Documentation Library

DataHub® WebView™

Version 1.4

Cogent Real-Time Systems, Inc.

May 31, 2013

DataHub® WebView™: Version 1.4

A user's guide to DataHub WebView.

Published May 31, 2013
Cogent Real-Time Systems, Inc.

162 Guelph Street, Suite 253
Georgetown, Ontario
Canada, L7G 5X7

Toll Free: 1 (888) 628-2028
Tel: 1 (905) 702-7851
Fax: 1 (905) 702-7850

Information Email: info@cogent.ca
Tech Support Email: support@cogent.ca
Web Site: www.cogent.ca

Revision History

Revision 1.4-1 January 2011
Initial release of documentation.

Copyright, trademark, and software license information.

Copyright Notice

Trademark Notice

END-USER LICENSE AGREEMENT FOR COGENT SOFTWARE

IMPORTANT - READ CAREFULLY: This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Cogent Real-Time Systems Inc. ("Cogent") of 162 Guelph Street, Suite 253, Georgetown, Ontario, L7G 5X7, Canada (Tel: 905-702-7851, Fax: 905-702-7850), from whom you acquired the Cogent software product(s) ("SOFTWARE PRODUCT" or "SOFTWARE"), either directly from Cogent or through one of Cogent's authorized resellers.
The SOFTWARE PRODUCT includes computer software, any associated media, any printed materials, and any "online" or electronic documentation. By installing, copying or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree with the terms of this EULA, Cogent is unwilling to license the SOFTWARE PRODUCT to you. In such event, you may not use or copy the SOFTWARE PRODUCT, and you should promptly contact Cogent for instructions on return of the unused product(s) for a refund.

SOFTWARE PRODUCT LICENSE

The SOFTWARE PRODUCT is protected by copyright laws and copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

1. **EVALUATION USE:** This software is distributed as "Free for Evaluation", and with a per-use royalty for Commercial Use, where "Free for Evaluation" means to evaluate Cogent's software and to do exploratory development and "proof of concept" prototyping of software applications, and where "Free for Evaluation" specifically excludes without limitation:
    i. use of the SOFTWARE PRODUCT in a business setting or in support of a business activity,
    ii. development of a system to be used for commercial gain, whether to be sold or to be used within a company, partnership, organization or entity that transacts commercial business,
    iii. the use of the SOFTWARE PRODUCT in a commercial business for any reason other than exploratory development and "proof of concept" prototyping, even if the SOFTWARE PRODUCT is not incorporated into an application or product to be sold,
    iv. the use of the SOFTWARE PRODUCT to enable the use of another application that was developed with the SOFTWARE PRODUCT,
    v. inclusion of the SOFTWARE PRODUCT in a collection of software, whether that collection is sold, given away, or made part of a larger collection.
    vi. inclusion of the SOFTWARE PRODUCT in another product, whether or not that other product is sold, given away, or made part of a larger product.

2. **COMMERCIAL USE:** COMMERCIAL USE is any use that is not specifically defined in this license as EVALUATION USE.
3. **GRANT OF LICENSE:** This EULA covers both COMMERCIAL and EVALUATION USE of the SOFTWARE PRODUCT. Either clause (A) or (B) of this section will apply to you, depending on your actual use of the SOFTWARE PRODUCT. If you have not purchased a license of the SOFTWARE PRODUCT from Cogent or one of Cogent's authorized resellers, then you may not use the product for COMMERCIAL USE.
    A. **GRANT OF LICENSE (EVALUATION USE):** This EULA grants you the following non-exclusive rights when used for EVALUATION purposes:

Software: You may use the SOFTWARE PRODUCT on any number of computers, either stand-alone, or on a network, so long as every use of the SOFTWARE PRODUCT is for EVALUATION USE. You may reproduce the SOFTWARE PRODUCT, but only as reasonably required to install and use it in accordance with this LICENSE or to follow your normal back-up practices.

Subject to the license expressly granted above, you obtain no right, title or interest in or to the SOFTWARE PRODUCT or related documentation, including but not limited to any copyright, patent, trade secret or other proprietary rights therein. All whole or partial copies of the SOFTWARE PRODUCT remain property of Cogent and will be considered part of the SOFTWARE PRODUCT for the purpose of this EULA.

Unless expressly permitted under this EULA or otherwise by Cogent, you will not:

i. use, reproduce, modify, adapt, translate or otherwise transmit the SOFTWARE PRODUCT or related components, in whole or in part;

ii. rent, lease, license, transfer or otherwise provide access to the SOFTWARE PRODUCT or related components;

iii. alter, remove or cover proprietary notices in or on the SOFTWARE PRODUCT, related documentation or storage media;

iv. export the SOFTWARE PRODUCT from the country in which it was provided to you by Cogent or its authorized reseller;

v. use a multi-processor version of the SOFTWARE PRODUCT in a network larger than that for which you have paid the corresponding multi-processor fees;

vi. decompile, disassemble or otherwise attempt or assist others to reverse engineer the SOFTWARE PRODUCT;

vii. circumvent, disable or otherwise render ineffective any demonstration time-outs, locks on functionality or any other restrictions on use in the SOFTWARE PRODUCT;

viii. circumvent, disable or otherwise render ineffective any license verification mechanisms used by the SOFTWARE PRODUCT;

ix. use the SOFTWARE PRODUCT in any application that is intended to create or could, in the event of malfunction or failure, cause personal injury or property damage; or

x. make use of the SOFTWARE PRODUCT for commercial gain, whether directly, indirectly or incidentally.

