43.JS 物件和 JS 庫

本章介紹 JS 物件和 JS 庫的語法、編輯、及使用方法。

43.1. 概要

在實際 HMI 的應用中，有些需求無法透過 EasyBuilder Pro 內建的功能達成時，可以考慮使用 JS 物件。透過撰寫 JavaScript 程式，自由地控制物件行為及外觀，達成特定目標。

43.1.1. 使用建議

應儘可能使用 EasyBuilder Pro 其它內建功能或物件來達成需求，除非需求難以 (或完全無法) 透過 EasyBuilder Pro 內建功能來完成時，才考慮使用 JS 物件。

43.1.2. 環境規格

- EasyBuilder Pro V6.05.01 及之後的版本。
- 支援的 JavaScript 版本：ECMAScript 2017 (SharedArrayBuffer 和 Atomics 除外)。
- 支援的 HMI 型號：cMT X 系列。
- 32-bit Android 裝置不支援 JS Object。

43.1.3. 警告

JS 物件提供了強大的物件定制能力，但錯誤的使用方式可能造成系統行為異常或效能降低，請小心使用。

43.1.4. JS 物件 SDK

JS 物件支援的 SDK 請參考以下連結：
https://dl.weintek.com/public/Document/JS_Object_SDK/Current/index.html

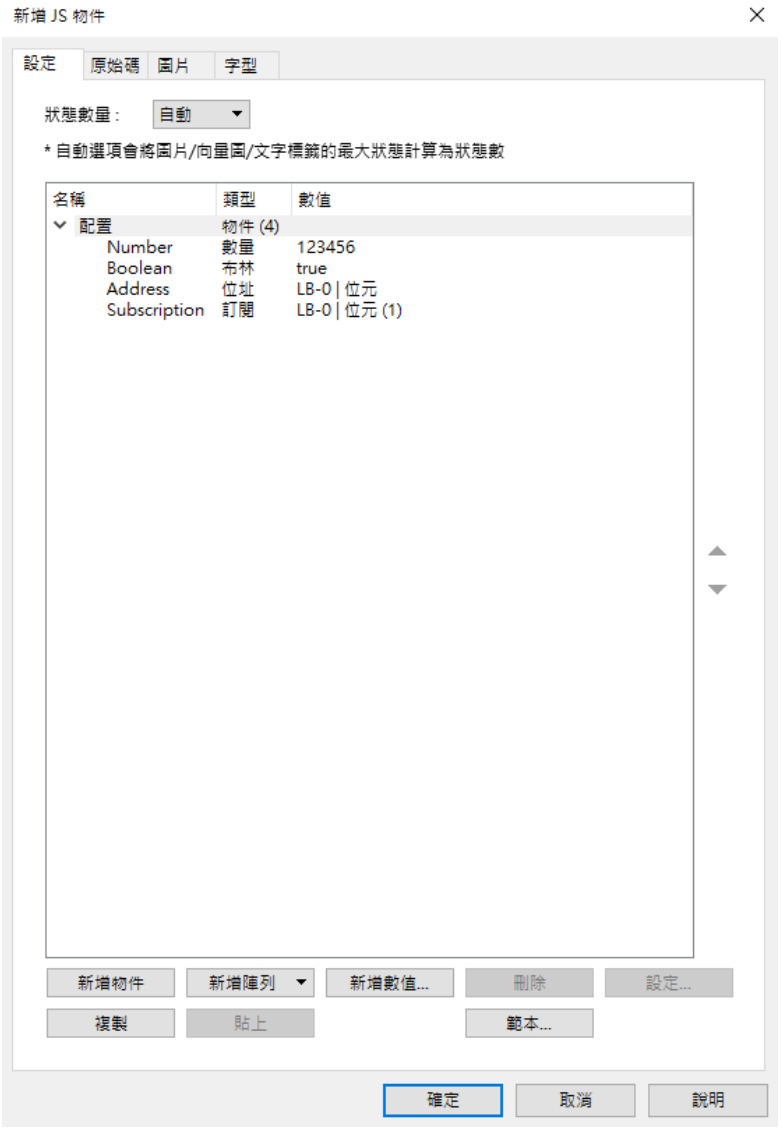
43.2. JS 物件

設定

JS 物件的**屬性配置**，包含下列幾種資料：

1. Address
2. Subscription
3. 非固定，且希望由 JS 物件的使用者，依使用情境決定其內容，進而影響 JS 物件執行期 (runtime) 行為的參數。

屬性配置在執行期會經由一個 JavaScript 物件 – **'config'**，注入到 JS 物件身上 ⇨ **this.config**。



設定	描述
新增物件	新增一個 JavaScript 物件。物件中可再建立多個屬性，每個

	屬性各自有名稱及值。
新增陣列	新增一個資料陣列。
