DataHub WebView Scripting

DataHub WebView 是一个 web-based 的数据视觉工具，具有后端的数据传输平台和一个能够透过设计动画来显示 DataHub 的数据，并检视这些资料的 Browser-based 编辑器，无论从任何地方，利用因特网或企业网络，以及标准的 Web 浏览器就可以使用此编辑器。它可以与其他 Cogent DataHub 产品的网络功能链接，并进行实时的数据交换。

DataHub WebView1.4 的释出包含了一个新的综合性的 Script 功能。此功能是由 S#(Script.NET)，具有揭露和扩展应用程序的模式，并且能够与内部的.NET 交换对象、类型和组件互动。

DataHub 的 WebView 内有几个可以被宏化的领域。概括地说明，它们为：

Script Bindings	能够链接至 Script 结果的任何属性的控制组件。
Dynamic Point Bindings	这些是由 Script 表示式计算的 DataHub 点名称。
Events and Event Handlers	事件处理可以被加在三种类型的事件：控制组件、页面和应用程序。

以附加的 WebView Script，你可以利用三个不同的 Script 领域来协同 Cogent DataHub 运作：

- DataHub 中执行的 Gamma
- 浏览器中执行的 Javascript
- 在 DataHub WebView 中执行的 S#

Scripting 能力

DataHub WebView 1.4 的发行，包括一个新的综合性 Scripting 功能，能够使网页设计师和使用者在 DataHub WebView 中自行客制化动作以得到令人满意的使用者体验。

有了这个新功能，您可以：

- 在执行模式下以用户选择的结果来设定动态点链接（dynamic point bindings)。
- 以用户名称或是其他的范围数据为数据源链接，来设计模板页面(templated pages)。
- 构建多页的解决方案，可与页面资料和全局变量交互动作。
- 新增动画效果以仿真流程及增强告警通知。

范围和环境

变量的范围和 Script 环境表示式是每一个编辑程序环境的重要课题。 S#也不例外。

Variable Scope	变量的范围，有时也被称为作为一个变量的可存取性，是指其中的变量可以读取和/或写入，和变量的寿命，或它能够在内存内保留多久。对不同的程序语言/Script 语言和工具而言，变量宣告语法和范围规则是不同的。
Script Context	Script 环境是 Script 执行时，存储讯息的一部分，如：范围、范围内的变量、旗标、函数等。在 DataHub WebView 中的变量，环境取决于 Script 是如何被关联的。环境通常是一个：页面，控制组件或事件处理程序。

变量范围

所有 S#变量都是无类型的。他们可以接受布尔值、数字、字符串和任何其它数据类型。变量存储在 Script 环境内，这意味着它们和它们宣告的 Script 具有相同的范围，。

区域范围 – 局部变量

执行 Script 时，它执行在自己私有的变量范围。私有范围内宣告的局部变量在 Script 完成后才算有效。最好的做法是使用关键词 var 来宣告变量，但它不是绝对必要的。例如，变量 x 和 y 都是在 Script 片段中被宣告的环境内。测试功能具有它自己的环境，宣告的变量 z 被限制在函数的范围。

```
x = 5;
var y = 20;
var result = x * y;
DEBUG("The result is " + result);

function test() {
  var z = 10;
}

DEBUG(z);  /* generates error: 'Identifier not found: Namespace 'z' is not found' */
```

由于函数有它们自己的环境，他们只能存取其内宣告的变量。例如，test2 的函数没有存取变量 msg 的信息。

```
var msg = "Hello World!";

function test2() {
  DEBUG(msg);  /* generates error: 'Identifier not found: Namespace 'msg' is not found' */
}
```

区域范围 - 内部对象

除了局部变量，DataHub WebView 中也有一些特殊的变量，称为内部对象，它提供方便地存取所使用的 Script 链接和事件处理程序的共享对象。例如，以下程序代码可用于控制组件事件处理程序：

```
var thisPageName = Page.PageName;
var thisElement = Element;
var thisEvent = EventName;
```

thisPageName、thisElement，还有 thisEvent 都是局部变量。Page、Element 和 EventName 是内部对象。内部对象仅在私有范围内是有效的。内部对象的可用性取决于 Script 环境。

全局范围

除了私有的范围外，所有 Script 都能够存取全局范围内的资源。全局范围是让所有 Script 和所有 DataHub WebView 页面在 WebView session 的持续时间内共享的。重设全局范围的唯一方法是重新启动 DataHub WebView。若使用在 Script 中的变量，没有明确使用 var 这个关键词宣告，则该变量属于全局范围之内。在全局范围内分配一个值给全局变量的简单动作，就可让该变量存在。

函数可以透过全局关键词来存取全局变量。

数据点

数据点被存储在一个分隔开的命名空间，且不被视为 Script 变量。也就是说，数据点不能做为内部引用。相反的，数据点可使用 VAL、GET 和 SET 函数来引用。

动态的全局变量

动态的全局变量被存储在一个分隔开的命名空间，如同数据点。他们也使用 VAL、GET 和 SET 函数来引用。

元素属性

控制组件的属性也不被视为 Script 变量。像数据点一样，他们可以使用特殊的存取函数，GETP 和 SETP 来引用。

Script 环境

Script 执行时，它有对于一些特殊变量的存取，取决于它执行的环境。这些特殊的变量被称为内部对象。此外，所有的 Script 可存取在全局范围内、动态的全局变量和数据点（通过存取函数）。环境可以是：

- DataHub WebView 页面（页面環境）
- 页面上的控制组件（控制组件環境）
- 页面上控制组件的事件处理程序（控制组件事件環境）

页面环境

网页事件处理程序执行在页面环境，可存取 Page 和 EventName 内部对象。

控制组件环境

当 script 是一个点名称表示式、script 连结，或一个事件处理程序时，script 与控制组件有着链接。当 script 被执行时，控制组件本身是可使用 Element 存取的内部对象。此对象引用承载控制组件可视化呈现的容器。

注：在本文中，术语“控制组件(control)”和“元素(element)”是同义词，交替使用。

Element 对象公开一些属性和方法可以用来操纵控制组件的属性。

所有的控制中有一个特殊的讯息存储，称为 ElementData。这是一个特殊的 expando 对象，它的成员变量是在它们被存取时动态产生的。这使 script 编写者能够简单、自然地新增新项目至 ElementData。例如，此 script 分配值至 ElementData 的 LastInput 属性：

```
ElementData.LastInput = the_value;
```

之后，ElementData.LastInput 在同一 Script 环境内所有执行的 Script 中都为有效（Script 与同一控制组件关联）。每个控制组件都有其自己的 ElementData，这是只适用于与该控制组件链接的 Script。任意数量的成员都可被加入 ElementData。

