you need to call the `self.RM.__del__() function for these variables and not \verb self...`— because the update of these variables is different from the others.

```
def __del__(self):

        del self.Files_Directories
        del self.Pot
        del self.Smoke
        del self.PB_Driver
        del self.PB_Electronic
        del self.Emergency_Line
        del self.Control_Line
        del self.Power_PC
        del self.Beam
        self.RM.__del__()
```

2.6 Function update

Update all the variables. Only the Rack Monitor variables are not updated in this function.

```
def update(self):

        self.Files_Directories.update()
        self.Pot.update()
        self.Smoke.update()
        self.PB_Driver.update()
        self.PB_Electronic.update()
        self.Emergency_Line.update()
        self.Control_Line.update()
        self.Power_PC.update()
        self.Beam.update()
```

Chapter 3

FPDMenuFrame class

3.1 Introduction

This class creates the menubar.

3.2 Definition

The superclasses are :

- Frame (python class);
- FPDLogFileFrame (FPD class the FPDOptionGui.py file).

```
class FPDMenuFrame(Frame, FPDLogFileFrame):
```

3.3 Function __init__

This function creates variables and call `__CreateMenuBar` and `CreateGuiLogFile` functions located in this class and in the `FPDLogFileFrame` class.

```
def __init__(self,parent):

        Frame.__init__(self, parent, relief = GROOVE, borderwidth = 4)

        self.parent = parent

        self.ALL_PB_ELECTRONIC = IntVar()
        self.ALL_PB_ELECTRONIC.set(0)

        self.parent = parent
        self.expert_frame = 0

        self.__CreateMenuBar()
        self.CreateGuiLogFile(parent)

        return
```

3.4 Function __CreateMenuBar

This function calls other functions where each command is defined.

```
def __CreateMenuBar(self):

        f = Frame(self)

        for menu in ['File', 'Alarm', 'Command', 'Expert', 'Init']:

                m = eval('self.' + menu + 'Menu(self)')
                m.pack(side = LEFT)

        m=self.HelpMenu(self)
        m.pack(side = RIGHT)

        f.pack(side = TOP,fill = X)
```

3.5 Function FileMenu

Create the `Exit` command in the `File` menu, This command calls the `quit` function defined in this file.

```
def FileMenu(self, parent):

        mbutton=Menubutton(parent, text = 'File')
        mbutton.pack(side = LEFT)

        menu = Menu(mbutton)
        menu.add_command(label = 'Exit', command = quit,font = Style)
        mbutton['menu'] = menu

        return mbutton
```

3.6 Function AlarmMenu

Create the `Alarm` menu. This menu contains the name of all pots. By selecting one, it calls the `Log_Files` function defined in the `FPDLogFileFrame` class.

```
def AlarmMenu(self, parent):

        mbutton = Menubutton(parent, text = 'Alarms')
        mbutton.pack(side = LEFT)

        menu = Menu(mbutton)

        rack = 'DIPOLES'

        for key in POT_ORDER:

                if POT[key]['RACK'] is not rack:
                        menu.add_separator({})
                        rack = POT[key]['RACK']

                menu.add_command(label = key, font = Style,
                                  command = lambda s = self,k = key: FPDMenuFrame.Log_Files(s,k))

        mbutton['menu'] = menu

        return mbutton
```

3.7 Function CommandMenu

The commands of this menu are:

- `Pot In` which calls the `Run_Start` function;
- `Pot Out` which calls the `Run_Stop` function;
- `Amplifier & LVDTs` which pops up a new menu. This new menu allows to set each `CONTROL` variables of `PB_ELECTRONIC` or calls the `AllAmplifierCrate` function;
- `High Voltage` which calls the `HV` function;
- `Smoke Detector Reset` which calls the `SmokeDetector` function.

```
def CommandMenu(self, parent):

        mbutton = Menubutton(parent, text = 'Command')
        mbutton.pack(side = LEFT)

        menu = Menu(mbutton)

        menu.add_command(label = 'Pot In', font = Style,
                        command = self.Run_Start)
        menu.add_command(label = 'Pot Out', font = Style,
                        command = self.Run_Stop)

        menu.add_separator({})

        submenu_amplifier = Menu(menu)

        for key  in RACK_ORDER:
                submenu_amplifier.add_checkbutton(label = key, font = Style,
                                                  indicatoron = 'on', selectcolor = Color[2],
                                                  command = self.AmplifierCrate,
                                                  variable = PB_ELECTRONIC[key]['CONTROL'],
                                                  onvalue = 1, offvalue = 0)

        submenu_amplifier.add_checkbutton(label = 'All', font = Style,
                                          indicatoron = 'on', selectcolor = Color[2],
                                          command = self.AllAmplifierCrate,
                                          variable = self.ALL_PB_ELECTRONIC,
                                          onvalue = 1, offvalue = 0)

        menu.add_cascade(label = 'Amplifiers & LVDTs',font = Style,
                         menu = submenu_amplifier)

        menu.add_separator({})
```

```
        menu.add_command(label = 'High Voltage', font = Style,
                                command = self.HV)

        menu.add_separator({})

        menu.add_command(label = 'Smoke Detector Reset', font = Style,
                         command = self.SmokeDetector)
        mbutton['menu'] = menu

        return mbutton
```

3.8 Function Run_Start

This function will set at 1 the `MODE` variable of `POT` when the automatic mode will be available.

```
    def Run_Start(self):
        for key in POT_ORDER:
            POT[key]['MODE'].set(0)                  # Need to be 1 for the version with the automatic mode
```

3.9 Function Run_Stop

This function sets at 0 the `MODE` variable of `POT`.

```
    def Run_Stop(self):
        for key in POT_ORDER:
            POT[key]['MODE'].set(0)
```

3.10 Function HV

Load the high voltage software. The name and path are defined in `FILE NAME` and `HV FILE` of `FILES['HV FILE']`.

```
    def HV(self):
        file = os.path.join(FILES['HV FILE']['DIRECTORY NAME'].get(), FILES['HV FILE']['FILE NAME'].get())
        os.system(file + ' &')
```

3.11 Function ExpertMenu

Calls the function `Expert`.

```
    def ExpertMenu(self, parent):

        mbutton = Menubutton(parent, text = 'Expert')
        mbutton.pack(side = LEFT)

        menu = Menu(mbutton)

        menu.add_command(label = 'Expert', command = self.Expert, font = Style)
        mbutton['menu'] = menu

        return mbutton
```

3.12 Function InitMenu

Drawns the Init menu and Init function will be called when this menu will be selected.

```
    def InitMenu(self, parent):

        mbutton = Menubutton(parent, text = 'Init')
        mbutton.pack(side = LEFT)

        menu = Menu(mbutton)

        menu.add_command(label = 'Init', command = self.Init, font = Style)
        mbutton['menu'] = menu

        return mbutton
```

3.13 Function Init

Set the variables `HOME`, `STEP`, `SPEED` and `MOTION` of `POT`. This will move all pots at home position.

```
    def Init(self):

        for key in POT_ORDER:
            if not POT[key]['HOME'].get():
                POT[key]['STEP'].set(20)
                POT[key]['SPEED'].set(1)
            else:
                POT[key]['STEP'].set(0)
                POT[key]['MOTION'].set(0)
```

3.14 Function HelpMenu

In this menu, 2 commands are defined:

- `Help` which calls the `HelpFrame` function;
- `About...` which calls the `AboutDialogFrame` function.

```
def HelpMenu(self, parent):

        mbutton = Menubutton(parent, text = 'Help')
        mbutton.pack(side = RIGHT)

        menu = Menu(mbutton)

        menu.add_command(label = 'Help', command = self.HelpFrameself.A, font = Style)
        menu.add_separator({})
        menu.add_command(label = 'About...', command = self.AboutDialogFrame, font = Style)
        mbutton['menu'] = menu

        return mbutton
```

3.15 Function AboutDialogFrame

Pop up the About dialog.

```
def AboutDialogFrame(self):

        Pmw.aboutversion('02.04.02')
        Pmw.aboutcontact(
                'For information about this application contact : \n'
                '    Lionel Peyrichoux (peyricli@fnal.gov) \n'
                )
        Pmw.AboutDialog(self.parent, applicationname = 'Forward Proton Detector').show()
```

3.16 Function HelpFrame

Calls the `HelpFrame` class defined in `helpD0Dialog` file. The text `HelpFPDStandard.__doc__` is located in `FPDHelp.py` file.

```
def HelpFrame(self):

            HelpDialog(self.parent, HelpFPDStandard.__doc__)
```

3.17 Function AmplifierCrate

Set at 0 the `ALL_PB_ELECTRONIC` variable.

```
def AmplifierCrate(self):

        self.ALL_PB_ELECTRONIC.set(0)

        return
```

3.18 Function AllAmplifierCrate

Set the `CONTROL` variable of `PB_ELECTRONIC` at the value of `ALL_PB_ELECTRONIC` variable (0 or 1).

```
def AllAmplifierCrate(self):

        for key in PB_ELECTRONIC.keys():
                PB_ELECTRONIC[key]['CONTROL'].set(self.ALL_PB_ELECTRONIC.get())

        return
```

3.19 Function SmokeDetector

Set at 0 the `CONTROL` variable of `SMOKE_DETECTOR`.

```
def SmokeDetector(self):

        for key in SMOKE_DETECTOR.keys():
                SMOKE_DETECTOR[key]['CONTROL'].set(0)
```

3.20 Function Expert

Pop up a new dialog window and calls the `Expert_Activate` function when one of the 2 button (`OK` `Cancel`) is pressed.

```
def Expert(self):

        if not self.expert_frame:
                self.dialog_pwd = Pmw.PromptDialog(self.parent,
                                                   title = 'Password',
                                                   label_text = 'Password :',
                                                   entryfield_labelpos = 'n', entry_show = '*',
                                                  defaultbutton = 0, buttons = ('OK', 'Cancel'),
                                                   command = self.Expert_Activate)
                self.dialog_pwd.withdraw()
                self.dialog_pwd.activate()
```

3.21 Function Expert_Activate

Load the expert mode if `OK` button is clicked and the password is correct.

```
def Expert_Activate(self, result):

        if result is 'OK':
                if self.dialog_pwd.get() == 'FPD2000':
                        self.expert = ExpertFrame(self)
                        self.expert_frame = 1

        self.dialog_pwd.clear()
        self.dialog_pwd.deactivate()

        return
```

3.22 Function update

Try to update the display of the expert mode.

```
def update(self):

        try:
                self.expert.update_display()
                self.expert_frame = 1
        except:
                self.expert_frame = 0

        return
```

Chapter 4

FPDCreatePositionPotMenu class

4.1 Introduction

This class creates the scales and buttons for each pot in the main window. The scales show the position of each pot in millimeter. The buttons pop up a new window where all the informations (status, alarms...) are printed.

4.2 Definition

The superclass is `Frame` (python class).

Two global variables `POT` and `State_Checkbutton` are created. The value of `State_Checkbutton` is `DISABLED` or `NORMAL`.

```
class FPDCreatePositionPotMenu(Frame):

        global POT
        global State_Checkbutton

        State_Checkbutton = [DISABLED,NORMAL]
```

4.3 Function __init__

Creates 2 variables `Scale_Status` and `Button_Status` and calls the function `__CreateGui` and the class `FPDStatusPotFrame` defined in `FPDOptionGui.py` file.

```
        def __init__(self,parent):

                Frame.__init__(self,parent,relief=GROOVE,borderwidth=4)

                self.Scale_Status = []
                self.Button_Status = []
                self.__CreateGui(parent)

                self.__fpdSPF = FPDStatusPotFrame(self)

                return
```

4.4 Function __CreateGui

Create the scale and checkbutton from each pot. The checkbutton will call the function `Status_Pot` defined in this class.

Calls the `Balloon` class.

```
        def __CreateGui(self,parent):

                self.balloon = Pmw.Balloon(parent)

                group = Pmw.Group(self,
                                  tag_text = 'Position')
                group.pack()

                row = 0
                column = 0
                num = 0

                for key in POT_ORDER:
```

```
row = num / 2
column = 5 * (num % 2)

self.Button_Status.insert(num, IntVar())

cb = Checkbutton(group.interior(),
                text = key, anchor = 's', font = Style,
                indicatoron = 'off', selectcolor = 'GREY',
                variable = self.Button_Status[num],
                command = lambda s = self, n = num : FPDCreatePositionPotMenu.Status_Pot(s, n),
                width = 15)
cb.grid(row = row, column = column, sticky = 'w')

self.Scale_Status.insert(num, Scale(group.interior(),
                                 to=50, length=100, width=10,
                                 resolution = 0.01, tickinterval = 50, sliderlength = 5,
                                 variable = POT[key]['CURRENT POSITION'],
                                 orient = HORIZONTAL, showvalue = TRUE, font = Style))
 self.Scale_Status[num].grid(row = row, column = column + 1)

 Label(group.interior() ,anchor='s' ,text='        ').grid(row = row, column = column + 4, sticky = 'w')

 num = num + 1
```

4.5 Function Status_Pot

calls the function `DrawGui` of the class `FPDStatusPotFrame`.

```
def Status_Pot(self, num):

        self.__fpdSPF.DrawGui(num,self.Button_Status[num].get())

        return
```

4.6 Function update

Update status, color and text in the balloon of each scale.

```
def update(self):

        num = 0
        for key in POT_ORDER:

                failures = (POT[key]['DRIVER FAILURE'].get()                    # 0 if no failures
                           | POT[key]['MECHANICAL FAILURE'].get()
                           | POT[key]['CABLE FAILURE'].get())

                status = ((not POT[key]['ENABLED-DISABLED'].get()) * 3 +
                          POT[key]['ENABLED-DISABLED'].get() * (not failures) *  (
                                      (not PB_ELECTRONIC[POT[key]['RACK']]['CONTROL'].get()) +
                                       PB_ELECTRONIC[POT[key]['RACK']]['CONTROL'].get() * (
                                               (not POT[key]['INIT'].get()) * (POT[key]['STATUS'].get() * 2 + (not POT[key]['STATUS'].get()) * 4) +
                                              (POT[key]['INIT'].get() * 6))))

                self.Scale_Status[num].config(troughcolor = Color[status])

                text = ''

                if status != 3:
                        if POT[key]['STATUS'].get():
                                text = 'Running, '
                        else:
                                text = 'Stopped at '
                        if POT[key]['HOME'].get():
                                text = text + 'Home postion'
                        else:
                                text = text + ('position : %6.2f' %(POT[key]['CURRENT POSITION'].get()))
                        if POT[key]['INIT'].get():
                                text = 'Initialization'
                else:
                        text = Status[status]

                self.balloon.bind(self.Scale_Status[num], text)

                num = num + 1

        return
```

Chapter 5

FPDCreateLogbookPot class

5.1 Introduction

The superclass is `Frame` (python class). This class creates the logbook in the main window.

5.2 Definition

The superclass is `Frame` (python class).

```
class FPDCreateLogbookPot(Frame):
```

5.3 Function __init__

Calls the `__CreateGui` function.

```
    def __init__(self, parent):

        Frame.__init__(self, parent, relief = GROOVE, borderwidth = 4)

        self.__CreateGui()

        return
```

5.4 Function __CreateGui

Create the text zone. Three type of text are used :

- Alarm in black but will be red;
- Normal in black but will be green;
- Warning in black but will be orange.

```
    def __CreateGui(self):

        group = Pmw.Group(self,
                        tag_text = 'Logbook')
        group.pack(fill = 'both', expand = 'yes')

        scrollbar = Scrollbar(group.interior())
        scrollbar.pack(side = RIGHT, fill = Y)

        LogbookText.insert(0, 'instance')
        LogbookText[0] = Text(group.interior(), height = 10, width = 50, yscrollcommand = scrollbar.set)
        LogbookText[0].pack(side = TOP)

        LogbookText[0].tag_config("a", foreground = 'black')              # Alarm tag
        LogbookText[0].tag_config("n", foreground = 'black')              # Normal tag
        LogbookText[0].tag_config("w", foreground = 'black')              # Warning tag

        scrollbar.config(command = LogbookText[0].yview)

        LogbookText[0].insert(END, 'Start Software', "n")
```

5.5 Function update

Nothing

```
def update(self):
        return
```

Chapter 6

FPDCreateHighVoltage class

6.1 Introduction

Creates the high voltage area. This show the status of the high voltage but doesn't control it.

6.2 Definition

The superclass is `Frame` (python class).

```
class FPDCreateHighVoltage(Frame):
```

6.3 Function __init__

Calls the `__CreateGui` function.

```
def __init__(self, parent):

        Frame.__init__(self, parent, relief = GROOVE, borderwidth = 4)

        self.__CreateGui(parent)

        return
```

6.4 Function __CreateGui

Create 2 labels `Level 0` and `MAPMT` and the balloons.

```
def __CreateGui(self, parent):

        self.balloon = Pmw.Balloon(parent, label_font = Style)

        group = Pmw.Group(self,
                        tag_text = 'High Voltage')

        group.pack(side = TOP, expand = 'yes', fill = BOTH)

        self.HV_L0 = Label(group.interior(),
                              text = 'Level 0',
                              relief = RAISED, bd = 2)
        self.HV_L0.pack(side = LEFT, expand = 'yes', fill = BOTH)
        self.HV_L0.bind('<Enter>', self._HV_L0Balloon)

        self.HV_MAPMT = Label(group.interior(),
                             text ='MAPMT',
                             relief = RAISED, bd = 2)
        self.HV_MAPMT.pack(side = LEFT, expand = 'yes', fill = BOTH)
        self.HV_MAPMT.bind('<Enter>', self._HV_MAPMTBalloon)
```

6.5 Function _HV_L0Balloon

Updates the text balloon of `Level 0` label.

```
def _HV_L0Balloon(self, event):

        text_L0 = ''
        rack = ''

        for key in POT_ORDER:
```

```
		if POT[key]['HV LO'].get() != 3:
			if rack != POT[key]['RACK']:
				text_LO = text_LO + '\n' + POT[key]['RACK'] + ' On'
				rack = POT[key]['RACK']
		else:
			if rack != POT[key]['RACK']:
				text_LO = text_LO + '\n' + POT[key]['RACK'] + ' Off'
				rack = POT[key]['RACK']

	self.balloon.bind(self.HV_LO, text_LO)

	return
```

6.6 Function _HV_MAPMTBalloon

Updates the text balloon of the `MAPMT` label.

```
def _HV_MAPMTBalloon(self, event):

	text_PM = ''

	for key in POT_ORDER:

		if POT[key]['HV PM'].get() != 3:
			text_PM = text_PM + '\n' + key + ' On'
		else:
			text_PM = text_PM + '\n' + key + ' Off'

	self.balloon.bind(self.HV_MAPMT, text_PM)

	return
```

6.7 Function update

Update the colors :

- red if all high voltages are off;
- green if one of them is on.

```
def update(self):

	color_PM = 0
	color_LO = 0

	for key in POT_ORDER:

		if POT[key]['HV PM'].get() != 3:
			color_PM = 1

		if POT[key]['HV LO'].get() != 3:
			color_LO = 1

	self.HV_MAPMT.config(bg = Color[2 * color_PM])
	self.HV_LO.config(bg = Color[2 * color_LO])

	return
```

Chapter 7

FPDCreateAmplifierLVDT class

7.1 Introduction

Creates the amplifer and LVDT area. This show the status of the electronic power box (Amplifier crate and LVDTs) in each rack.

7.2 Definition

The superclass is `Frame` (python class).

```
class FPDCreateAmplifierLVDT(Frame):
```

7.3 Function __init__

Calls the `__CreateGui` function.

```
def __init__(self, parent):

        Frame.__init__(self, parent, relief = GROOVE, borderwidth = 4)

        self.__CreateGui(parent)

        return
```

7.4 Function __CreateGui

Create a label for each rack.

```
def __CreateGui(self,parent):

        group = Pmw.Group(self,
                          tag_text = 'Amplifiers & LVDTs')

        group.pack(side = TOP, expand = 'yes', fill = BOTH)

        self.Amplifier = {}

        for key in RACK_ORDER:

                self.Amplifier[key] = Label(group.interior(),
                                            text=key, font = Style,
                                            relief = RAISED, bd = 2)
                self.Amplifier[key].pack(side = LEFT, expand = 'yes', fill = BOTH)
```

7.5 Function update

Updates the color and the text of each label.

```
def update(self):

        for key in RACK_ORDER:

                self.Amplifier[key].config(bg = Color[2 * PB_ELECTRONIC[key]['CONTROL'].get()])

                if PB_ELECTRONIC[key]['STATUS'].get():
                        self.Amplifier[key].config(text = key + ' On')
                else:
```

```
            self.Amplifier[key].config(text = key + ' Off')
    del key
```

Chapter 8

FPDCreatePower class

8.1 Introduction

This area shows the status of the driver's power box and control line.

8.2 Definition

The superclass is `Frame` (python class).

```
class FPDCreatePower(Frame):
```

8.3 Function __init__

Calls the `__CreateGui` function.

```
def __init__(self,parent):

        Frame.__init__(self,parent,relief=GROOVE,borderwidth=4)

        self.__CreateGui(parent)
        self.parent = parent

        return
```

8.4 Function __CreateGui

Creates one label for the control line and one label for each castle.

```
def __CreateGui(self,parent):

        group = Pmw.Group(self,
                          tag_text = 'Power')

        group.pack(side = TOP, expand = 'yes', fill = BOTH)

        self.ControlLine = Label(group.interior(),
                                 text='Control Line', font = Style,
                                 relief = RAISED, bd = 2)
        self.ControlLine.pack(side = LEFT, expand = 'yes', fill = 'both')

        self.PB = {}

        for key in RACK_ORDER:

                self.PB[key] = Label(group.interior(),
                                     text = key, font = Style,
                                     relief = RAISED, bd = 2)
                self.PB[key].pack(side = LEFT, expand = 'yes', fill = 'both')
```

8.5 Function update

Update color and text of each label.

```
def update(self):

        controlline = 0

        for key in RACK_ORDER:

                self.PB[key].config(bg = Color[2 * PB_DRIVER[key]['STATUS'].get()])

                if PB_DRIVER[key]['STATUS'].get():
                        self.PB[key].config(text = key + ' On')
                else:
                        self.PB[key].config(text = key + ' Off')

                controlline = controlline | CONTROL_LINE[key]['STATUS'].get()

        if controlline:
                self.ControlLine.config(text = 'Control Line On', bg = Color[2])
        else:
                self.ControlLine.config(text = 'Control Line Off', bg = Color[0])

        return
```

Chapter 9

FPDCreateInfo class

9.1 Introduction

This class creates 2 notebooks (`Beam` and `Power PC`) where the beam parameters and power pc parameters are updated.