新增數值	新增一個屬性，可以是下列型別： **String** 長度最大可支援 1000 個字元。 **Number** 可支援 64 位元浮點數的範圍。 **Boolean** True 或是 False。 **Address** 設備地址。 **Subscription** 地址訂閱。用於監測設備地址的資料變化，當資料改變時，系統會透過在 Subscription.onResponse 方法中註冊的回調 (callback) 函式通知。
刪除	刪除選定的欄位。
設定	修改選定的欄位。
複製	複製選定的欄位。
貼上	在選擇的欄位貼上之前複製的內容。
範本	將目前屬性配置以 JSON 格式呈現。修改 JSON 格式並在按下 [確認] 後， JSON 內容將轉換回屬性配置。 熟悉 JSON 的用戶可以此方式縮短屬性配置的設定時間。

原始碼

[原始碼] 為 JavaScript 程式，其內容決定了此 JS 物件的執行期**行為**。

形式(Type)	描述
Compile	編譯程式碼，可確認程式碼內容是否有效。
	使用彈出視窗編寫程式碼。
JS Object SDK	開啟 JS Object SDK Documentation 連結，用於協助開發人員創建及編寫 JS 物件。

43.3. JS Resource

當在 JS 物件中需要使用外部 JS 模塊時，必須先將 JS 模塊加到 EBpro 專案的 JS Resource 中，然後才能在 JS 物件的 [原始碼] 中引入該模塊。

下圖為 JS Resource 對話窗，以下逐一解釋：

設定	描述
開啟包含資料夾	當開啟專案時，EBpro 會在電腦系統中暫時建立一資料夾，並將 JS Resource 內容匯出到此資料夾。當儲存專案時，EBpro 也會反向地將資料夾的內容同步回專案中。
新資料夾	建立新資料夾。
新增資料夾	匯入一整個資料夾。
新增檔案	匯入一個檔案。
刪除	刪除資料夾或檔案。
重新命名	修改資料夾或 JS Resource 名稱。
加密	可以將已匯入的.js 檔案加密成.js.enc 檔案，直接開啟加密後的檔案會顯示為亂碼。
解密	將.js.enc 檔案以加密時所設置的密碼解密。
複製路徑	複製選中的檔案或資料夾路徑到剪貼簿，方便在 JS 物件的 [原始碼] 中使用。

Note

- JS Resource 內容上限為：10 MB。
- 包含資料夾在開啟專案時自動產生，在關閉專案時自動刪除。
- 包含資料夾名稱在每次開啟專案時以亂數產生，非固定。
- 包含資料夾的內容變化會讓專案變為 [已修改] 狀態，但直到用戶執行儲存專案時，此資料夾的內容才真正被同步回專案中。
- 加密的密碼長度上限為：32 Bytes。

