PICTURE Graphic Drawing Library 使用說明

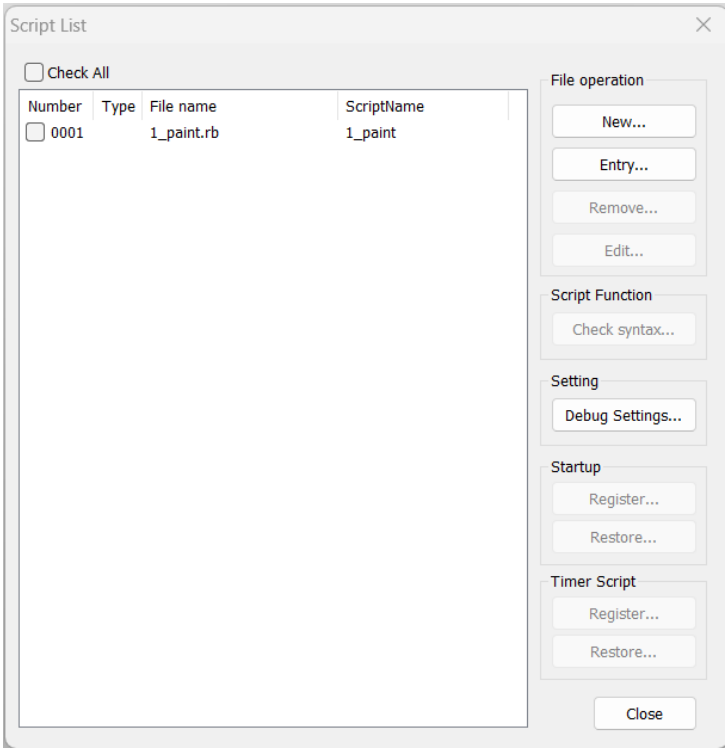
0 FANUC PICTURE Graphic Drawing Library 簡介

FANUC PICTURE Graphic Drawing Library (本文件將以繪圖函數庫代稱) 為 FANUC PICTURE 11.3 版本後支援的程式開發資源。繪圖函數庫專用於開發 PICTURE 中 Numeral/String 控制項可呼叫的自訂義函數，透過建立繪圖函數庫提供的類別所建立的實例去取用繪圖所需的變數與函數。您可以使用此函數庫開發出根據不同輸入資訊實時算繪自訂義圖形的 CNC Application。

本文件將提供使用繪圖函數庫進行具 2D 繪圖功能之 Ruby script 的開發指引。

1 範例程式

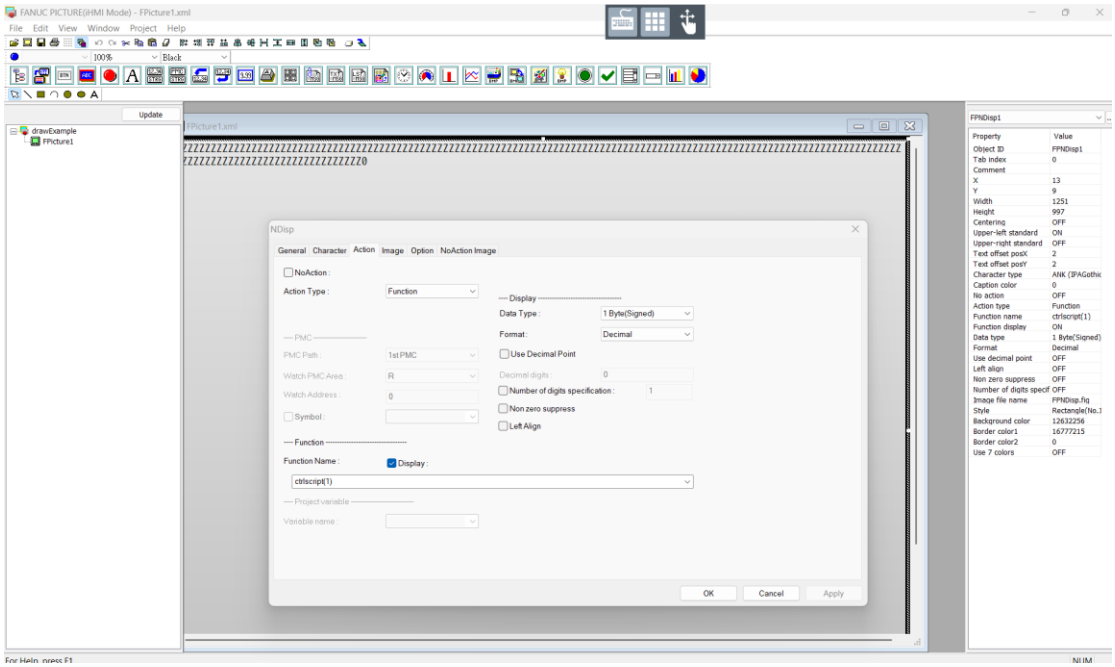
在 FANUC PICTURE 專案中，新增一個 Ruby Script – 1_paint.rb，並將 Number 設定為 1。



圖一、新增 Ruby Script

新增一個 Numeral/String 元件。設定資訊如下:

- Action Type: Function
- Function Name: ctrlscript(1)



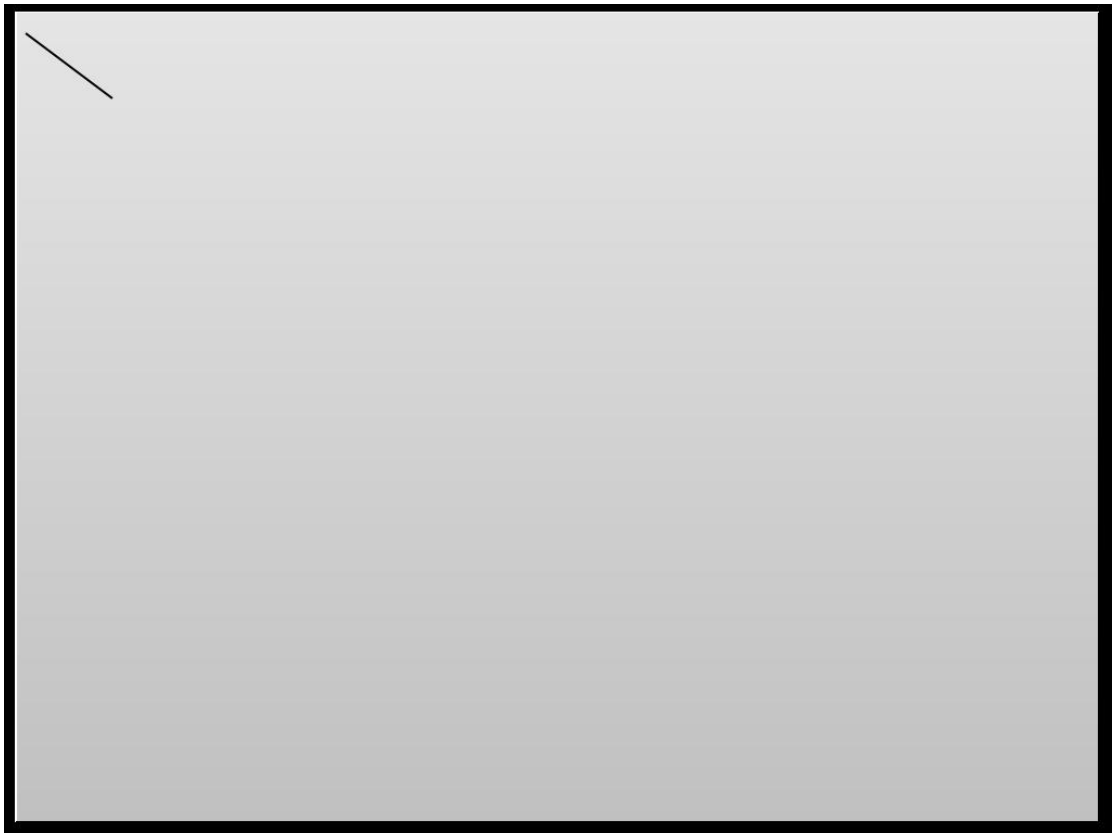
圖二、新增 Numeral/String 並設定呼叫繪圖函數專用的 function

將以下程式碼貼至空白的 1_paint.rb 中

```
class Paint < FpUserControl
  def paint()
    draw_line(10, 20, 90, 80, FP_LINE_SOLID, 2, fpcolor(0, 0, 0))
  end
end

Test = Paint.new()
Test.start()
```

程式碼一、範例程式

在 NC 上執行出來的效果如下圖所示:



圖三-範例程式執行結果

範例程式主要分為三個部分:

a. Class 定義：類別名稱可自行命名，首字母需大寫。以此範例來說，我們將類別名稱命名為“ Paint”。名稱後的 “< FpUserControl”為「此類別使用繪圖函數庫內容」的語法。建議一個類別設計於負責「一種」特定類型的繪圖任務，例如: 時鐘顯示、座標資訊顯示...等。

```
class Paint < FpUserControl #Class 定義↵
```

圖四、類別定義

- paint()函數定義: 繪圖時必須使用到的繪圖函數庫函數，函數中可定義必要的繪圖運算程式碼以及使用繪圖函數庫中的函數以進行繪圖運算。

```
def paint()
  draw_line(10, 20, 90, 80, FP_LINE_SOLID, 2, fpcolor(0, 0, 0))
end
end
```

圖五、paint()函數定義

i. **draw_line(10,20,90,80,FP_LINE_SOLID,2,fpcolor(0,0,0))**

引數	型態	值	說明
1	int	10	起始點的 x 座標值
2	int	20	起始點的 y 座標值
3	int	90	終點的 x 座標值
4	int	80	終點的 y 座標值
5	int	**FP_LINE_SOLID**	線段種類
6	int	**2**	線段寬度
7	int	**fpcolor(0,0,0)**	線段顏色

b. 建立實例

圖六、建立類別實例

(1) 實例名稱: 以大寫開頭，可自行命名

(2) 類別名稱: 根據前面類別定義的名稱做使用

c. 函數呼叫

```
Test.start()↵
```

圖七、函數呼叫

以【實例名稱】.start() 的語法呼叫，FANUC PICTURE 編譯器會自動判斷此類別的執行起點。若您希望只使用類別中的特定函數，可使用【實例名稱】.【函數名稱】的方式進行呼叫。

2 專案開發

2.1. 程式開發環境

- FANUC PICTURE 版本: 11.1 版以後
- 建議的程式編輯器(※註記二):
  - Nodepad++(圖八)
  - VS Code(圖九)