事件处理程序环境

有些事件处理程序可存取额外的内部对象。

范例

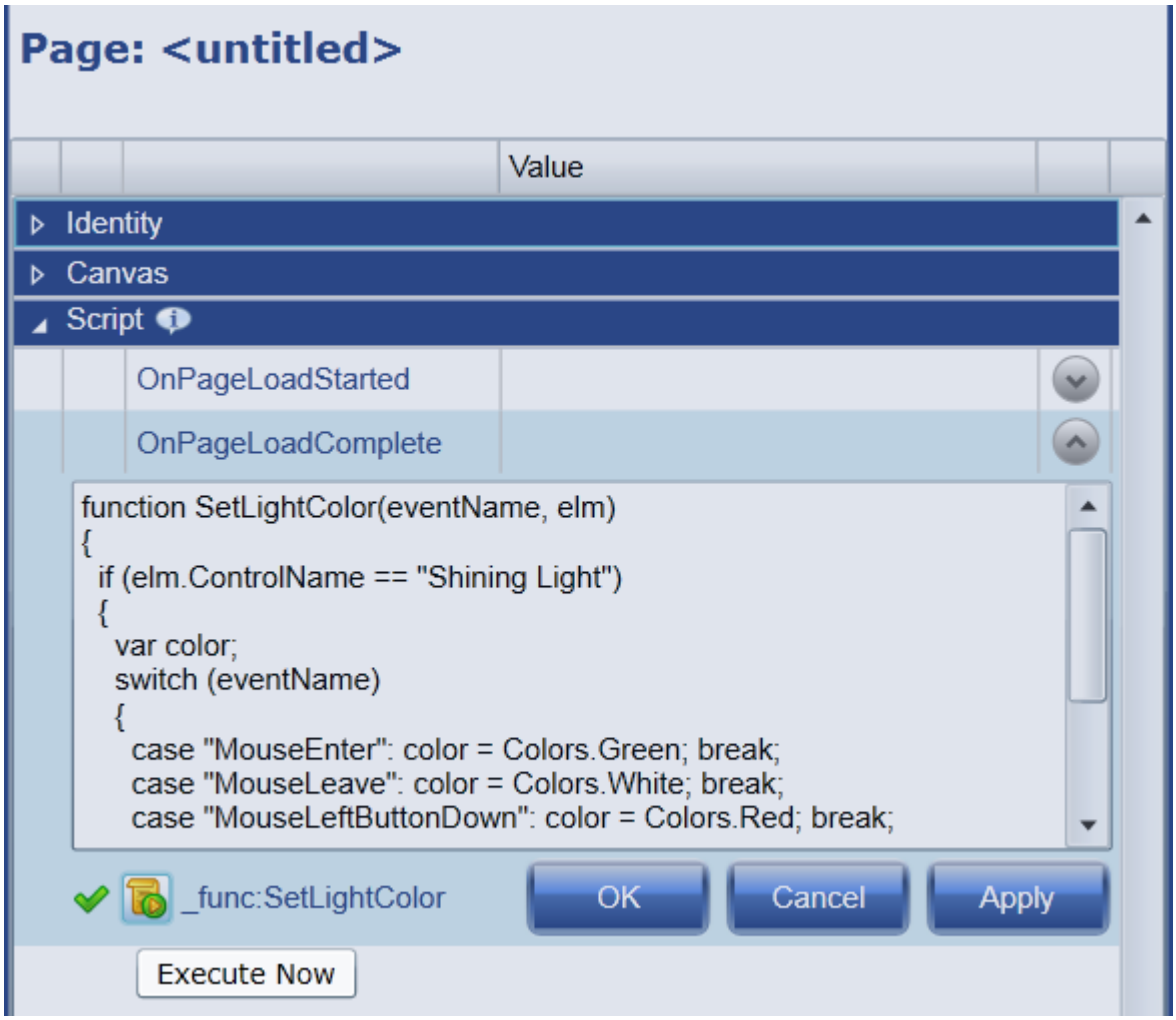
这个例子说明如何在页面加载时宣告一个 Script 函数，然后从各种控制组件事件处理程序中呼叫该函数。每个事件处理过程调用函数，传递 EventName 和引用给 Element，就是拥有事件处理程序 Script 的控制组件。此函数依据触发呼叫的事件名称来设定控制组件的颜色属性。

第 1 步：宣告函数

启动一个新的 WebView 页面，并新增下面 OnPageLoadComplete 事件处理程序码。

```
function SetLightColor(eventName, elm)
 {
   if (elm.ControlName == "Shining Light")
   {
     var color;
     switch (eventName)
     {
       case "MouseEnter": color = Colors.Green; break;
       case "MouseLeave": color = Colors.White; break;
       case "MouseLeftButtonDown": color = Colors.Red; break;
       case "MouseLeftButtonUp": color = Colors.Green; break;
     }
     SETP(elm.PageElementName + "@PrimaryLightColor", color);
   }
 }
```

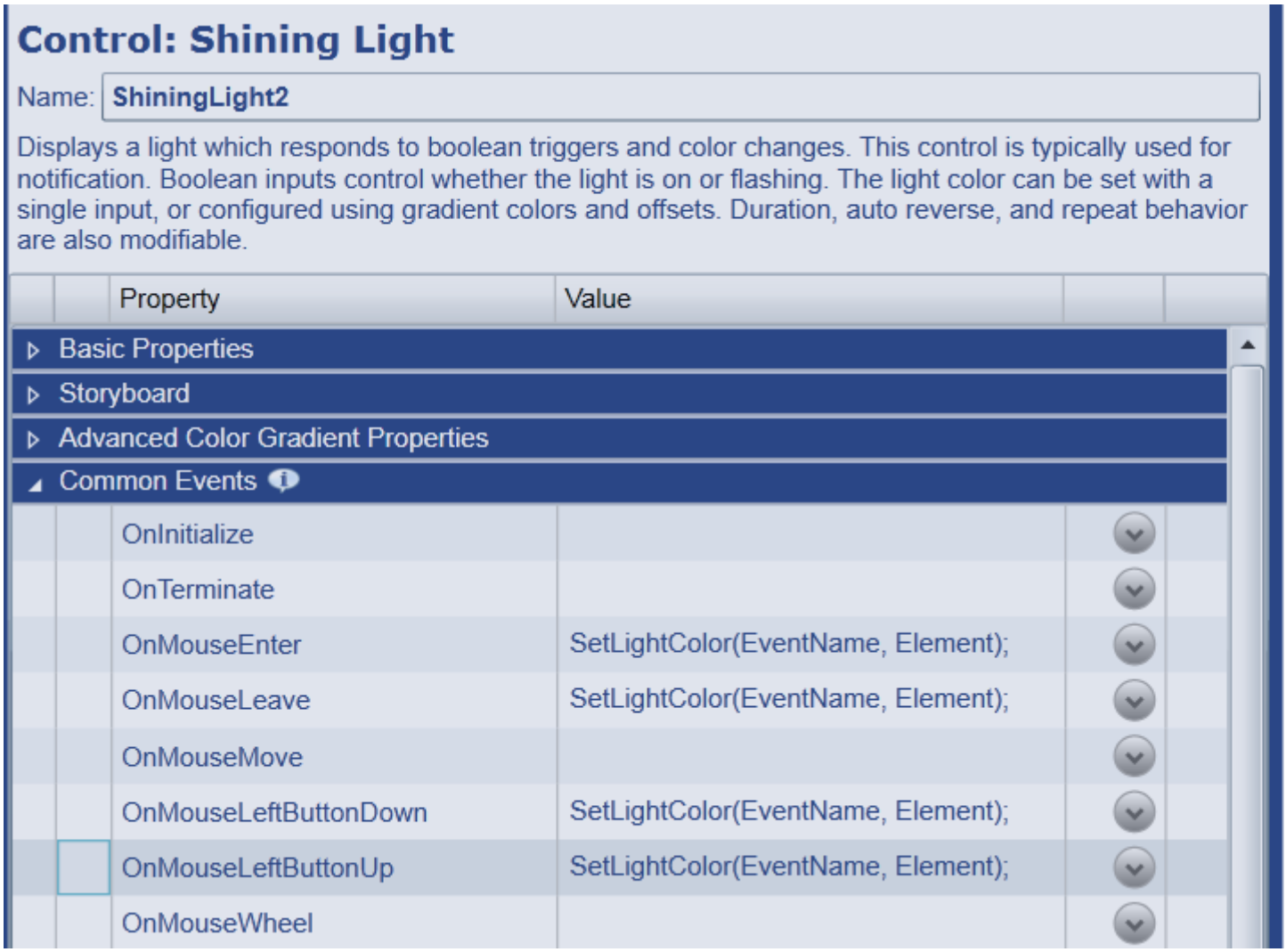
当页面加载和 Script 被执行时，SetLightColor 函数被注册。可以点击“立即执行”按钮来立即注册函数（如下图所示）。