43.4. 範例講解

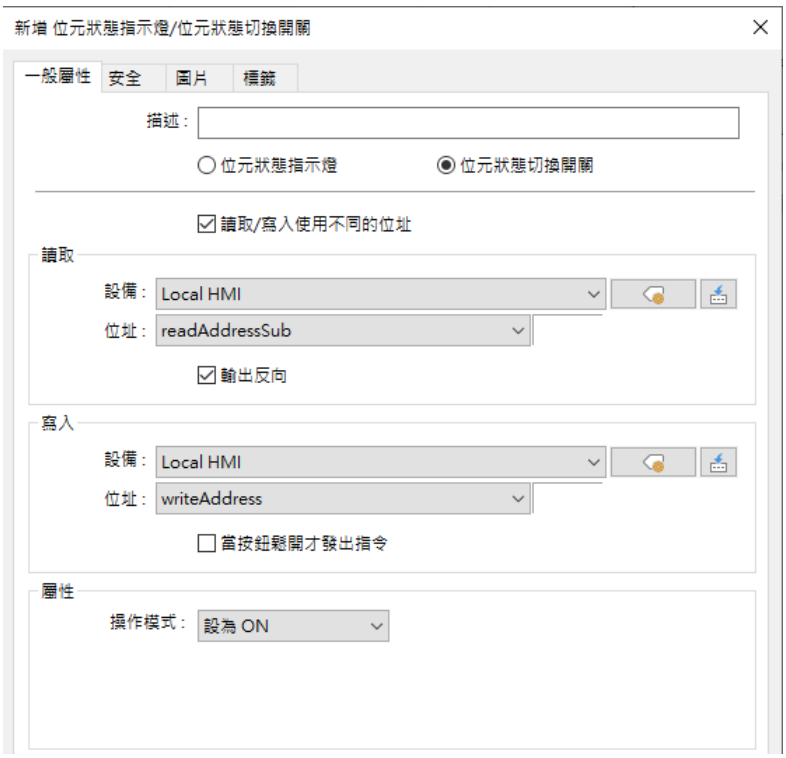
此章節的目的為透過一系列的教學範例，讓使用者更熟悉如何使用 JS 物件實作。本章節程式碼所使用的函式及類別可參考 JS Object SDK。

43.4.1. 位元切換開關

此章節的目的為學習如何使用 JS 物件逐步實作出 [位元切換開關] 物件的功能與行為，設計流程如下：

1. 設計出可讀取指定位址及輸出反向的 JS 物件
2. 加上另可將指定位址設為 ON 的 JS 物件
3. 加上另可將指定位址設為 OFF 的 JS 物件
4. 加上另可將指定位址做狀態反向的 JS 物件
5. 加上另可將指定位址做復歸型的 JS 物件

為了設計出該物件的功能，在 JS 物件中建立的對應的屬性的名稱標示如下圖。



43.4.1.1. 設計出可讀取指定位址及輸出反向的 JS 物件

本節會說明如何設計出可讀取指定位址及輸出反向的 JS 物件。為了達到 [可讀取指定位址及輸出反向] 的功能，JS 物件須建立 2 個屬性：*readAddressSub* 和 *invertSignal*。

設定

屬性	描述
readAddressSub	讀取指定位元位址的狀態。類型選擇 Subscription，Value Type 選擇 Bit，訂閱地址選擇 LB-100。 
invertSignal	設定是否輸出反向。Type 選擇 Boolean，Value 設定為 False (不進行輸出反向)。 

原始碼

本節將逐行解釋各程式碼。

```
1
2  this.config.readAddressSub.onResponse((err, data) => {
3      if (err) {
4          console.log('[response] error =', err.message);
5      } else if (this.config.invertSignal) {
6          this.state = (data.values[0]) ? 0 : 1;
7      } else {
8          this.state = (data.values[0]) ? 1 : 0;
9      }
10 });
11
```

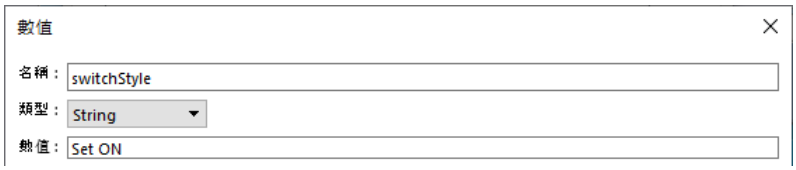
- Line2：***'this'*** 即代表 JS 物件本身。透過 ***'this.config'*** <object> 取得在 [Config] 設定頁中所加入的值(***'readAddressSub'*** 和 ***'invertSignal'***)。
- Line2：呼叫***'this.config.readAddressSub'***<Subscription> 的 'onResponse' 函式註冊 response callback，讓 [讀取位址] 在數據有變化時主動通知。
- Line 3 ~ 9：根據 response callback 內容進行後續動作。
- Line 3 ~ 4：判斷是否有錯誤。若 err 有值，代表讀取 [讀取位址] 發生錯誤，並簡單列印出錯誤訊息。
- Line 5 ~ 8：若 err 為空值，表示讀取 [讀取位址] 成功且資料有改變。根據讀取到的資料及 ***'invertSignal'*** 屬性設定 JS 物件狀態。