9.2 Definition

The superclass is `Frame` (python class).

```
class FPDCreateInfo(Frame):
```

9.3 Function __init__

Calls the `__CreateNotebook` function.

```
    def __init__(self, parent):

        Frame.__init__(self, parent, relief = GROOVE, borderwidth = 4)

        self.__CreateNotebook()

        return
```

9.4 Function __CreateNotebook

Create the 2 notebooks `Beam` and `Power PC`

```
    def __CreateNotebook(self):

        group = Pmw.Group(self,
                          tag_text = 'Informations')

        group.pack(side = TOP, expand = 'yes', fill = 'both')

        notebook = Pmw.NoteBook(group.interior())
        notebook.pack(fill = BOTH, expand = 'yes')

        beam = notebook.add('Beam')

        powerpc = notebook.add('Power PC')

        num = 0

        for key in POWERPC.keys():

            l = Label(powerpc,
                      text = POWERPC[key]['DESC'])
            l.grid(row = num, column = 0)

            l = Label(powerpc,
                      textvariable = POWERPC[key]['VALUE'])
            l.grid(row = num, column = 1)

            l = Label(powerpc,
                      text = POWERPC[key]['UNIT'])
            l.grid(row = num, column = 2)

            num = num + 1

        for key in BEAM.keys():
```

```
l = Label(beam,
          text = BEAM[key]['DESC'])
l.grid(row = num, column = 0)

l = Label(beam,
          textvariable = BEAM[key]['VALUE'])
l.grid(row = num, column = 1)

l = Label(beam,
          text = BEAM[key]['UNIT'])
l.grid(row = num, column = 2)

num = num + 1
```

9.5 Function update

Nothing

```
def update(self):

        return
```

Chapter 10

FPDCreateStatusBar class

10.1 Introduction

This class creates the status bar in the bottom of the main window.

10.2 Definition

The superclass is `Frame` (python class).

```
class FPDCreateStatusBar(Frame):
```

10.3 Function __init__

Creates 2 variables `date` and `num_image` and calls the `__CreateStatusBar` function. In the first label will be printed the animation, in the second the time.

```
    def __init__(self, parent):

        Frame.__init__(self, parent, relief=GROOVE, borderwidth=4)

        self.date = IntVar()
        self.num_image = IntVar()

        self.num_image.set(1)

        self.__CreateStatusBar()
        self.date.set(time.asctime(time.localtime(time.time())))

        return
```

10.4 Function __CreateStatusBar

Create 2 labels in the status bar. In the first label will be printed the images animation, in the second label date and time.

```
    def __CreateStatusBar(self):

        f = Frame(self)

        Label(f, image = Animation, relief = SUNKEN).pack(side = LEFT)
        Label(f, textvariable = self.date, bd = 2).pack(side = RIGHT)

        f.pack(side = RIGHT, expand = 'yes', fill = 'both')
```

10.5 Function update

Update the `date` variable.

```
    def update(self):

        self.date.set(time.asctime(time.localtime(time.time())))

        return
```

Chapter 11

FPDGui class

11.1 Introduction

This class is the main class. It calls all the others classes and update it.

11.2 Definition

The superclass is `Frame` (python class).

```
class FPDGui(Frame):
```

11.3 Comments

These comments will be print on xterm window by calling the command `print FPDGui.__doc__`.

```
     """
############################################################################

   INITIALIZATION GUI :
        - MenuBar
        - Frames
        - StatusBar

     """
```

11.4 Function __init__

calls the classes :

- `Database` in `FPDDatabase` class;
- `FPDVariable`;
- `FPDLogFileFrame`;
- `FPDMenuFrame`;
- `FPDCreatePositionPotMenu`;
- `FPDCreateLogbookPot`;
- `FPDCreateHighVoltage`;
- `FPDCreateAmplifierLVDT`;
- `FPDCreatePower`;
- `FPDCreateInfo`;
- `FPDCreateStatusBar`;

- FPDRunMotor in FPDRun class;

and the update function.

```
def __init__(self, parent, quitCallback = None):

        print FPDGui.__doc__

        db = Database()

        Frame.__init__(self,parent)

        self.__FPDVariable = FPDVariable(parent)

        FPDLogFileFrame(parent)

        self.__CreateMenuBar = FPDMenuFrame(parent)
        self.__CreateMenuBar.pack(side = TOP, fill = X)

        f = Frame(parent)

        self.__CreatePositionPot = FPDCreatePositionPotMenu(f)
        self.__CreatePositionPot.pack(side = LEFT, fill = BOTH)

        f1 = Frame(f)

        self.__CreateLogbookPot = FPDCreateLogbookPot(f1)
        self.__CreateLogbookPot.pack(side = TOP, fill = X)

        self.__CreateHighVoltage = FPDCreateHighVoltage(f1)
        self.__CreateHighVoltage.pack(side = TOP, fill = 'both', expand = 'yes')

        self.__CreateAmplifierLVDT = FPDCreateAmplifierLVDT(f1)
        self.__CreateAmplifierLVDT.pack(side = TOP, fill = 'both', expand = 'yes')

        self.__CreatePower = FPDCreatePower(f1)
        self.__CreatePower.pack(side = TOP, fill = 'both', expand = 'yes')

        self.__CreateInfo = FPDCreateInfo(f1)
        self.__CreateInfo.pack(side = TOP, fill = 'both', expand = 'yes')

        f1.pack(side = LEFT)

        f.pack(side = TOP)

        self.__CreateStatusBar = FPDCreateStatusBar(parent)
        self.__CreateStatusBar.pack(side = TOP, fill = X)

        self.__FPDRunMotor = FPDRunMotor(parent)

        self.update()

        return
```

11.5 Function destroy

Deletes all the variables.

```
def destroy(self):

        del self.__FPDVariable
        print 'Variables destroyed'
```

11.6 Function update

Calls all update function of each class, reset the Time_Run and waits 1 second before calling the same function.

```
def update(self):

        self.__FPDVariable.update()
        self.__CreatePositionPot.update()
        self.__CreateMenuBar.update()
        self.__CreateLogbookPot.update()
        self.__CreateHighVoltage.update()
        self.__CreateAmplifierLVDT.update()
        self.__CreatePower.update()
        self.__CreateInfo.update()
        self.__CreateStatusBar.update()
        self.__FPDRunMotor.update()

        Time_Run.set(0)

        self.after(1000, self.update)

        return
```

Chapter 12

Function destroy

12.1 Introduction

This function destroies the main window.

12.2 Function destroy

Calls the `destroy` function of `FPDGui` class.

```
def destroy():
        fpdgui.destroy()
        return
```

Chapter 13

Function quit

13.1 Introduction

This function shows a new dialog window before leaving of the software.

13.2 Function quit

Show the dialog window and calls the `quit` and `destroy` functions.

```
def quit():

        text='Are you sure you want to quit?' + chr(13)

        ans = Dialog(title = 'Exit',
                     text = text, font = ("Times", 12),
                     bitmap = 'question',
                     strings = ('Yes', 'No',), default = 0)

        if ans.num is 0:
                fpdgui.quit()
                destroy()

        return
```

Chapter 14

FPDSoft class

14.1 Introduction

This class verifies the update time of the software.

14.2 Definition

The superclass is `Frame` (python class).

```
class FPDSoft(Frame):
```

14.3 Function __init__

Calls the `update` function.

```
	def __init__(self, parent):

		Frame.__init__(self,parent)

		self.update()

		return
```

14.4 Function update

Changes the image in the label defined in the `FPDCreateStatusBar` class. Prints `Software not updated` and sounds if the update time is too long.

```
	def update(self):

		if Num_Animation.get() >= 26:
			Num_Animation.set(1)
		elif Time_Run.get() < 400:
			Num_Animation.set(Num_Animation.get() + 1)

		if Time_Run.get() > 100:
			print 'Software not updated'
			self.bell()

		Time_Run.set(Time_Run.get() + 1)

		name_image = 'ANIM/anim_%02d.gif'  %(Num_Animation.get())
		Animation.config(file=name_image)

		self.after(150, self.update)

		return
```

Chapter 15

Main Program

15.1 Introduction

This is the main program.

15.2 Main Program

Check if the software is not running (`fpd.lock`) and then calls the `FPDGui` and `FPDSoft` classes.

```
if (__name__ == '__main__'):

        global fpdgui
        global Animation
        global Num_Animation
        global Time_Run

        print __doc__

        try:
                tempfile.template = 'fpd.lock'

                root = Tk()
                root.title('Forward Proton Detector')

                Animation = PhotoImage(file='ANIM/anim_01.gif')
                Num_Animation = IntVar()
                Num_Animation.set(0)
                Time_Run = IntVar()
                Time_Run.set(0)

                fpdgui = FPDGui(root)
                fpdgui.pack(fill = X, expand = YES, ipadx = 0, ipady = 0)
                FPDSoft(root)
                root.mainloop()

                print 'BYE     -       FPD SOFTWARE'
        except:
                print 'Software is running'
```

Part II

FPDConstantVariables

Chapter 1

Introduction

All constant variables are defined in this files :

- `POT` and `POT_ORDER`;
- `RM` (Rack monitor);
- `POWERPC` (power pc or VME);
- `RACK` and `RACK_ORDER`;
- `PB_ELECTRONIC` (Power box for the electronic);
- `PB_DRIVER` (Power box for the drivers);
- `SMOKE_DETECTOR`;
- `EMERGENCY_LINE`;
- `CONTROL_LINE`;
- `FILES`;
- `BEAM`.

1.1 Libraries

Only one library is used in this file `D0GuiDefaults`. it's a D0 library.

```
from D0GuiDefaults import *
```

Chapter 2

Constant variables definition

2.1 Global definition

Define the variables as global variables used i by the others python files.

```
global POWERPC
global POT
global RM
global PB_ELECTRONIC
global MOTION
global SWITCHES
global LogbookText
```

2.2 Other variables

The variables `YES_NO`, `RUN_STOP`, `SPEED` and `DIRECTION` are used some `POT` variables in `FPDVariables.py` file.

The variable `Style` defines the style and size of the fonts.

The `LogbookText` will be defined in the `FPDCreateLogbookPot` class in `FPDGui.py` file.

The `Bad_Status_Text` variable will containt the text when the motor will move without the RUN command.

The `Status` variable is used for the balloon window in the `FDPCreatePositionPotMenu` class in the `FPDGui.py` file.

The `Color` variable uses the D0 colors defined in `D0GuiDefaults.py` file.

```
YES_NO  = ["NO","YES"]
RUN_STOP = ["STOP","RUN"]
SPEED = ["LOW","HIGH"]
DIRECTION = ["OUT","IN"]
Style = ("Times",12)
LogbookText = []
Bad_Status_Text = []
Status = ['Failures', 'OK', '', 'Disabled']
Color = [d0_red, d0_orange, d0_green, d0_grey, d0_blue, d0_purple, d0_yellow]
```

2.3 POWERPC Variables

the 3 informations from the Power PC are :

- CPU usage;
- Memory usage;
- files descriptor.

```
POWERPC = {
    'CTL_PROC_02/CPU':{
        'DESC':'CPU usage', 'UNIT':'%',
        'GUI':''},
    'CTL_PROC_02/MEM':{
        'DESC':'memory usage', 'UNIT':'%',
        'GUI':''},
    'CTL_PROC_02/FD':{
        'DESC':'file descriptors', 'UNIT':'%',
        'GUI':''}}
```

2.4 POT_ORDER Variables

This table is used to prin in a correct order the pot name.

```
POT_ORDER = [   'D1 IN', 'D2 IN',
                'A1 UP', 'A1 DOWN', 'A1 IN', 'A1 OUT',
                'A2 UP', 'A2 DOWN', 'A2 IN', 'A2 OUT',
                'P1 UP', 'P1 DOWN', 'P1 IN', 'P1 OUT',
                'P2 UP', 'P2 DOWN', 'P2 IN', 'P2 OUT']
```

2.5 RACK_ORDER Variables

Same as `POT_ORDER` variables.

```
RACK_ORDER = [ 'DIPOLES',
               'A1',
               'A2',
               'P1',
               'P2']
```

2.6 POT Variables

In this table are defined for each pot :

- Pot name;
- Name of the rack where the pot is located;
- Name of the rack monitor and the output connected to the interface indexer board;
- Name of the rack monitor, the input and channels (0 for bits 1 to 7, 1 for the bits 8 to 15) connected to the interface indexer board;
- Name of the Epics variables for the current pot position (LVDT);
- LVDT scale;
- File name;
- Epics variable for the high voltages (Level 0 and PM).

```
POT ={
    'D2 IN':{
        'RACK'        : 'DIPOLES',
        'RM OUTPUT'   : 'FPD_RM_02-05', 'CONNECTOR OUTPUT' : 'P08',
        'RM INPUT'    : 'FPD_RM_02-05', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 0,
        'EPICS LVDT' : 'FPD_RM_TND2-02-05/IN-POS', 'LVDT SCALE' : 0.0132626,
        'FILENAME'    : 'D2_IN',
        'EPIC HV LO' : 'FPD_HVC_PMTDI.STATE' ,'EPIC HC PM' : 'FPD_HVC_MADI2.STATE'},
    'D1 IN':{
        'RACK'        : 'DIPOLES',
        'RM OUTPUT'   : 'FPD_RM_02-05', 'CONNECTOR OUTPUT' : 'P02',
        'RM INPUT'    : 'FPD_RM_02-05', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 1,
        'EPICS LVDT' : 'FPD_RM_TND1-02-05/IN-POS', 'LVDT SCALE' : 0.0131926,
        'FILENAME'    : 'D1_IN',
        'EPIC HV LO' : 'FPD_HVC_PMTDI.STATE' ,'EPIC HC PM' : 'FPD_HVC_MADI1.STATE'},
    'A2 UP':{
        'RACK'        : 'A2',
        'RM OUTPUT'   : 'FPD_RM_02-03', 'CONNECTOR OUTPUT' : 'P08',
        'RM INPUT'    : 'FPD_RM_02-03', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 0,
        'EPICS LVDT' : 'FPD_RM_TNA2-02-03/UP-POS', 'LVDT SCALE' : 0.011905,
        'FILENAME'    : 'A2_UP',
        'EPIC HV LO' : 'FPD_HVC_PMTA2.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAA2U.STATE'},
    'A2 DOWN':{
        'RACK'        : 'A2',
        'RM OUTPUT'   : 'FPD_RM_02-04', 'CONNECTOR OUTPUT' : 'P08',                        # 'RM OUTPUT' : 'FPD_RM_02-03', 'CONNECTOR OUTPUT' : 'P02',
        'RM INPUT'    : 'FPD_RM_02-04', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 0,  # 'RM INPUT'  : 'FPD_RM_02-03', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 1,
        'EPICS LVDT' : 'FPD_RM_TNA2-02-03/DN-POS', 'LVDT SCALE' : 0.012755,
        'FILENAME'    : 'A2_DOWN',
        'EPIC HV LO' : 'FPD_HVC_PMTA2.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAA2D.STATE'},
    'A2 OUT':{
        'RACK'        : 'A2',
        'RM OUTPUT'   : 'FPD_RM_02-04', 'CONNECTOR OUTPUT' : 'P02',
        'RM INPUT'    : 'FPD_RM_02-04', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 1,
        'EPICS LVDT' : 'FPD_RM_TNA2-02-03/OUT-POS', 'LVDT SCALE' : 0.01252993,
        'FILENAME'    : 'A2_OUT',
        'EPIC HV LO' : 'FPD_HVC_PMTA2.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAA2O.STATE'},
    'A2 IN':{
        'RACK'        : 'A2',
        'RM OUTPUT'   : 'FPD_RM_02-03', 'CONNECTOR OUTPUT' : 'P02',                        # 'RM OUTPUT' : 'FPD_RM_02-04', 'CONNECTOR OUTPUT' : 'P08',
        'RM INPUT'    : 'FPD_RM_02-03', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 1,  # 'RM INPUT'  : 'FPD_RM_02-04', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 0,
        'EPICS LVDT' : 'FPD_RM_TNA2-02-03/IN-POS', 'LVDT SCALE' : 0.01252993,
        'FILENAME'    : 'A2_IN',
        'EPIC HV LO' : 'FPD_HVC_PMTA2.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAA2I.STATE'},
    'A1 UP':{
```

```
        'RACK'       : 'A1',
        'RM OUTPUT'  : 'FPD_RM_02-01', 'CONNECTOR OUTPUT' : 'P08',
        'RM INPUT'   : 'FPD_RM_02-01', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 0,
        'EPICS LVDT' : 'FPD_RM_TNA1-02-01/UP-POS', 'LVDT SCALE' : 0.013123,
        'FILENAME'   : 'A1_UP',
        'EPIC HV LO' : 'FPD_HVC_PMTA1.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAA1U.STATE'},
    'A1 DOWN':{
        'RACK'       : 'A1',
        'RM OUTPUT'  : 'FPD_RM_02-02', 'CONNECTOR OUTPUT' : 'P08',                       # 'RM OUTPUT' : 'FPD_RM_02-01', 'CONNECTOR OUTPUT' : 'P02',
        'RM INPUT'   : 'FPD_RM_02-02', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 0,  # 'RM INPUT'  : 'FPD_RM_02-01', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 1,
        'EPICS LVDT' : 'FPD_RM_TNA1-02-01/DN-POS', 'LVDT SCALE' : 0.012987,
        'FILENAME'   : 'A1_DOWN',
        'EPIC HV LO' : 'FPD_HVC_PMTA1.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAA1D.STATE'},
    'A1 OUT':{
        'RACK'       : 'A1',
        'RM OUTPUT'  : 'FPD_RM_02-02', 'CONNECTOR OUTPUT' : 'P02',
        'RM INPUT'   : 'FPD_RM_02-02', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 1,
        'EPICS LVDT' : 'FPD_RM_TNA1-02-01/OUT-POS', 'LVDT SCALE' : 0.01252993,
        'FILENAME'   : 'A1_OUT',
        'EPIC HV LO' : 'FPD_HVC_PMTA1.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAA1O.STATE'},
    'A1 IN':{
        'RACK'       : 'A1',
        'RM OUTPUT'  : 'FPD_RM_02-01', 'CONNECTOR OUTPUT' : 'P02',                       # 'RM OUTPUT' : 'FPD_RM_02-02', 'CONNECTOR OUTPUT' : 'P08',
        'RM INPUT'   : 'FPD_RM_02-01', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 1,  # 'RM INPUT'  : 'FPD_RM_02-02', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 0,
        'EPICS LVDT' : 'FPD_RM_TNA1-02-01/IN-POS', 'LVDT SCALE' : 0.01252993,
        'FILENAME'   : 'A1_IN',
        'EPIC HV LO' : 'FPD_HVC_PMTA1.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAA1I.STATE'},
    'P1 UP':{
        'RACK'       : 'P1',
        'RM OUTPUT'  : 'FPD_RM_02-11', 'CONNECTOR OUTPUT' : 'P08',
        'RM INPUT'   : 'FPD_RM_02-11', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 0,
        'EPICS LVDT' : 'FPD_RM_TNP1-02-11/UP-POS', 'LVDT SCALE' : 0.0133333,
        'FILENAME'   : 'P1_UP',
        'EPIC HV LO' : 'FPD_HVC_PMTP1.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAP1U.STATE'},
    'P1 DOWN':{
        'RACK'       : 'P1',
        'RM OUTPUT'  : 'FPD_RM_02-12', 'CONNECTOR OUTPUT' : 'P08',                       # 'RM OUTPUT' : 'FPD_RM_02-11', 'CONNECTOR OUTPUT' : 'P02',
        'RM INPUT'   : 'FPD_RM_02-12', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 0,  # 'RM INPUT'  : 'FPD_RM_02-11', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 1,
        'EPICS LVDT' : 'FPD_RM_TNP1-02-11/DN-POS', 'LVDT SCALE' : 0.013123,
        'FILENAME'   : 'P1_DOWN',
        'EPIC HV LO' : 'FPD_HVC_PMTP1.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAP1D.STATE'},
    'P1 OUT':{
        'RACK'       : 'P1',
        'RM OUTPUT'  : 'FPD_RM_02-12', 'CONNECTOR OUTPUT' : 'P02',
        'RM INPUT'   : 'FPD_RM_02-12', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 1,
        'EPICS LVDT' : 'FPD_RM_TNP1-02-11/OUT-POS', 'LVDT SCALE' : 0.01252993,
        'FILENAME'   : 'P1_OUT',
        'EPIC HV LO' : 'FPD_HVC_PMTP1.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAP1O.STATE'},
    'P1 IN':{
        'RACK'       : 'P1',
        'RM OUTPUT'  : 'FPD_RM_02-11', 'CONNECTOR OUTPUT' : 'P02',                       # 'RM OUTPUT' : 'FPD_RM_02-12', 'CONNECTOR OUTPUT' : 'P08',
        'RM INPUT'   : 'FPD_RM_02-11', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 1,  # 'RM INPUT'  : 'FPD_RM_02-12', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 0,
        'EPICS LVDT' : 'FPD_RM_TNP1-02-11/IN-POS', 'LVDT SCALE' : 0.01252993,
        'FILENAME'   : 'P1_IN',
        'EPIC HV LO' : 'FPD_HVC_PMTP1.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAP1I.STATE'},
    'P2 UP':{
        'RACK'       : 'P2',
        'RM OUTPUT'  : 'FPD_RM_02-13', 'CONNECTOR OUTPUT' : 'P08',
        'RM INPUT'   : 'FPD_RM_02-13', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 0,
        'EPICS LVDT' : 'FPD_RM_TNP2-02-13/UP-POS', 'LVDT SCALE' : 0.0132979,
        'FILENAME'   : 'P2_UP',
        'EPIC HV LO' : 'FPD_HVC_PMTP2.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAP2U.STATE'},
    'P2 DOWN':{
        'RACK'       : 'P2',
        'RM OUTPUT'  : 'FPD_RM_02-14', 'CONNECTOR OUTPUT' : 'P08',                       # 'RM OUTPUT' : 'FPD_RM_02-13', 'CONNECTOR OUTPUT' : 'P02',
        'RM INPUT'   : 'FPD_RM_02-14', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 0,  # 'RM INPUT'  : 'FPD_RM_02-13', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 1,
        'EPICS LVDT' : 'FPD_RM_TNP2-02-13/DN-POS', 'LVDT SCALE' : 0.0132626,
        'FILENAME'   : 'P2_DOWN',
        'EPIC HV LO' : 'FPD_HVC_PMTP2.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAP2D.STATE'},
    'P2 OUT':{
        'RACK'       : 'P2',
        'RM OUTPUT'  : 'FPD_RM_02-14', 'CONNECTOR OUTPUT' : 'P02',
        'RM INPUT'   : 'FPD_RM_02-14', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 1,
        'EPICS LVDT' : 'FPD_RM_TNP2-02-13/OUT-POS', 'LVDT SCALE' : 0.01252993,
        'FILENAME'   : 'P2_OUT',
        'EPIC HV LO' : 'FPD_HVC_PMTP2.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAP2O.STATE'},
    'P2 IN':{
        'RACK'       : 'P2',
        'RM OUTPUT'  : 'FPD_RM_02-13', 'CONNECTOR OUTPUT' : 'P02',                       # 'RM OUTPUT' : 'FPD_RM_02-14', 'CONNECTOR OUTPUT' : 'P08',
        'RM INPUT'   : 'FPD_RM_02-13', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 1,  # 'RM INPUT'  : 'FPD_RM_02-14', 'CONNECTOR INPUT'  : 'P07', 'CHANNELS INPUT' : 0,
        'EPICS LVDT' : 'FPD_RM_TNP2-02-13/IN-POS', 'LVDT SCALE' : 0.01252993,
        'FILENAME'   : 'P2_IN',
        'EPIC HV LO' : 'FPD_HVC_PMTP2.STATE' ,'EPIC HC PM' : 'FPD_HVC_MAP2I.STATE'}
        }
```