第 2 步：新增事件处理程序

新增一个 Shining Light 控制组件实体到页面上，并新增四个鼠标事件到以下程序代码：OnMouseEnter、OnMouseLeave、OnMouseLeftButtonDown 和 OnMouseLeftButtonUp。

```
SetLightColor(EventName, Element);
```



步骤 3：执行模式下触发事件

进入执行模式。当你移动鼠标到灯号上，点击鼠标左键然后释放，灯号就会改变颜色。只有灯号接收到事件会改变颜色，因为一个 Element 的引用被传递到 SetLightColor 函数。

链接

任何控制组件的属性都可以被链接。连结类型有四种：

类型	說明
None	属性没有链接。
Point	属性链接到一个数据点。
Simple	属性链接到另一个页面上其他控制组件的另一个属性。
Script	属性链接到一个 Script 的结果。

S#的 Script 表示式可以被用来建立：

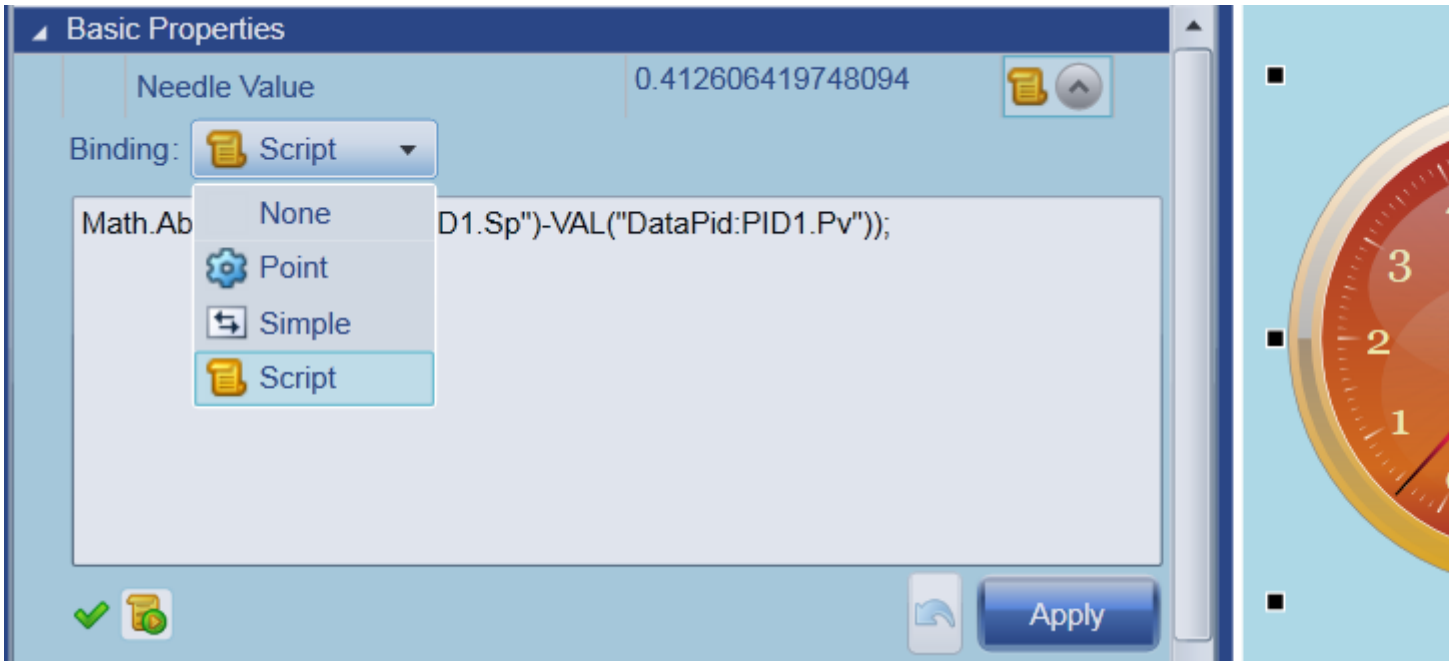
- Script Binding – 其 script 的结果为控制组件属性所用。
- Dynamic Point Binding - Script 的结果为动态设定点名称所用。

注：简单的链结不支持 Script。

Script 链接

一个 Script 连结可以包含任意数量的表示式。属性的值将会是 Script 最后一行执行的结果。

链接类型可以在设计模式中交互地选择使用属性浏览，如下图所示。



连结类型也可以在 script 中透过 IPageElementParameter.SetBindingTypemethod 方法，以程序语言编辑的方式设定。在 script 中设定连结类型时，你必须使用 BindingType 列举，如下面的程序代码。

```
var p = Page.GetParameter("myGauge@NeedleValue");
if (p != null)
{
  var expr =
"Math.Abs(VAL(\"DataPid:PID1.Sp\")-VAL(\"DataPid:PID1.Pv\"));";
  p.SetBindingExpression(BindingType.Script, expr);
  p.SetBindingType(BindingType.Script);
}
```