讀取位址數據	輸出反向	JS 物件狀態
0	False	0
1	False	1
0	True	1
1	True	0

43.4.1.2. 設計位元狀態切換開關設定為 ON 模式功能

承前一節範例，本節會再加上可將指定位址 [設為 ON] 模式的功能。如同位元切換開關使用設為 ON 模式功能。為了達到上述功能，JS 物件須再建立 2 個屬性：*writeAddress* 和 *switchStyle*。

設定

屬性	描述
readAddressSub	讀取指定位元位址的狀態。
invertSignal	是否輸出反向。
writeAddress	設定被寫入的位址。當使用者按下物件時，會根據 ***'switchStyle'*** 中的設定將數值寫入至此位址。
switchStyle	設定開關的模式，數值設定為 [Set ON]。 

原始碼

```
1
2  this.config.readAddressSub.onResponse((err, data) => {
3      if (err) {
4          console.log('[response] error =', err.message);
5      } else if (this.config.invertSignal) {
6          this.state = (data.values[0]) ? 0 : 1;
7      } else {
8          this.state = (data.values[0]) ? 1 : 0;
9      }
10 });
11
12 var mouseArea = new MouseArea();
13 this.widget.add(mouseArea);
14
15 mouseArea.on('mousedown', (mouseEvent) => {
16     if (this.config.switchStyle === 'Set ON') {
17         driver.setData(this.config.writeAddress, 1);
18     }
19 });
20
```

透過使用 MouseArea 物件，接收手指按下/或釋放的事件，以進而後續的寫入動作。

- Line 1 ~ 11：參考 CH43.4.1.1。
- Line 12：建立一個名為***'mouseArea'*** 的 MouseArea 物件。
- Line 13：***'this.widget'***<Container>代表 JS 物件的 Graphical widget，負責畫面的呈現及與使用者的互動。
- Line 13：將***'mouseArea'***加入 JS 物件。
- Line 15：向***'mouseArea'***註冊***'mousedown'***事件的監聽者。當使用者手指按下時，系統會呼叫

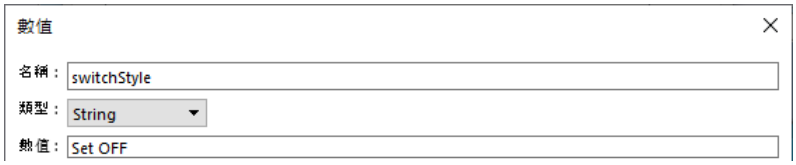
此監聽者，並執行 MouseEvent。

- Line 15 ~ 19：***'mousedown'*** 事件的監聽者。
- Line 16 ~ 18：若 [switchStyle] 的值為 'Set ON'，使用 **driver** module 的 setData Method 將數值 1 寫入到 [writeAddress]。

43.4.1.3. 設計位元切換開關設為 OFF 模式功能

承前一節範例，本節會改為可將指定位址 [設為 OFF] 模式的功能。如同位元切換開關使用設為 OFF 模式功能。

設定

屬性	描述
readAddressSub	讀取指定位元位址的狀態。
invertSignal	是否輸出反向。
writeAddress	設定被寫入的位址。當使用者按下物件時，會根據 ***'switchStyle'*** 中的設定將數值寫入至此位址。
switchStyle	設定開關的模式，數值設定為 [Set OFF]。 

原始碼

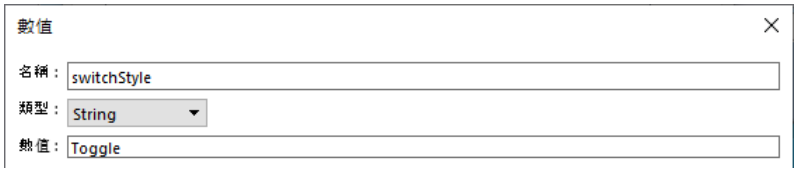
```
1
2  this.config.readAddressSub.onResponse((err, data) => {
3      if (err) {
4          console.log('[response] error =', err.message);
5      } else if (this.config.invertSignal) {
6          this.state = (data.values[0]) ? 0 : 1;
7      } else {
8          this.state = (data.values[0]) ? 1 : 0;
9      }
10 });
11
12 var mouseArea = new MouseArea();
13 this.widget.add(mouseArea);
14
15 mouseArea.on('mousedown', (mouseEvent) => {
16     if (this.config.switchStyle === 'Set ON') {
17         driver.setData(this.config.writeAddress, 1);
18     } else if (this.config.switchStyle === 'Set OFF') {
19         driver.setData(this.config.writeAddress, 0);
20     }
21 });
22
```