2.7 RACK Variables

This table has the name of all detectors in each castle.

```
RACK = { 'DIPOLES' : {'DETECTOR' : ('D1 IN', 'D2 IN')},
         'A1'      : {'DETECTOR' : ('A1 UP', 'A1 DOWN', 'A1 IN', 'A1 OUT')},
         'A2'      : {'DETECTOR' : ('A2 UP', 'A2 DOWN', 'A2 IN', 'A2 OUT')},
         'P1'      : {'DETECTOR' : ('P1 UP', 'P1 DOWN', 'P1 IN', 'P1 OUT')},
         'P2'      : {'DETECTOR' : ('P2 UP', 'P2 DOWN', 'P2 IN', 'P2 OUT')}
        }
```

2.8 RM Variables

List of all rack monitors in the tunnel.

```
RM =  {
        'FPD_RM_02-01':{},
        'FPD_RM_02-02':{},
        'FPD_RM_02-03':{},
        'FPD_RM_02-04':{},
        'FPD_RM_02-05':{},
        'FPD_RM_02-06':{},
        'FPD_RM_02-11':{},
        'FPD_RM_02-12':{},
        'FPD_RM_02-13':{},
        'FPD_RM_02-14':{}}
```

2.9 PB_ELECTRONIC Variables

Status and control variables for each electronic power box :

- control is defined by the name, output and channel of the rack monitor;
- status is defined by the name, input and channel of the rach monitor.

```
PB_ELECTRONIC = {
        'DIPOLES':{
                'RM INPUT' : 'FPD_RM_02-05', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5,
                'RM OUTPUT' : 'FPD_RM_02-05','CONNECTOR OUTPUT':'P03','CHANNEL OUTPUT':5},
        'A1':{
                'RM INPUT' : 'FPD_RM_02-01', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5,
                'RM OUTPUT' : 'FPD_RM_02-01','CONNECTOR OUTPUT':'P03','CHANNEL OUTPUT':5},
        'A2':{
                'RM INPUT' : 'FPD_RM_02-03', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5,
                'RM OUTPUT' : 'FPD_RM_02-03','CONNECTOR OUTPUT':'P03','CHANNEL OUTPUT':5},
        'P1':{
                'RM INPUT' : 'FPD_RM_02-11', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5,
                'RM OUTPUT' : 'FPD_RM_02-11','CONNECTOR OUTPUT':'P03','CHANNEL OUTPUT':5},
        'P2':{
                'RM INPUT' : 'FPD_RM_02-13', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5,
                'RM OUTPUT' : 'FPD_RM_02-13','CONNECTOR OUTPUT':'P03','CHANNEL OUTPUT':5}}
```

2.10 PB_DRIVER Variables

Same as `PB_ELECTERONIC`.

```
PB_DRIVER = {
        'DIPOLES':{
                'RM INPUT' : 'FPD_RM_02-06', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5,
                'RM OUTPUT' : 'FPD_RM_02-06','CONNECTOR OUTPUT':'P03','CHANNEL OUTPUT':13},
        'A1':{
                'RM INPUT' : 'FPD_RM_02-02', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5,
                'RM OUTPUT' : 'FPD_RM_02-02','CONNECTOR OUTPUT':'P03','CHANNEL OUTPUT':13},
        'A2':{
                'RM INPUT' : 'FPD_RM_02-04', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5,
                'RM OUTPUT' : 'FPD_RM_02-04','CONNECTOR OUTPUT':'P03','CHANNEL OUTPUT':13},
        'P1':{
                'RM INPUT' : 'FPD_RM_02-12', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5,
                'RM OUTPUT' : 'FPD_RM_02-12','CONNECTOR OUTPUT':'P03','CHANNEL OUTPUT':13},
        'P2':{
                'RM INPUT' : 'FPD_RM_02-14', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5,
                'RM OUTPUT' : 'FPD_RM_02-14','CONNECTOR OUTPUT':'P03','CHANNEL OUTPUT':13}}
```

2.11 SMOKE_DETECTOR Variables

Same as `PB_ELECTERONIC`.

```
SMOKE_DETECTOR = {
        'DIPOLES':{
                'RM INPUT'  : 'FPD_RM_02-02',       'CONNECTOR INPUT'  : 'P03',        'CHANNEL INPUT'  : 1,
                'RM OUTPUT' : 'FPD_RM_02-05',        'CONNECTOR OUTPUT' : 'P03',        'CHANNEL OUTPUT' : 1},
        'A1':{
                'RM INPUT'  : 'FPD_RM_02-02',        'CONNECTOR INPUT'  : 'P03',       'CHANNEL INPUT'  : 1,
                'RM OUTPUT' : 'FPD_RM_02-01',        'CONNECTOR OUTPUT' : 'P03',       'CHANNEL OUTPUT' : 1},
        'A2':{
                'RM INPUT'  : 'FPD_RM_02-04',        'CONNECTOR INPUT'  : 'P03',       'CHANNEL INPUT'  : 1,
                'RM OUTPUT' : 'FPD_RM_02-03',        'CONNECTOR OUTPUT' : 'P03',       'CHANNEL OUTPUT' : 1},
        'P1':{
                'RM INPUT'  : 'FPD_RM_02-12',        'CONNECTOR INPUT'  : 'P03',       'CHANNEL INPUT'  : 1,
                'RM OUTPUT' : 'FPD_RM_02-11',        'CONNECTOR OUTPUT' : 'P03',       'CHANNEL OUTPUT' : 1},
        'P2':{
                'RM INPUT'  : 'FPD_RM_02-14',        'CONNECTOR INPUT'  : 'P03',       'CHANNEL INPUT'  : 1,
                'RM OUTPUT' : 'FPD_RM_02-13',        'CONNECTOR OUTPUT' : 'P03',       'CHANNEL OUTPUT' : 1}}
```

2.12 EMERGENCY_LINE Variables

Same as PB_ELECTERONIC but without control.

```
EMERGENCY_LINE = {
        'DIPOLES' : {
                'RM INPUT' : 'FPD_RM_02-06', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5},
        'A1' : {
                'RM INPUT' : 'FPD_RM_02-02', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5},
        'A2' : {
                'RM INPUT' : 'FPD_RM_02-04', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5},
        'P1' : {
                'RM INPUT' : 'FPD_RM_02-12', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5},
        'P2' : {
                'RM INPUT' : 'FPD_RM_02-14', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 5}}
```

2.13 CONTROL_LINE Variables

Same as EMERGENCY_LINE.

```
CONTROL_LINE = {
        'DIPOLES' : {
                'RM INPUT' : 'FPD_RM_02-06', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 9},
        'A1' : {
                'RM INPUT' : 'FPD_RM_02-02', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 9},
        'A2' : {
                'RM INPUT' : 'FPD_RM_02-04', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 9},
        'P1' : {
                'RM INPUT' : 'FPD_RM_02-12', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 9},
        'P2' : {
                'RM INPUT' : 'FPD_RM_02-14', 'CONNECTOR INPUT' : 'P03', 'CHANNEL INPUT' : 9}}
```

2.14 FILES Variables

```
FILES = {'LOG FILE' : {
                'NAME'      : "Directory Data",
                'DIRECTORY' : "self.load('FPDConfig','Logfiles','directory','string')",
                'FILE'      : ".",
                'TYPE'      : {'Log File' : '.dat', 'Alarms' : '.alm'}},
         'HV FILE' : {
                'NAME'      : "High Voltage",
                'DIRECTORY' : "self.load('FPDConfig','HVfiles','directory','string')",
                'FILE'      : "self.load('FPDConfig','HVfiles','file','string')"}}
```

2.15 BEAM Variables

Epics variables of the beam parameters writed in the Beam Notebook

```
BEAM = {
        'ACCE72626':{'GUI':'',
                'NAME':'T:IBEAM','DESC':'Beam Intensity','UNIT':'e12','FORMAT':'%4.1f'},
           'ACCE26472':{'GUI':'',
                'NAME':'T:STORE','DESC':'Store Number','UNIT':'','FORMAT':'%02d'},
        'ACCE9202':{'GUI':'',
                'NAME':'T:ERING','DESC':'Ring Energy','FORMAT':'%4.0f','UNIT':'GeV'},
        'ACCE9203':{'GUI':'',
                'NAME':'T:IRING','DESC':'Ring Current','FORMAT':'%4.0f','UNIT':'Amp'}
        }
```

Part III

FPDVariables.py

Chapter 1

Introduction

In this file, all variables which will be updated are defined. The `VARIABLES` class is a superclass and called by the other classes.

1.1 Libraries

3 type of librairies are used :

- python : Tkinter, Dialog, os;
- CaChannel : CaChannel;
- FPD : FPDConstantVariables, FPDOptionGui and FPDDatabase.

```
from Tkinter import *
from Dialog import *
import os
from CaChannel import *
from FPDConstantVariables import *
from FPDOptionGui import *
from FPDDatabase import *
```

Chapter 2

Variables Definition

2.1 Introduction

In the first part of this file, all variables are defined. The Structure is for all of them the same :

- `TYPE` : StringVar, IntVar, DoubleVar, BooleanVar or CaChannel;
- `SET` : number, string or nothing;
- `UPDATE` : formula or nothing.

2.2 FILES Variables

Define `DIRECTORY NAME` and `FILE NAME` variables.

```
FILES_VARIABLES = { 'DIRECTORY NAME' : {'TYPE' : "StringVar()", 'SET' : "eval(FILES[key]['DIRECTORY'])", 'UPDATE' : ""},
                    'FILE NAME' :      {'TYPE' : "StringVar()", 'SET' : "eval(FILES[key]['FILE'])",      'UPDATE' : ""}}
```

2.3 POT Variables

The variables for each pot are the following :

- `MODE` : 0 = no motion, 1 = automatic mode, 2 = manual mode, 3 = random mode, 4 = switches mode;
- `INIT` : 1 = pots send at Home position;
- `INPUT` : rack monitor input (between 0x0000 and 0x007f);
- `STEP` : step in the motion 0 = no motion, 1 = direction checking, 2 = start motion, 3 = motion, 4 and 5 = correct position checking, 6 = correct position;
- `WAIT` : time for each step;
- `WANTED POSITION` and `WANTED POSITION STRING` : wanted position;
- `POSITION` : last correct position;
- `CURRENT POSITION` and `CA CURRENT POSITION` : current position (LVDTs);
- `TIME RUN` and `TIME STOP` : time when the motor is running or stopped;
- `MOTION` : 0 = stop, 1 = run, 2 = initialization;
- `SAFE POSITION` : safe position (not used);
- `MAX POSITION` : max position (used in the expert mode only);

- `MAX RATE` : max rate (not used);
- `ENABLED-DISABLED` : 0 = disabled, 1 = enabled;
- `HOME` and `STRING HOME` : 0 or 'NO' = not at home, 1 or 'YES' = home position;
- `IN` and `STRING IN` : 0 or 'NO' = not at in, 1 or 'YES' = in position;
- `HV L0` and `CA HV L0` : 3 = high voltage Level 0 On, 3 = high voltage Level 0 Off;
- `HV PM` and `CA HV` : 3 = high voltage Photomulplier On, 3 = high voltage Photomulplier Off;
- `TRIGGER` : trigger rate (not used);
- `DISABLED FAILURES` : 0 = no failure disable, +1 = cable failure disables, +2 = mechanical failure disabled, +4 = driver failure disabled (not used);
- `DRIVER FAILURE` and `STRING DRIVER FAILURE` : 0 or 'NO' = no driver failure, 1 or 'YES' = driver failure;
- `CABLE FAILURE` and `STRING CABLE FAILURE` : 0 or 'NO' = no cable failure, 1 or 'YES' = cable failure;
- `MECHANICAL FAILURE` and `STRING MECHANICAL FAILURE` : 0 or 'NO' = no mechanical failure, 1 or 'YES' = mechanical failure;
- `CA ASLO` : Scale for the Epics variable (LVDT information) = 0 (pot disabled), diff 0 (pot enabled).

The variable `key` will be the pots name ('A1 IN' for instance).

```
POT_VARIABLES = {

        'MODE' :                      {'TYPE'   : "IntVar()",     'SET'    : "0",                                         'UPDATE' : ""},
        'INIT' :                      {'TYPE'   : "BooleanVar()", 'SET'    : "1",                                         'UPDATE' : "(not POT[key]['HOME'].get()) * POT[key]['INIT'].get()"},

        'INPUT' :                     {'TYPE'   : "IntVar()",     'SET'    : "",
                                       'UPDATE' : "(RM[POT[key]['RM INPUT']]['VAL ' + POT[key]['CONNECTOR INPUT']].get() >> (8 * POT[key]['CHANNELS INPUT'])) & 0x007f"},
        'STEP' :                      {'TYPE'   : "IntVar()",     'SET'    : "0",                                         'UPDATE' : ""},
        'WAIT' :                      {'TYPE'   : "IntVar()",     'SET'    : "0",                                         'UPDATE' : "POT[key]['WAIT'].get() + 1"},

        'WANTED POSITION' :           {'TYPE'   : "IntVar()",     'SET'    : "0",                                         'UPDATE' : ""},
        'WANTED POSITION STRING' :    {'TYPE'   : "StringVar()",  'SET'    : "0",                                         'UPDATE' : ""},
        'POSITION' :                  {'TYPE'   : "IntVar()",     'SET' : "0",                                            'UPDATE' : "POT[key]['POSITION'].get() * (not POT[key]['HOME'].get())"},
        'CURRENT POSITION' :          {'TYPE'   : "DoubleVar()",  'SET'    : "0",                                         'UPDATE' : "POT[key]['CA CURRENT POSITION'].getw()"},
        'CA CURRENT POSITION' :       {'TYPE'   : "CaChannel()",  'SET'    : "POT[key]['EPICS LVDT']",                    'UPDATE' : ""},

        'TIME RUN' :                  {'TYPE' : "IntVar()",       'SET' : "0",                                            'UPDATE' : ""},
        'TIME STOP' :                 {'TYPE' : "IntVar()",       'SET' : "0",                                            'UPDATE' : ""},

        'SPEED' :                     {'TYPE' : "BooleanVar()",   'SET' : "1",                                            'UPDATE' : ""},
        'STRING SPEED' :              {'TYPE' : "StringVar()",    'SET' : "",
                                       'UPDATE' : "SPEED[POT[key]['SPEED'].get() & (POT[key]['TIME RUN'].get() > 2)]"},

        'WANTED DIRECTION' :          {'TYPE' : "BooleanVar()",   'SET' : "0",                                            'UPDATE' : ""},
        'GO DIRECTION' :              {'TYPE' : "IntVar()",       'SET' : "-1",                                           'UPDATE' : ""},
        'GO HOME' :                   {'TYPE' : "BooleanVar()",   'SET' : "0",                                            'UPDATE' : "(not POT[key]['HOME'].get()) * POT[key]['GO HOME'].get()"},
        'DIRECTION' :                 {'TYPE' : "BooleanVar()",   'SET' : "",                                             'UPDATE' : "(POT[key]['INPUT'].get() & 0x0040) == 0x0040"},
        'STRING DIRECTION' :          {'TYPE' : "StringVar()",    'SET' : "",                                             'UPDATE' : "DIRECTION[POT[key]['DIRECTION'].get()]"},

        'STATUS' :                    {'TYPE' : "BooleanVar()",   'SET' : "",                                             'UPDATE' : "(POT[key]['INPUT'].get() & 0x0008) == 0x0008"},
        'STRING STATUS' :             {'TYPE' : "StringVar()",    'SET' : "",                                             'UPDATE' : "RUN_STOP[POT[key]['STATUS'].get()]"},

        'MOTION' :                    {'TYPE' : "IntVar()",      'SET' : "0",                                             'UPDATE' : ""},

        'SAFE POSITION' :             {'TYPE' : "DoubleVar()",   'SET' : "self.load('Potdbase',key,'safe_pos','double')",      'UPDATE' : ""},
        'MAX POSITION' :              {'TYPE' : "DoubleVar()",  'SET' : "self.load('Potdbase',key,'max_pos','double')",       'UPDATE' : ""},
        'MAX RATE' :                  {'TYPE' : "IntVar()",      'SET' : "self.load('Potdbase',key,'max_rate','int')",        'UPDATE' : ""},
        'ENABLED-DISABLED' :          {'TYPE' : "IntVar()",      'SET' : "self.load('Potdbase',key,'enabled_disabled','int')", 'UPDATE' : ""},

        'HOME' :                      {'TYPE' : "BooleanVar()",  'SET' : "",                                              'UPDATE' : "(POT[key]['INPUT'].get() & 0x0002) == 0x0002"},
        'STRING HOME' :               {'TYPE' : "StringVar()",   'SET' : "",                                              'UPDATE' : "YES_NO[POT[key]['HOME'].get()]"},

        'IN' :                        {'TYPE' : "BooleanVar()",  'SET' : "",                                              'UPDATE' : "(POT[key]['INPUT'].get() & 0x0004) == 0x0004"},
        'STRING IN' :                 {'TYPE' : "StringVar()",   'SET' : "",                                              'UPDATE' : "YES_NO[POT[key]['IN'].get()]"},

        'CA HV L0' :                  {'TYPE' : "CaChannel()",   'SET' : "POT[key]['EPIC HV L0']",                        'UPDATE' : ""},
        'HV L0' :                     {'TYPE' : "IntVar()",      'SET' : "0",                                             'UPDATE' : "POT[key]['CA HV L0'].getw()"},
        'CA HV PM' :                  {'TYPE' : "CaChannel()",   'SET' : "POT[key]['EPIC HC PM']",                        'UPDATE' : ""},
        'HV PM' :                     {'TYPE' : "IntVar()",      'SET' : "0",                                             'UPDATE' : "POT[key]['CA HV PM'].getw()"},
        'TRIGGER' :                   {'TYPE' : "IntVar()",      'SET' : "0",                                             'UPDATE' : ""},
        'DISABLED FAILURES' :         {'TYPE' : "IntVar()",      'SET' : "0",                                             'UPDATE' : ""},

        'DRIVER FAILURE' :            {'TYPE' : "BooleanVar()",  'SET' : "",                                              'UPDATE' : "not ((POT[key]['INPUT'].get() & 0x0020) == 0x0020)"},
        'STRING DRIVER FAILURE' :     {'TYPE' : "StringVar()",   'SET' : "",                                              'UPDATE' : "YES_NO[POT[key]['DRIVER FAILURE'].get()]"},
        'CABLE FAILURE' :             {'TYPE' : "BooleanVar()",  'SET' : "",                                              'UPDATE' : "(POT[key]['INPUT'].get() & 0x0001) == 0x0001"},
        'STRING CABLE FAILURE' :      {'TYPE' : "StringVar()",   'SET' : "",                                              'UPDATE' : "YES_NO[POT[key]['CABLE FAILURE'].get()]"},
        'MECHANICAL FAILURE' :        {'TYPE' : "BooleanVar()",  'SET' : "",                                              'UPDATE' : "((POT[key]['INPUT'].get() & 0x0010) == 0x0010)"},
        'STRING MECHANICAL FAILURE' : {'TYPE' : "StringVar()",   'SET' : "",                                              'UPDATE' : "YES_NO[POT[key]['MECHANICAL FAILURE'].get()]"},

        'CA ASLO' :                   {'TYPE' : "CaChannel()",   'SET' : "POT[key]['EPICS LVDT'] + '.ASLO'",              'UPDATE' : ""}
        }
```

2.4 RM Variables

The variables for each rack monitor are the following :

- `STATUS` and `VAL STATUS` : connection and status of the rack monitor, 0 = status OK, diff 0 = bad status;
- `P08`, `VAL P08` and `PREVIOUS VAL P08` : connection, actual value and previous value of the digital Input/Output P08;
- `P02`, `VAL P02` and `PREVIOUS VAL P02` : connection, actual value and previous value of the digital Input/Output P02;
- `P03`, `VAL P03` and `PREVIOUS VAL P03` : connection, actual value and previous value of the digital Input/Output P03;
- `P07`, `VAL P07` and `PREVIOUS VAL P07` : connection, actual value and previous value of the digital Input/Output P04;
- `BMASK` : configuration Input/Output, +1 = P02 is an Output, +2 = P07, +4 = P02, +8 = P08.