圖八、Nodepad++　　圖九、VS Code

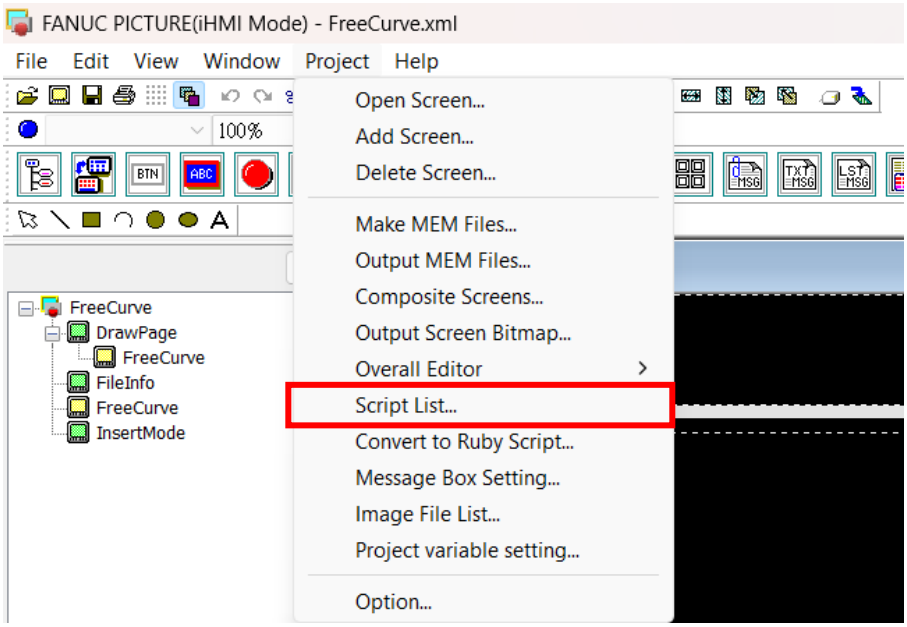
※註記一: 在 FANUC CNC GUIDE2 環境安裝具繪圖函數庫功能的 CNC Application 時，可模擬 iHMI Pro/iPC 環境的使用效果。

※註記二: 您也可以使用 FANUC PICTURE 原生的程式編輯器進行程式撰寫。這邊所提及的程式編輯器皆不包含編譯、製作執行檔...等功能，僅作為一個文字編輯器使用。FANUC PICTURE 中使用到的 Ruby Script 都需要按照本文章所介紹的方法才能成為一個對 FANUC PICTURE 專案有效之程式碼。

2.2. 登錄 Ruby Script

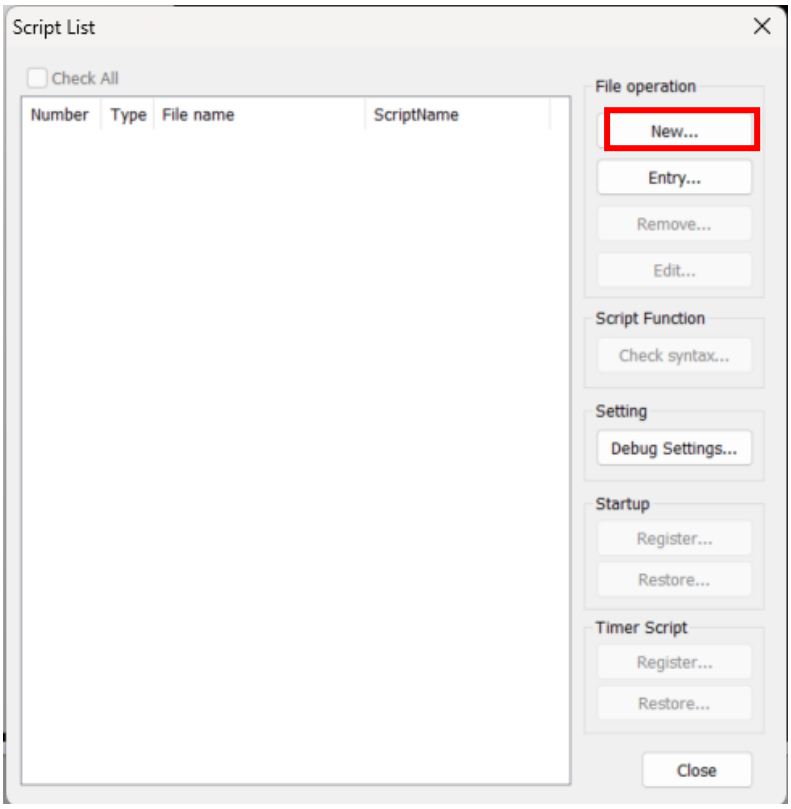
【腳本新增說明】

◆ 點擊【Project】->【Script List】進入 Script List 編輯視窗



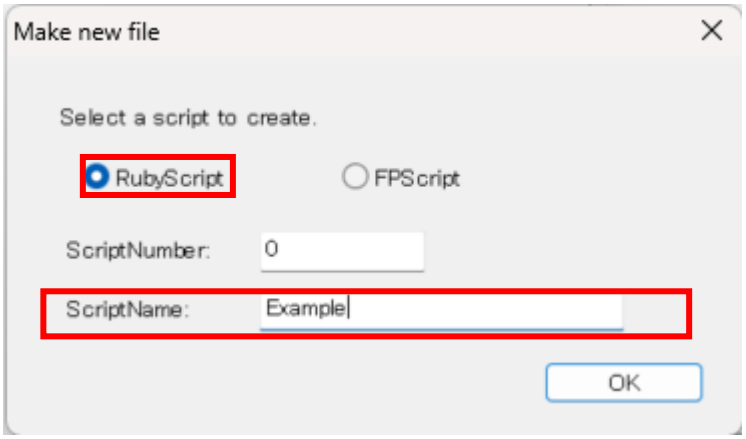
圖十、打開 Script List 編輯視窗

◆ 剛創立的專案中，您所看到的 Script List 不會有任何已登錄的 Ruby 腳本。您可透過【File operation】中的【New】按鈕來新增腳本。



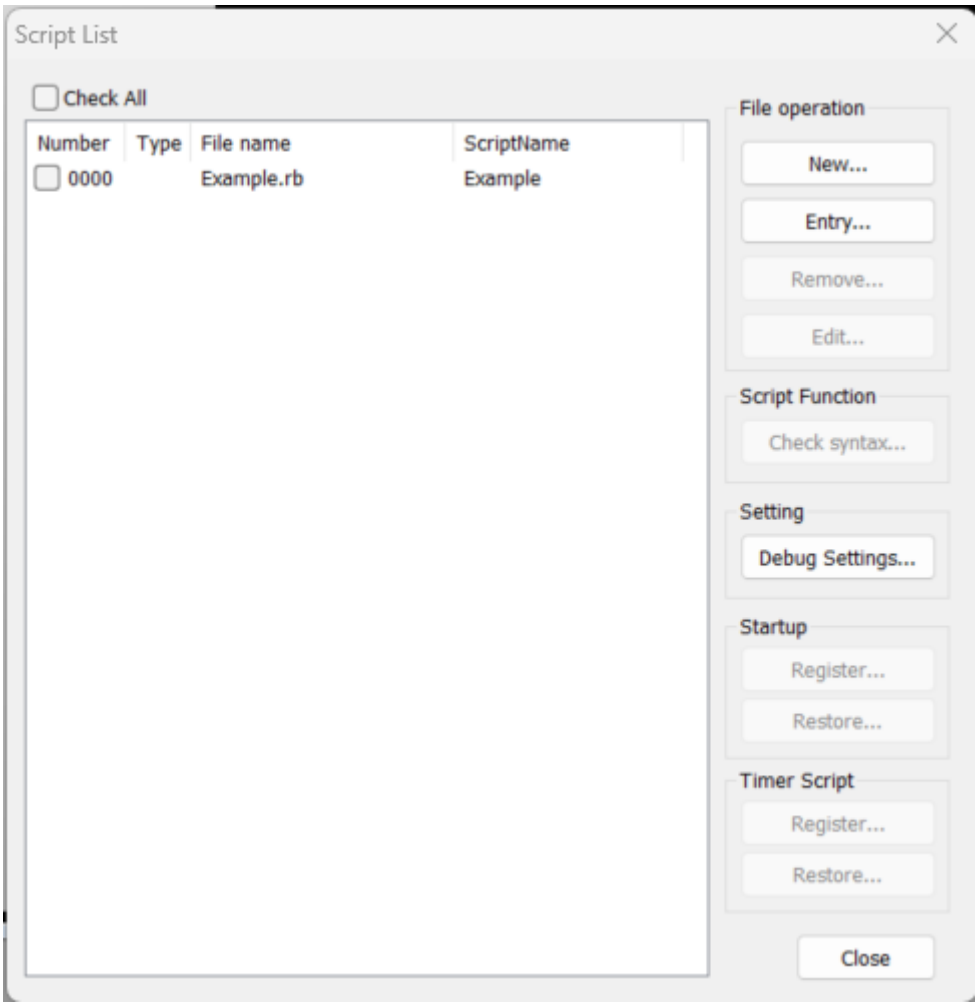
圖十一、新增腳本按鈕

勾選 【RubyScript】並輸入 Script Number 以及 ScriptName(英文)並點擊【OK】即可創建並登錄一個空白腳本至當前專案中。



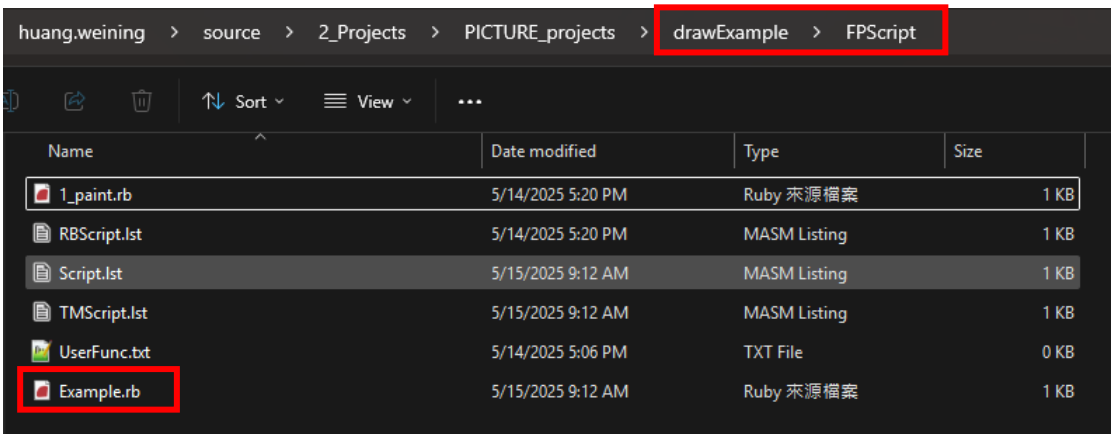
圖十二、新增腳本視窗

創建成功時，您可以在 Script List 中看見您剛新增的腳本程式。



圖十三、新增成功

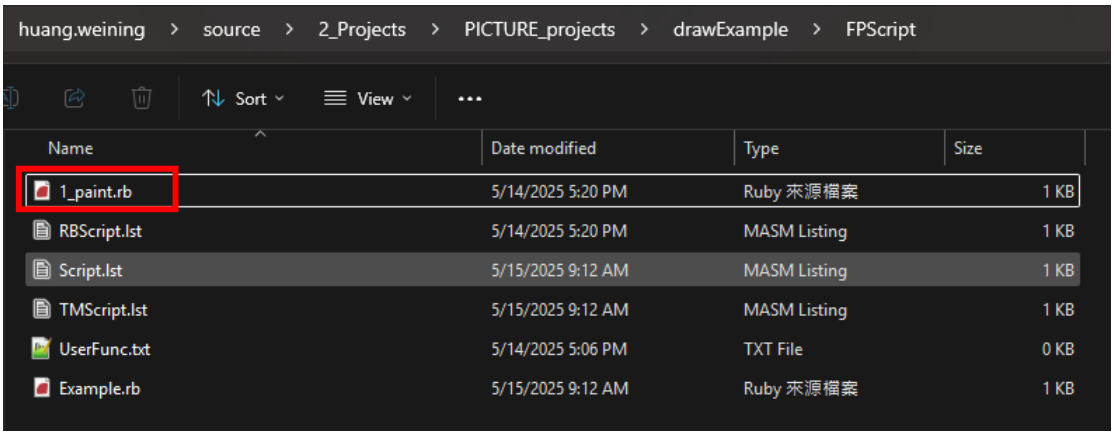
而創建出來的 Ruby Script 會在專案中的 FPScript 資料夾中。