动态点链接

动态点链接提供了强大的能力来建立：

- 在执行模式中，让用户能够选择数据点连结的页面。
- 设备或程序的呈现类似数组的模板页面。

动态点链接让您能够建立一个单一的 DataHub WebView 页面，取决于控制组件状态、其他的 Script、使用者的输入或页面启起始参数来显示不同的数据。

点连结是以“=”表示式开始，后面接着回传数据点名称的 Script 表示式，动态地被生成。

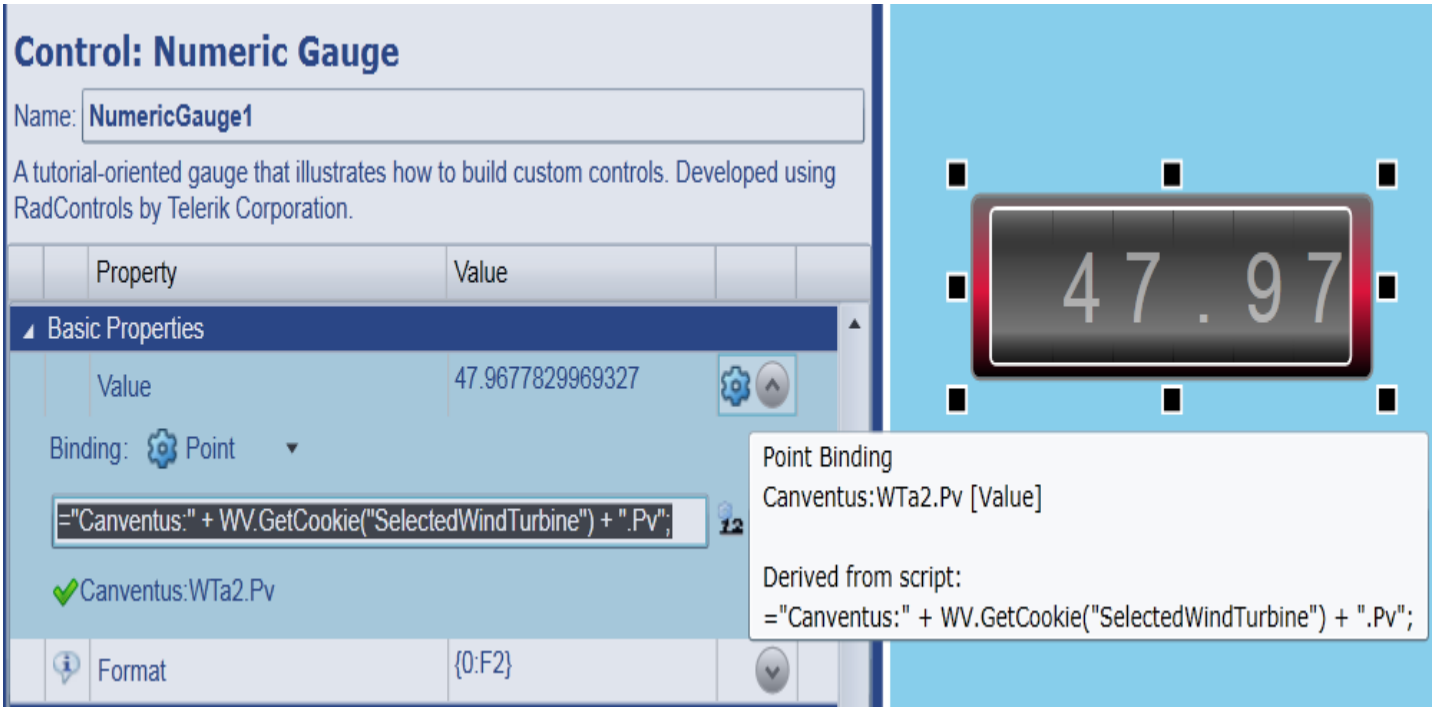
点名称表示式的结果必须是一个字符串，其中包含完全符合规定的点名称，以资料 domain 和一个冒号为起始。点名称的表示式是自动执行，所以动态全局变量、数据点和控制组件属性的引用，将在引用值变化是导致点名称被重新计算。

范例

这个例子说明了如何建立一个使用 Script 表示式来确定数据点名称的点连结，。

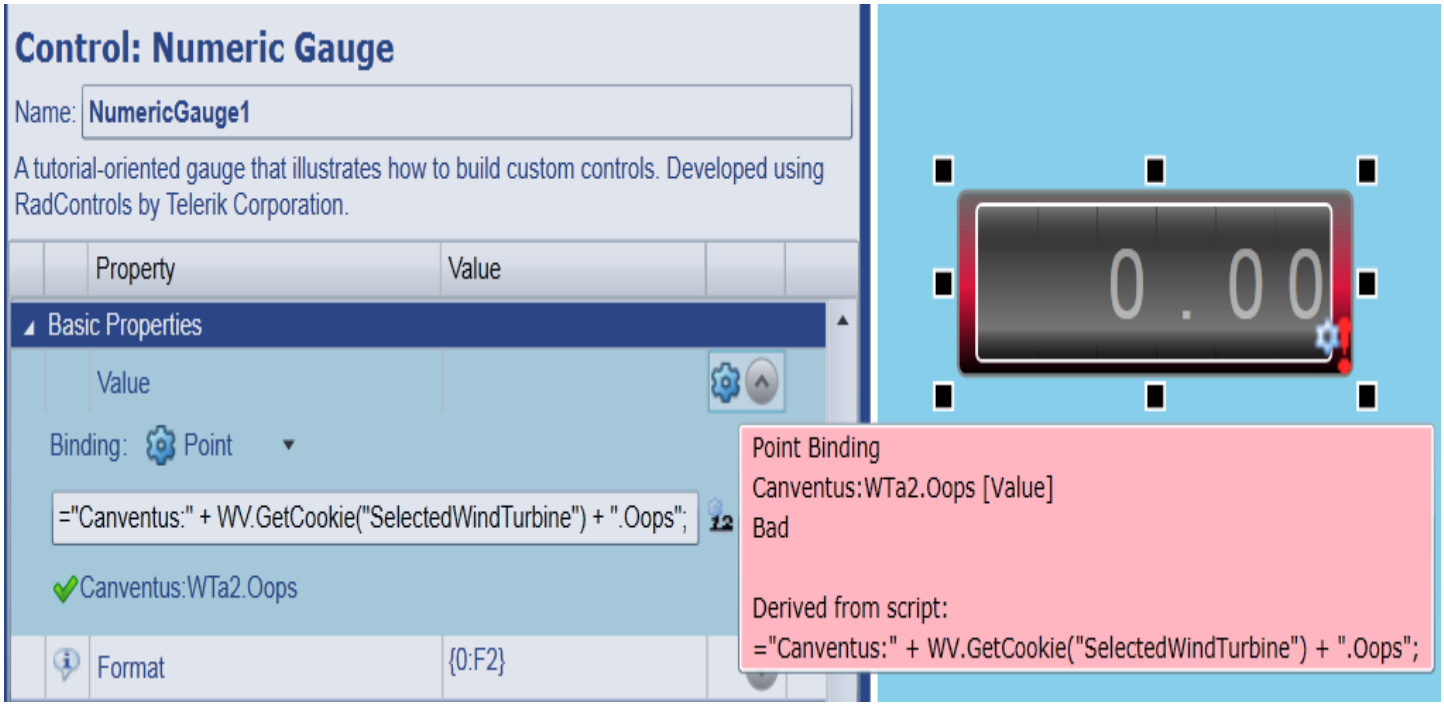
```
="Canventus:" + WV.GetCookie("SelectedWindTurbine") + ".Pv";
```

此 Scriptc 藉由链接一个表示资料 domain(“Canventus：”)的字符串，与一个 Script cookie 的值，和点名称(“.PV”)，来回传一个点名称。当前的 cookie 值是“WTa2”，所以数字压力表被链接到 Canventus：WTa2.Pv.