The variable `key` is the name of the rack monitors ('FPD_RM_02-01' for inst.).

```
RM_VARIABLES = {
        'STATUS' :           {'TYPE' : "CaChannel()", 'SET' : "key + '.DSTAT'", 'UPDATE' : ""},
        'BMASK' :            {'TYPE' : "IntVar()",    'SET' : "",               'UPDATE' : ""},
        'P08' :              {'TYPE' : "CaChannel()", 'SET' : "key + '.BIN01'", 'UPDATE' : ""},
        'P02' :              {'TYPE' : "CaChannel()", 'SET' : "key + '.BIN02'", 'UPDATE' : ""},
        'P03' :              {'TYPE' : "CaChannel()", 'SET' : "key + '.BIN00'", 'UPDATE' : ""},
        'P07' :              {'TYPE' : "CaChannel()", 'SET' : "key + '.BIN03'", 'UPDATE' : ""},
        'VAL STATUS' :       {'TYPE' : "IntVar()",    'SET' : "",               'UPDATE' : ""},
        'VAL P08' :          {'TYPE' : "IntVar()",    'SET' : "0",              'UPDATE' : ""},
        'VAL P02' :          {'TYPE' : "IntVar()",    'SET' : "0",              'UPDATE' : ""},
        'VAL P03' :          {'TYPE' : "IntVar()",    'SET' : "0",              'UPDATE' : ""},
        'VAL P07' :          {'TYPE' : "IntVar()",    'SET' : "0",              'UPDATE' : ""},
        'PREVIOUS VAL P08' : {'TYPE' : "IntVar()",    'SET' : "0",              'UPDATE' : ""},
        'PREVIOUS VAL P02' : {'TYPE' : "IntVar()",    'SET' : "0",              'UPDATE' : ""},
        'PREVIOUS VAL P03' : {'TYPE' : "IntVar()",    'SET' : "0",              'UPDATE' : ""},
        'PREVIOUS VAL P07' : {'TYPE' : "IntVar()",    'SET' : "0",              'UPDATE' : ""}
        }
```

2.5 SMOKE_DETECTOR Variables

The variables for each smoke detector (one per rack) are the following :

- `STATUS` : 0 = no smoke detected, 1 = smoke detected;
- `CONTROL` : 0 = reset rack monitor interface, 1 = normal mode.

```
SMOKE_DETECTOR_VARIABLES = {
        'STATUS' :  {'TYPE' : "BooleanVar()", 'SET' : "",  'UPDATE' : ""},
        'CONTROL' : {'TYPE' : "BooleanVar()", 'SET' : "1", 'UPDATE' : ""}
        }
```

2.6 PB_ELECTRONIC Variables

The variables for each powerbox electronic (one per rack) are the following :

- `STATUS` : 0 = powerbox Off, 1 = powerbox On;
- `CONTROL` : 0 = turn Off, 1 = turn On.

Only the `CONTROL` variable is working, `STATUS` variable must be set when the control circuit will be installed.

```
PB_ELECTRONIC_VARIABLES = {
        'STATUS' :  {'TYPE' : "BooleanVar()", 'SET' : "0", 'UPDATE' : ""},
        'CONTROL' : {'TYPE' : "BooleanVar()", 'SET' : "1", 'UPDATE' : ""}
        }
```

2.7 PB_DRIVER Variables

Same as `PB_ELECTRONIC` variables. These variables are not used (control circuit not installed).

```
PB_DRIVER_VARIABLES = {
        'STATUS' :  {'TYPE' : "BooleanVar()", 'SET' : "0", 'UPDATE' : ""},
        'CONTROL' : {'TYPE' : "BooleanVar()", 'SET' : "0", 'UPDATE' : ""}
        }
```

2.8 EMERGENCY_LINE Variables

Get the status of the emergency line : 0 = On (emergency), 1 = Off (no emergency). This variable is not used (control circuit not installed).

```
EMERGENCY_LINE_VARIABLES = {
        'STATUS' : {'TYPE' : "BooleanVar()", 'SET' : "0", 'UPDATE' : ""}
        }
```

2.9 CONTROL_LINE Variables

Get the status of the control line : 0 = Off, 1 = On. This variable is not used (control circuit not installed).

```
CONTROL_LINE_VARIABLES = {
        'STATUS' : {'TYPE' : "BooleanVar()", 'SET' : "1", 'UPDATE' : ""}
        }
```

2.10 POWERPC Variables

Set the connection and the value for the powerpc variables (CPU, Memory and file descriptor).

```
POWERPC_VARIABLES = {
        'CA' :    {'TYPE' : "CaChannel()", 'SET' : "key", 'UPDATE' : ""},
        'VALUE' : {'TYPE' : "StringVar()", 'SET' : "0",   'UPDATE' : ""}
        }
```

2.11 BEAM Variables

Set the connection and the value for the beam variables.

```
BEAM_VARIABLES = {
        'CA' :    {'TYPE' : "CaChannel()", 'SET' : "key", 'UPDATE' : ""},
        'VALUE' : {'TYPE' : "StringVar()", 'SET' : "0",   'UPDATE' : ""}
        }
```

Chapter 3

Variables Class

3.1 Introduction

This class is a superclass. It creates, updates and deletes all variables.

3.2 Definition

The superclasses are :

- Frame (python class);
- Database defined in `FPDDatabase.py` file.

```
class Variables(Frame, Database):
```

3.3 Function __init__

This function creates all the variables and calls the `InitVar` function. The input variable `name` should be the name of a table (`POT`, `RM`...).

```
    def __init__(self, parent, name = ""):

        self.name = name
        self.parent = parent

        Data_Variables = eval(self.name + '_VARIABLES')
        Data = eval(self.name)

        for key in Data.keys():
            for key_v in Data_Variables.keys():

                Data[key][key_v] = eval(Data_Variables[key_v]['TYPE'])

                if Data_Variables[key_v]['TYPE'] == "CaChannel()":
                    try:
                        Data[key][key_v].searchw(eval(Data_Variables[key_v]['SET']))
                    except CaChannelException, status:
                        print ca.message(status), key, key_v
                elif Data_Variables[key_v]['SET'] is not "":
                    try:
                        Data[key][key_v].set(eval(Data_Variables[key_v]['SET']))
                    except:
                        pass

        self.InitVar(parent)

        return
```

3.4 Function InitVar

```
    def InitVar(self, parent):
        pass
```

3.5 Function __del__

Delete all variables and call the `DelVar` function.

```
def __del__(self):

        print self.name, 'Variables Destroyed'

        Data_Variables = eval(self.name + '_VARIABLES')
        Data = eval(self.name)

        for key in Data.keys():
                for key_v in Data_Variables.keys():
                        del Data[key][key_v]

        self.DelVar()

        return
```

3.6 Function DelVar

This function can be used in the child classes.

```
def DelVar(self):
        pass
```

3.7 Function update

Update all variables when `UPDATE` is not an empty string, and call the `UpdateVar` function.

```
def update(self):

        Data_Variables = eval(self.name + '_VARIABLES')
        Data = eval(self.name)

        for key in Data.keys():
                for key_v in Data_Variables.keys():

                        if (Data_Variables[key_v]['TYPE'] == "CaChannel()") and (Data_Variables[key_v]['UPDATE'] is not ""):
                                try:
                                        Data[key][key_v].getw(eval(Data_Variables[key_v]['UPDATE']))
                                except CaChannelException, status:
                                        print ca.message(status), key, key_v
                        elif (Data_Variables[key_v]['TYPE'] != "CaChannel()") and (Data_Variables[key_v]['UPDATE'] is not ""):
                                try:
                                        Data[key][key_v].set(eval(Data_Variables[key_v]['UPDATE']))
                                except:
                                        pass

        self.UpdateVar()

        return
```

3.8 Function UpdateVar

This function can be used in the child classes.

```
def UpdateVar(self):
        pass
```

Chapter 4

Files_DirectoriesVariables Class

4.1 Introduction

This class checkes if the logfiles exist or creates it.

4.2 Definition

The superclass is `Variables`.

```
class Files_DirectoriesVariables(Variables):
```

4.3 Function InitVar

```
    def InitVar(self, parent):

        for key in POT_ORDER:
            for key_file in FILES['LOG FILE']['TYPE'].keys():
                name = os.path.join(FILES['LOG FILE']['DIRECTORY NAME'].get(), POT[key]['FILENAME'] + FILES['LOG FILE']['TYPE'][key_file])
                file = open(name, 'a+')
                file.close()

        return
```

Chapter 5

PotVariables Class

5.1 Introduction

Create the `POT` variables, update and delete.

5.2 Definition

the superclass is `Variables`.

```
class PotVariables(Variables):
```

5.3 Function InitVar

Set at 2 the `MOTION` variable if the pot is not at home position.

```
def InitVar(self, parent):

        for key in POT_ORDER:
                POT[key]['MOTION'].set(2 * (not POT[key]['HOME'].get()))
```

5.4 Function UpdateVar

Update the `CA ASLO` variable : 0 (pot disabled or smoke detected) or LVDT scale.

```
def UpdateVar(self):

        for key in POT_ORDER:
                try:
                        if POT[key]['ENABLED-DISABLED'].get() and (not SMOKE_DETECTOR[POT[key]['RACK']]['STATUS'].get()):
                                POT[key]['CA ASLO'].putw(POT[key]['LVDT SCALE'])
                        else:
                                POT[key]['CA ASLO'].putw(0.000000001)
                except:
                        pass

                if (POT[key]['MOTION'].get() == 2) and POT[key]['HOME'].get():
                        POT[key]['MOTION'].set(0)
        return
```

Chapter 6

RMVariables Class

6.1 Introduction

This class creates, update and deletes the variables from the rack monitors.

6.2 Definition

The superclass is `Variables`.

```
class RMVariables(Variables):
```

6.3 Function InitVar

Set the frequency at 10 KHz (`SCAN` set at 9) and get the `BMASK` parameter.

```
def InitVar(self, parent):

        Frame.__init__(self, parent)

        for key in RM.keys():
                try:
                        status = RM[key]['STATUS'].getw()
                        if status:
                                self.InitVar(parent)
                        else:
                                pv = CaChannel()
                                pv.searchw(key + '.SCAN')
                                pv.putw(9)
                                pv.searchw(key + '.BMASK')
                                RM[key]['BMASK'].set(pv.getw())
                                bmask = RM[key]['BMASK'].get()
                                del pv

                                for key_slot, num in [['P03', 1], ['P08', 2], ['P02', 4], ['P07', 8]]:
                                        if (bmask & num) == 0:
                                                RM[key][key_slot].putw(0)
                                                RM[key]['PREVIOUS VAL ' + key_slot].set(0)
                except:
                        self.InitVar(parent)

        self.UpdateVar()
        self.UpdateVar()

        self.update()
```

6.4 Function UpdateVar

Update the rack monitors Inputs/Outputs when the status is OK.

```
def UpdateVar(self):

        status = 0
        text = ''
        val = 0
        key_slot = ''
        num = 0
        bmask = 0

        for key in RM.keys():
                try:
                        status = RM[key]['STATUS'].getw()
                        bmask = RM[key]['BMASK'].get()
```

```
            if RM[key]['VAL STATUS'].get() and status:
                text = 'Error communication : ' + key + '\n'
                LogbookText[0].insert(END, text, "a")
                LogbookText[0].yview('moveto', 1.0)

            elif not status:
                for key_slot, num in [['P03', 1], ['P08', 2], ['P02', 4], ['P07', 8]]:
                    if (bmask & num) == num:
                        val = RM[key][key_slot].getw()
                        if RM[key]['PREVIOUS VAL ' + key_slot].get() == val:
                            if RM[key]['VAL ' + key_slot].get() != val:
                                RM[key]['VAL ' + key_slot].set(val)
                        RM[key]['PREVIOUS VAL ' + key_slot].set(val)
                    elif (RM[key]['VAL ' + key_slot].get() != RM[key][key_slot].getw()) and (bmask & num) == 0:
                        RM[key][key_slot].putw(RM[key]['VAL ' + key_slot].get())
#                        RM[key]['PREVIOUS VAL ' + key_slot].set(RM[key]['VAL ' + key_slot].get())

            RM[key]['VAL STATUS'].set(status)

        except CaChannelException, status:
            print ca.message(status), key
```

6.5 Function update

Calls the `UpdateVar` function after 150 ms.

```
def update(self):

    self.UpdateVar()

    self.after(150, self.update)

    return
```

Chapter 7

SmokeVariables Class

7.1 Introduction

The class check the smoke detectors and pop up a window when the smoke is detected.

7.2 Definition

2 superclasses are used :

- `Variables`;
- `FPDLogFileFrame` defined in the `FPDOptionGui.py` file.

```
class SmokeVariables(Variables, FPDLogFileFrame):
```

7.3 Function InitVar

Create the smoke window.

```
    def InitVar(self, parent):

        self.smoke_dialog = Pmw.MessageDialog(self.parent,
                                              title = 'ALARM SMOKE DETECTOR',
                                              buttons = ('RESET',),
                                              defaultbutton = 'RESET',
                                              command = self.Reset_Smoke)
        self.smoke_dialog.withdraw()

        return
```

7.4 Function Reset_Smoke

Reset the `CONTROL` variable and deactivate the smoke window.

```
    def Reset_Smoke(self,result):

        for key in SMOKE_DETECTOR.keys():

            SMOKE_DETECTOR[key]['CONTROL'].set(0)
            self.smoke_dialog.deactivate()
        return
```

7.5 Function update

Update the variable `STATUS` and pop up the smoke window when some smoke are detected.

```
    def update(self):

        smoke_text = 'Smoked Detected in rack:\n'
        input_smoke = 0
        text = ''
        detector = ''

        for key in SMOKE_DETECTOR.keys():
```

```
        input_smoke = ((RM[SMOKE_DETECTOR[key]['RM INPUT']]['VAL ' + SMOKE_DETECTOR[key]['CONNECTOR INPUT']].get()
                                        >> (SMOKE_DETECTOR[key]['CHANNEL INPUT'] - 1)) & 0x0001)

        if input_smoke and SMOKE_DETECTOR[key]['STATUS'].get():

                text = 'Smoke detected'
                smoke_text = smoke_text + key + '\n'

                for num in range(len(RACK[key]['DETECTOR'])):
                        detector = RACK[key]['DETECTOR'][num]
                        self.Write_File(detector, FILES['LOG FILE']['DIRECTORY NAME'].get(), POT[detector]['FILENAME'], text, ('Log File','Alarms',))

        SMOKE_DETECTOR[key]['STATUS'].set(input_smoke)

        Output = ((RM[SMOKE_DETECTOR[key]['RM OUTPUT']]['VAL ' + SMOKE_DETECTOR[key]['CONNECTOR OUTPUT']].get() & 0xfffe) +
                  + SMOKE_DETECTOR[key]['CONTROL'].get() * pow(2, SMOKE_DETECTOR[key]['CHANNEL OUTPUT'] - 1))
        RM[SMOKE_DETECTOR[key]['RM OUTPUT']]['VAL ' + SMOKE_DETECTOR[key]['CONNECTOR OUTPUT']].set(Output)

        if not SMOKE_DETECTOR[key]['CONTROL'].get():
                text = 'Reset Smoke Detector ' + key + ' rack \n'
                LogbookText[0].yview('moveto', 1.0)
                LogbookText[0].insert(END, text, "n")
                SMOKE_DETECTOR[key]['CONTROL'].set(1)

if smoke_text != 'Smoked Detected in rack:\n':
        if self.smoke_dialog.active():
                self.smoke_dialog.deactivate()
        self.smoke_dialog.configure(message_text = smoke_text)
        self.smoke_dialog.show()
elif self.smoke_dialog.active():
        self.smoke_dialog.deactivate()

return
```

Chapter 8

PB_ElectronicVariables Class

8.1 Introduction

Create, update and delete the variables from `PB_ELECTRONIC` table.

8.2 Definition

The superclass is `Variables`.

```
class PB_ElectronicVariables(Variables):
```

8.3 Function update

Update the `CONTROL` and `STATUS` variables.

```
    def update(self):

        for key in PB_ELECTRONIC.keys():
            PB_ELECTRONIC[key]['CONTROL'].set(PB_ELECTRONIC[key]['CONTROL'].get() * SMOKE_DETECTOR[key]['CONTROL'].get())
            Output = ((RM[PB_ELECTRONIC[key]['RM OUTPUT']]['VAL ' + PB_ELECTRONIC[key]['CONNECTOR OUTPUT']].get() & 0xffef)
                    + PB_ELECTRONIC[key]['CONTROL'].get() * pow(2,PB_ELECTRONIC[key]['CHANNEL OUTPUT']-1))

            RM[PB_ELECTRONIC[key]['RM OUTPUT']]['VAL ' + PB_ELECTRONIC[key]['CONNECTOR OUTPUT']].set(Output)

            PB_ELECTRONIC[key]['STATUS'].set((RM[PB_ELECTRONIC[key]['RM INPUT']]['VAL ' + PB_ELECTRONIC[key]['CONNECTOR INPUT']].get()
                                        >> (PB_ELECTRONIC[key]['CHANNEL INPUT'] - 1)) & 0x0001)

        return
```

Chapter 9

PB_DriverVariables Class

9.1 Introduction

Create, update and delete the variables from `PB_DRIVER` table.

9.2 Definition

The superclass is `Variables`.

```
class PB_DriverVariables(Variables):
```

9.3 Function update

Update `STATUS` and `CONTROL` variables.

```
    def update(self):

        for key in PB_DRIVER.keys():
            PB_DRIVER[key]['STATUS'].set((RM[PB_DRIVER[key]['RM INPUT']]['VAL ' + PB_DRIVER[key]['CONNECTOR INPUT']].get()
                                    >> (PB_DRIVER[key]['CHANNEL INPUT'] - 1)) & 0x0001)

            control = 0

            for PotKey in POT_ORDER:
                if POT[PotKey]['RACK'] == key:
                    control = control | POT[PotKey]['MOTION'].get()

            PB_DRIVER[key]['CONTROL'].set(control)

        del key
        del control

        return
```

Chapter 10

Emergency_LineVariables Class

10.1 Introduction

Create, update and delete the variables from `EMERGENCY_LINE` table.

10.2 Definition

The superclass is `Variables`.

```
class Emergency_LineVariables(Variables):
```

10.3 Function update

This function will update the `STATUS` variable.

```
        def update(self):

#               for key in EMERGENCY_LINE.keys():
#                       EMERGENCY_LINE[key]['STATUS'].set((RM[EMERGENCY_LINE[key]['RM INPUT']]['VAL ' + EMERGENCY_LINE[key]['CONNECTOR INPUT']].get()
#                                                       >> (EMERGENCY_LINE[key]['CHANNEL INPUT'] - 1)) & 0x0001)

                return
```

Chapter 11

Control_LineVariables Class

11.1 Introduction

Create, update and delete the variables from `CONTROL_LINE` table.

11.2 Definition

The superclass is `Variables`.

```
class Control_LineVariables(Variables):
```

11.3 Function update

This function updates the `STATUS` variable.

```
def update(self):

        for key in CONTROL_LINE.keys():
                CONTROL_LINE[key]['STATUS'].set((RM[CONTROL_LINE[key]['RM INPUT']]['VAL ' + CONTROL_LINE[key]['CONNECTOR INPUT']].get()
                                                >> (CONTROL_LINE[key]['CHANNEL INPUT'] - 1)) & 0x0001)
        return
```

Chapter 12

Power_PCVariables Class

12.1 Introduction

Create, update and delete the variables from `POWERPC` table.

12.2 Definition

The superclass is `Variables`.

```
class Power_PCVariables(Variables):
```

12.3 Function update

Update the `VALUE` variable.

```
    def update(self):

        try:
            for key in POWERPC.keys():
                POWERPC[key]['VALUE'].set(str('%6.2f' %(POWERPC[key]['CA'].getw())))
        except CaChannelException, status:
            print ca.message(status)

        return
```

Chapter 13

Beam_Variables Class

13.1 Introduction

Create, update and delete the variables from `BEAM` table.

13.2 Definition

The superclass is `Variables`.

```
class Beam_Variables(Variables):
```

13.3 Function update

Update the `VALUE` variable.

```
        def update(self):

                try:
                        for key in BEAM.keys():
                                BEAM[key]['VALUE'].set(str('%6.2f' %(BEAM[key]['CA'].getw())))
                        del key
                except CaChannelException, status:
                        print ca.message(status)

                return
```

Part IV

FPDDatabase.py

Chapter 1

Introduction

This file defines :

- `PotData` class;
- `FPDConfigdbase` class;
- `Database` class.

1.1 Libraries

The libraries are python libraries.

```
import os
import sys
import shelve
import anydbm
```

Chapter 2

PotData class

2.1 Introduction

This class defines the variables for the local database. This database will save the characteristic of each pots.

2.2 Definition

No superclass used.

```
class PotData:
```

2.3 Function __init__

The variables are :

- `name` : pot name;
- `safe_pos` : safe position;
- `mas_pos` : maximum position;
- `max_rate` : maximum rate;
- `enabled_disabled` : pot enabled or disabled.

```
    def __init__(self, name = '', safe_pos = 0, max_pos = 0, max_rate = 0, enabled_disabled = 0):

        self.name = name
        self.safe_pos = safe_pos
        self.max_pos = max_pos
        self.max_rate = max_rate
        self.enabled_disabled = enabled_disabled
```

2.4 Function info

Return the parameters when the function is called.

```
    def info(self):

        return self.name, self.safe_pos, self.max_pos, self.max_rate, self.enabled_disabled
```

Chapter 3

FPDConfigdbase class

3.1 Introduction

This class defined the variables write in the `FPDConfig.dbm` file.