圖十四、新增腳本的實際路徑

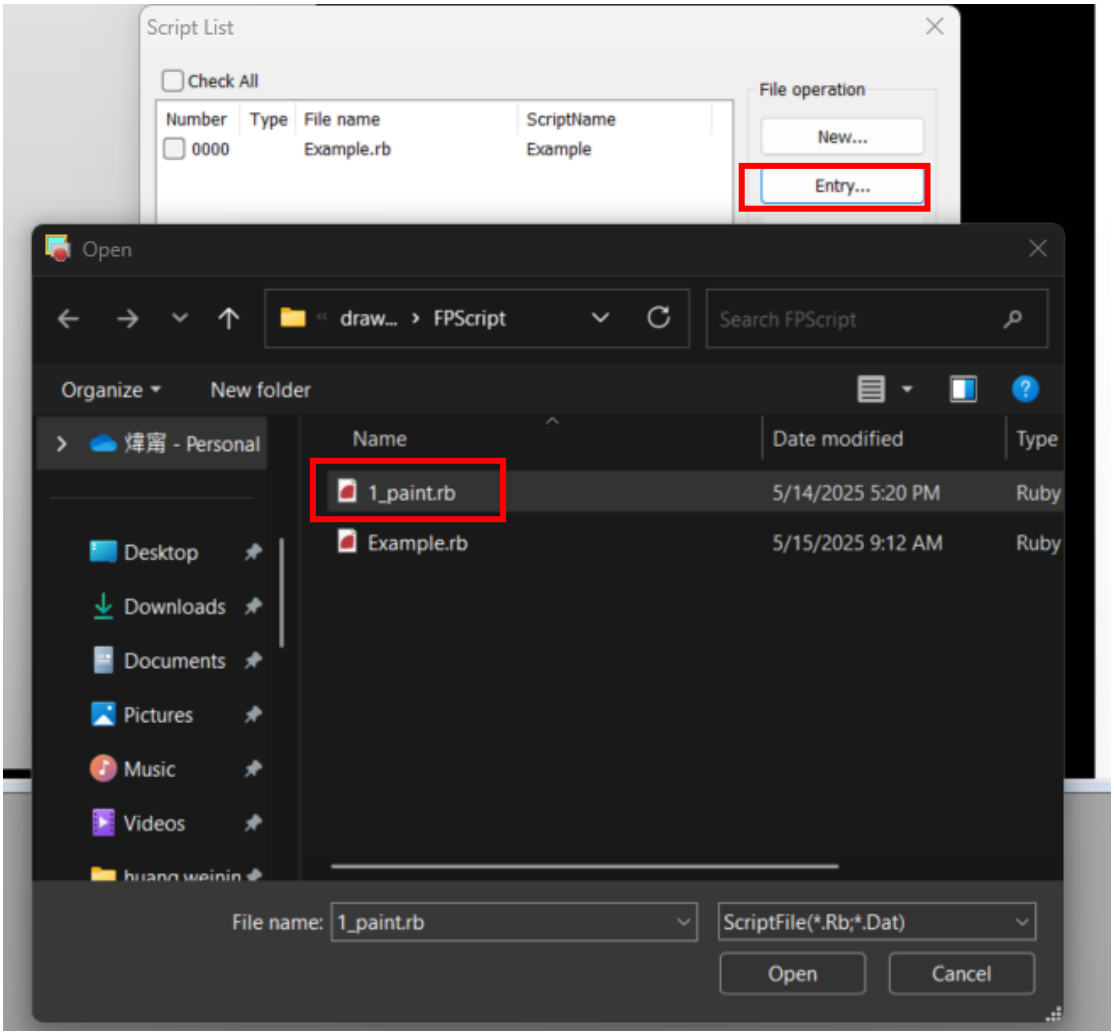
【腳本登錄】

您也可以透過其他編輯器創建一個以「.rb」為附檔名的 Ruby 腳本，並將腳本儲存於 FPScript 資料夾中。再透過 Script List 中的【Entry】將該腳本登錄進專案中。



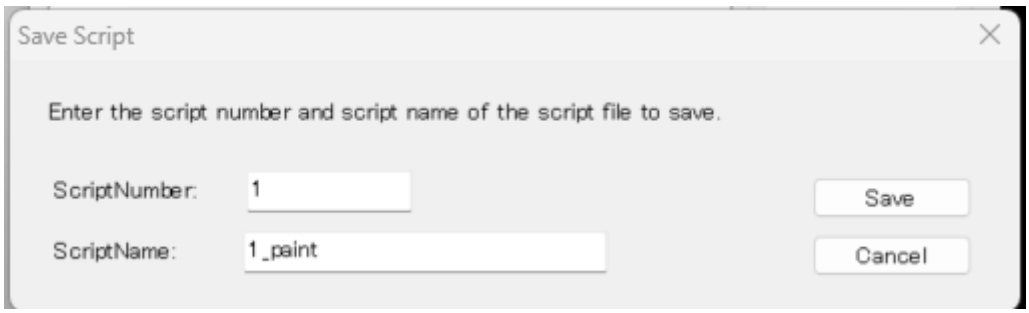
圖十五、欲登錄的腳本

以上圖路徑為範例，我們想要將 1_paint.rb 登錄進專案中。首先需要將此腳本至於 FPScript 資料夾中。在 PICTURE 的 Script List 中點選【Entry】以選擇該腳本。