- Line 1 ~ 17：參考 CH43.4.1.2。
- Line 18 ~ 19：若 [switchStyle] 的值為 'Set OFF'，寫入 0 到 [writeAddress]。

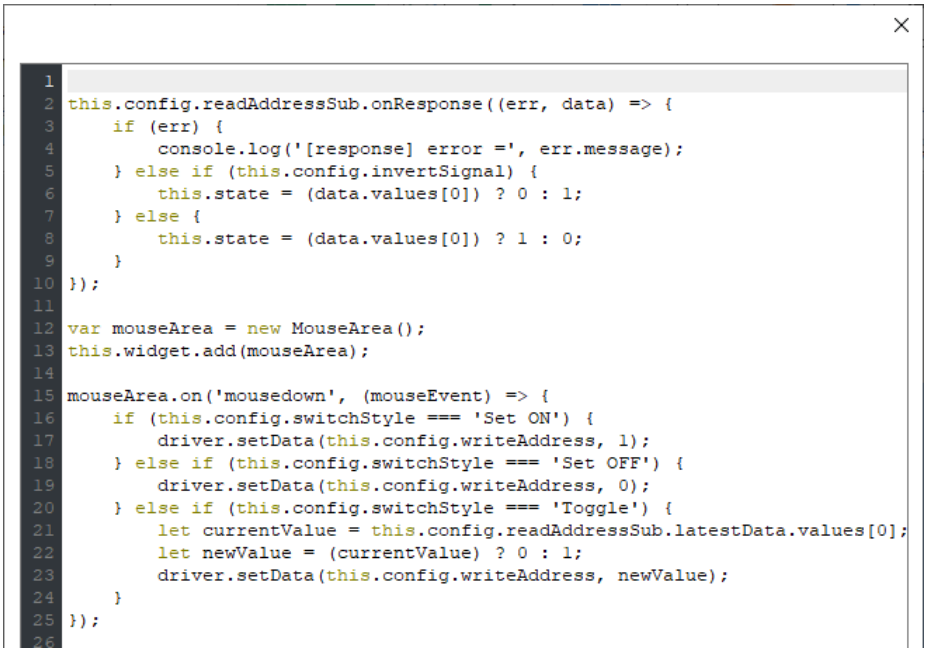
43.4.1.4. 設計位元切換開關設為切換開關模式功能

承前一節範例，本節會改為可將指定位址做狀態反向的功能。如同位元切換開關使用設為切換開關模式功能。

設定

屬性	描述
readAddressSub	讀取指定位元位址的狀態。
invertSignal	是否輸出反向。
writeAddress	設定被寫入的位址。當使用者按下物件時，會根據 ***'readAddressSub'*** 的狀態和 ***'switchStyle'*** 中的設定將數值寫入至此位址。
switchStyle	設定開關的模式，數值設定為 [Toggle]。 

原始碼



```
this.config.readAddressSub.onResponse((err, data) => {
    if (err) {
        console.log('[response] error =', err.message);
    } else if (this.config.invertSignal) {
        this.state = (data.values[0]) ? 0 : 1;
    } else {
        this.state = (data.values[0]) ? 1 : 0;
    }
});

var mouseArea = new MouseArea();
this.widget.add(mouseArea);

mouseArea.on('mousedown', (mouseEvent) => {
    if (this.config.switchStyle === 'Set ON') {
        driver.setData(this.config.writeAddress, 1);
    } else if (this.config.switchStyle === 'Set OFF') {
        driver.setData(this.config.writeAddress, 0);
    } else if (this.config.switchStyle === 'Toggle') {
        let currentValue = this.config.readAddressSub.latestData.values[0];
        let newValue = (currentValue) ? 0 : 1;
        driver.setData(this.config.writeAddress, newValue);
    }
});
```