3.2 Definition

No superclass used.

```
class FPDConfigdbase:
```

3.3 Function __init__

The variables are :

- `name` printed in `POT` notebook in the Expert mode;
- `directory` : directory chosen;
- `file` : file chosen.

```
def __init__(self, name = '', directory = '', file = ''):

        self.name = name
        self.directory = directory
        self.file = file
```

3.4 Function info

Return the parameters when the function is called.

```
def info(self):

        return self.directory, self.file
```

Chapter 4

Database class

4.1 Introduction

4.2 Definition

Create the global variable `dbase`.

```
class Database:

	global dbase

	dbase = {}
```

4.3 Function __init__

Initialize the `dbase` variables. `Potdbase.dbm` and `FPDConfig.dbm` are 2 files located in the same directory that the pot motion software.

```
	def __init__(self):

		dbase['Potdbase'] = shelve.open('Potdbase.dbm')
		dbase['FPDConfig'] = shelve.open('FPDConfig.dbm')

		return
```

4.4 Function save

Save the `value` of `attribut` variable in the database `name` with the key `key`.

```
	def save(self, name = '', key = '', attribut = None, value = 0, vartype = 'int'):

		db = ''

		if name is not '':
			if (vartype is 'double') and (value == 0):
				value = 0.0
			if (vartype is 'string') and (value == 0):
				value = ''

			db = dbase[name][key]
			setattr(db, attribut, value)
			dbase[name][key] = db

		return
```

4.5 Function load

Load the `value` in `attribut` variable in the database `name` with the key `key` or create it.

```
	def load(self, name = '', key = '', attribut = '', vartype = 'int'):

		result = 0

		if name is not '':

			if dbase[name].has_key(key):
				try:
					result = getattr(dbase[name][key],attribut)
				except:
```

```
                print 'create new data :', key,', attribut :', attribut, ' in database :', name
                self.save(name, key, attribut, 0, vartype)
                result = getattr(dbase[name][key],attribut)
    else:
            print 'create new key :', key, ' in database :', name
            if name == 'Potdbase':
                db = PotData(key)
            if name == 'FPDConfig':
                db = FPDConfigdbase(key)
            dbase[name][key] = db
            result = getattr(dbase[name][key],attribut)

return result
```

Part V

FPDExpertGui.py

Chapter 1

Introduction

This file define the graphical interface for the Expert mor

1.1 Librairies

2 types of librairies are used :

- python : Tkinter, Pmw, Dialog, random, CaChannel;
- FPD : MyTkinter, MyD0Gui, MyPmw, FPDConstantVariables, FPDRun, FPDDatabase

The `State_Button` variable is defined.

```
import Tkinter
import Pmw
from Dialog import Dialog
from random import *
from CaChannel import *
from MyTkinter import *
from MyD0Gui import *
from MyPmw import *
from FPDConstantVariables import *
from FPDRun import *
from FPDDatabase import *

State_Button = (NORMAL, DISABLED, ACTIVE)
```

Chapter 2

AccPage class

2.1 Introduction

This class is used by the classes `RandomFrame`, `ManualFrame`, `DisabledFailureFrame`, `SwitchesFrame`, `PotFrame`, `HVFrame` and `ConfigFrame`.

2.2 Definition

The superclass is `Pmw.MegaWidget`.

```
class AccPage(Pmw.MegaWidget):
```

2.3 Function __init__

Init the `self.name` variable and call the `self.initPage()` function.

```
	def __init__(self, name):
		self.name = name
		self.initPage()
```

2.4 Function initPage

This function will be more defined in the other classes.

```
	def initPage(self):
		pass
```

2.5 Function raisePage

This function will be defined in the other classes. It call when the page is no longer selected.

```
	def raisePage(self):
		pass
```

2.6 Function lowerPage

This function will be defined in the other classes. It call when the page is selected.

```
	def lowerPage(self):
		pass
```

2.7 Function makePage

Create the notebook page.

```
	def makePage(self,notebook,tab):
		self.area = notebook.add(self.name, tab_text = tab)

		self.display()
```

2.8 Function display

This function will be defined in the other classes.

```
def display(self):
        pass
```

2.9 Function update

This function will be defined in the other classes.

```
def update(self):
        pass
```

Chapter 3

myNotebook class

3.1 Introduction

Create the notebookes.

3.2 Definition

The superclass is `Pmw.MegaWidget`.

```
class myNotebook(Pmw.MegaWidget):
```

3.3 Function __init__

Create the notebook by calling the `Pmw.NoteBook` class.

```
	def __init__(self, master = None):
		self.pages = {}
		self.nb = Pmw.NoteBook(master,
					createcommand = self.initPage,
					raisecommand = self.raisePage,
					lowercommand = self.lowerPage)
		self.nb.pack(fill = 'both', expand = 1)
```

3.4 Function createPage

Create the page by calling the `makePage` function.

```
	def createPage(self, page, text, type = 'AccPage'):
		self.pages[page]=eval(type + '(page)')
		self.pages[page].makePage(self.nb, text)
```

3.5 Function initPage

Init the page.

```
	def initPage(self,page):
		self.pages[page].initPage()
```

3.6 Function raisePage

Call the `raisePage` function when the page is no longer selected.

```
	def raisePage(self,page):
		self.pages[page].raisePage()
```

3.7 Function lowerPage

Reset the variables and call the `lowerPage` function when the page is selected.

```
def lowerPage(self,page):

        for key in POT_ORDER:
                POT[key]['GO HOME'].set(0)
                POT[key]['WANTED POSITION'].set(1)
                POT[key]['SPEED'].set(0)
                POT[key]['MOTION'].set(0)

        self.pages[page].lowerPage()
```

3.8 Function display

Defined later.

```
def display(self):
        pass
```

3.9 Function quit

Call the `quit` function.

```
def quit(self):

        for page in  self.nb.pagenames():
                self.pages[page].quit()
```

3.10 Function update_display

Update the display.

```
def update_display(self, name):

        self.nb.update_idletasks()
        pageInfo = self.nb._pageAttrs[name]
        page = pageInfo['page']
         pageBorder = self.nb._borderWidth + self.nb._pageMargin

        if page.winfo_reqwidth() > 500:
                 width = page.winfo_reqwidth() + 2 * pageBorder + 10
        else:
                 width = 500
        if page.winfo_reqheight() == 1:
                 height = 500
        else:
                 height = page.winfo_reqheight() + 2 * pageBorder + 35
        self.nb.configure(hull_width = width, hull_height = height)
```

3.11 Function update

Call the `update_display` function and the `update` function.

```
def update(self):
        foo = self.nb.getcurselection()
        self.update_display(foo)
        self.pages[foo].update()
```

Chapter 4

RandomFrame class

4.1 Introduction

Used for the randon motion.

4.2 Definition

```
class RandomFrame(AccPage):

        global potbuttonrandom

        potbuttonrandom = {}
```

4.3 Function initPage

```
        def initPage(self):
                pass
```

4.4 Function display

Create the display.

```
        def display(self):

                group = Pmw.Group(self.area,
                                  tag_pyclass = None)
                group.pack(fill = 'both',expand = 'yes')

                num = 0
                for key in POT_ORDER:
                        potbuttonrandom[num] = Tkinter.Checkbutton(group.interior(),
                                                                   indicatoron = 'off', selectcolor = Color[2], width = 12,
                                                                   text = key, font = Style,
                                                                   variable = POT[key]['MODE'],
                                                                   onvalue = 3, offvalue = 0,
                                                                   command = lambda  s = self,k = key : RandomFrame.start(s,k))
                        potbuttonrandom[num].grid(column = (num % 2) * 2, row = num / 2)

                        l = Tkinter.Label(group.interior(),
                                          textvariable = POT[key]['WANTED POSITION STRING'])
                        l.grid(column = (num % 2) * 2 + 1, row = num / 2)

                        num = num +1

                return
```

4.5 Function start

Call when the pot is selected.

```
        def start(self,key):

                if POT[key]['MODE'].get():
                        POT[key]['GO HOME'].set(not POT[key]['HOME'].get())
                        POT[key]['WANTED POSITION'].set(1)
                        POT[key]['POSITION'].set(0)
                        POT[key]['SPEED'].set(1)
                        POT[key]['MOTION'].set(0)
                else:
                        POT[key]['GO HOME'].set(0)
```

```
        POT[key]['MOTION'].set(0)

    return
```

4.6 Function lowerPage

```
def lowerPage(self):
    for key in POT_ORDER:
        POT[key]['MODE'].set(0)
```

4.7 Function raisePage

```
def raisePage(self):
    for key in POT_ORDER:
        POT[key]['MODE'].set(0)
```

4.8 Function quit

```
def quit(self):

    print 'Random Frame destroyed'
```

4.9 Function update

```
def update(self):

    num = 0
    position = 0
    InputData = 0

    for key in POT_ORDER:

        if POT[key]['MODE'].get() == 3:

            Inputdata = (RM[POT[key]['RM INPUT']]['VAL ' + POT[key]['CONNECTOR INPUT']].get() >> (8 * POT[key]['CHANNELS INPUT'])) & 0x007f

            if POT[key]['HOME'].get():
                POT[key]['GO DIRECTION'].set(1)                                 # 1 to go In

            if POT[key]['IN'].get():
                POT[key]['GO DIRECTION'].set(-1)                        # -1 to go Home

            if ((POT[key]['TIME STOP'].get() > 60) & (POT[key]['WANTED POSITION'].get() != 9999)):
                position = (POT[key]['GO DIRECTION'].get() * randint(50,1000)) + POT[key]['WANTED POSITION'].get()
                POT[key]['WANTED POSITION'].set(position)
                POT[key]['MOTION'].set(1)
                POT[key]['SPEED'].set(1)
                POT[key]['TIME STOP'].set(0)

                if POT[key]['IN'].get():
                    POT[key]['WANTED POSITION'].set(4000)

            if POT[key]['WANTED POSITION'].get() <= 0:
                POT[key]['WANTED POSITION'].set(0)

        else:

            POT[key]['MOTION'].set(0)
            POT[key]['GO HOME'].set(0)

        status = PB_ELECTRONIC[POT[key]['RACK']]['CONTROL'].get() & POT[key]['ENABLED-DISABLED'].get()
        potbuttonrandom[num].config(state = State_Button[not status])

        num = num + 1

    return
```

Chapter 5

ManualFrame class

5.1 Introduction

5.2 Definition

```
class ManualFrame(AccPage):

        global speed_item
        global potbuttonmanual

        speed_item = ['Low','High']
        potbuttonmanual = {}
```

5.3 Function initPage

```
        def initPage(self):

                self.var_speed = Tkinter.StringVar()
                self.var_speed.set('Low')

                self.go_home = Tkinter.IntVar()
                self.go_home.set(0)

                self.motion = Tkinter.IntVar()
                self.motion.set(1)

                self.pos = Tkinter.DoubleVar()
                self.pos.set(0)
```

5.4 Function display

```
        def display(self):

                group = Pmw.Group(self.area, tag_text = 'Commands')
                group.pack(side = 'top', fill = 'both',expand = 'yes')

                self.positionfield = Pmw.EntryField(group.interior(),
                                                        labelpos = 'w',
                                                                label_text='Position :',
                                                                entry_width = 5,
                                                                validate = {'validator' : 'real','min' : 0,'max' : 45})
                self.positionfield.grid(row = 0, column = 0)

                l = Tkinter.Label(group.interior(),
                                  text = 'mm', font = Style)
                l.grid(row = 0, column = 1)

                self.speed = Pmw.OptionMenu(group.interior(),
                                            labelpos = 'w',
                                            label_text = 'Speed :',
                                            initialitem = 'Low',
                                            items = speed_item)
                self.speed.grid(row = 0, column = 1)

                b = Tkinter.Button(group.interior(),
                                   text = 'RUN', font = Style,
                                   command = self.run,
                                   width = 10)
                b.grid(row = 1, column = 0)

                b = Tkinter.Button(group.interior(),
                                   text = 'HOME', font = Style,
                                   command = self.home,
                                   width = 10)
                b.grid(row = 1, column = 1)

                b = Tkinter.Button(group.interior(),
                                   text='STOP', font = Style,
                                   command = self.stop,
                                   width = 10)
```

```
        b.grid(row = 1, column = 2)

        group = Pmw.Group(self.area,
                    tag_text = 'Pots')
        group.pack(fill = 'both',expand=1)

        num = 0

        for key in POT_ORDER:

                potbuttonmanual[num] = Pmw.OptionMenu(group.interior(),
                                                labelpos = 'w',
                                                label_text = key,
                                                label_width = 10,
                                                items = ('Lock', 'Unlock'),
                                                menubutton_width = 10)

                potbuttonmanual[num].grid(column = (num % 2) * 2, row = num / 2)

                num = num + 1

        return
```

5.5 Function run

```
def run(self):

        self.motion.set(1)
        self.go_home.set(0)
        self.activate_dialog('MOTION \n' + 'Speed :' + self.speed.getcurselection() + '\n')
```

5.6 Function stop

```
def stop(self):

        self.motion.set(0)
        self.go_home.set(0)

        num = 0
        for key in POT_ORDER:
                if (potbuttonmanual[num].getcurselection() == 'Motion') and (POT[key]['STEP'].get()):
                        POT[key]['STEP'].set(0)
                        POT[key]['WAIT'].set(0)
                        POT[key]['MOTION'].set(0)
                        POT[key]['GO HOME'].set(0)
                num = num + 1
```

5.7 Function home

```
def home(self):

        self.go_home.set(1)
        self.motion.set(0)
        self.positionfield.setentry('0')
        self.activate_dialog('Home Position ' + '\n' + 'Speed : ' + self.speed.getcurselection() + '\n')
```

5.8 Function activate_dialog

```
def activate_dialog(self,text):

        if not self.positionfield.valid():
                return

        start_text = text

        self.dialog = Pmw.Dialog(self.area,
                            buttons = ('OK','Cancel'),
                            defaultbutton = 'OK',
                            title = 'Run dialog',
                            command = self.execute)
        self.dialog.withdraw()

        num = 0
        for key in POT_ORDER:

                pos = (float(self.positionfield.get()) * (float(self.positionfield.get()) <= POT[key]['MAX POSITION'].get())
                        + POT[key]['MAX POSITION'].get() * (float(self.positionfield.get()) > POT[key]['MAX POSITION'].get()))

                if ((potbuttonmanual[num].getcurselection() == 'Unlock') and
                    ((POT[key]['POSITION'].get() != int(pos * 120)) or self.go_home.get())):

                        if self.motion.get():
                                text = text + key + (' : %6.2f mm \n' %(pos))
                        else:
                                text = text + key + '\n'
                num = num + 1
        w = Tkinter.Label(self.dialog.interior(), text = text)
        w.pack()

        if text != start_text:
                self.dialog.activate()
```

5.9 Function execute

```
def execute(self,result):
        self.dialog.deactivate()

        if result is 'Cancel':
                del result
                return
        pos = 0
        num = 0
        for key in POT_ORDER:

                if ((potbuttonmanual[num].getcurselection() == 'Unlock') and
                    (POT[key]['POSITION'].get() != self.positionfield.get())):
                        pos = (float(self.positionfield.get()) * (float(self.positionfield.get()) <= POT[key]['MAX POSITION'].get())
                                + POT[key]['MAX POSITION'].get() * (float(self.positionfield.get()) > POT[key]['MAX POSITION'].get()))

                        POT[key]['WANTED POSITION'].set(int(float(pos) * 120.) * (not self.go_home.get()))
                        POT[key]['SPEED'].set(self.speed.index(self.speed.getcurselection()))
                        POT[key]['MOTION'].set(self.motion.get() * (not self.go_home.get()))
                        POT[key]['GO HOME'].set((not POT[key]['HOME'].get()) * self.go_home.get())

                        if self.go_home.get():
                                POT[key]['POSITION'].set(0)

                num = num + 1
```

5.10 Function lowerPage

```
def lowerPage(self):
        for key in POT_ORDER:
                POT[key]['MODE'].set(0)
```

5.11 Function raisePage

```
def raisePage(self):
        for key in POT_ORDER:
                POT[key]['MODE'].set(0)
```

5.12 Function quit

```
def quit(self):

        del self.var_speed
        del self.go_home
        del self.motion

        print 'Manual Frame destroyed'
```

5.13 Function update

```
def update(self):

        num=0
        for key in POT_ORDER:

                if PB_ELECTRONIC[POT[key]['RACK']]['CONTROL'].get() & POT[key]['ENABLED-DISABLED'].get():

                        potbuttonmanual[num].configure(menubutton_state = 'normal')

                        if POT[key]['INIT'].get():
                                potbuttonmanual[num].setitems(items = ('Initialization',))
                                potbuttonmanual[num].configure(menubutton_bg = d0_yellow)
                        elif POT[key]['STATUS'].get():
                                if not(potbuttonmanual[num].getcurselection() in ('Motion','Stop',)):
                                        potbuttonmanual[num].setitems(items = ('Motion','Stop',), index = 'Motion')
                                potbuttonmanual[num].configure(menubutton_bg = Color[2 - 2 * potbuttonmanual[num].index(potbuttonmanual[num].getcurselection())])
                        elif (POT[key]['STEP'].get() == 0):
                                if not(potbuttonmanual[num].getcurselection() in ('Lock', 'Unlock',)):
                                        potbuttonmanual[num].setitems(items = ('Lock', 'Unlock',), index = 'Lock')
                                POT[key]['MODE'].set((potbuttonmanual[num].getcurselection() == 'Unlock') * 2)
                                potbuttonmanual[num].configure(menubutton_bg = Color[4 + potbuttonmanual[num].index(potbuttonmanual[num].getcurselection())])
                        else:
                                potbuttonmanual[num].setitems(items = ('Check',))
                                potbuttonmanual[num].configure(menubutton_bg = d0_orange)

                        if (potbuttonmanual[num].getcurselection() == 'Stop'):
                                POT[key]['STEP'].set(0)
                                POT[key]['WAIT'].set(0)
                                POT[key]['MOTION'].set(0)

                else:
                        potbuttonmanual[num].configure(menubutton_bg = d0_grey, menubutton_state = 'disabled')
                        potbuttonmanual[num].setitems(items = ('Lock', 'Unlock',), index = 'Lock')

                if (POT[key]['STEP'].get() == 0) and (POT[key]['MODE'].get() == 0):
                        POT[key]['MOTION'].set(0)

                num = num + 1

        return
```

Chapter 6

DisabledFailuresFrame class

6.1 Introduction

6.2 Definition

```
class DisabledFailuresFrame(AccPage):

        global mycb
        mycb={}
```

6.3 Function initPage

```
        def initPage(self):
                pass
```

6.4 Function display

```
        def display(self):

                self.scroll_area = Pmw.ScrolledFrame(self.area,
                                                hscrollmode = 'dynamic', vscrollmode = 'dynamic',
                                                horizflex = 'expand', vertflex = 'expand')
                self.scroll_area.pack(expand = 'yes', fill = 'both')

                i=0

                for key in POT_ORDER:

                        group = Pmw.Group(self.scroll_area.interior(), tag_text=key)
                        group.grid(row = i/2, column = i%2)

                        mycb[i] = MyCheckbutton(group.interior(),
                                        text = ('Driver','Cable','Mechanical'),
                                        font = Style,
                                        onvalue = (4,1,2), offvalue = (0,0,0),
                                        command = lambda s = self, k = key, num = i : DisabledFailuresFrame.Value_Failure(s,k,num))

                        mycb[i].grid(row = 0, column = 0)

                        i = i + 1

                return
```

6.5 Function Value_Failure

```
        def Value_Failure(self,key,num):
                POT[key]['DISABLED FAILURES'].set(mycb[num].variables())

                return
```

6.6 Function lowerPage

```
        def lowerPage(self):
                for key in POT_ORDER:
                        POT[key]['MODE'].set(0)

                pass
```

6.7 Function raisePage

```
def raisePage(self):
        for key in POT_ORDER:
                POT[key]['MODE'].set(0)
        return
```

6.8 Function quit

```
def quit(self):

        print 'Disabled Failures Frame destroyed'
```

6.9 Function update

```
def update(self):

        return
```

Chapter 7

SwitchesFrame

7.1 Introduction

7.2 Definition

```
class SwitchesFrame(AccPage):

        global positionbutton
        global homebutton
        global statusbutton
        global speedbutton
        global groupswitches

        positionbutton = {}
        homebutton = {}
        statusbutton = {}
        speedbutton = {}
        groupswitches = {}
```

7.3 Function initPage

```
        def initPage(self):
                pass
```

7.4 Function display

```
        def display(self):

                num = 0

                self.scroll_area = Pmw.ScrolledFrame(self.area,
                                        hscrollmode = 'dynamic', vscrollmode = 'dynamic',
                                        horizflex = 'expand', vertflex = 'expand')
                self.scroll_area.pack(expand = 'yes', fill = 'both')

                for key in POT_ORDER:

                        groupswitches[num] = Pmw.Group(self.scroll_area.interior(),
                                     tag_pyclass = Tkinter.Checkbutton,
                                     tag_indicatoron = 'off',
                                     tag_variable = POT[key]['MODE'],
                                     tag_onvalue = 4, tag_offvalue = 0,
                                     tag_text = key,
                                     tag_selectcolor = Color[2], tag_width = 15,
                                     tag_command = lambda s = self, k = key, num = num : SwitchesFrame.motion(s,k,num))
                        groupswitches[num].pack(fill = 'both', expand = 'yes')

                        homebutton[num] = Tkinter.Checkbutton(groupswitches[num].interior(),
                                             text = 'Home', font = Style,
                                             variable = POT[key]['GO HOME'],
                                             onvalue = 1,
                                             offvalue = 0)
                        homebutton[num].grid(row = num,column = 0)

                        speedbutton[num] = Tkinter.Checkbutton(groupswitches[num].interior(),
                                              text = 'Low/High', font = Style,
                                              variable = POT[key]['SPEED'],
                                              onvalue = 1,
                                              offvalue = 0)
                        speedbutton[num].grid(row = num,column = 1)

                        statusbutton[num] = Tkinter.Checkbutton(groupswitches[num].interior(),
                                               text = 'Stop/Run', font = Style,
                                               variable = POT[key]['MOTION'],
                                               onvalue = 1,
                                               offvalue = 0)
                        statusbutton[num].grid(row = num,column = 2)

                        positionbutton[num] = MyCheckbutton(groupswitches[num].interior(),
                                            text = ('4096','2048','1024','512','256','128','64','32','16','8','4','2','1'),
```

```
                                     font = Style,
                                     onvalue = (4096,2048,1024,512,256,128,64,32,16,8,4,2,1),
                                     offvalue = (0,0,0,0,0,0,0,0,0,0,0,0,0))
            positionbutton[num].grid(row = num,column = 3)

            num = num + 1

        return
```