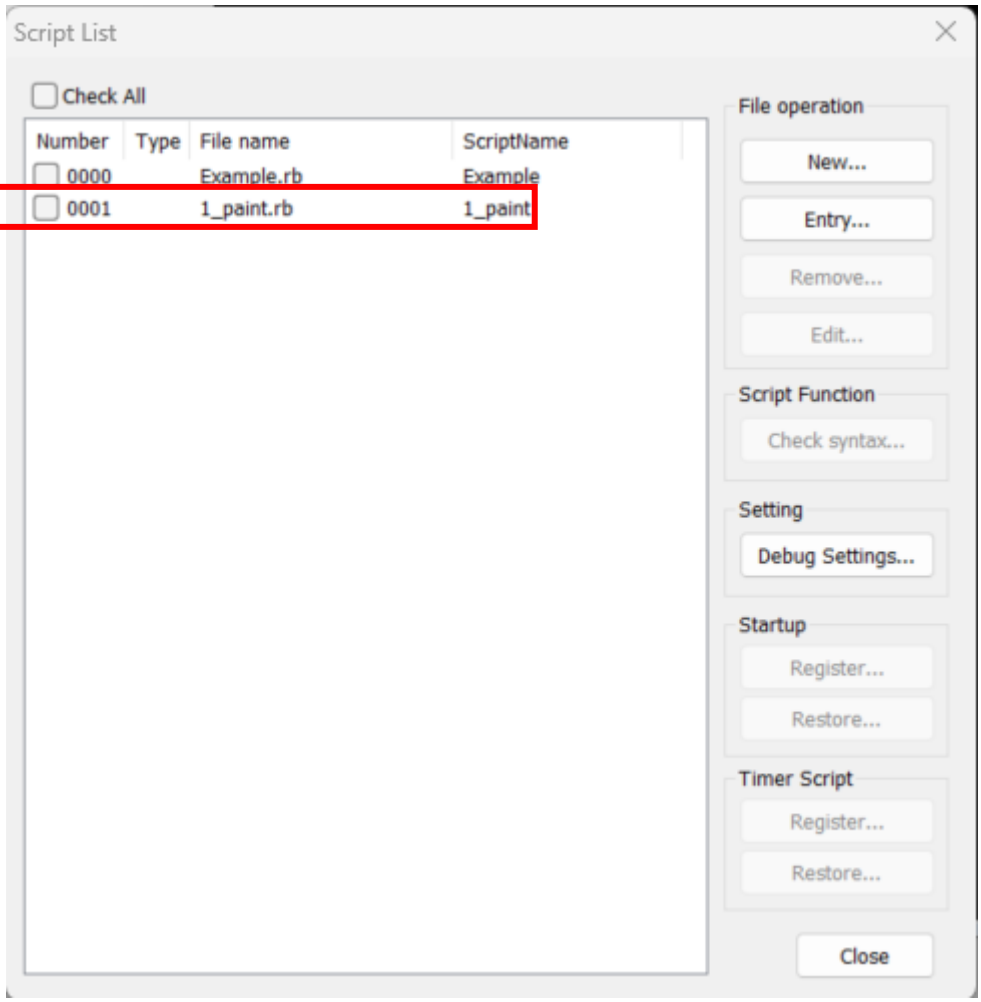
圖十六、登錄腳本設定

設定 ScriptNumber 及 ScriptName 後點擊【Save】。



圖十七、登錄腳本設定視窗

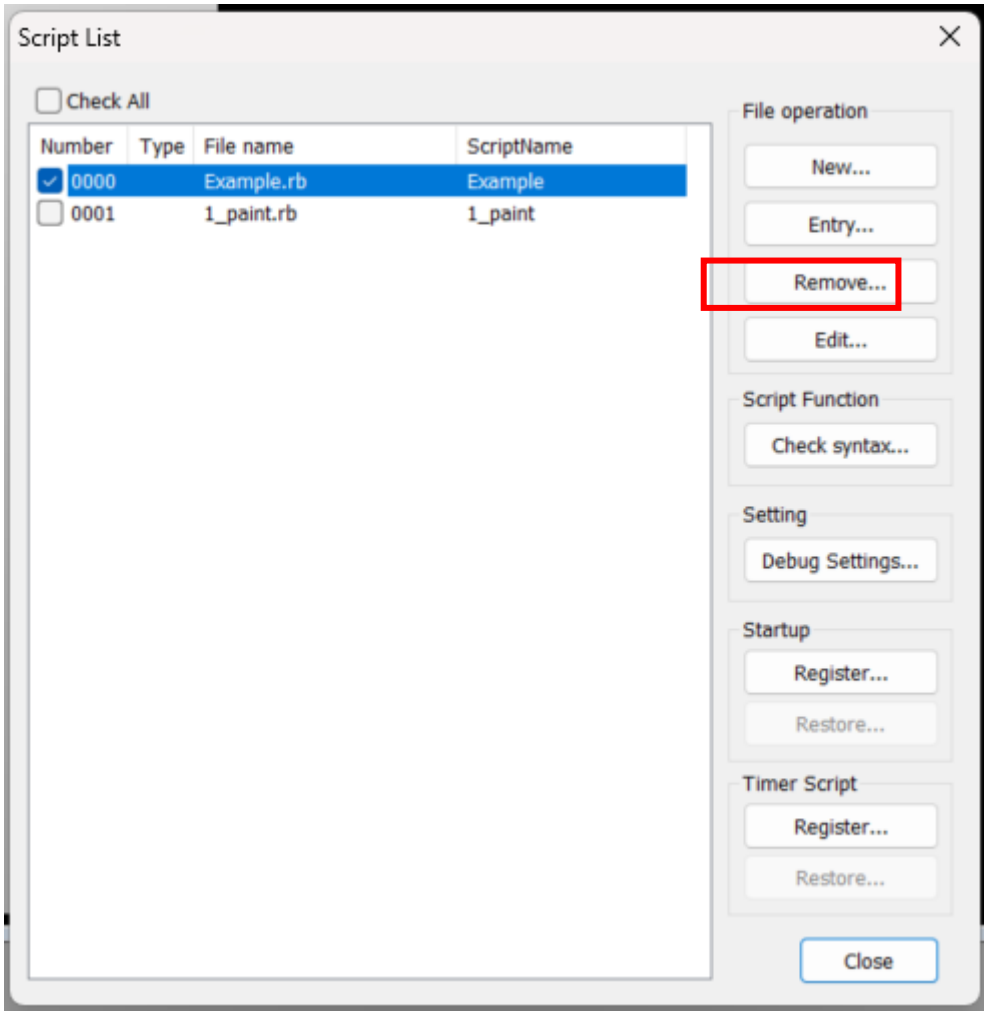
登錄成功時，您可以在 Script List 中看見您剛登錄的腳本程式。


Script List
Check All
Number
Type
File name
ScriptName
0000
Example.rb
Example
0001
1_paint.rb
1_paint
File operation
New...
Entry...
Remove...
Edit...
Script Function
Check syntax...
Setting
Debug Settings...
Startup
Register...
Restore...
Timer Script
Register...
Restore...
Close

圖十八、登錄成功

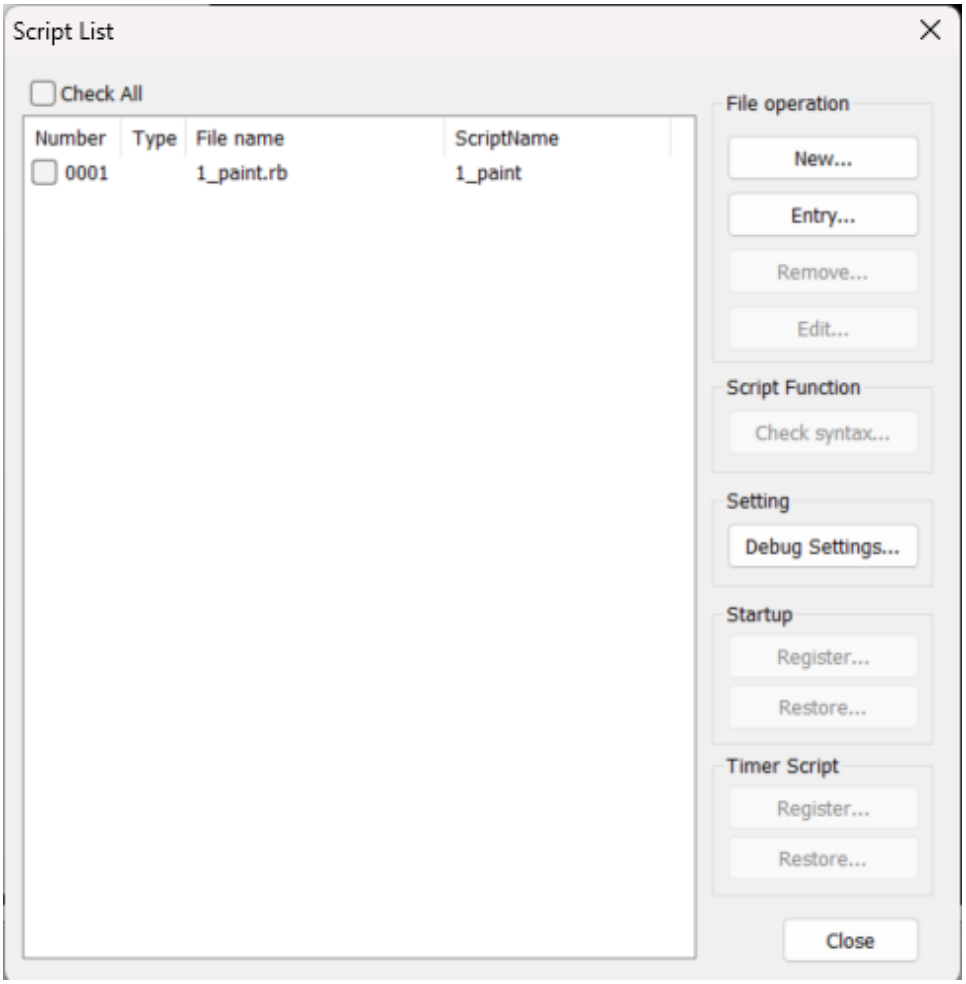
【腳本移除】

若今天有不再繼續於專案中使用的腳本時，您可以在 Script List 中勾選該腳本。並點選【Remove】以進行移除。



圖十九、移除腳本

移除成功時，您將不會在 Script List 上看見您所移除的腳本程式資訊。


Script List
Check All
Number
Type
File name
ScriptName
0001
1_paint.rb
1_paint
File operation
New...
Entry...
Remove...
Edit...
Script Function
Check syntax...
Setting
Debug Settings...
Startup
Register...
Restore...
Timer Script
Register...
Restore...
Close

圖二十、移除成功

2.3. 使用繪圖函數庫進行開發

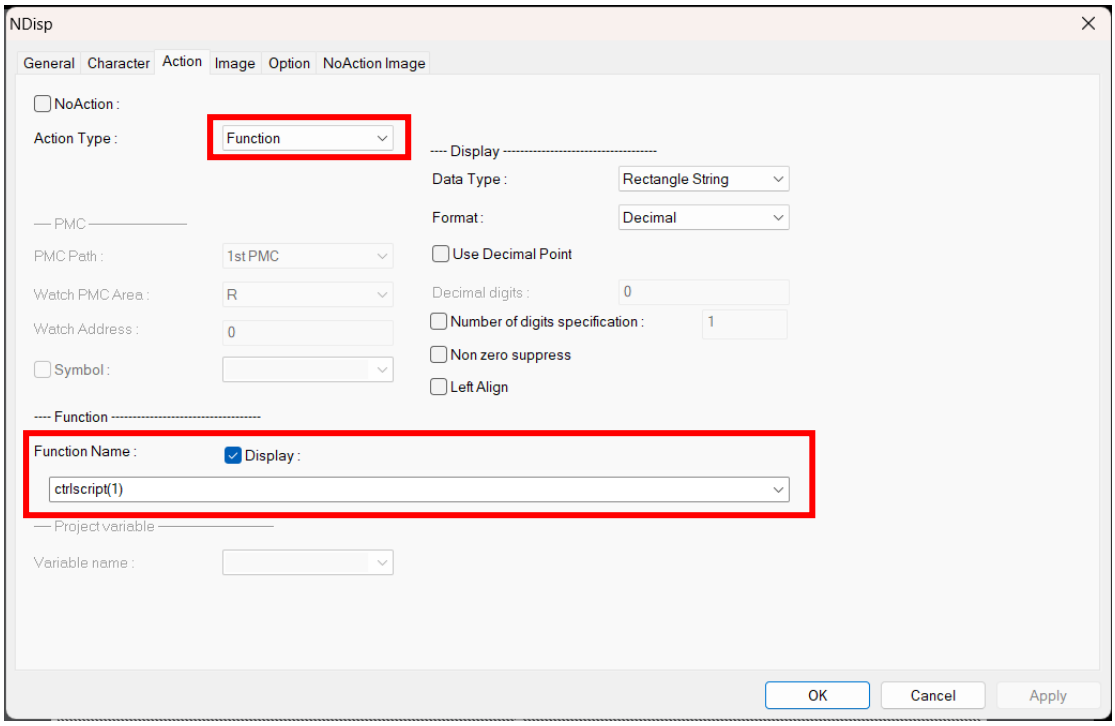
使用您認為合適的編輯器，參考本文件第一章的範例程式進行程式開發。

2.4. 控制項設定

在您的專案中設定 Numeral/String 的 Action。

- 將 Action Type 選擇為 Function
- 在 Function Name 欄位選擇【ctrlscript()】，括弧中請填入對應的 Script Number，如 ctrlscript(1)。

設定完成後點擊【OK】完成設定。



圖二十一、控制項設定

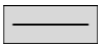
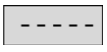
3 函數庫常用變數、函數介紹

3.1. 參考文件

FANUC PICTURE 光碟中的 A-42148-00830EN.pdf。該檔案存放於【FP 光碟路徑】/FP/document/底下。

3.2. 繪圖函數庫專用變數

◆ Line Style 皆為 Integer 型態變數，共有 5 種，只可用於繪圖函數庫中的線段資訊設定。

- FP_LINE_SOLID : 實心線 
- FP_LINE_DASH : 虛線 
- FP_LINE_DOT : 虛線 
- FP_LINE_DASHDOT : 
- FP_LINE_DASHDOTDOT : 

3.3. 事件函數

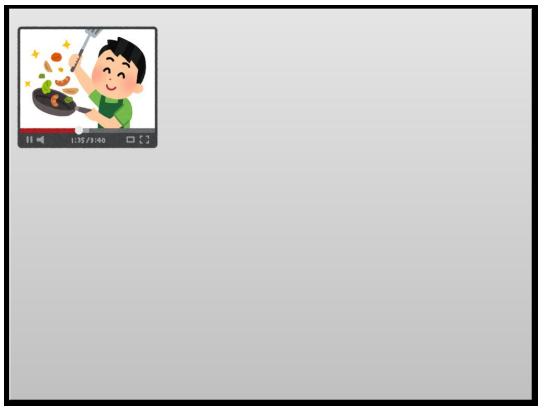
◆ paint()