使用 Script Cookies 窗口可以对 script cookies 做修改，可以透过选择 Tools | Script Cookies 来显示该窗口。

如果 Script 的表示式不回传一个可识别的点名称，或者如果点的值联机质量不好，控制系统将显示一个警告图标（取决于控制组件的 BadDataAdornmentType 属性）。在设计模式中，您可以透过停留在连结类型的图标，排除故障点连结问题，如下图所示。



在运行模式选择点

动态点链接表示式可以引用全局变量、Element 属性和其他数据点。

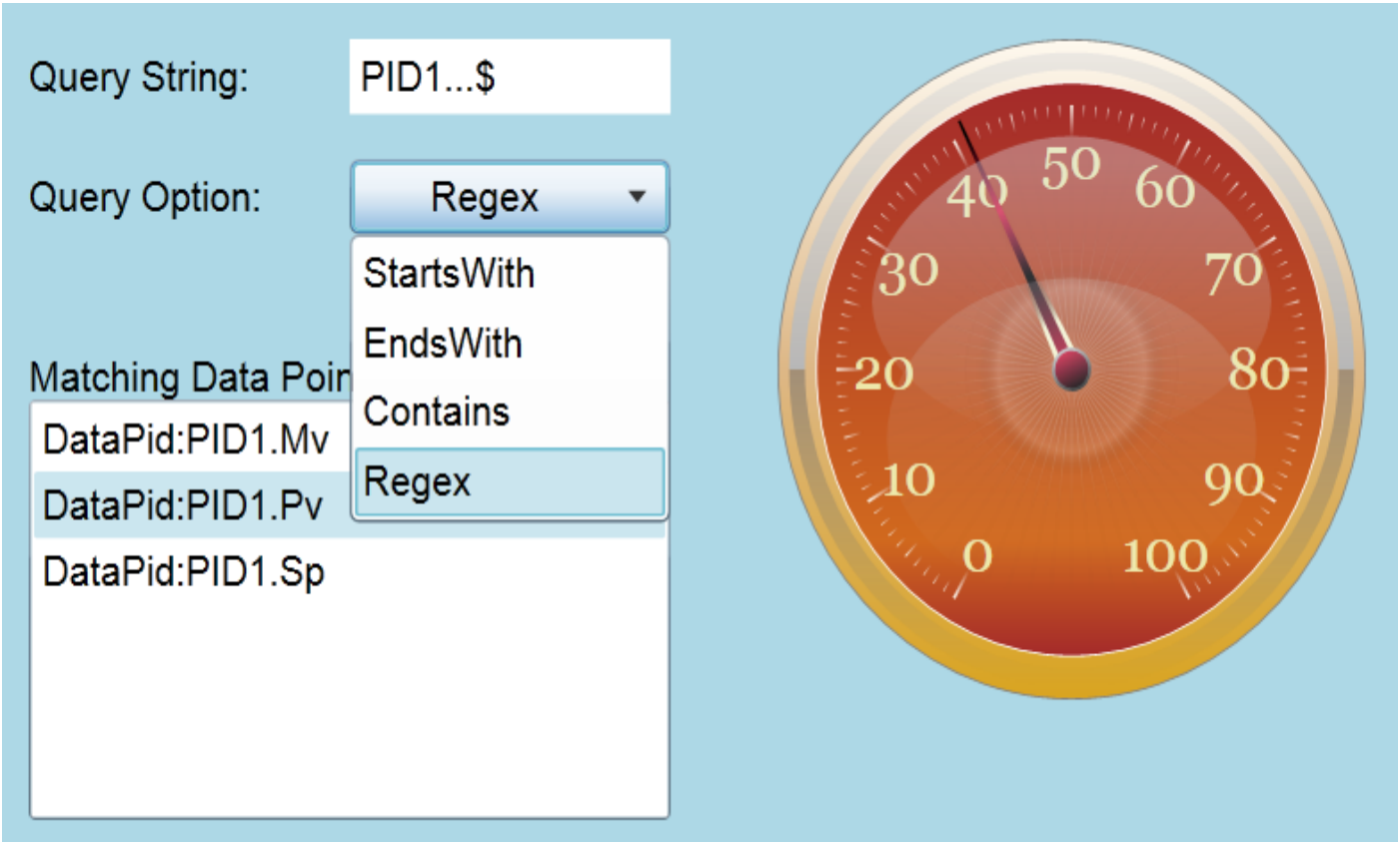
例如，你可以建立一个页面，代表 PLC 的配置盘，然后使用点名称的表示式填入配置盘作为数值。页面上的其他地方，您可以提供一个选择控制组件，来选择您的制造程序中的别组 PLC。当你选取 PLC，面板将变更数据点和它的显示，允许用户快速地在 PLC 之间做切换而不用更换页面。

范例

这个例子说明了如何建立一个数据点的选择机制，让用户选择执行模式中的数据点，并连结一个仪表。

第 1 步：页面设计

此页面包含一个 Text Entry element（txtQueryString），一个 Combo Box（cbQueryOption），一个 List Box（lstMatchingPointNames），和一个 Simple Radial Gauge（gauge）。



第 2 步：Script 链接

Combo Box 项目对象来源属性是一个 Script 链接从 PointQueryOptions 列举回传的名称列表。

```
WV.GetEnumNames(PointQueryOptions);
```

List Box 项目来源属性是一个 Script 链接使用 GETP 来取得用户输入的查询字符串和选定的查询选项，然后呼叫 GetPointNames 来回传一个符合条件的数据点清单。

```
var query = GETP("txtQueryString@Value");
```

```
var option = GETP("cbQueryOption@SelectedItem");
WV.GetPointNames(query, Enum.Parse(PointQueryOptions, option, true));
```

第 3 步：动态点链接

仪表的指针数值的属性使用了点链接，从符合条件的数据点的 List Box 中，动态地引用配对到的对象。

```
=GETP("lstMatchingPointNames@SelectedValue");
```

当用户输入一个查询字符串，并选择查询选项，符合条件的数据点将会出现在清单中。在清单中选择一个数据点，将触发仪表指针数的点数值连结表示式重计算，然后开始将所选定的数据点反映在仪表的数值上。

事件和事件处理程序

事件处理程序是当某个事件发生时所要执行的 Script。例如，当光标在一个控制组件上，用户按下鼠标按钮使事件发生的时候。

有些事件处理程序需要特定事件的讯息。例如，控制组件的 OnMouseMove 事件可能需要知道鼠标的位置。DataHub WebView 中使用的一般事件是以 Microsoft Silverlight 为基础，但使用不同的事件处理程序机制。对于大多数的事件，都是 DataHub WebView 内展现的内部对象，而不需事件处理程序来要求的函数参数。

Type 類型	Event 參數
Element Events	透过内部对象来进行存取：Element、ElementData、Me、EventName 以及 EventArgs。
Page Events	透过内部对象来进行存取：Page 以及 EventName。
Application Events	作为函数参数来传递。应用程序的事件处理程序是被编码为完整的 Script 函数（通常是.ss 檔）。Application_UserChanged 事件处理程序必须进行以用户的名称编码，以作为函数参数。内部对象则是无效的。

Element 事件

所有控制组件都支持几种常见的事件处理程序。一些控制组件还支持特定控制组件的事件处理程序（称为“客制化事件”）。

例如，所有的控制组件支持按下，然后松开鼠标左键(OnMouseLeftButtonDown)的事件。一个按钮控制组件还支持一个 OnClick 事件，而一个 checkbox，支持 OnChecked 和 OnUnchecked 的事件。

Event handlers common to all controls include:

所有控制组件常用的事件处理程序包括：

事件名稱	何時發生
OnInitialize	此 Element 被新增到页面上(在主页面被加载时也可以新增)。您可以使用这个事件新增自行定义的初始化作业。
OnTerminate	这个元素在页面上被删除（在主页面被卸除时也可以移除）。您可以使用这个事件新增自行定义的清理作业。
OnMouseEnter	鼠标指针进入此元素的边界。
OnMouseLeave	鼠标指针离开该元素的边界。
OnMouseMove	鼠标指针在这个元素上移动。
OnMouseLeftButtonDown	当鼠标光标在这个元素上，同时按下鼠标左键。
OnMouseLeftButtonUp	当鼠标光标在这个元素上，同时放开鼠标左键。
OnMouseWheel	当鼠标光标在这个元素上，同时转动鼠标滚轮。

Element 事件处理程序存取内部对象：Element、ElementData、Me、EventName 以及 EventArgs。

可以使用属性浏览器来编辑 Element 的事件处理程序。

页面事件

有两页的事件在加载页面时被启动。您可以为任何一个事件的处理程序新增 Script。

事件名稱	何時發生
OnPageLoadStarted	页面开始被加载，且立即的，在页面上的控制组件被加载之前（如，预载的 Script）。在此时，它对页面上引用的元素是不安全的。
OnPageLoadComplete	所有其他页面设定完成（如，后载 Script）。在此时，页面元素可以安全地引用，且初始化或操纵。

使用这些 Script 事件处理程序来建立自行定义的 Script 函数，和初始化全局变量是很常见的。

页面事件处理程序可存取内部对象：Page 以及 EventName。

页面事件处理程序可以使用属性浏览器来编辑。

应用程序事件

应用程序事件的触发是独立于页面的加载和全局环境。

事件名稱	何時發生
Application_Startup	应用程序启动。
Application_UserChanged	当应用程序为了特定的使用者而重新启动。该用户的名称将会作为函数参数，并传递给事件处理程序。

使用 Application_UserChanged 事件处理程序来加载用户别的起始页面是很常见的。Application_UserChanged 事件处理程序必须进行以用户的名声编码，以作为函数参数。内部对象则是无效的。

应用程序事件处理程序通常会被建立及管理在应用程序之外的应用程序，并自动由 DataHub WebView 加载.SS 档案，并存在于 DataHub server 中。

应用程序事件处理程序不使用属性浏览器做编辑。

函数和特殊对象

为强化对 DataHub WebView Scripting 的了解，有必要去熟悉每个主题：

Working with Data Points	取得/設定資料點的函数。
Working with Element Properties	取得/设定元素属性的函数。
WV Functions	存取应用程序的命令和行为的函数。
Intrinsic Objects	允许事件处理程序去引用和操纵相关的 Page，Element 和 EventArgs 对象。

数据点工作

每个存储在 Cogent DataHub 中的值在被称为一个点。 点具有以下属性：

- Name - 一个字符串。目前唯一的上限是内部缓冲区的大小，预设情况下，约 1000 个字符。
- Value - 一个整数、浮点数，或字符串。
- Timestamp - 最不显著变化的点数值、联机质量或其他状态消息的日期和时间。
- Quality – 连接质量，由 Cogent DataHub 分派给此点，如 Good、Bad、Last known、Local override 等

DataHub WebView 支持区域和 server 端的数据点。区域数据点是由 WebView client 端进行管理，但不能持续保持。Server 端的数据点是由 Cogent DataHub 管理。

大多数情况下，您将使用点的值。不过，您也可以使用点对象来存取其他属性。要使用点作为对象，你可以使用 IVqt 属性。

您可以使用下面的 Script 函数来运用数据点：

VAL	取得指定点的值。
GET	取得指定的点以作为对象。支持 IVqt 和 IDataTable 接口。
SET	设定指定点的值。
WV.GetPointNames	使用指定的标准，来取得一个点名称数组。
WV.GetPoint	取得指定的点以作为对象。支持 IVqt 接口。

元素属性工作

在设计模式中，您可以在您的页面新增元素（或控制组件）。每个元素都有几个属性。大多数情况下，您可以使用属性浏览器来配置元素。不过，你也可以在 Script 中使用以下函数来操纵属性：

GETP	取得引用属性的值。
SETP	设定引用属性为指定的值。
IPageElementParameter.Value Property	透过 IPageElementParameter 界面成员，直接使用属性对象来取得/设定属性的值。