7.5 Function motion

```
def motion(self,key,num):

         if POT[key]['MODE'].get() == 4:
                POT[key]['WANTED POSITION'].set(positionbutton[num].variables() & (not POT[key]['GO HOME'].get()))
        else:
                POT[key]['MOTION'].set(0)

        return
```

7.6 Function lowerPage

```
def lowerPage(self):
        for key in POT_ORDER:
                POT[key]['MODE'].set(0)

        return
```

7.7 Function raisePage

```
def raisePage(self):
        for key in POT_ORDER:
                POT[key]['MODE'].set(0)

        return
```

7.8 Function quit

```
def quit(self):

        print 'Switches Frame destroyed'
```

7.9 Function update

```
def update(self):

        num = 0
        for key in POT_ORDER:
                status = PB_ELECTRONIC[POT[key]['RACK']]['CONTROL'].get() & POT[key]['ENABLED-DISABLED'].get()

                homebutton[num].config(state = State_Button[(not status) | ((POT[key]['MODE'].get() == 4) & status)])
                statusbutton[num].config(state = State_Button[(not status) | (((POT[key]['MODE'].get() == 4) | POT[key]['GO HOME'].get()) & status)])
                speedbutton[num].config(state = State_Button[(not status) | ((POT[key]['MODE'].get() == 4) & status)])
                positionbutton[num].config(state = State_Button[(not status) | (((POT[key]['MODE'].get() == 4) | POT[key]['GO HOME'].get()) & status)])
                groupswitches[num].configure(tag_state = State_Button[not status])
                num = num + 1

        return
```

Chapter 8

PotFrame class

8.1 Introduction

8.2 Definition

```
class PotFrame(AccPage, Database):

        global safepositionfield
        global maxpositionfield
        global maxratefield
        global en_disfield
        global grouppot

        safepositionfield = {}
        maxpositionfield = {}
        maxratefield = {}
        en_disfield = {}
        grouppot = {}
```

8.3 Function initPage

```
        def initPage(self):
                pass
```

8.4 Function display

```
        def display(self):
                num = 0

                self.scroll_area = Pmw.ScrolledFrame(self.area,
                                                hscrollmode = 'dynamic', vscrollmode = 'dynamic',
                                                horizflex = 'expand', vertflex = 'expand')
                self.scroll_area.pack(expand = 'yes', fill = 'both')

                for key in POT_ORDER:

                        grouppot[num] = Pmw.Group(self.scroll_area.interior(),
                                                tag_text = key)
                        grouppot[num].pack(side = TOP, fill = 'both', expand = 'yes')

                        safepositionfield[num] = Pmw.EntryField(grouppot[num].interior(),
                                                        labelpos = 'w',
                                                        label_text='Safe Position :',
                                                        entry_width = 5,
                                                        value = '%d' %(POT[key]['SAFE POSITION'].get()),
                                                        validate = {'validator' : 'real','min' : 0,'max' : 10})
                        safepositionfield[num].grid(row = 0, column = 0)

                        maxpositionfield[num] = Pmw.EntryField(grouppot[num].interior(),
                                                        labelpos = 'w',
                                                        label_text='Max Position :',
                                                        entry_width = 5,
                                                        value = '%d' %(POT[key]['MAX POSITION'].get()),
                                                        validate = {'validator' : 'real','min' : 0,'max' : 45})
                        maxpositionfield[num].grid(row = 0, column = 1)

                        maxratefield[num] = Pmw.EntryField(grouppot[num].interior(),
                                                        labelpos = 'w',
                                                        label_text='Max Rate :',
                                                        entry_width = 8,
                                                        value = '%i' %(POT[key]['MAX RATE'].get()),
                                                        validate = {'validator' : 'numeric','min' : 0,'max' : 100000})
                        maxratefield[num].grid(row = 0, column = 2)

                        en_disfield[num] = Pmw.OptionMenu(grouppot[num].interior(),
                                                        labelpos = 'w',
                                                        label_text='Enabled Disabled :',
                                                        label_width = 20,
                                                        items = ('Disabled', 'Enabled'),
                                                        menubutton_width = 15)
```

```
                en_disfield[num].grid(row = 0, column = 3)

                num = num + 1

        group = Pmw.Group(self.area,
                        tag_pyclass = None)
        group.pack(side = TOP, fill = 'x', expand = 'no')

        b = Tkinter.Button(group.interior(),
                        text = 'SAVE', font = Style,
                        command = self.save_database,
                        width = 10)
        b.pack(side = RIGHT, fill = 'x', expand = 'yes'),

        b = Tkinter.Button(group.interior(),
                        text = 'RELOAD', font = Style,
                        command = self.reload,
                        width = 10)
        b.pack(side = LEFT, fill = 'x', expand = 'yes')

        return
```

8.5 Function save_database

```
def save_database(self):

        num = 0

        for key in POT_ORDER:
                en = en_disfield[num].index(en_disfield[num].getcurselection())

                self.save('Potdbase', key, 'safe_pos', safepositionfield[num].get(), 'double')
                self.save('Potdbase', key, 'max_pos', maxpositionfield[num].get(), 'double')
                self.save('Potdbase', key, 'max_rate', maxratefield[num].get(), 'int')
                self.save('Potdbase', key, 'enabled_disabled', en, 'int')

                POT[key]['SAFE POSITION'].set(safepositionfield[num].get())
                POT[key]['MAX POSITION'].set(maxpositionfield[num].get())
                POT[key]['MAX RATE'].set(maxratefield[num].get())
                POT[key]['ENABLED-DISABLED'].set(en)

                num = num + 1
```

8.6 Function reload

```
def reload(self):

        num = 0

        for key in POT_ORDER:
                POT[key]['SAFE POSITION'].set(self.load('Potdbase', key, 'safe_pos', 'double'))
                POT[key]['MAX POSITION'].set(self.load('Potdbase', key, 'max_pos', 'double'))
                POT[key]['MAX RATE'].set(self.load('Potdbase', key, 'max_rate', 'int'))
                POT[key]['ENABLED-DISABLED'].set(self.load('Potdbase', key, 'enabled_disabled', 'int'))

                safepositionfield[num].setentry(POT[key]['SAFE POSITION'].get())
                maxpositionfield[num].setentry(POT[key]['MAX POSITION'].get())
                maxratefield[num].setentry(POT[key]['MAX RATE'].get())
                en_disfield[num].invoke(POT[key]['ENABLED-DISABLED'].get())

                num = num + 1
```

8.7 Function lowerPage

```
def lowerPage(self):
        for key in POT_ORDER:
                POT[key]['MODE'].set(0)
        pass
```

8.8 Function raisePage

```
def raisePage(self):
        for key in POT_ORDER:
                POT[key]['MODE'].set(0)
        self.reload()

        pass
```

8.9 Function quit

```
def quit(self):
        print 'Pot Frame destroyed'
```

8.10 Function update

```
def update(self):

        for num in range(len(POT_ORDER)):
                en_disfield[num].configure(menubutton_bg = Color[en_disfield[num].index(en_disfield[num].getcurselection()) * 2])

        return
```

Chapter 9

HVFrame class

9.1 Introduction

9.2 Definition

```
class HVFrame(AccPage):
```

9.3 Function initPage

```
def initPage(self):
        pass
```

9.4 Function display

```
def display(self):
        return
```

9.5 Function lowerPage

```
def lowerPage(self):
        for key in POT_ORDER:
                POT[key]['MODE'].set(0)

        pass
```

9.6 Function raisePage

```
def raisePage(self):
        for key in POT_ORDER:
                POT[key]['MODE'].set(0)

        pass
```

9.7 Function quit

```
def quit(self):
        print 'High Voltage Frame destroyed'
```

9.8 Function update

```
def update(self):
        return
```

Chapter 10

ConfigFrame class

10.1 Introduction

10.2 Definition

```
class ConfigFrame(AccPage, Database):
```

10.3 Function initPage

```
    def initPage(self):

        pass
```

10.4 Function display

```
    def display(self):

        for key in FILES.keys():

            group = Pmw.Group(self.area,
                          tag_text = FILES[key]['NAME'])
            group.pack(side = TOP, fill = 'both', expand = 'yes')

            l = Tkinter.Label(group.interior(),
                         text = 'Directory :', font = Style)
            l.grid(row = 0, column = 0)

            e = Tkinter.Entry(group.interior(),
                         textvariable = FILES[key]['DIRECTORY NAME'], width = 25)
            e.bind('<Return>', self.validate)
            e.grid(row = 0, column = 1)

            if FILES[key]['FILE'] == ".":
                output = 'directory'
                text_button = 'Directory ...'
            else:

                l = Tkinter.Label(group.interior(),
                             text = 'File :', font = Style)
                l.grid(row = 0, column = 2)

                e = Tkinter.Entry(group.interior(),
                             textvariable = FILES[key]['FILE NAME'], width = 12)
                e.grid(row = 0, column = 3)

                output = 'files'
                text_button = 'File ...'

            b = Tkinter.Button(group.interior(),
                          text = text_button, font = Style,
                         command = lambda s = self, type = key, output = output : ConfigFrame.opendialog(s, type, output),
                         width = 10)
            b.grid(row = 0, column = 4)

        group = Pmw.Group(self.area,
                    tag_pyclass = None)
        group.pack(side = TOP, fill = 'x', expand = 'no')

        b = Tkinter.Button(group.interior(),
                    text = 'SAVE', font = Style,
                    command = self.save_database,
                    width = 10)
        b.pack(side = RIGHT, fill = 'x', expand = 'yes')

        b = Tkinter.Button(group.interior(),
                    text = 'RELOAD', font = Style,
                    command = self.reload,
                    width = 10)
        b.pack(side = LEFT, fill = 'x', expand = 'yes')
```

```
        return
```

10.5 Function validate

```
def validate(self, event):

        if not os.path.exists(event.widget.get()):
                print 'Bad directory'
        else:
                print 'OK'

        print event.widget['textvariable'].cget()

        return
```

10.6 Function opendialog

```
def opendialog(self, type, output):

        openfile = OpenDialog(self.area,
                              title = 'Open',
                              directory = FILES[type]['DIRECTORY NAME'].get(),
                              file = FILES[type]['FILE NAME'].get(),
                              output = output)

        openfile.activate()

        if openfile.directory() != '.':
                FILES[type]['DIRECTORY NAME'].set(openfile.directory())
        FILES[type]['FILE NAME'].set(openfile.file())
```

10.7 Function save_database

```
def save_database(self):

        for key in ['LOG FILE', 'HV FILE']:
                FILES[key]['DIRECTORY NAME'].set(os.path.normpath(FILES[key]['DIRECTORY NAME'].get()))

        self.save('FPDConfig', 'Logfiles', 'directory', FILES['LOG FILE']['DIRECTORY NAME'].get(), 'string')
        self.save('FPDConfig', 'HVfiles', 'directory', FILES['HV FILE']['DIRECTORY NAME'].get(), 'string')
        self.save('FPDConfig', 'HVfiles', 'file', FILES['HV FILE']['FILE NAME'].get(), 'string')

        self.reload()
        pass
```

10.8 Function reload

```
def reload(self):
        FILES['LOG FILE']['DIRECTORY NAME'].set(self.load('FPDConfig','Logfiles','directory','string'))
        FILES['HV FILE']['DIRECTORY NAME'].set(self.load('FPDConfig','HVfiles','directory','string'))
        FILES['HV FILE']['FILE NAME'].set(self.load('FPDConfig','HVfiles','file','string'))

        for key in ['LOG FILE', 'HV FILE']:
                FILES[key]['DIRECTORY NAME'].set(os.path.normpath(FILES[key]['DIRECTORY NAME'].get()) + '/')

        pass
```

10.9 Function lowerPage

```
def lowerPage(self):
        for key in POT_ORDER:
                POT[key]['MODE'].set(0)
        pass
```

10.10 Function raisePage

```
def raisePage(self):
        for key in POT_ORDER:
                POT[key]['MODE'].set(0)
        pass
```

10.11 Function quit

```
def quit(self):
        print 'Config Frame destroyed'
```

10.12 Function update

```
def update(self):
        return
```

Chapter 11

ExpertFrame class

11.1 Introduction

11.2 Definition

```
class ExpertFrame(DOGui):
```

11.3 Function appInit

```
        def appInit(self):

                self.title('Forward Proton Dectector - Expert Mode')
```

11.4 Function createDataArea

Random, Switches, Disabled failures and High Voltages are disabled.

```
        def createDataArea(self):

                da = self.data_area()

                self.notebook = myNotebook(da)

                self.notebook.createPage('Manual','Manual',type = 'ManualFrame')
#                self.notebook.createPage('Random','Random',type = 'RandomFrame')
#                self.notebook.createPage('Switches','Switches',type = 'SwitchesFrame')
#                self.notebook.createPage('Disabled failures','Disabled failures',type = 'DisabledFailuresFrame')
                self.notebook.createPage('Pots','Pots',type = 'PotFrame')
#                self.notebook.createPage('High Voltages','High Voltages',type = 'HVFrame')
                self.notebook.createPage('Config','Config',type = 'ConfigFrame')
```

11.5 Function quit

Destroy the display

```
        def quit(self):

                for key in POT_ORDER:
                        POT[key]['MODE'].set(0)
                self.notebook.quit()

                self.root.destroy()

                print 'Expert destroyed'

                return
```

11.6 Function update_display

Update the notebook.

```
        def update_display(self):

                self.notebook.update()
```

11.7 Function guiInit

```
def guiInit(self):
        pass
```

Part VI

FPDHelp.py

Chapter 1

Introduction

Chapter 2

HelpFPDStandard

2.1 Introduction

2.2 Definition

2.3 HelpFPDStandard class

```
class HelpFPDStandard:
```

2.4 FPD Help

```
"""
#
FPD HELP
#
Normal Mode #1
Nothing
#
Expert Mode #2
Password required
#
"""
```

2.5 Function __init__

```
def __init__(self):
  print __doc__
```

Part VII

FPDOptionGui.py

Chapter 1

Introduction

1.1 Librairies

```
from Tkinter import *
import Pmw
from Dialog import *
import sys
import os
import time
from FPDConstantVariables import *
from FPDDatabase import *
from D0GuiDefaults import *
```

1.2 Global Variables

```
InfoPot = { 'Position' :     {'text' : ['Home', 'In', 'Wanted', 'Current'],
                              'textvariable' : ['STRING HOME', 'STRING IN', 'WANTED POSITION STRING', 'CURRENT POSITION']},
            'Status' :       {'text' : ['Direction', 'Speed', 'Motion'],
                              'textvariable' : ['STRING DIRECTION', 'STRING SPEED', 'STRING STATUS']},
            Failures' :      {'text' : ['Driver', 'Cable', 'Mechanical'],
                   'textvariable' : ['STRING DRIVER FAILURE', 'STRING CABLE FAILURE', 'STRING MECHANICAL FAILURE']},
            Trigger' :       {'text' : ['Rate'],
                   'textvariable' : ['TRIGGER']},
            High Voltages' : {'text' : ['LO', 'PM'],
                   'textvariable' : ['HV LO', 'HV PM']},
            Info' :      {'text' : ['step', 'wait', 'Direction', 'Wanted Direction', 'Last Position','Init'],
                   'textvariable' : ['STEP', 'WAIT', 'DIRECTION', 'WANTED DIRECTION', 'POSITION','INIT']},
#           'RM' :      {'text' : ['Output', 'Wanted Position'],
#                             'textvariable' : ['RM OUTPUT', 'WANTED POSITION']}
          }

global InfoPot
```

Chapter 2

FPDStatusPotFrame class

2.1 Introduction

2.2 Definition

```
class FPDStatusPotFrame(Toplevel):
```

2.3 Function __init__

```
def __init__(self, parent):

        self.parent = parent

        self.top = []
        self.__CreateGui()
```

2.4 Function __CreateGui

```
def __CreateGui(self):

        for num in range(len(POT_ORDER)):

                self.top.insert(num, 'instance')
                self.top[num] = Toplevel(self.parent)
                self.top[num].title(POT_ORDER[num])

                self.top[num].protocol('WM_DELETE_WINDOW', self.error)

                self.top[num].withdraw()

        del num

        return
```

2.5 Function DrawGui

```
def DrawGui(self, num, var = 0):

        key = ''
        numInfo = 0
        group = ''

        if (var == 0) and (self.top[num].state() == 'normal'):
                self.top[num].withdraw()
                self.top[num] = Toplevel(self.parent)
                self.top[num].title(POT_ORDER[num])

                self.top[num].protocol('WM_DELETE_WINDOW', self.error)
                self.top[num].withdraw()

        else:
                for key in ['Position', 'Status', 'Failures', 'Trigger', 'High Voltages', 'Info']:

                        group = Pmw.Group(self.top[num], tag_text = key)
                        group.pack(fill = 'both', expand = 1)

                        for numInfo in range(len(InfoPot[key]['text'])):

                                Label(group.interior(), text = InfoPot[key]['text'][numInfo]).grid(row = numInfo, column = 0)
                                Label(group.interior(), textvariable = POT[POT_ORDER[num]][InfoPot[key]['textvariable'][numInfo]]).grid(row = numInfo, column = 1)

                self.top[num].deiconify()

        del num
        del var
        del key
```

```
        del numInfo
        del group
```

2.6 Function error

```
def error(self):
        print 'Error : Window can not be deleted with this button.'
```

Chapter 3

FPDLogFileFrame class

3.1 Introduction

3.2 Definition

```
class FPDLogFileFrame(Database):
```

3.3 Function __init__

```
def __init__(self,parent):

        pass
```

3.4 Function CreateGuiLogFile

```
def CreateGuiLogFile(self, parent):

        buttons = FILES['LOG FILE']['TYPE'].keys()
        buttons.sort()
        buttons.reverse()
        buttons.append('Exit')

        self.dialog_logfile = Pmw.TextDialog(parent,
                                buttons = buttons,
                                command = self.__Log_file_visu,
                                defaultbutton = 'Exit')

        self.dialog_logfile.withdraw()

        del buttons
```

3.5 Function Write_File

```
def Write_File(self, key, directory = '', filename = '', text = '', typefile = ('Log File',)):

        if (text == '') or (filename == ''):
                del text
                del filename
                del typefile

                return

        if 'Alarms' in typefile:
                colortype = 'a'
        elif 'Warning' in typefile:
                colortype = 'w'
        else:
                colortype = 'n'

        for type in typefile:
                name = os.path.join(directory, filename + FILES['LOG FILE']['TYPE'][type])
                file = open(name, 'a+')
                textfile = time.asctime(time.localtime(time.time()))+ ' ' + text + (', LVDT : %7.2f' %(POT[key]['CURRENT POSITION'].get())) + '\n'
                file.write(textfile)
                file.close()

        try:
                LogbookText[0].yview('moveto', 1.0)
                LogbookText[0].insert(END, key + ' : ' + text + '\n', colortype)
        except:
                pass

        del directory
        del name
        del type
        del typefile
```

```
        del text
        del filename
        del file

        return
```

3.6 Function Log_Files

```
def Log_Files(self, key):

        self.key_log_file = key

        try:
                self.filetype = self.filetype
        except:
                self.filetype = 'Log File'

        self.__Log_file_visu(self.filetype)

        self.dialog_logfile.deiconify()

        del key

        return
```

3.7 Function __Log_file_visu

```
def __Log_file_visu(self,result):

        if result != 'Exit':
                if not FILES['LOG FILE']['TYPE'].get(result):
                        result = 'Log File'
                self.filetype = result
                self.dialog_logfile.configure(title = self.key_log_file + ' - ' + self.filetype)
                self.dialog_logfile.clear()
                try:
                        filename = os.path.join(FILES['LOG FILE']['DIRECTORY NAME'].get(), POT[self.key_log_file]['FILENAME'] + FILES['LOG FILE']['TYPE'].get(self.filetype))
                        self.dialog_logfile.importfile(filename)
                except:
                        pass
        else:
                self.dialog_logfile.withdraw()


        del result
```

Part VIII

FPDRun.py

Chapter 1

Introduction

1.1 Librairies

```
from Tkinter import *
import Pmw
from Dialog import *
import os
import sys
import time
from CaChannel import *
from FPDConstantVariables import *
from FPDVariables import *
from FPDOptionGui import *
```

Chapter 2

Step class

2.1 Introduction

2.2 Definition

```
class Step(FPDLogFileFrame):
```

2.3 Function __init__

```
        def __init__(self, parent, min_wait = 0, max_wait = 0):

                self.min_wait = min_wait
                self.max_wait = max_wait

                return
```

2.4 Function lowerStep

```
        def lowerStep(self, key):

                if POT[key]['STEP'].get() !=20:
                        POT[key]['STEP'].set(POT[key]['STEP'].get() + 1)
                POT[key]['WAIT'].set(0)

                self.MoreLowerStep(key)

                return
```

2.5 Function MoreLowerStep

```
        def MoreLowerStep(self, key):
                pass
```

2.6 Function raiseStep

```
        def raiseStep(self, key):

                self.MoreRaiseStep(key)

                return
```