- 用途: 觸發繪圖程序，繪圖函數必須在此函數範圍內使用。
- 範例_顯示圖片:

```
class Paint < FpUserControl
  def paint()
    image = "C:\\Users\\huang.weining\\Pictures\\video_cooking_man.png"
    draw_image(10, 20, 290, 280, image, true)
  end
end


Test = Paint.new()
Test.paint()
```

程式碼二、paint()函數範例

- 程式執行結果:



圖二十二、paint()範例執行結果

◆ timer()

- 用途: 根據設定的時間間隔觸發函數內容，可用於需要定時刷新或定頻處理的功能開發。
- 範例_時鐘顯示:

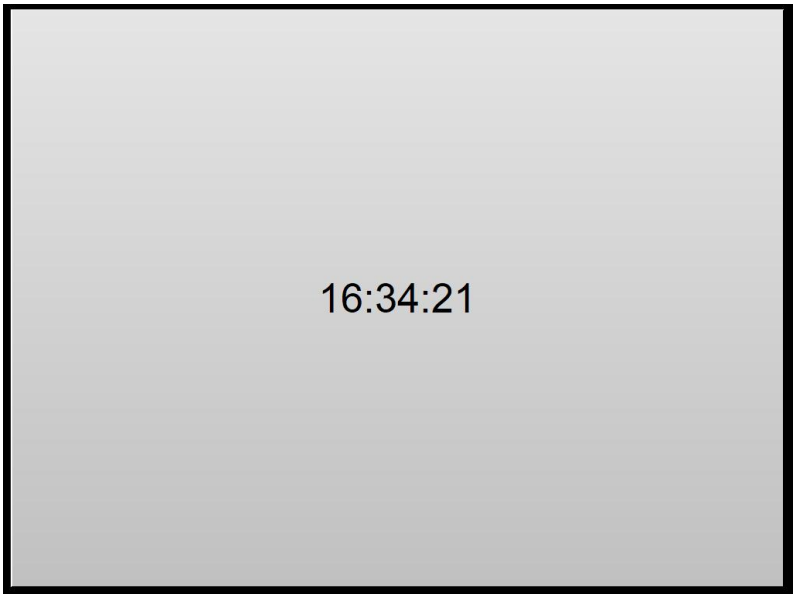
```
class DrawClock < FpUserControl
    def initialize() # Start of displaying screen
        super
        self.timer_interval = 1000 # timer cycles 1 second (=1000ms)
    end
    def paint() # Screen drawing
        s = ctime(time()[1])[1].slice(11, 8) # s = hh:mm:ss
        x, y, w, h = rect
        # Create font
        font = FpFont.new("Arial", 32)
        # Display time
        draw_text(x, y, x + w, y + h, FP_TEXT_CENTER, s, font,
fpcolor(0,0,0))
        # Release font
        font.dispose()
    end
    def timer() # Every timer cycle
        restore_rect() # Redraw background
        paint()
        update_rect() # Update drawing
    end
end
Test = DrawClock.new()
Test.start()
```

程式碼三、timer()範例

※restore_rect()及 update_rect()函數必須加在 paint()前後，才可正確刷新畫面。

※此範例執行結果會因您當下的時間而與圖二十三所示的圖片不同。

- 程式執行結果:



圖二十三、timer()範例執行結果

3.4. 屬性相關函數或類別變數

- rect()
  - 用途：用於取得 Numeral/String 控制項的座標資訊，包含:
    - x : Numeral/String 控制項左上角的 x 座標
    - y : Numeral/String 控制項右上角的 x 座標
    - width : Numeral/String 控制項的寬(在程式中抓到的控制項寬度會與 PICTURE 上設定的寬度不同，進行程式開發時要以 rect()產生的 width 資訊為主)
    - height : Numeral/String 控制項的高(在程式中抓到的控制項高度會與 PICTURE 上設定的高度不同，進行程式開發時要以 rect()產生的 width 資訊為主)

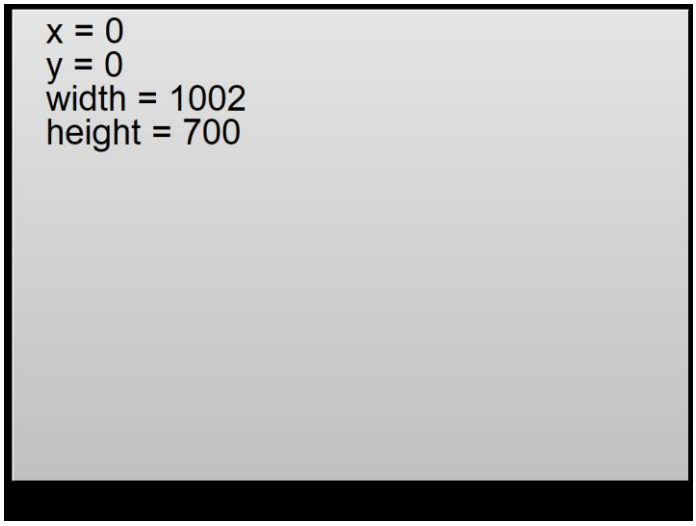
    若需要使用其中一項座標資訊需以呼叫實例屬性成員的語法進行。
  - 回傳值型態: FpRect 實例。
  - 範例_檢查 Numeral/String 的座標資訊:

```
class Paint < FpUserControl
    def paint() # Screen drawing
        r = rect
        font = FpFont.new("Arial", 32)
        draw_text(50, 50, "x = " + r.x.to_s, font, fpcolor(0,0,0))
        draw_text(50, 100,"y = " + r.y.to_s, font, fpcolor(0,0,0))
        draw_text(50, 150,"width = " + r.w.to_s, font, fpcolor(0,0,0))
        draw_text(50, 200,"height = " + r.h.to_s, font, fpcolor(0,0,0))
    end
```

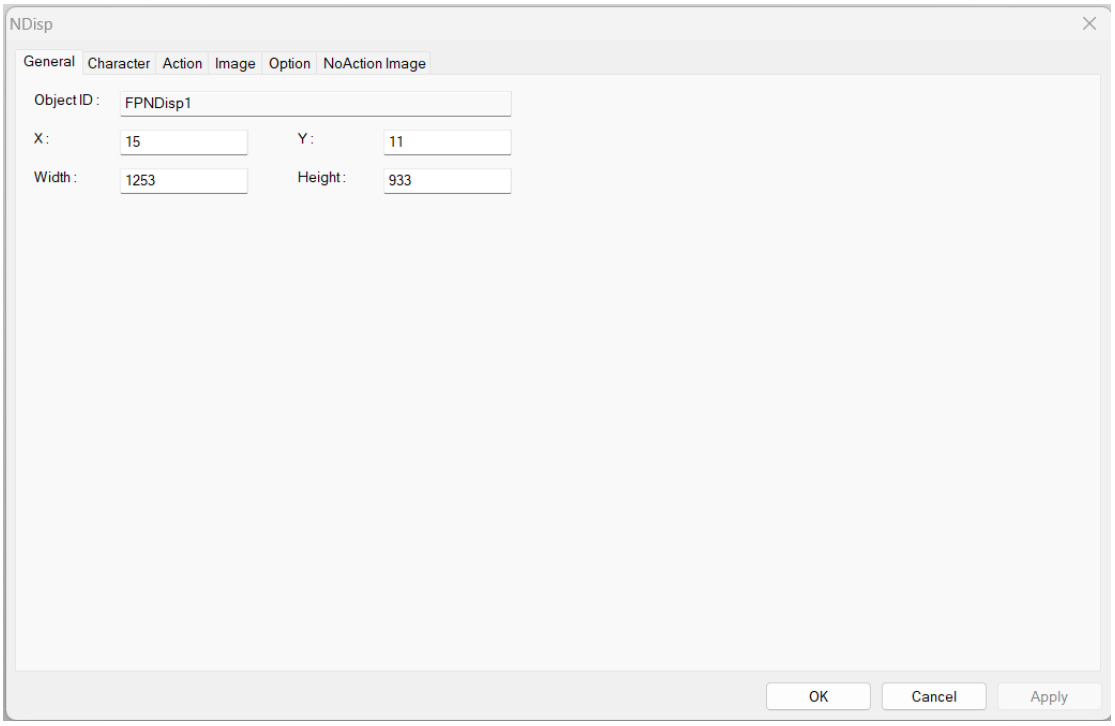
```
end

Example = Paint.new() #新增 DrawClock 實例
Example.start()
```

程式碼四、rect()範例

■ 程式執行結果:



圖二十四、rect()抓取道的座標資訊



圖二十五、PICTURE 上設定的座標資訊

我們可以在圖二十四跟圖二十五中看到程式與 PICTURE 上設定的座標資訊是不同的。

- restore_rect()、update_rect()
  - 用途：restore_rect()用於清除畫面內容；update_rect()用於更新畫面內容。若在 timer()中使用 paint()函數時需要在前後各自加上這兩個函數(restore_rect()在前，update_rect()在後)才可正確更新畫面。錯誤的使用可能導致畫面無法如期更新或有圖型疊加...等的問題發生。
  - 範例: 請參考 timer()函數說明

3.5. 繪圖函數

◆ fpcolor(r,g,b) -> Integer

- 用途: 以 RGB 數值設定 FANUC PICTURE 專用的色彩資訊，用於設定繪圖函數庫中提供的繪圖函數顏色設定引數。
- 引數說明:

引數	型態	值	說明
1	int	r	設定紅色亮度
2	int	g	設定綠色亮度
3	int	b	設定藍色亮度

- 回傳值型態: Integer
- 範例_設定線段的色彩資訊為黑色:

```
class Paint < FpUserControl
  def paint()
    draw_line(10, 20, 90, 80, FP_LINE_SOLID, 2, fpcolor(0, 0, 0))#黑色線
  end
end

Test = Paint.new()
Test.paint()
```

程式碼五、fpcolor()範例程式

- 程式執行結果:

圖二十六、fpcolor()範例執行結果

◆ **FpFont.new(type, size) -> FpFont**

- 用途: 用以設定繪圖函數庫可用之字體資訊
- 引數說明:

引數	型態	值	說明
1	string	type	設定字體種類
2	int	size	設定字體大小

- 回傳值型態: FpFont 類別實例
- 範例_設定字體資訊為 Arial 10 級字:

```
class Paint < FpUserControl
  def paint()
    font =  FpFont.new("Arial",80) #設定字體
    draw_text(120,240,"Hello World",font, fpcolor(0,0,0))
  end
end

Test = Paint.new()
Test.paint()
```

程式碼六、FpFont.new()範例程式

- 程式執行結果:

圖二十七、FpFont()範例執行結果

◆ **draw_line(sx, sy, ex, ey, style, width, color)**

- 用途: 繪製線段
- 引數說明:

引數	型態	值	說明
1	int	sx	起始點的 x 座標值
2	int	sy	起始點的 y 座標值
3	int	ex	終點的 x 座標值
4	int	ey	終點的 y 座標值
5	int	**style**	線段種類
6	int	**width**	線段寬度
7	int	**color**	線段顏色

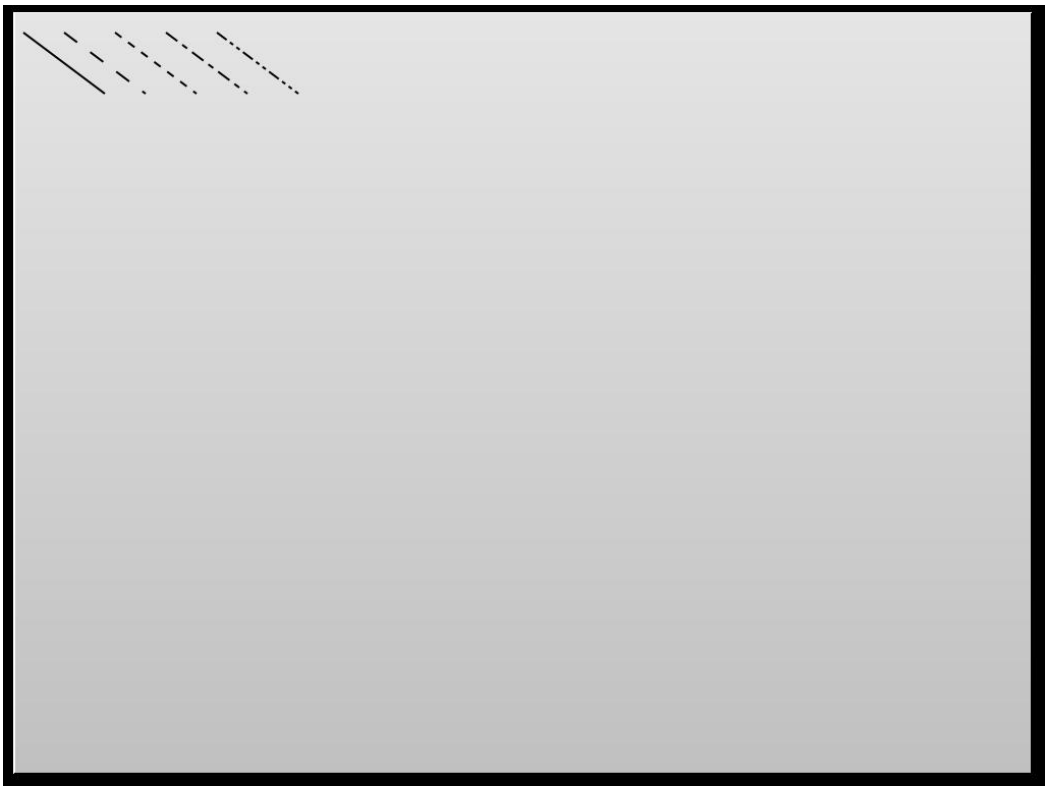
- 回傳值型態: 無回傳值
- 範例_繪製線段:

```
class Paint < FpUserControl

  def paint()

    draw_line(10, 20, 90, 80, FP_LINE_SOLID, 2, fpcolor(0, 0, 0)) #左一

    draw_line(50, 20, 130, 80, FP_LINE_DASH, 2, fpcolor(0, 0, 0)) #左二

    draw_line(100, 20, 180, 80, FP_LINE_DOT, 2, fpcolor(0, 0, 0)) #中

    draw_line(150, 20, 230, 80, FP_LINE_DASHDOT, 2, fpcolor(0, 0, 0))#右
二

    draw_line(200, 20, 280, 80, FP_LINE_DASHDOTDOT, 2, fpcolor(0, 0,
0))#右一
```

```
  end
end


Test = Paint.new()
Test.paint()
```

程式碼六、draw_line()範例程式

- 程式執行結果:



圖二十八、draw_line()範例執行結果

◆ **draw_rect(sx, sy, ex, ey, style, width, color)**

- 用途: 繪製長方形
- 引數說明:

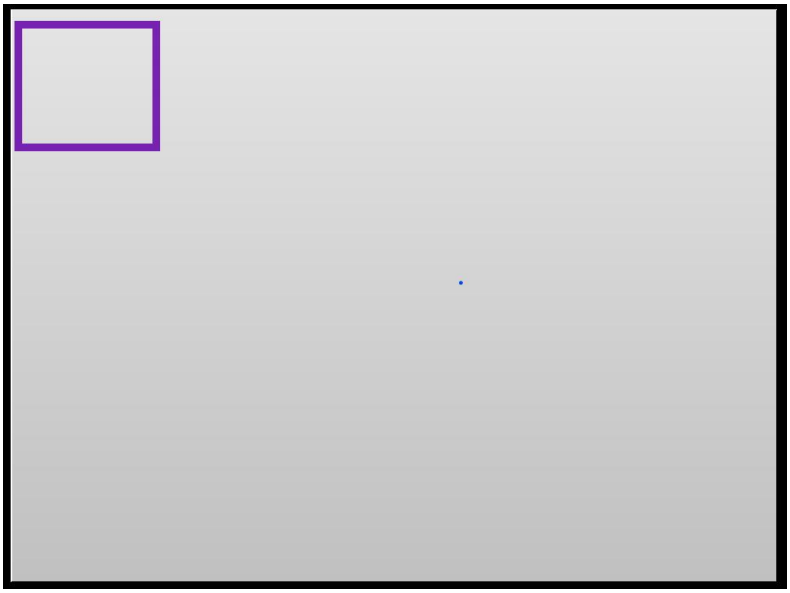
引數	型態	值	說明
1	int	sx	左上角的 x 座標值
2	int	sy	左上角的 y 座標值
3	int	ex	右下角的 x 座標值
4	int	ey	右下角的 y 座標值
5	int	style	線段種類
6	int	**width**	線段寬度
7	int	**color**	線段顏色

- 回傳值型態: 無回傳值
- 範例_繪製紫色長方形:

```
class Paint < FpUserControl
  def paint()
    draw_rect(10, 20, 190, 180, FP_LINE_SOLID, 10, fpcolor(120, 34,
179))
  end
end

Test = Paint.new()
Test.paint()
```

程式碼七、draw_rect()範例程式

- 程式執行結果:



圖二十九、draw_rect()範例執行結果

◆ **fill_rect(sx, sy, ex, ey, color)**

- 用途: 繪製填實心長方形
- 引數說明:

引數	型態	值	說明
1	int	sx	左上角的 x 座標值
2	int	sy	左上角的 y 座標值
3	int	ex	右下角的 x 座標值
4	int	ey	右下角的 y 座標值
5	int	**color**	圖形顏色

- 回傳值型態: 無回傳值
- 範例_繪製藍色實心長方形:

```
class Paint < FpUserControl
  def paint()
    fill_rect(10, 20, 190, 180, fpcolor(0, 120, 255))
  end
end


Test = Paint.new()
Test.paint()
```

程式碼八、fill_rect()範例程式

■ 程式執行結果:

圖三十、fill_rect()範例執行結果

◆ **draw_ellipse(cx, cy, rx, ry, style, width, color)**

■ 用途: 繪製橢圓

■ 引數說明:

引數	型態	值	說明
1	int	cx	圓心的 x 座標值
2	int	cy	圓心的 y 座標值
3	int	rx	x 方向的半徑長
4	int	ry	y 方向的半徑長
5	int	style	線段種類
6	int	width	線段寬度
7	int	**color**	線段顏色

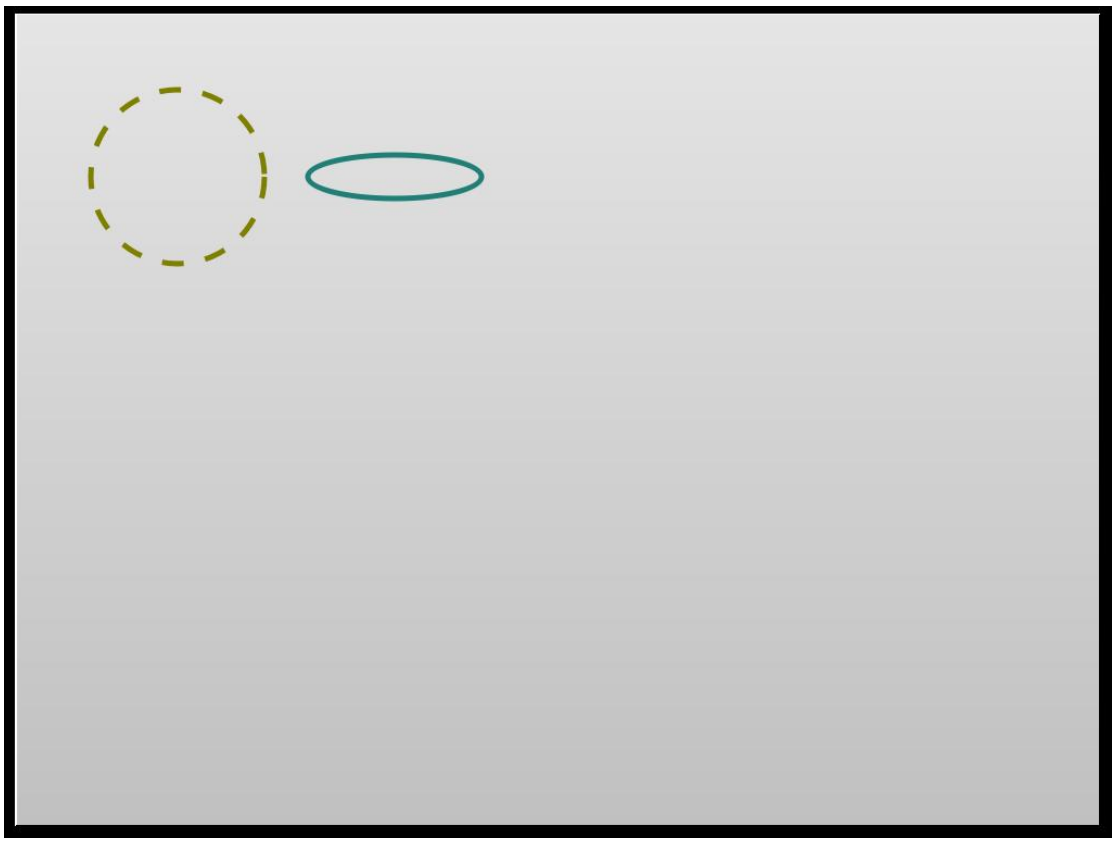
■ 回傳值型態: 無回傳值

■ 範例_繪製虛線圓形與實線橢圓:

```
class Paint < FpUserControl
  def paint()
    draw_ellipse(150, 150, 80, 80, FP_LINE_DOT, 5, fpcolor(127, 127, 0))
    draw_ellipse(350, 150, 80, 20, FP_LINE_SOLID, 5, fpcolor(33, 127,
120))
  end
end
Test = Paint.new()
Test.paint()
```