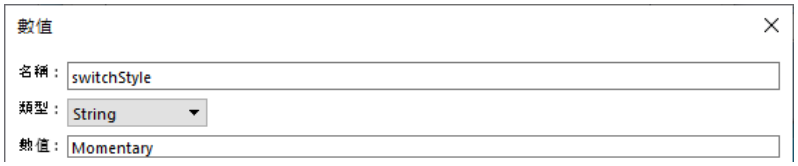
- Line 1 ~ 19：參考 CH43.4.1.3。
- Line 20 ~ 23：若 [switchStyle] 的值為 'Toggle'，將 [writeAddress] 的狀態反相。

43.4.1.5. 設計位元切換開關復歸型模式功能

承前一節範例，本節會改為可將指定位址使用 [復歸型] 模式的功能。如同位元切換開關使用復歸型模式功能。

設定

屬性	描述
readAddressSub	讀取指定位元位址的狀態。
invertSignal	是否輸出反向。
writeAddress	設定被寫入的位址。當使用者按下物件時，會將數值 1 寫入此位址；當使用者手放開物件時，會將 0 寫入此位址。
switchStyle	設定開關的模式，數值設定為 [Momentary]。



原始碼

```
1
2  this.config.readAddressSub.onResponse((err, data) => {
3      if (err) {
4          console.log('[response] error =', err.message);
5      } else if (this.config.invertSignal) {
6          this.state = (data.values[0]) ? 0 : 1;
7      } else {
8          this.state = (data.values[0]) ? 1 : 0;
9      }
10 });
11
12 var mouseArea = new MouseArea();
13 this.widget.add(mouseArea);
14
15 mouseArea.on('mousedown', (mouseEvent) => {
16     if (this.config.switchStyle === 'Set ON') {
17         driver.setData(this.config.writeAddress, 1);
18     } else if (this.config.switchStyle === 'Set OFF') {
19         driver.setData(this.config.writeAddress, 0);
20     } else if (this.config.switchStyle === 'Toggle') {
21         let currentValue = this.config.readAddressSub.latestData.values[0];
22         let newValue = (currentValue) ? 0 : 1;
23         driver.setData(this.config.writeAddress, newValue);
24     } else if (this.config.switchStyle === 'Momentary') {
25         driver.setData(this.config.writeAddress, 1);
26     }
27 });
28
29 mouseArea.on('mouseup', (mouseEvent) => {
30     if (this.config.switchStyle === 'Momentary') {
31         driver.setData(this.config.writeAddress, 0);
32     }
33 });
34
```

- Line 1 ~ 23：參考 43.4.1.4。
- Line 24 ~ 25：若 [switchStyle] 的值為 'Momentary'，寫入 1 到 [寫入位址]。
- Line 29 ~ 33：當使用者手指離開物件時，寫入 0 到 [writeAddress]。

43.4.2. JS Resource 使用範例

此章節的目的為學習 JS 物件如何匯入在 JS Resource 中的 JS 模塊。
JS 物件目前支援以下兩種引入模塊方式：

1. ES6 動態匯入 (dynamic import)：透過 import() 函式。注意：JS 物件不支援 ES6 靜態匯入 (static import)。
2. CommonJS：透過 require() 函式。

43.4.2.1. 使用 ES6 動態匯入

此範例示範如何使用 ES6 動態匯入 JS 模塊。

在此 JS 物件中，當圓半徑改變時，將使用外部 JS 模塊 ***(circle.js)*** 裡的 ***area*** 及 ***circumference*** 函式，計算圓的面積及周長，並將結果輸出到指定地址。

circle.js 程式碼如下：

```
// circle.js

export function area(radius) {
  return Math.PI * radius * radius;
}

export function circumference(radius) {
  return 2 * Math.PI * radius;
}
```

1. 建立 JS 物件的屬性如下圖。

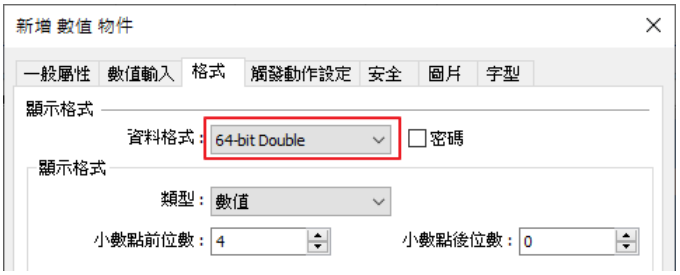
屬性	描述
radiusSub	用於取得當前圓半徑的值及觀察其變化。(選擇 LW-100，64-bit Double 型別)
areaAddress	用於輸出圓面積的計算結果。(選擇 LW-110，64-bit Double 型別)
circumferenceAddress	用於輸出圓周長的計算結果。(選擇 LW-120，64-bit Double 型別)