2.7 Function MoreRaiseStep

```
        def MoreRaiseStep(self, key):
                pass
```

2.8 Function alarm

```
        def alarm(self, key):

                if (self.min_wait == 0) and (self.max_wait == 0):
                        return
```

```
        if ((POT[key]['WAIT'].get() == self.min_wait) and (self.min_wait != 0) and
             (POT[key]['WANTED POSITION'].get() != 0)):
                POT[key]['WANTED POSITION'].set(POT[key]['WANTED POSITION'].get() - 1)

        if POT[key]['WAIT'].get() > self.max_wait:
                POT[key]['WANTED POSITION'].set(0)

        self.MoreAlarm(key)

        return
```

2.9 Function MoreAlarm

```
def MoreAlarm(self, key):
        pass
```

2.10 Function update

```
def update(self, key):

        self.MoreUpdate(key)

        if POT[key]['WAIT'].get() >= self.min_wait:
                self.alarm(key)

        return
```

2.11 Function MoreUpdate

```
def MoreUpdate(self, key):
        pass
```

Chapter 3

Step_00 class

3.1 Introduction

3.2 Definition

```
class Step_00(Step):
```

3.3 Function MoreAlarm

```
def MoreAlarm(self, key):

        text = ''

        if POT[key]['STATUS'].get() and (POT[key]['WAIT'].get() > 10):
                text = 'Motor in bad status (run activated)'
                Bad_Status_Text[0] = Bad_Status_Text[0] + key + 'Motor in bad status (run activated) \n'
                POT[key]['GO HOME'].set(0)
                POT[key]['MOTION'].set(0)

        if text:
                self.Write_File(key, FILES['LOG FILE']['DIRECTORY NAME'].get(), POT[key]['FILENAME'], text, ('Log File','Alarms',))

        del text

        return
```

3.4 Function MoreUpdate

```
def MoreUpdate(self, key):

        Output = RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].get() & 0x1fff
        if (Output != RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].get()):
                RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].set(Output)

        POT[key]['WAIT'].set(POT[key]['WAIT'].get() * POT[key]['STATUS'].get())

        if ((POT[key]['MOTION'].get() and
            (POT[key]['WANTED POSITION'].get() != POT[key]['POSITION'].get()))
            or POT[key]['INIT'].get() or POT[key]['GO HOME'].get()):

                self.lowerStep(key)

        del Output

        return
```

Chapter 4

Step_01 class

4.1 Introduction

4.2 Definition

```
class Step_01(Step):
```

4.3 Function MoreRaiseStep

```
	def MoreRaiseStep(self, key):

		POT[key]['WANTED POSITION STRING'].set(str('%6.2f' %(POT[key]['WANTED POSITION'].get() * 25. / 3000.)))
		POT[key]['WANTED DIRECTION'].set(POT[key]['POSITION'].get() < POT[key]['WANTED POSITION'].get())

		if POT[key]['INIT'].get():
			POT[key]['WANTED DIRECTION'].set(1)

		return
```

4.4 Function MoreAlarm

```
	def MoreAlarm(self, key):

		text = ''

		if (POT[key]['WAIT'].get() == self.min_wait) and (self.min_wait != 0):
			text = 'Bad direction choosed, Wanted Position decreased by 1/2 turns.'

		if (POT[key]['WAIT'].get() > self.max_wait):
			text = 'Bad direction choosed : home position activated.'
			POT[key]['STEP'].set(20)

		if text:
			self.Write_File(key, FILES['LOG FILE']['DIRECTORY NAME'].get(), POT[key]['FILENAME'], text, ('Log File','Alarms',))

		del text

		return
```

4.5 Function MoreUpdate

```
	def MoreUpdate(self, key):

		Output = POT[key]['WANTED POSITION'].get()
		if (Output != RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].get()):
			RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].set(Output)

		if (((Output == RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].get()) and
		    (POT[key]['DIRECTION'].get() == POT[key]['WANTED DIRECTION'].get())) or
		    (POT[key]['INIT'].get())):

			self.lowerStep(key)

		del Output

		return
```

Chapter 5

Step_02 class

5.1 Introduction

5.2 Definition

```
class Step_02(Step):
```

5.3 Function MoreRaiseStep

```
    def MoreRaiseStep(self, key):

        PB_ELECTRONIC[POT[key]['RACK']]['CONTROL'].set(1)

        return
```

5.4 Function MoreAlarm

```
    def MoreAlarm(self, key):

        text = ''

        if (POT[key]['WAIT'].get() == self.min_wait):
            text = 'no motion detected : Wanted Position decreased by 1/2 turns'
        if (POT[key]['WAIT'].get() > self.max_wait):
            text = 'no motion detected'
            POT[key]['MOTION'].set(0)
            POT[key]['GO HOME'].set(0)
            POT[key]['STEP'].set(0)

        if text:
            self.Write_File(key, FILES['LOG FILE']['DIRECTORY NAME'].get(), POT[key]['FILENAME'], text, ('Log File','Alarms',))

        del text

        return
```

5.5 Function MoreUpdate

```
    def MoreUpdate(self, key):

        text = ''

        Output = (POT[key]['GO HOME'].get() * 0x8000 +
                  POT[key]['MOTION'].get() * 0x2000 +
                  POT[key]['WANTED POSITION'].get())
        if (Output != RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].get()):
            RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].set(Output)

        if ((Output == RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].get()) and
            (POT[key]['STATUS'].get())):
            text = 'running'
            self.Write_File(key, FILES['LOG FILE']['DIRECTORY NAME'].get(), POT[key]['FILENAME'], text, ('Log File',))

            self.lowerStep(key)

        if POT[key]['INIT'].get() and POT[key]['HOME'].get():
            POT[key]['STEP'].set(20)

        del Output
        del text

        return
```

Chapter 6

Step_03 class

6.1 Introduction

6.2 Definition

```
class Step_03(Step):
```

6.3 Function MoreUpdate

```
def MoreUpdate(self, key):

        text = ''

        Output = (POT[key]['GO HOME'].get() * 0x8000 +
                  POT[key]['SPEED'].get() * 0x4000 +
                  0x2000 +
                  POT[key]['WANTED POSITION'].get())
        if (Output != RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].get()):
                RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].set(Output)

        if (not POT[key]['STATUS'].get()):

                text = 'stopped'
                self.Write_File(key, FILES['LOG FILE']['DIRECTORY NAME'].get(), POT[key]['FILENAME'], text, ('Log File',))

                self.lowerStep(key)

        del Output
        del text

        return
```

Chapter 7

Step_04 class

7.1 Introduction

7.2 Definition

```
class Step_04(Step):
```

7.3 Function MoreAlarm

```
    def MoreAlarm(self, key):

        text = ''

        if (POT[key]['WAIT'].get() == self.min_wait):
            text = 'Bad verification of the position, new verification.'
        if (POT[key]['WAIT'].get() > self.max_wait):
            text = 'Bad verification of the position : pot probably in wrong position - home position activated.'
            POT[key]['STEP'].set(20)

        if text:
            self.Write_File(key, FILES['LOG FILE']['DIRECTORY NAME'].get(), POT[key]['FILENAME'], text, ('Log File','Alarms',))

        del text

        return
```

7.4 Function MoreUpdate

```
    def MoreUpdate(self, key):

        if (POT[key]['HOME'].get() or POT[key]['IN'].get()):
            POT[key]['STEP'].set(6)
            POT[key]['WAIT'].set(0)

            return

        Output = 0x4000 + POT[key]['WANTED POSITION'].get() + 10
        if (Output != RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].get()):
            RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].set(Output)
        POT[key]['WANTED DIRECTION'].set(POT[key]['WANTED POSITION'].get() < (Output & 0x1fff))

        if ((POT[key]['WAIT'].get() > 0) and
            (POT[key]['DIRECTION'].get() == POT[key]['WANTED DIRECTION'].get())):

            self.lowerStep(key)

        del Output

        return
```

Chapter 8

Step_05 class

8.1 Introduction

8.2 Definition

```
class Step_05(Step):
```

8.3 Function MoreAlarm

```
def MoreAlarm(self, key):

        text = ''

        if (POT[key]['WAIT'].get() == self.min_wait):
                text = 'Bad verification of the position, new verification.'
        if (POT[key]['WAIT'].get() > self.max_wait):
                text = 'Bad verification of the position : pot probably in wrong position - home position activated.'
                POT[key]['STEP'].set(20)

        if text:
                self.Write_File(key, FILES['LOG FILE']['DIRECTORY NAME'].get(), POT[key]['FILENAME'], text, ('Log File','Alarms',))

        del text

        return
```

8.4 Function MoreUpdate

```
def MoreUpdate(self, key):

        Output = 0x4000 + POT[key]['WANTED POSITION'].get() - 10
        if (Output != RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].get()):
                RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].set(Output)
        POT[key]['WANTED DIRECTION'].set(POT[key]['WANTED POSITION'].get() < (Output & 0x1fff))

        if ((POT[key]['WAIT'].get() > 0) and
            (POT[key]['DIRECTION'].get() == POT[key]['WANTED DIRECTION'].get())):

                self.lowerStep(key)

        del Output

        return
```

Chapter 9

Step_06 class

9.1 Introduction

9.2 Definition

```
class Step_06(Step):
```

9.3 Function MoreUpdate

```
        def MoreUpdate(self, key):

                POT[key]['POSITION'].set(POT[key]['WANTED POSITION'].get())
#                Output = POT[key]['POSITION'].get()
                Output = 0
                if (Output != RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].get()):
                        RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].set(Output)

                POT[key]['STEP'].set(0)
                POT[key]['WAIT'].set(0)

                if POT[key]['IN'].get():
                        text = 'IN position.'
                elif POT[key]['HOME'].get():
                        text = 'HOME position.'
                else:
                        text = 'Correct position : ' + (" %7.2f " %(float(POT[key]['POSITION'].get() / 120.))) + 'mm.'

                self.Write_File(key, FILES['LOG FILE']['DIRECTORY NAME'].get(), POT[key]['FILENAME'], text, ('Log File',))

                del text
                del Output

                return
```

Chapter 10

Step_20 class

10.1 Introduction

10.2 Definition

```
class Step_20(Step):
```

10.3 Function lowerStep

```
def lowerStep(self, key):

        POT[key]['STEP'].set(1)
        POT[key]['WAIT'].set(0)

        return
```

10.4 Function update

```
def update(self, key):

        POT[key]['GO HOME'].set(not POT[key]['HOME'].get())
        POT[key]['SPEED'].set(not POT[key]['HOME'].get())
        POT[key]['WANTED POSITION'].set(0)
        POT[key]['WANTED DIRECTION'].set(0)
        POT[key]['POSITION'].set(0)
        POT[key]['MOTION'].set(0)

        POT[key]['INIT'].set(not POT[key]['HOME'].get())

        self.lowerStep(key)

        return
```

Chapter 11

FPDRunMotor class

11.1 Introduction

11.2 Definition

```
class FPDRunMotor(Pmw.MegaWidget, FPDLogFileFrame):
```

11.3 Function __init__

```
def __init__(self, parent):

        self.init_home_pot(parent)
        self.init_bad_status_dialog(parent)

        self.step = {}
        Bad_Status_Text.insert(0,'')

        self.step[0] = Step_00(parent, 10, 10)
        self.step[1] = Step_01(parent, 10, 20)
        self.step[2] = Step_02(parent, 10, 15)
        self.step[3] = Step_03(parent)
        self.step[4] = Step_04(parent, 10, 15)
        self.step[5] = Step_05(parent, 10, 15)
        self.step[6] = Step_06(parent)
        self.step[20] = Step_20(parent)

        return
```

11.4 Function init_home_pot

```
def init_home_pot(self, parent):

        text = ''

        for key in POT_ORDER:
                if (not POT[key]['HOME'].get()) and POT[key]['ENABLED-DISABLED'].get():
                        text = text + key + ' not at home position' + '\n'
                        POT[key]['INIT'].set(1)
                        POT[key]['STEP'].set(20)
                else:
                        POT[key]['INIT'].set(0)
                        POT[key]['STEP'].set(0)

        if text == '':
                text = 'Initialization OK'
        else:
                text = text + 'The pots will be moved at Home position by the software'

        InitDialog = Pmw.MessageDialog(parent,
                                title = 'Initialization',
                                defaultbutton = 0,
                                message_text = text)
        InitDialog.activate()

        del InitDialog
        del text
        del key
```

11.5 Function init_bad_status_dialog

```
def init_bad_status_dialog(self, parent):

        self.BadStatusDialog = Pmw.MessageDialog(parent,
                                title = 'Bad Status')
        self.BadStatusDialog.withdraw()
```

11.6 Function stop

```
def stop(self, key):
        Output = 0
        if (not PB_ELECTRONIC[POT[key]['RACK']]['CONTROL'].get()):

                    Output = RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].get() & 0x1fff

                RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].set(Output)

                if POT[key]['STEP'].get() != 0:
                        POT[key]['WAIT'].set(0)
                        POT[key]['STEP'].set(0)

        del Output

        return
```

11.7 Function alarms

```
def alarms(self, key):

        failures = ((POT[key]['DRIVER FAILURE'].get() and CONTROL_LINE[POT[key]['RACK']]['STATUS'].get()) |     # 0 if not failure
                    POT[key]['MECHANICAL FAILURE'].get() |
                     POT[key]['CABLE FAILURE'].get())

        direction = ((POT[key]['DIRECTION'].get() != POT[key]['WANTED DIRECTION'].get()) & POT[key]['STATUS'].get()
                        & (POT[key]['STEP'].get() == 3) & (not POT[key]['INIT'].get()))

        text = ''

        if (failures or SMOKE_DETECTOR[POT[key]['RACK']]['STATUS'].get() or direction):

                    if direction:
                        text = 'POT pass the position without stopping'
                        POT[key]['STEP'].set(20)

                    if POT[key]['DRIVER FAILURE'].get():
                        text = 'Driver failure, MOTOR STOPPED BY SOFTWARE'
                        POT[key]['STEP'].set(0)
                        POT[key]['MOTION'].set(0)

                    if POT[key]['MECHANICAL FAILURE'].get():
                        text = text + 'Mechanical failure, MOTOR STOPPED BY SOFTWARE'
                        POT[key]['STEP'].set(0)

                    if POT[key]['CABLE FAILURE'].get():
                        text = text + 'Cable failure, MOTOR STOPPED BY SOFTWARE'
                        POT[key]['STEP'].set(0)
                        POT[key]['MOTION'].set(0)

                if SMOKE_DETECTOR[POT[key]['RACK']]['STATUS'].get():
                        POT[key]['MOTION'].set(0)
                        POT[key]['GO HOME'].set(1)
                        POT[key]['WANTED POSITION'].set(0)
                        POT[key]['POSITION'].set(0)

        if EMERGENCY_LINE[POT[key]['RACK']]['STATUS'].get():

                text = 'EMERGENCY LINE ACTIVATED : Pot goes home'

                POT[key]['MOTION'].set(0)
                POT[key]['GO HOME'].set(1)
                POT[key]['WANTED POSITION'].set(0)
                POT[key]['POSITION'].set(0)

        if text != '':

                POT[key]['STEP'].set(0)
                POT[key]['WAIT'].set(0)

                self.Write_File(key, FILES['LOG FILE']['DIRECTORY NAME'].get(), POT[key]['FILENAME'], text, ('Log File','Alarms',))

        del failures
        del direction
        del text

        return
```

11.8 Function update

```
def update(self):

        num = 0
        num_step = 0
        Bad_Status_Text[0] = ''

        for key in POT_ORDER:

                try:
                        if POT[key]['ENABLED-DISABLED'].get():

                                num_step = POT[key]['STEP'].get()

                                POT[key]['TIME STOP'].set((POT[key]['TIME STOP'].get() + 1) * (not POT[key]['STATUS'].get()) *
                                                            (POT[key]['STEP'].get() == 0))                          # +1 if motor's not running
                                POT[key]['TIME RUN'].set((POT[key]['TIME RUN'].get() + 1) * (POT[key]['STATUS'].get()) *
                                                           (POT[key]['STEP'].get() in range (1,4)))                # +1 if motor's running

                                self.stop(key)
                                self.step[num_step].update(key)
                                if num_step != POT[key]['STEP'].get():
```

```
                                self.step[POT[key]['STEP'].get()].raiseStep(key)
                        self.alarms(key)
                else:
                        RM[POT[key]['RM OUTPUT']]['VAL ' + POT[key]['CONNECTOR OUTPUT']].set(0x0000)
        except:
                pass

        num = num + 1

if Bad_Status_Text[0]:
        Bad_Status_Text[0] = Bad_Status_Text[0] + 'TURN OFF THE CONTROL LINE'
        if self.BadStatusDialog.active():
                self.BadStatusDialog.deactivate()
        self.BadStatusDialog.configure(message_text = Bad_Status_Text[0])
        self.BadStatusDialog.show()
elif self.BadStatusDialog.active():
        self.BadStatusDialog.deactivate()

del num
del key
del num_step

return
```

Part IX

MyD0Gui.py

Chapter 1

Introduction

1.1 Librairies

```
from Tkinter import *
import Pmw
import D0GuiDefaults
import helpD0Dialog
import sys
```

Chapter 2

D0Gui class

2.1 Introduction

2.2 Definition

```
class D0Gui(Toplevel):
```

2.3 Function __init__

```
    def __init__(self,parent, title='No Title', help=None):

Toplevel.__init__(self,parent)

self.transient(parent)

self.appInit()
self.root = self

        self.help = help
        Pmw.aboutversion('02.04.02')
        Pmw.aboutcontact(
'For information about this application contact : \n'
'    Lionel Peyrichoux peyricli@fnal.gov \n')

        self.__createInterface()
self.guiInit()
```

2.4 Function appInit

```
    def appInit(self):
        pass
```

2.5 Function guiInit

```
    def guiInit(self):
pass
```

2.6 Function __createInterface

```
    def __createInterface(self):
        self.__createMenuBar()
self.createMenuBar()
self.__exitMenuBar()

self.__createStatusArea()

        self.__createDataArea()
self.createDataArea()

        self.__placeStatusArea()
```

2.7 Function __createMenuBar

```
    def __createMenuBar(self):
        self.__menuBar = Pmw.MenuBar(self.root,
   hull_relief=RAISED,
                                hull_borderwidth=2,
```

```
padx=5)
     mb = self.__menuBar
     mb.pack(side=TOP, fill=X)
     mb.addmenu('File', 'File menu', side='left')

     mb.addmenu('Help', 'Help menu', side='right')
     mb.addmenuitem('Help', 'command', label='About', command=self.__showAbout)
     mb.addmenuitem('Help', 'command', label='Help', command=self.__showHelp)
```

2.8 Function createMenuBar

```
    def createMenuBar(self):
mb = self.menu_bar()
```

2.9 Function menu_bar

```
    def menu_bar(self):
        return self.__menuBar
```

2.10 Function __exitMenuBar

```
    def __exitMenuBar(self):
mb = self.menu_bar()
        mb.addmenuitem('File', 'command', label='Exit', command=self.quit)
```

2.11 Function __createDataArea

```
    def __createDataArea(self):
        self.__dataFrame = Frame(self.root)
        df = self.__dataFrame
        df.pack(side=TOP, fill=BOTH, padx=0, pady=0, expand=1)
```

2.12 Function data_area

```
    def data_area(self):
        return self.__dataFrame
```

2.13 Function createDataArea

```
    def createDataArea(self):
da = self.data_area()
```

2.14 Function __createStatusArea

```
    def __createStatusArea(self):
        self.__statusText = Pmw.MessageBar(self.root,
   entry_width=30,
   entry_relief=SUNKEN,
   labelpos=W,
   label_text='Status:')
```

2.15 Function __placeStatusArea

```
    def __placeStatusArea(self):
        sa = self.__statusText
        sa.pack(side=TOP, fill=X, expand=NO, padx=10, pady=10)
```

2.16 Function statusMessage

```
    def statusMessage(self, msg, messagetype='state'):
        self.__statusText.message(messagetype, msg)
```

2.17 Function __createAboutDialog

```
    def __createAboutDialog(self):
self.__about = Pmw.AboutDialog(self.root, applicationname = 'Forward Proton Detector')
        self.__about.withdraw()
```

2.18 Function __showAbout

```
def __showAbout(self):
    if None == self.about:
        self.__createAboutDialog()
    self.__about.show()
    self.__about.focus_set()
```

2.19 Function __showHelp

```
def __showHelp(self):
        helpDODialog.HelpDialog(self, HelpFPDStandard.__doc__)
```

Part X

MyPmw.py

Chapter 1

Introduction

In this files, all class related to the PMW library are defined. Now only one is used `OpenDialog`.

1.1 Librairies

All libraries come from the python libraries.

```
from Tkinter import *
import Pmw
from Dialog import Dialog
import os
from os.path import *
import sys
```

Chapter 2

OpenDialog class

2.1 Introduction

This class `OpenDialog` is used in Option of the Expert Mode. It pops up a window where the user can choose the directory and/or file.

2.2 Definition

To call this class, you need to use this command "Your variable" = OpenDialog("Your area (self...)", title = "Your title", directory = "Your directory", file = "Your file", output = "Your Output (file or directory)") and then "Your variable".activate().