WV 函数

WV 函数是一组为所有 Script 提供资源查询的方法。

例如，可以使用 WV.GetPoint 从 DataHub 内取得一个点，而 WV.MsgBox 可用来在一个对话窗口中显示点的属性。

所有 WV 方法是静态的，这意味着并不需要建立 WV 对象的实体。相反的，你只需简单地通过直接使用 WV 类别来引用方法，。

WV 函数在整个本文内被广泛使用在实例中。

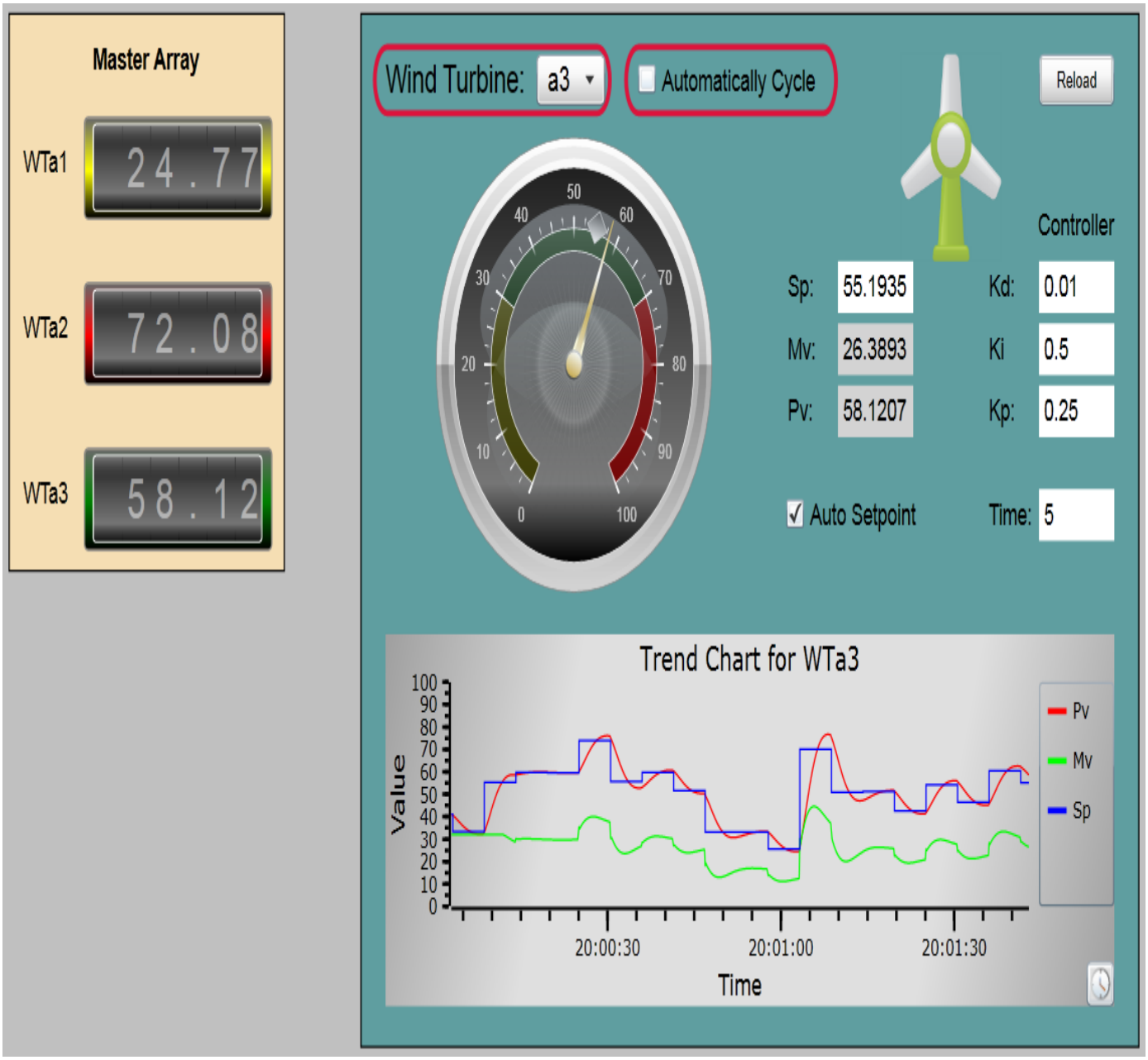
内部对象

内部对象使用各种 Script 链接和事件处理程序，提供存取通用对象的便利性。

例如，引用 Page 对象能够提供存取像是 PageName和 andPageDescription 等属性，以及像是 GetNamedElement 和 GetParameter 等方法。

范例

本实例说明一个从数组中显示三个风力涡轮机的讯息。仪表、趋势图和其他显示组件反映目前选定的涡轮机。使用者可以从”Wind Turbine”的 combo box 中选择一个涡轮机，或在清单中可以选择”Automatically Cycle”。



为让使用者的选择能够持续，已新增下面的 Script 事件处理程序。

Wind Turbine combobox ('wtSelector')OnDropDownClosed：设定一个 cookie 来代表选定的风力涡轮机。

```
WV.SetCookie("SelectedWindTurbine", GETP("@SelectedIndex"));
```

Automatically Cycle checkbox ('chkAutoCycle')OnChecked：设定一个 cookie 来表示 auto-cycle 已被选择。

```
WV.SetCookie("AutoCycleSelectedWindTurbine", 1);
```

Automatically Cycle checkbox ('chkAutoCycle')OnUnchecked：设定一个 cookie 代表自动循环不被选择。

```
WV.SetCookie("AutoCycleSelectedWindTurbine", 0);
```

OnPageLoadStarted：宣告一个 Script 函数，以及为控制组件储存 Cookie 的值。

```
function ApplyCookies()
{
  var wt = WV.GetCookie("SelectedWindTurbine");
  var auto = WV.GetCookie("AutoCycleSelectedWindTurbine");

  if (wt != null)
    SETP("wtSelector@SelectedIndex", wt);

  if (auto != null)
    SETP("chkAutoCycle@Value", auto);
}
```

OnPageLoadComplete：呼叫注册的 Script 函数。

```
ApplyCookies();
```

Reload button ('btnReloadPage')OnClick：重整当前页面。

```
WV.ExecuteCommand("OpenPage", Page.PageFullPath);
```

在执行模式下，每当用户点击 Automatically Cycle 的 checkbox，或从 Wind Turbine 的 combobox 进行选择， Script Cookie 会被设定和储存。

当用户点击刷新按钮时，会重新开启页面，并保持 cookie 值的取得和套用。即使使用者关闭并重新启动 DataHub WebView，Script Cookie 依然会持续保持。

最佳实作方法

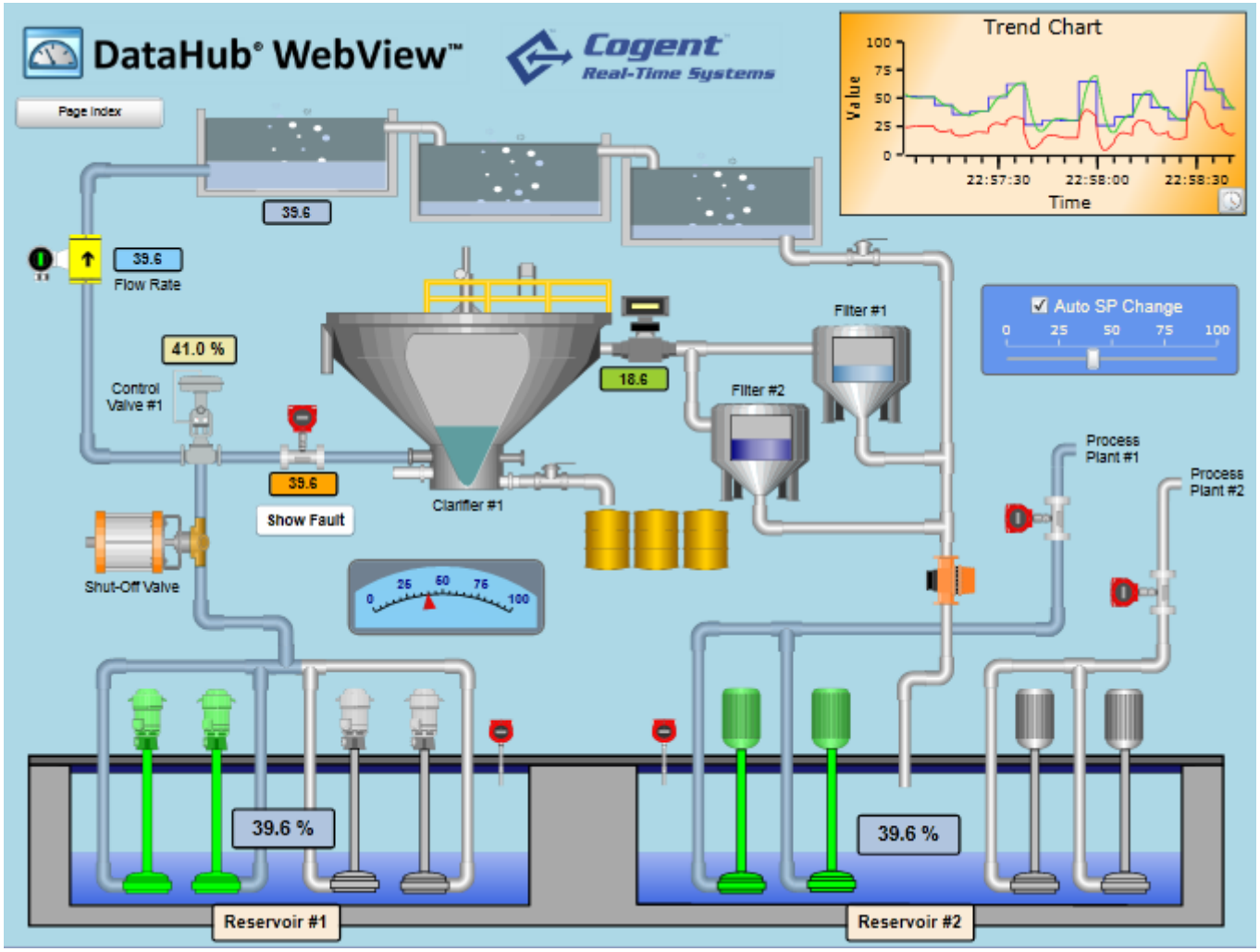
采用最佳实作方法，降低了维护需要和简化故障排除。当您建立 DataHub WebView 解决方案，我们建议您考虑采取以下做法。