B. **GRANT OF LICENSE (COMMERCIAL USE):** This EULA grants you the following non-exclusive rights when used for COMMERCIAL purposes:

Software: You may use the SOFTWARE PRODUCT on one computer, or if the SOFTWARE PRODUCT is a multi-processor version - on one node of a network, either: (i) as a development systems for the purpose of creating value-added software applications in accordance with related Cogent documentation; or (ii) as a single run-time copy for use as an integral part of such an application. This includes reproduction and configuration of the SOFTWARE PRODUCT, but only as reasonably required to install and use it in association with your licensed processor or to follow your normal back-up practices.

Storage/Network Use: You may also store or install a copy of the SOFTWARE PRODUCT on one computer to allow your other computers to use the SOFTWARE PRODUCT over an internal network, and distribute the SOFTWARE PRODUCT to your other computers over an internal network. However, you must acquire and dedicate a license for the SOFTWARE PRODUCT for each computer on which the SOFTWARE PRODUCT is used or to which it is distributed. A license for the SOFTWARE PRODUCT may not be shared or used concurrently on different computers.

Subject to the license expressly granted above, you obtain no right, title or interest in or to the SOFTWARE PRODUCT or related documentation, including but not limited to any copyright, patent, trade secret or other proprietary rights therein. All whole or partial copies of the SOFTWARE PRODUCT remain property of Cogent and will be considered part of the SOFTWARE PRODUCT for the purpose of this EULA.

Unless expressly permitted under this EULA or otherwise by Cogent, you will not:

i. use, reproduce, modify, adapt, translate or otherwise transmit the SOFTWARE PRODUCT or related components, in whole or in part;

ii. rent, lease, license, transfer or otherwise provide access to the SOFTWARE PRODUCT or related components;

iii. alter, remove or cover proprietary notices in or on the SOFTWARE PRODUCT, related documentation or storage media;

iv. export the SOFTWARE PRODUCT from the country in which it was provided to you by Cogent or its authorized reseller;

v. use a multi-processor version of the SOFTWARE PRODUCT in a network larger than that for which you have paid the corresponding multi-processor fees;

vi. decompile, disassemble or otherwise attempt or assist others to reverse engineer the SOFTWARE PRODUCT;

vii. circumvent, disable or otherwise render ineffective any demonstration time-outs, locks on functionality or any other restrictions on use in the SOFTWARE PRODUCT;

viii. circumvent, disable or otherwise render ineffective any license verification mechanisms used by the SOFTWARE PRODUCT, or

ix. use the SOFTWARE PRODUCT in any application that is intended to create or could, in the event of malfunction or failure, cause personal injury or property damage.

4. **WARRANTY:** Cogent cannot warrant that the SOFTWARE PRODUCT will function in accordance with related documentation in every combination of hardware platform, software environment and SOFTWARE PRODUCT configuration. You acknowledge that software bugs are likely to be identified when the SOFTWARE PRODUCT is used in your particular application. You therefore accept the responsibility of satisfying yourself that the SOFTWARE PRODUCT is suitable for your intended use. This includes conducting exhaustive testing of your application prior to its initial release and prior to the release of any related hardware or software modifications or enhancements.

   Subject to documentation errors, Cogent warrants to you for a period of ninety (90) days from acceptance of this EULA (as provided above) that the SOFTWARE PRODUCT as delivered by Cogent is capable of performing the functions described in related Cogent user documentation when used on appropriate hardware. Cogent also warrants that any enclosed disk(s) will be free from defects in material and workmanship under normal use for a period of ninety (90) days from acceptance of this EULA. Cogent is not responsible for disk defects that result from accident or abuse. Your sole remedy for any breach of warranty will be either: i) terminate this EULA and receive a refund of any amount paid to Cogent for the SOFTWARE PRODUCT, or ii) to receive a replacement disk.

5. **LIMITATIONS:** Except as expressly warranted above, the SOFTWARE PRODUCT, any related documentation and disks are provided "as is" without other warranties or conditions of any kind, including but not limited to implied warranties of merchantability, fitness for a particular purpose and non-infringement. You assume the entire risk as to the results and performance of the SOFTWARE PRODUCT. Nothing stated in this EULA will imply that the operation of the SOFTWARE PRODUCT will be uninterrupted or error free or that any errors will be corrected. Other written or oral statements by Cogent, its representatives or others do not constitute warranties or conditions of Cogent.