The superclass is a Pmw class.

```
class OpenDialog(Pmw.MegaWidget):
```

2.3 Function __init__

Define the options for this class and create the window.

```
    def __init__(self, parent = None, **kw):

        INITOPT = Pmw.INITOPT
        optiondefs = (
            ('title',     'Open',                   None),
            ('directory', os.path.expanduser('~'), None),
            ('file',      '.',                      None),
            ('output',    'files',                  None),
            ('filter',    '*',                      None))

        self.defineoptions(kw, optiondefs)

        Pmw.MegaWidget.__init__(self, parent)
        self.kw = self

        self.openarea = parent

        self.output = self.kw['output']

        self.directory_input = self.kw['directory']
        self.directory_current = self.kw['directory']
        self.directory_select = self.kw['directory']

        self.file_input = self.kw['file']

        self.filter = self.kw['filter']

        self.opendialog = Pmw.Dialog(self.openarea,
                            title = self.kw['title'],
                            buttons = ('Open','Filter','Cancel'),
                            defaultbutton = 'Open',
                            command = self.openbutton)
        self.opendialog.withdraw()

        l = Label(self.opendialog.interior(), text = 'Directories :', anchor = 'w')
        l.grid(row = 0, column = 0, sticky = 'w')

        self.directoriesListBox = Pmw.ScrolledListBox(self.opendialog.interior(),
                                    listbox_height = 10,
                                    hscrollmode = 'static', vscrollmode = 'static',
                                    selectioncommand = self.selectdirectory,
                                    dblclickcommand = self.changedirectory)
        self.directoriesListBox.grid(row = 1, column = 0)

        if self.output == 'files':
```

```
                l = Label(self.opendialog.interior(), text = 'Files :', anchor = 'w')
                l.grid(row = 0, column = 1, sticky = 'w')

                self.filesListBox = Pmw.ScrolledListBox(self.opendialog.interior(),
                                        hscrollmode = 'static', vscrollmode = 'static',
                                        selectioncommand = self.selectfile,
                                        dblclickcommand = self.openfile)
                self.filesListBox.grid(row = 1, column = 1)
        else:
                self.file_input = '.'

        return
```

2.4 Function changedirectory

Changes the current directory when a new directory is chosen and call the `reseach` function.

```
def changedirectory(self):

        self.directory_current = self.directory_select
        self.research()
```

2.5 Function research

Write in the `listdir` and `listfile` tables, the directories and files contained in the current directory.

```
def research(self):
        self.directory_current = os.path.join(os.path.normpath(self.directory_current), '')
        try:
                list = os.listdir(self.directory_current)
        except:
                list = os.listdir(os.getcwd())
                self.directory_current = os.getcwd()

        listdir = ['.','..']
        listfile = []

        numfile = 0
        numdir = 2

        for i in range(len(list)):
                if isfile(self.directory_current + list[i]):
                        listfile.insert(numfile, list[i])
                        numfile = numfile + 1
                else:
                        listdir.insert(numdir, list[i])
                        numdir = numdir + 1

        listfile.sort()
        listdir.sort()

        if self.output == 'files':
                self.filesListBox.setlist(listfile)

        self.directoriesListBox.setlist(listdir)
```

2.6 Function selectdirectory

Select the directory chosen by the user.

```
def selectdirectory(self):
        result = self.directoriesListBox.getcurselection()
        if result != ():
                self.directory_select = os.path.join(os.path.normpath(self.directory_current), result[0], '')
```

2.7 Function selectfile

Select the file chosen by the user.

```
def selectfile(self):
        self.file_select = self.filesListBox.getcurselection()[0]
```

2.8 Function openfile

Exit from this class when the user doubleclick on a file.

```
def openfile(self):
        self.file_select = self.filesListBox.getcurselection()[0]
        self.opendialog.invoke(index = 'Open')
```

2.9 Function activate

Pop up the window.

```
def activate(self):
        self.research()
        self.opendialog.activate()
```

2.10 Function openbutton

Action when one of buttons (Open, Filter or Cancel) is activated.

```
        def openbutton(self, result):
if result != 'Filter':
self.opendialog.deactivate()

                if result == 'Open':
                        if self.output == 'files':
                                self.file_open = self.file_select
                        else:
                                self.file_open = self.file_input
                        self.directory_open = self.directory_select
                elif result == 'Filter':
self.changedirectory()
else:
                        self.directory_open = self.directory_input
                        self.file_open = self.file_input
```

2.11 Function directory

Send the selected directory.

```
def directory(self):
        self.directory_open = os.path.normpath(self.directory_open)
        return self.directory_open
```

2.12 Function file

Send the selected file.

```
def file(self):
        return self.file_open
```

Part XI

MyTkinter.py

Chapter 1

Introduction

1.1 Librairies

```
import Tkinter
import Pmw
from MyDOGui import *
```

Chapter 2

MyCheckbutton class

2.1 Introduction

2.2 Definition

```
class MyCheckbutton(Pmw.MegaWidget):

        status = ['Disabled','Normal']
```

2.3 Function __init__

```
        def  __init__(self, parent = None, **kw):

                # Define the megawidget options.
                INITOPT = Pmw.INITOPT
                optiondefs = (
                        ('command',       None,      None),
                        ('offvalue',         0,   INITOPT),
                        ('onvalue',          1,   INITOPT),
                        ('state',       NORMAL,      None),
                        ('font',          None,   INITOPT),
                        ('text',          None,   INITOPT),
                        ('indicatoron',     ON,   INITOPT),
                        ('anchor',      CENTER,   INITOPT)
                )
                self.defineoptions(kw, optiondefs)

                # Initialise the base class (after defining the options).
                Pmw.MegaWidget.__init__(self, parent)

                self.kw = self

                self.cb={}
                self.var={}

                for i in range(len(self.kw['text'])):

                        self.var[i] = IntVar()
                        self.var[i].set(self.kw['offvalue'][i])

                        self.cb[i] = Tkinter.Checkbutton(parent, indicatoron = self.kw['indicatoron'],
                                                text = self.kw['text'][i], font = self.kw['font'], anchor = self.kw['anchor'],
                                                variable = self.var[i],
                                                onvalue = self.kw['onvalue'][i],
                                                offvalue = self.kw['offvalue'][i],
                                                state = self.kw['state'],
                                                command = self.kw['command'])

                        if self['command'] != None:
                                self.cb[i].config(command=self.kw['command'])

                self.initialiseoptions(MyCheckbutton)


                return
```

2.4 Function pack

```
        def pack(self):

                for i in range(len(self.kw['text'])):

                        self.cb[i].pack()
```

2.5 Function grid

```
def grid(self,**grids):

        for i in range(len(self.kw['text'])):

                self.cb[i].grid(row=grids['row'],column=grids['column'] +i)
```

2.6 Function variables

```
def variables(self):

        MyCheckbutton_var = 0
        for i in range(len(self.kw['text'])):

                MyCheckbutton_var = self.var[i].get() + MyCheckbutton_var

        return MyCheckbutton_var
```

2.7 Function config

```
def config(self,state = NORMAL):
        for i in range(len(self.kw['text'])):
                self.cb[i].config(state=state)
```

Part XII

Local Databases

Chapter 1

FPDConfig.dbm

The data in this files are :

- directory for the logfile;
- directore and filename for the high voltage software.

Chapter 2

Potdbase.dbm

The pot parameters, saved in this file, are the following :

- maximum position (in mm);
- safe position (in mm);
- maximum rate (in Hz);
- status (enabled or disabled).

Table of contents

Contents

INTRODUCTION 1

I FPDGui.py 3

1 Introduction 5
1.1 Comment 5
1.2 Libraries 5

2 FPDVariable class 6
2.1 Introduction 6
2.2 Definition 6
2.3 Comments 6
2.4 Function __init__ 6
2.5 Function __del__ 7
2.6 Function update 7

3 FPDMenuFrame class 8
3.1 Introduction 8
3.2 Definition 8
3.3 Function __init__ 8
3.4 Function __CreateMenuBar 8
3.5 Function FileMenu 9
3.6 Function AlarmMenu 9
3.7 Function CommandMenu 9
3.8 Function Run_Start 10
3.9 Function Run_Stop 10
3.10 Function HV 10
3.11 Function ExpertMenu 10
3.12 Function InitMenu 10
3.13 Function Init 10
3.14 Function HelpMenu 11
3.15 Function AboutDialogFrame 11
3.16 Function HelpFrame 11
3.17 Function AmplifierCrate 11
3.18 Function AllAmplifierCrate 11
3.19 Function SmokeDetector 11
3.20 Function Expert 12
3.21 Function Expert_Activate 12
3.22 Function update 12

4 FPDCreatePositionPotMenu class 13
4.1 Introduction 13
4.2 Definition 13
4.3 Function __init__ 13
4.4 Function __CreateGui 13

4.5 Function Status_Pot . . . 14
4.6 Function update . . . 14

5 FPDCreateLogbookPot class **15**
5.1 Introduction . . . 15
5.2 Definition . . . 15
5.3 Function __init__ . . . 15
5.4 Function __CreateGui . . . 15
5.5 Function update . . . 16

6 FPDCreateHighVoltage class **17**
6.1 Introduction . . . 17
6.2 Definition . . . 17
6.3 Function __init__ . . . 17
6.4 Function __CreateGui . . . 17
6.5 Function _HV_L0Balloon . . . 17
6.6 Function _HV_MAPMTBalloon . . . 18
6.7 Function update . . . 18

7 FPDCreateAmplifierLVDT class **19**
7.1 Introduction . . . 19
7.2 Definition . . . 19
7.3 Function __init__ . . . 19
7.4 Function __CreateGui . . . 19
7.5 Function update . . . 19

8 FPDCreatePower class **21**
8.1 Introduction . . . 21
8.2 Definition . . . 21
8.3 Function __init__ . . . 21
8.4 Function __CreateGui . . . 21
8.5 Function update . . . 21

9 FPDCreateInfo class **23**
9.1 Introduction . . . 23
9.2 Definition . . . 23
9.3 Function __init__ . . . 23
9.4 Function __CreateNotebook . . . 23
9.5 Function update . . . 24

10 FPDCreateStatusBar class **25**
10.1 Introduction . . . 25
10.2 Definition . . . 25
10.3 Function __init__ . . . 25
10.4 Function __CreateStatusBar . . . 25
10.5 Function update . . . 25

11 FPDGui class **26**
11.1 Introduction . . . 26
11.2 Definition . . . 26
11.3 Comments . . . 26
11.4 Function __init__ . . . 26
11.5 Function destroy . . . 27
11.6 Function update . . . 27

12 Function destroy **28**
12.1 Introduction . . . 28
12.2 Function destroy . . . 28

13 Function quit **29**
13.1 Introduction . . . 29
13.2 Function quit . . . 29

14 FPDSoft class **30**
14.1 Introduction . . . 30
14.2 Definition . . . 30
14.3 Function __init__ . . . 30
14.4 Function update . . . 30

15 Main Program **31**
15.1 Introduction . . . 31
15.2 Main Program . . . 31

II FPDConstantVariables **33**

1 Introduction **35**
1.1 Libraries . . . 35

2 Constant variables definition **36**
2.1 Global definition . . . 36
2.2 Other variables . . . 36
2.3 POWERPC Variables . . . 36
2.4 POT_ORDER Variables . . . 37
2.5 RACK_ORDER Variables . . . 37
2.6 POT Variables . . . 37
2.7 RACK Variables . . . 38
2.8 RM Variables . . . 38
2.9 PB_ELECTRONIC Variables . . . 39
2.10 PB_DRIVER Variables . . . 39
2.11 SMOKE_DETECTOR Variables . . . 39
2.12 EMERGENCY_LINE Variables . . . 40
2.13 CONTROL_LINE Variables . . . 40
2.14 FILES Variables . . . 40
2.15 BEAM Variables . . . 40

III FPDVariables.py **41**

1 Introduction **43**
1.1 Libraries . . . 43

2 Variables Definition **44**
2.1 Introduction . . . 44
2.2 FILES Variables . . . 44
2.3 POT Variables . . . 44
2.4 RM Variables . . . 46
2.5 SMOKE_DETECTOR Variables . . . 46
2.6 PB_ELECTRONIC Variables . . . 46
2.7 PB_DRIVER Variables . . . 47
2.8 EMERGENCY_LINE Variables . . . 47
2.9 CONTROL_LINE Variables . . . 47

2.10 POWERPC Variables 47
2.11 BEAM Variables 47

3 Variables Class **48**
3.1 Introduction 48
3.2 Definition 48
3.3 Function __init__ 48
3.4 Function InitVar 48
3.5 Function __del__ 49
3.6 Function DelVar 49
3.7 Function update 49
3.8 Function UpdateVar 49

4 Files_DirectoriesVariables Class **50**
4.1 Introduction 50
4.2 Definition 50
4.3 Function InitVar 50

5 PotVariables Class **51**
5.1 Introduction 51
5.2 Definition 51
5.3 Function InitVar 51
5.4 Function UpdateVar 51

6 RMVariables Class **52**
6.1 Introduction 52
6.2 Definition 52
6.3 Function InitVar 52
6.4 Function UpdateVar 52
6.5 Function update 53

7 SmokeVariables Class **54**
7.1 Introduction 54
7.2 Definition 54
7.3 Function InitVar 54
7.4 Function Reset_Smoke 54
7.5 Function update 54

8 PB_ElectronicVariables Class **56**
8.1 Introduction 56
8.2 Definition 56
8.3 Function update 56

9 PB_DriverVariables Class **57**
9.1 Introduction 57
9.2 Definition 57
9.3 Function update 57

10 Emergency_LineVariables Class **58**
10.1 Introduction 58
10.2 Definition 58
10.3 Function update 58

11 Control_LineVariables Class **59**
11.1 Introduction 59
11.2 Definition 59
11.3 Function update 59

12 Power_PCVariables Class **60**
12.1 Introduction 60
12.2 Definition 60
12.3 Function update 60

13 Beam_Variables Class **61**
13.1 Introduction 61
13.2 Definition 61
13.3 Function update 61

IV FPDDatabase.py 63

1 Introduction **65**
1.1 Libraries 65

2 PotData class **66**
2.1 Introduction 66
2.2 Definition 66
2.3 Function __init__ 66
2.4 Function info 66

3 FPDConfigdbase class **67**
3.1 Introduction 67
3.2 Definition 67
3.3 Function __init__ 67
3.4 Function info 67

4 Database class **68**
4.1 Introduction 68
4.2 Definition 68
4.3 Function __init__ 68
4.4 Function save 68
4.5 Function load 68

V FPDExpertGui.py 71

1 Introduction **73**
1.1 Librairies 73

2 AccPage class **74**
2.1 Introduction 74
2.2 Definition 74
2.3 Function __init__ 74
2.4 Function initPage 74
2.5 Function raisePage 74
2.6 Function lowerPage 74
2.7 Function makePage 74
2.8 Function display 75
2.9 Function update 75

3 myNotebook class **76**
3.1 Introduction 76
3.2 Definition 76
3.3 Function __init__ 76
3.4 Function createPage 76

3.5 Function initPage . . . 76
3.6 Function raisePage . . . 76
3.7 Function lowerPage . . . 77
3.8 Function display . . . 77
3.9 Function quit . . . 77
3.10 Function update_display . . . 77
3.11 Function update . . . 77

4 RandomFrame class **78**
4.1 Introduction . . . 78
4.2 Definition . . . 78
4.3 Function initPage . . . 78
4.4 Function display . . . 78
4.5 Function start . . . 78
4.6 Function lowerPage . . . 79
4.7 Function raisePage . . . 79
4.8 Function quit . . . 79
4.9 Function update . . . 79

5 ManualFrame class **80**
5.1 Introduction . . . 80
5.2 Definition . . . 80
5.3 Function initPage . . . 80
5.4 Function display . . . 80
5.5 Function run . . . 81
5.6 Function stop . . . 81
5.7 Function home . . . 81
5.8 Function activate_dialog . . . 81
5.9 Function execute . . . 82
5.10 Function lowerPage . . . 82
5.11 Function raisePage . . . 82
5.12 Function quit . . . 82
5.13 Function update . . . 82

6 DisabledFailuresFrame class **83**
6.1 Introduction . . . 83
6.2 Definition . . . 83
6.3 Function initPage . . . 83
6.4 Function display . . . 83
6.5 Function Value_Failure . . . 83
6.6 Function lowerPage . . . 83
6.7 Function raisePage . . . 84
6.8 Function quit . . . 84
6.9 Function update . . . 84

7 SwitchesFrame **85**
7.1 Introduction . . . 85
7.2 Definition . . . 85
7.3 Function initPage . . . 85
7.4 Function display . . . 85
7.5 Function motion . . . 86
7.6 Function lowerPage . . . 86
7.7 Function raisePage . . . 86
7.8 Function quit . . . 86
7.9 Function update . . . 86

8 PotFrame class **87**
8.1 Introduction . . . 87
8.2 Definition . . . 87
8.3 Function initPage . . . 87
8.4 Function display . . . 87
8.5 Function save_database . . . 88
8.6 Function reload . . . 88
8.7 Function lowerPage . . . 88
8.8 Function raisePage . . . 88
8.9 Function quit . . . 88
8.10 Function update . . . 89

9 HVFrame class **90**
9.1 Introduction . . . 90
9.2 Definition . . . 90
9.3 Function initPage . . . 90
9.4 Function display . . . 90
9.5 Function lowerPage . . . 90
9.6 Function raisePage . . . 90
9.7 Function quit . . . 90
9.8 Function update . . . 90

10 ConfigFrame class **91**
10.1 Introduction . . . 91
10.2 Definition . . . 91
10.3 Function initPage . . . 91
10.4 Function display . . . 91
10.5 Function validate . . . 92
10.6 Function opendialog . . . 92
10.7 Function save_database . . . 92
10.8 Function reload . . . 92
10.9 Function lowerPage . . . 92
10.10 Function raisePage . . . 92
10.11 Function quit . . . 92
10.12 Function update . . . 93

11 ExpertFrame class **94**
11.1 Introduction . . . 94
11.2 Definition . . . 94
11.3 Function appInit . . . 94
11.4 Function createDataArea . . . 94
11.5 Function quit . . . 94
11.6 Function update_display . . . 94
11.7 Function guiInit . . . 95

VI FPDHelp.py **97**

1 Introduction **99**

2 HelpFPDStandard **100**
2.1 Introduction . . . 100
2.2 Definition . . . 100
2.3 HelpFPDStandard class . . . 100
2.4 FPD Help . . . 100
2.5 Function __init__ . . . 100

VII FPDOptionGui.py 101

1 Introduction 103
1.1 Librairies 103
1.2 Global Variables 103

2 FPDStatusPotFrame class 104
2.1 Introduction 104
2.2 Definition 104
2.3 Function __init__ 104
2.4 Function __CreateGui 104
2.5 Function DrawGui 104
2.6 Function error 105

3 FPDLogFileFrame class 106
3.1 Introduction 106
3.2 Definition 106
3.3 Function __init__ 106
3.4 Function CreateGuiLogFile 106
3.5 Function Write_File 106
3.6 Function Log_Files 107
3.7 Function __Log_file_visu 107

VIII FPDRun.py 109

1 Introduction 111
1.1 Librairies 111

2 Step class 112
2.1 Introduction 112
2.2 Definition 112
2.3 Function __init__ 112
2.4 Function lowerStep 112
2.5 Function MoreLowerStep 112
2.6 Function raiseStep 112
2.7 Function MoreRaiseStep 112
2.8 Function alarm 112
2.9 Function MoreAlarm 113
2.10 Function update 113
2.11 Function MoreUpdate 113

3 Step_00 class 114
3.1 Introduction 114
3.2 Definition 114
3.3 Function MoreAlarm 114
3.4 Function MoreUpdate 114

4 Step_01 class 115
4.1 Introduction 115
4.2 Definition 115
4.3 Function MoreRaiseStep 115
4.4 Function MoreAlarm 115
4.5 Function MoreUpdate 115

5 Step_02 class **116**
5.1 Introduction . . . 116
5.2 Definition . . . 116
5.3 Function MoreRaiseStep . . . 116
5.4 Function MoreAlarm . . . 116
5.5 Function MoreUpdate . . . 116

6 Step_03 class **117**
6.1 Introduction . . . 117
6.2 Definition . . . 117
6.3 Function MoreUpdate . . . 117

7 Step_04 class **118**
7.1 Introduction . . . 118
7.2 Definition . . . 118
7.3 Function MoreAlarm . . . 118
7.4 Function MoreUpdate . . . 118

8 Step_05 class **119**
8.1 Introduction . . . 119
8.2 Definition . . . 119
8.3 Function MoreAlarm . . . 119
8.4 Function MoreUpdate . . . 119

9 Step_06 class **120**
9.1 Introduction . . . 120
9.2 Definition . . . 120
9.3 Function MoreUpdate . . . 120

10 Step_20 class **121**
10.1 Introduction . . . 121
10.2 Definition . . . 121
10.3 Function lowerStep . . . 121
10.4 Function update . . . 121

11 FPDRunMotor class **122**
11.1 Introduction . . . 122
11.2 Definition . . . 122
11.3 Function __init__ . . . 122
11.4 Function init_home_pot . . . 122
11.5 Function init_bad_status_dialog . . . 122
11.6 Function stop . . . 123
11.7 Function alarms . . . 123
11.8 Function update . . . 123

IX MyD0Gui.py **125**

1 Introduction **127**
1.1 Librairies . . . 127

2 D0Gui class **128**
2.1 Introduction . . . 128
2.2 Definition . . . 128
2.3 Function __init__ . . . 128
2.4 Function appInit . . . 128
2.5 Function guiInit . . . 128

2.6 Function __createInterface . . . 128
2.7 Function __createMenuBar . . . 128
2.8 Function createMenuBar . . . 129
2.9 Function menu_bar . . . 129
2.10 Function __exitMenuBar . . . 129
2.11 Function __createDataArea . . . 129
2.12 Function data_area . . . 129
2.13 Function createDataArea . . . 129
2.14 Function __createStatusArea . . . 129
2.15 Function __placeStatusArea . . . 129
2.16 Function statusMessage . . . 129
2.17 Function __createAboutDialog . . . 129
2.18 Function __showAbout . . . 130
2.19 Function __showHelp . . . 130

X MyPmw.py 131

1 Introduction 133
1.1 Librairies . . . 133

2 OpenDialog class 134
2.1 Introduction . . . 134
2.2 Definition . . . 134
2.3 Function __init__ . . . 134
2.4 Function changedirectory . . . 135
2.5 Function research . . . 135
2.6 Function selectdirectory . . . 135
2.7 Function selectfile . . . 135
2.8 Function openfile . . . 135
2.9 Function activate . . . 136
2.10 Function openbutton . . . 136
2.11 Function directory . . . 136
2.12 Function file . . . 136

XI MyTkinter.py 137

1 Introduction 139
1.1 Librairies . . . 139

2 MyCheckbutton class 140
2.1 Introduction . . . 140
2.2 Definition . . . 140
2.3 Function __init__ . . . 140
2.4 Function pack . . . 140
2.5 Function grid . . . 141
2.6 Function variables . . . 141
2.7 Function config . . . 141

XII Local Databases 143

1 FPDConfig.dbm 145

2 Potdbase.dbm 146