定义有意义的控制组件名称

当控制组件在页面上被加入，它会被指定一个以其类型和后缀整数组成的预设名称。例如，第一次在页面上新增”SimpleButton”，它会命名为”SimpleButton1” - 除非页面上有个控制组件已经有这个名字了，在这种情况下，计数器会递增，直到找到一个可用的名称。

很显然，这样的页面，如下图所示，这样的 Script 变得很难理解，很难检查与排除问题。

```
if (GETP("Symbol23@Condition") > GETP("Symbol72@Condition"))
  SETP("TextLabel2@Input", "Overflow");
```



控制组件名称使用在 Script 表示式和简单的连结表示式中。给予页面上刚新增的控制组件适当的、有意义的名字是个很好的主意。

使用正确的变量环境

使用 var 宣告并初始化变量。这也适用于 Script 链接、事件处理程序，以及 Script 函数。没有 var 宣告，变量会被新增至全局 Script 环境，这意味着不管是在同一页上，或是其他页面上的其他 Script 而言，该变量都是有效的。除非这是您希望如此做，否则，最好使用局部变量以避免造成凌乱全局命名的空间。

使用有意义的名称给全局变量和 Script Cookie

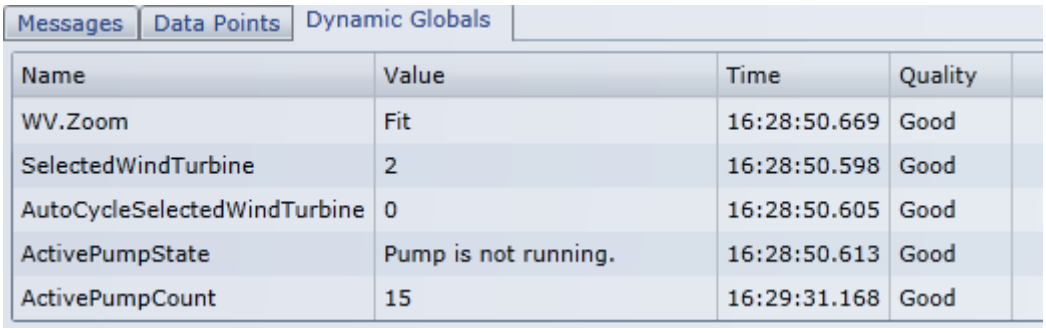
大多数 Script 很短，所以非常容易理解在函数、Script 链接或事件处理程序内如何使用局部变量。因为局部变量只提供相关的 Script 使用，所以混乱或名称冲突的风险不大。

然而，当使用全局 Script 变量的动态全局变量（通过 GET 和 SET）和 Script Cookie 时，使用很容易理解的名称、并提供目的指示，会是一个好的习惯。例如：

```
/* These variables and cookies are not well-named */
SET("c", 15);
WV.SetCookie("st", "Pump is not running.");

/* These are better */
SET("ActivePumpCount", 15);
WV.SetCookie("ActivePumpState", "Pump is not running.");
```

DataHub WebView 的动态全局变量和 Script Cookie 都是列示在Debug Window 中。



呈现结果在 DataHub WebView；产生逻辑在 DataHub

如同其他的 client－server 解决方案或是服务取向架构，数据和商业逻辑应该保留在服务器上。DataHub WebView 应该被用来管理用户接口（即表示层）来产生令人满意的使用者体验，同时依赖 Cogent DataHub 来处理数据和商业逻辑的复杂系统。

组织 Scripts 成可重复使用的函数

随着项目的发展，你可能会了解到相同的 Script 表示式或是编码区块会用在一个以上的地方。为了避免出现不一致的情况，以及维修问题，可以考虑建立 script 函数，甚至你自己的 script 函数库。script 函数库档案有.ss 延伸檔名，存放在 Web server 上 installDirectory/Silverlight/Script/活页夹中。Script 函数库可以由 DataHub 管理程序新增至 server 中。函数会在应用程序启动时自动加载和注册。

使用 PageData 与 initParams

Cogent DataHub 属性窗口为 DataHub Web Server 和 DataHub WebView 中提供配置选项。这些配置设定(例如，” 启动在执行模式” 和” 在启动时加载的页面”)影响 DataHub WebView 预设的启动 URL - installDirectory/Silverlight/DataHubWebView.asp。然而，你也可以建立自己的 HTML 或 ASP 页面，并提供这些网址让不同群体来启动不同的 DataHub WebView。例如，您可能每个部门都有一个网址，每个不同的启动页面。或者，你可能需要一个专门网页来启动禁止进入设计模式的应用程序。

除此之外，要在 Silverlight <object>tag 上指定 initParams，你也可以在 url 上传递参数。例如，这个 URL 传递使用者认证、设定一个起始页面，然后传送页面数据(可用于起始页面上的 Script，以链接控制组件到要求处理的数据点)。

```
http://localhost/Silverlight/AnnualShowKioskOnly.asp&username=demo&password
=demo&page=AnnualShow/Home&pagedata=process=mfg1
```

有两点需要注意...

1. 指定使用者认证在 url 或 ASP 页面是一个潜在的安全风险。如果你选择使用此功能，您应该确保这是仅被授权使用在执行模式，以及只是使用于公开展示的目的。此功能适合使用于像是会展中的自助服务计算机环境中，DataHub WebView 需要跳过登录画面，因为会议的参加者不会有使用者认证。
2. PageData 是一组 name-value 配对。每个名称都成为 S#的全局变量。在这种情况下，全局变量程序将会被分配字符串值“mfg1”，该值就是可以用在点连结表示式，像 this: `="demodata:" + process + "_LineSpeed";`来产生点连结 ofdemodata：mfg1_LineSpeed 。

以动态点链接建立模板页面

考虑建立模板页面，而不是建立多个相同的页面给不同的点连结。例如使用 Script 表示式来设定点连结的页面。

以 Script Cookie 记忆用户的选择

以 comboboxes 和 listboxes 提供给使用者选择，是相当常见的作法。在某些情况下，使用者所做的选择应该要保留，那么页面在下次开启时，就不需要再次做选择。大多数情况下，这些选择是使用者别，且不应该与其他使用者共享。Script Cookie 最适合使用于此目的。

以设计模式中的解释来提升页面维护效率

理解和维护复杂的页面设计可能是一个挑战，特别是试图支持页面的人不是当初的设计者。详细批注可以使用标准的文字卷标控制组件来新增。为了确保用户在执行模式下看不到批注，只需将“Visible in Run Mode”属性设定为 false(在”Common Properties: Content Visibility and Appearance”之下的清单)。

以控制组件工具提示提供在执行模式下的用户帮助，

创造一个简单而直观的页面设计，有时候并不容易。某些制造过程先天上就有点复杂。为了帮助操作者了解页面的目的，以及如何与控制组件互动，可以考虑增加工具提示控制（在” Common Properties: ToolTip and Tags”之下的清单）。您甚至可以使用 Script 来设定动态工具提示，或许是为了提供内容讯息给报警和其他情况。