程式碼九、**draw_ellipse()**範例程式

- 程式執行結果:



圖三十一、**draw_ellipse()**範例執行結果

◆ **fill_ellipse(cx, cy, rx, ry, color)**

■ 用途: 繪製實心橢圓

■ 引數說明:

引數	型態	值	說明
1	int	cx	圓心的 x 座標值
2	int	cy	圓心的 y 座標值
3	int	rx	x 方向的半徑長
4	int	ry	y 方向的半徑長
5	int	**color**	圖形顏色

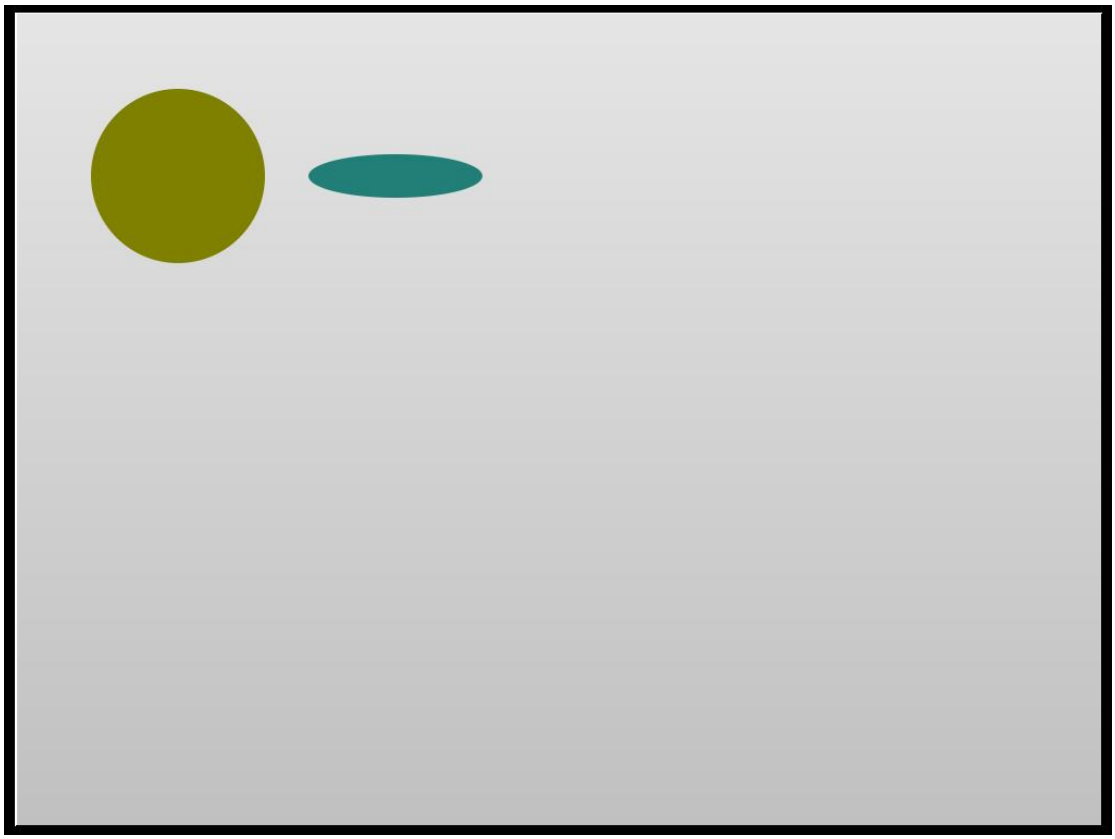
■ 回傳值型態: 無回傳值

■ 範例_繪製實心圓形與實心橢圓:

```
class Paint < FpUserControl
  def paint()
    fill_ellipse(150, 150, 80, 80, fpcolor(127, 127, 0))
    fill_ellipse(350, 150, 80, 20, fpcolor(33, 127, 120))
  end
end

Test = Paint.new()
Test.paint()
```

程式碼十、**fill_ellipse()**範例程式

■ 程式執行結果:



圖三十一、**fill_ellipse**()範例執行結果

◆ **draw_arc(cx, cy, rx, ry, sa, ea, style, width, color)**

■ 用途: 繪製圓弧

■ 引數說明:

引數	型態	值	說明
1	int	cx	圓心的 x 座標值
2	int	cy	圓心的 y 座標值
3	int	rx	x 方向的半徑長
4	int	ry	y 方向的半徑長
5	int	sa	起始角
6	int	ea	結束角
7	int	style	線段種類
8	int	width	線段寬度
9	int	**color**	線段顏色

■ 回傳值型態: 無回傳值

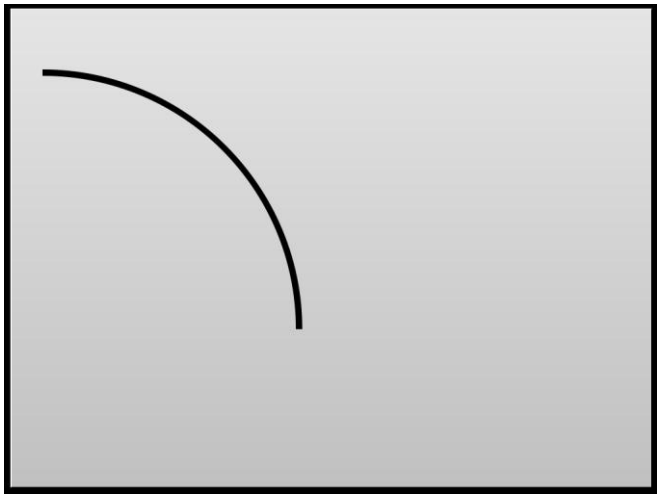
■ 範例_繪製 0 到 90 度的圓弧:

```
class Paint < FpUserControl
  def paint()   draw_arc(50, 500, 400, 400, 0.0, 90.0, FP_LINE_SOLID,
10, fpcolor(0, 0, 0))
  end
end

Test = Paint.new()
Test.paint()
```

```
end
```

程式碼十、**draw_arc()**範例程式

- 程式執行結果:



圖三十二、**draw_arc()**範例執行結果

- **draw_pie (cx, cy, rx, ry, sa, ea, style, width, color)**
  - 用途: 繪製圓餅圖
  - 引數說明:

引數	型態	值	說明
1	int	cx	圓心的 x 座標值
2	int	cy	圓心的 y 座標值
3	int	rx	x 方向的半徑長
4	int	ry	y 方向的半徑長
5	int	sa	起始角
6	int	ea	結束角
7	int	style	線段種類
8	int	width	線段寬度
9	int	**color**	線段顏色

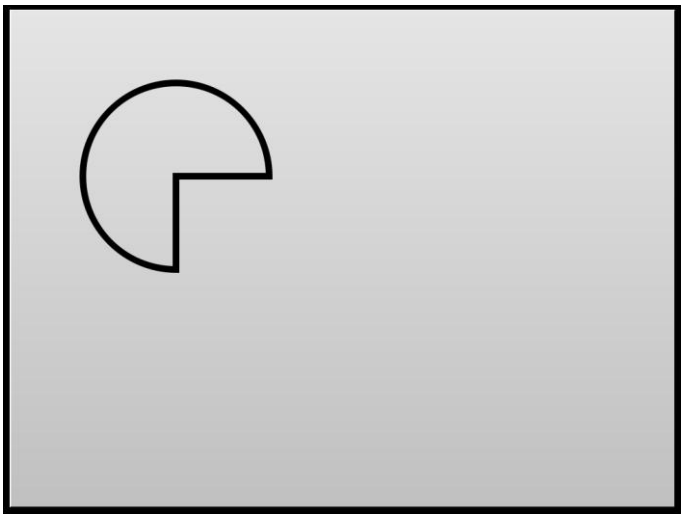
  - 回傳值型態: 無回傳值
  - 範例_繪製 0 到 270 度的圓餅圖:

```
class Paint < FpUserControl
  def paint()
   draw_pie(250, 250, 140, 140, 0.0, 270.0, FP_LINE_SOLID, 10,
fpcolor(0, 0, 0))
  end
end

Test = Paint.new()
```

```
Test.paint()
```

程式碼十一、**draw_pie()**範例程式

- 程式執行結果:



圖三十三、**draw_pie()**範例執行結果

◆ **fill_pie (cx, cy, rx, ry, sa, ea, color)**

■ 用途: 繪製實心圓餅圖

■ 引數說明:

引數	型態	值	說明
1	int	cx	圓心的 x 座標值
2	int	cy	圓心的 y 座標值
3	int	rx	x 方向的半徑長
4	int	ry	y 方向的半徑長
5	int	sa	起始角
6	int	ea	結束角
7	int	**color**	圖形顏色

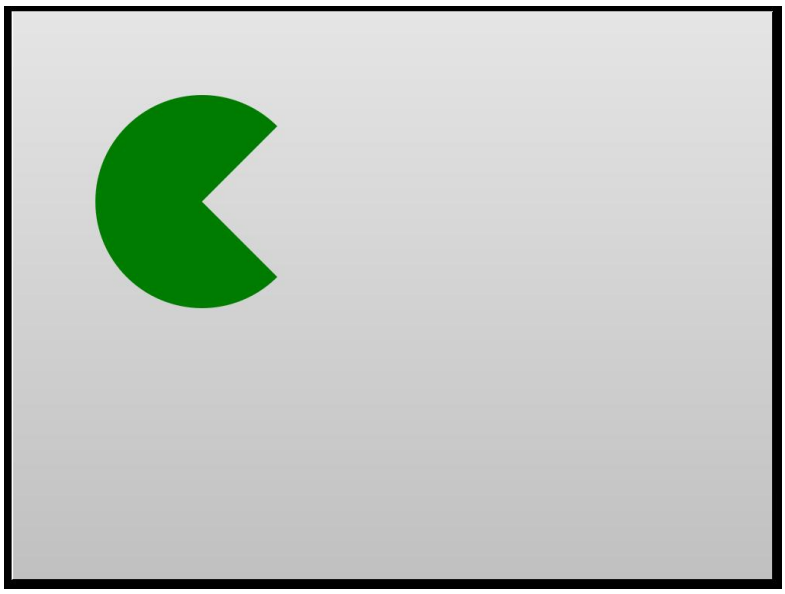
■ 回傳值型態: 無回傳值

■ 範例_繪製 45 到 315 度的實心圓餅圖:

```
class Paint < FpUserControl
  def paint()
   fill_pie(250, 250, 140, 140, 45.0, 315.0, fpcolor(0, 125, 0))
  end
end

Test = Paint.new()
Test.paint()
```