2. 在 [Source code] 設定頁編寫以下程式碼：

```
1  const circle = await import('/circle.js');
2
3  this.config.radiusSub.onResponse((err, data) => {
4      if (err) {
5          console.log('[response] error =', err.message);
6      } else {
7          let radius = data.values[0];
8          let area = circle.area(radius);
9          let circumference = circle.circumference(radius);
10         driver.setData(this.config.areaAddress, area);
11         driver.setData(this.config.circumferenceAddress, circumference);
12     }
13 });
```

- Line 1：匯入 JS Resource 中的 '/circle.js' 檔案，並將結果綁定到 ***circle*** 上。
- Line 3：呼叫***'this.config.radiusSub'*** 的 'onResponse' 函式註冊 response callback，讓 [circle radius] 在變化時通知 JS 物件。
- Line 8：呼叫 circle.js 的 area 方法計算圓面積，並將結果存在 area 變數。
- Line 9：呼叫 circle.js 的 circumference 方法計算圓周長，並將結果存在 circumference 變數
- Line 10 ~ 11：將圓面積與周長計算結果分別輸出到 areaAddress 與 circumferenceAddress 地址。

3. 在 Project 中建立三個 64 bit Double 數值物件，分別為 Radius (LW-100)、Area (LW-110)、circumference(LW-120)。



4. 輸入數值到 Radius (LW-100)，Area (LW-110) 和 circumference(LW-120) 會自動計算出數值。

Note

■ 若要匯入 JS 模塊的默認輸出 (/default export)，可以使用以下方式：

```
1  var myModule = (await import('my-module.js')).default;
2
```

43.4.2.2. 使用 CommonJS 匯入

此範例示範如何使用 require() 函式匯入 JS 模塊。

此 JS 物件行為與前例相同，但使用 CommonJS 方式匯入外部 JS 模塊。

circle.js 程式碼如下：

```
// circle.js

exports.area = function (radius) {
  return Math.PI * radius * radius;
};

exports.circumference = function (radius) {
  return 2 * Math.PI * radius;
}
```

1. JS 物件屬性與前例相同。
2. 在 [原始碼] 設定頁編寫以下程式碼：

```
1  const circle = require('/circle.js');
2
3  this.config.radiusSub.onResponse((err, data) => {
4      if (err) {
5          console.log('[response] error =', err.message);
6      } else {
7          let radius = data.values[0];
8          let area = circle.area(radius);
9          let circumference = circle.circumference(radius);
10         driver.setData(this.config.areaAddress, area);
11         driver.setData(this.config.circumferenceAddress, circumference);
12     }
13 });
14
```

- Line 1：匯入 JS Resource 中的 '/circle.js' 檔案，並將結果綁定到 ***circle*** 上。
- Line 3 ~ 13：參考 CH43.4.2.1。

3. 在 Project 中建立三個 64 bit Double 數值物件，分別為 Radius (LW-100)、Area (LW-110)、circumference(LW-120)。
4. 輸入數值到 Radius (LW-100)，Area (LW-110) 和 circumference(LW-120) 會自動計算出數值。

43.5. 規格限制

1. JS 物件原始碼長度限制：100KB。
2. JS Resource 總計檔案大小限制：10MB。
3. JS Context (*) 的記憶體用量限制：20MB。

(*) JS context 為 Javascript 執行環境，有一些內建物件與函數。在人機上，同一個視窗的所有 JS 物件會共用同一個 JS context，因此也會共用 memory heap 及 global object。

43.6. JS 物件的運作流程

1. 建立 JS Context。
2. 建立 JS 物件。
3. 為 JS 物件初始化 'config' (=> this.config)。
4. 為 JS 物件初始化 'widget' (=>this.widget)。
5. 等待所有 subscriptions 的回應。
6. 將 JS 物件的 [原始碼] 包裝成一個非同步函數並呼叫它。