程式碼十一、**fill_pie()**範例程式

■ 程式執行結果:



圖三十三、**fill_pie()**範例執行結果

◆ **draw_polygon(points, style, width, color)**

◆ 用途: 繪製多邊形

◆ 引數說明:

引數	型態	值	說明
1	int array[][2]	points	點座標
2	int	**style**	線段種類
3	int	**width**	線段寬度
4	int	**color**	線段顏色

■ 回傳值型態: 無回傳值

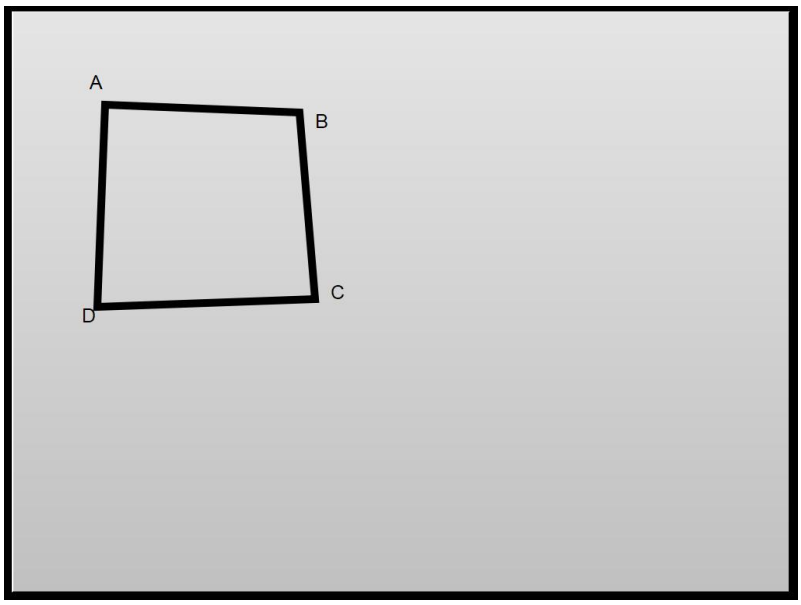
■ 範例_繪製多邊形:

```
class Paint < FpUserControl
  def paint()
    font = FpFont.new("Arial",15)
    #點 A、B、C、D 設定
    points =[[120,120], [370,130], [390,370], [110,380]]
    draw_text(100,100, "A", font, fpcolor(0, 0, 0))
    draw_text(390,150, "B", font, fpcolor(0, 0, 0))
    draw_text(410,370, "C", font, fpcolor(0, 0, 0))
    draw_text(90,400, "D", font, fpcolor(0, 0, 0))
    draw_polygon(points, FP_LINE_SOLID, 10, fpcolor(0, 0, 0))
  end
end
```

```
Test = Paint.new()

Test.paint()
```

程式碼十二、**draw_polygon()**範例程式

- 程式執行結果:



圖三十四、**draw_polygon()**範例執行結果

◆ **fill_polygon(points, color, winding)**

◆ 用途: 繪製實心多邊形

◆ 引數說明:

引數	型態	值	說明
1	int array[][2]	points	點座標
2	int	color	圖形顏色
3	Boolean	**winding**	繞行設定

■ 回傳值型態: 無回傳值

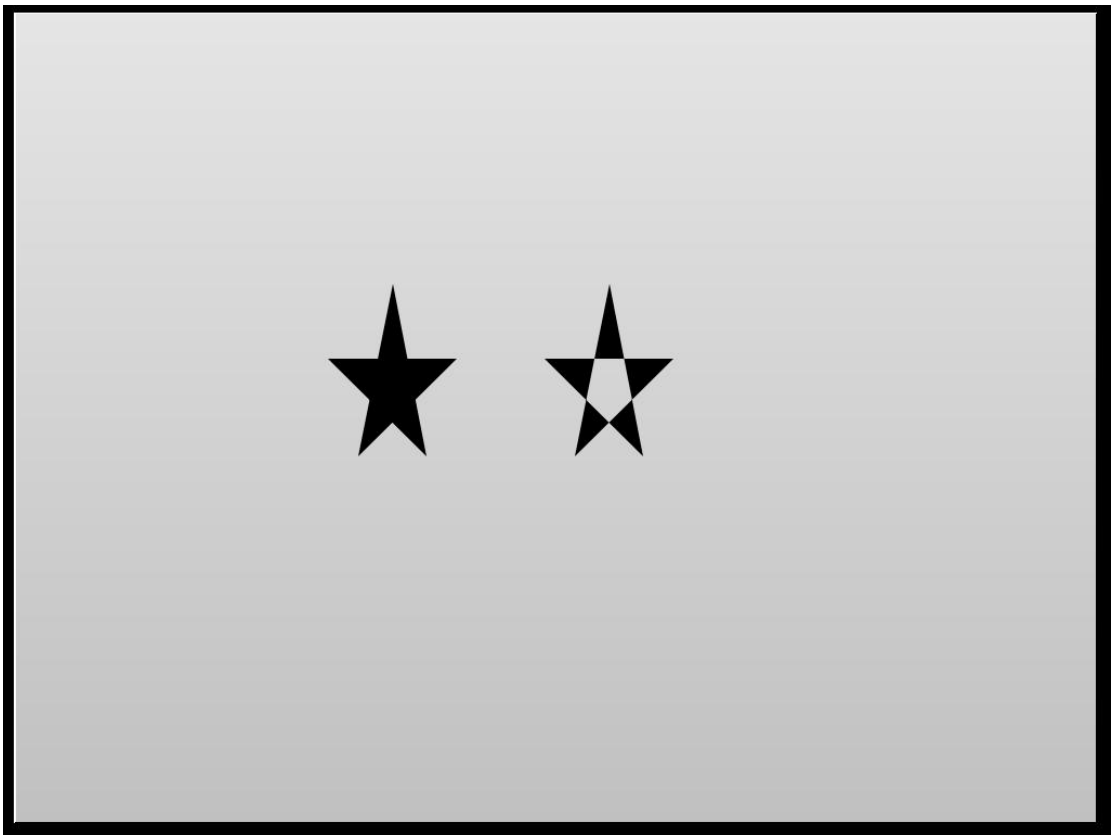
■ 範例_繪製多邊形:

```
class Paint < FpUserControl
  def paint()
    points_1 = [[350.0, 250.0],    # 頂點 1（最上方）
      [381.23, 409.1],   # 頂點 3
      [290.49, 319.1],   # 頂點 5
      [409.51, 319.1],   # 頂點 2
      [318.77, 409.1],   # 頂點 4
      [350.0, 250.0]     # 回到頂點 1，封閉路徑
      ]
      points_2 = [[550.0, 250.0],    # 頂點 1（最上方）
      [581.23, 409.1],   # 頂點 3
      [490.49, 319.1],   # 頂點 5
      [609.51, 319.1],   # 頂點 2
      [518.77, 409.1],   # 頂點 4
      [550.0, 250.0]     # 回到頂點 1，封閉路徑
      ]
    fill_polygon(points_1, fpcolor(0, 0, 0),true) #winding mode
    fill_polygon(points_2, fpcolor(0, 0, 0),false) #alternate mode
  end
end
```

```
Test = Paint.new()
Test.paint()
```

程式碼十三、**fill_polygon**()範例程式

■ 程式執行結果，左為 winding mode, 右為 alternate mode:



圖三十五、**fill_polygon**()範例執行結果

◆ **draw_ image(sx, sy, ex, ey, fname, fit)**

◆ 用途: 顯示圖片

◆ 引數說明:

引數	型態	值	說明
1	int	sx	左上角的 x 座標值
2	int	sy	左上角的 y 座標值
3	int	ex	右下角的 x 座標值
4	int	ey	右下角的 y 座標值
5	string	fname	圖檔路徑
6	Boolean	fit	是否貼合長方形

■ 回傳值型態: Boolean

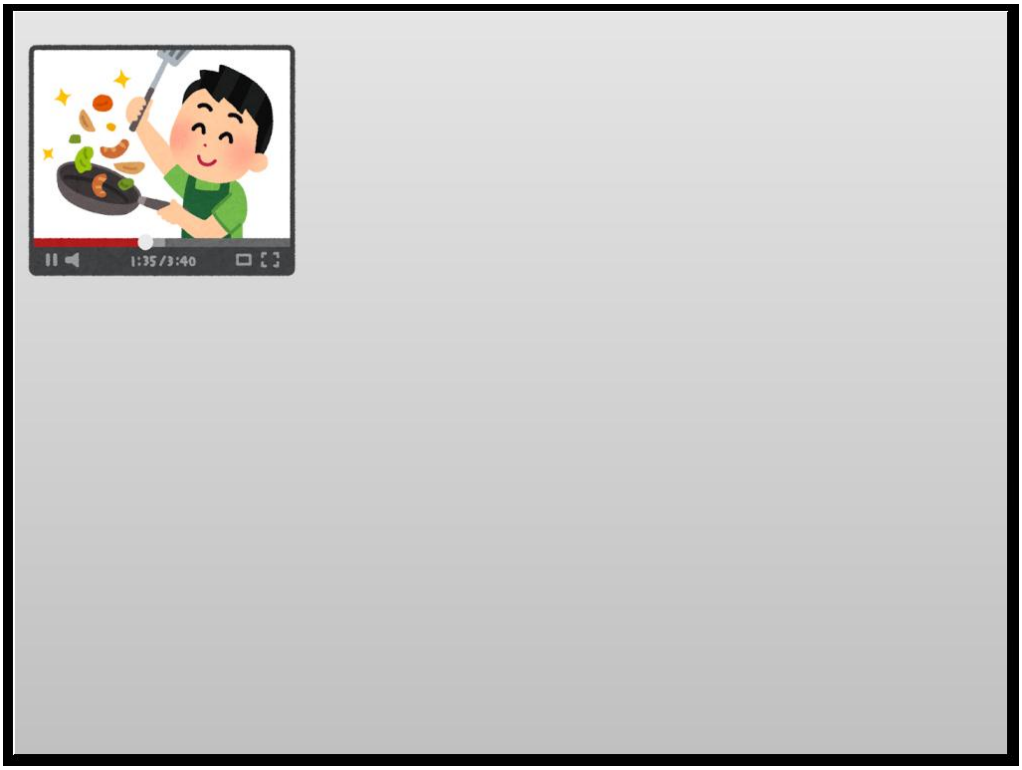
■ 範例_顯示圖片:

```
class Paint < FpUserControl
  def paint()
    image = "C:\\Users\\huang.weining\\Pictures\\video_cooking_man.png"
    draw_image(10, 20, 290, 280, image, true)
  end
end

Test = Paint.new()
Test.paint()
```

程式碼十四、**draw_ image()**範例程

※此範例的圖檔路徑請更換為您實際儲存圖片的路徑與名稱

■ 程式執行結果:



圖三十六、**draw_image()**範例執行結果