   In no event will Cogent (or its officers, employees, suppliers, distributors, or licensors: collectively "Its Representatives") be liable to you for any indirect, incidental, special or consequential damages whatsoever, including but not limited to loss of revenue, lost or damaged data or other commercial or economic loss, arising out of any breach of this EULA, any use or inability to use the SOFTWARE PRODUCT or any claim made by a third party, even if Cogent (or Its Representatives) have been advised of the possibility of such damage or claim. In no event will the aggregate liability of Cogent (or that of Its Representatives) for any damages or claim, whether in contract, tort or otherwise, exceed the amount paid by you for the SOFTWARE PRODUCT.

   These limitations shall apply whether or not the alleged breach or default is a breach of a fundamental condition or term, or a fundamental breach. Some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, or certain limitations of implied warranties. Therefore the above limitation may not apply to you.

6. **DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS:**

   Separation of Components. The SOFTWARE PRODUCT is licensed as a single product. Its component parts may not be separated for use on more than one computer.

Termination. Without prejudice to any other rights, Cogent may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such an event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts.

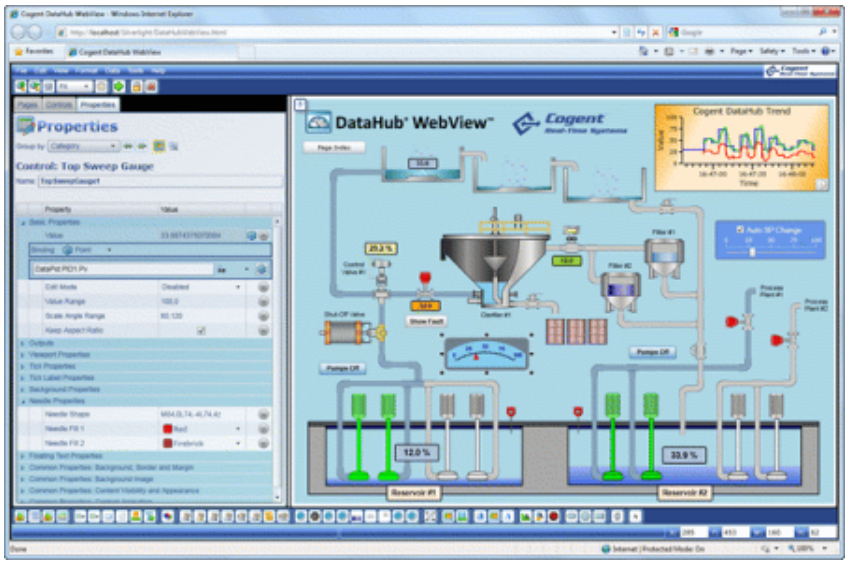
7. **UPGRADES:** If the SOFTWARE PRODUCT is an upgrade from another product, whether from Cogent or another supplier, you may use or transfer the SOFTWARE PRODUCT only in conjunction with that upgrade product, unless you destroy the upgraded product. If the SOFTWARE PRODUCT is an upgrade of a Cogent product, you now may use that upgraded product only in accordance with this EULA. If the SOFTWARE PRODUCT is an upgrade of a component of a package of software programs which you licensed as a single product, the SOFTWARE PRODUCT may be used and transferred only as part of that single product package and may not be separated for use on more than one computer.
8. **COPYRIGHT:** All title and copyrights in and to the SOFTWARE PRODUCT (including but not limited to any images, photographs, animations, video, audio, music, text and ’applets", incorporated into the SOFTWARE PRODUCT), any accompanying printed material, and any copies of the SOFTWARE PRODUCT, are owned by Cogent or its suppliers. You may not copy the printed materials accompanying the SOFTWARE PRODUCT. All rights not specifically granted under this EULA are reserved by Cogent.
9. **PRODUCT SUPPORT:** Cogent has no obligation under this EULA to provide maintenance, support or training.
10. **RESTRICTED RIGHTS:** Use, duplication, or disclosure by the U.S. Government is subject to restrictions as provided in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii) (OCT 1988), FAR 12.212(a)(1995), FAR 52.227-19, or FAR 52.227-14 (ALT III), as appropriate. Manufacturer is Cogent Real-Time Systems Inc. 162 Guelph Street, Suite 253, Georgetown, Ontario, L7G 5X7, Canada.
11. **GOVERNING LAW:** This Software License Agreement is governed by the laws of the Province of Ontario, Canada. You irrevocably attorn to the jurisdiction of the courts of the Province of Ontario and agree to commence any litigation that may arise hereunder in the courts located in the Judicial District of Peel, Province of Ontario.

Table of Contents

List of Tables

Chapter 1. Introduction

DataHub WebView is a state-of-the-art, rich internet application for designing and delivering high-quality, real-time displays. All page editing is done in a standard web browser, and page updates can be automatically published as soon as changes are saved. Page designers have access to standard controls, gauges, and 4,000 industry-standard symbols, each of which can be easily configured to recognize condition states and to graphically notify operators of process status and anomalies. All controls feature powerful, "anything-to-anything" data binding.



1.1. System Requirements

Windows: X86 or x64(64-bit mode support for IE only) 1.6 GHz or higher processor with 512-MB of RAM.

Macintosh (Intel-based): Intel Core Duo 1.83 GHz or higher processor with 512 MB of RAM.

Minimum and Recommended

Hard disk: minimum 70 MB, recommended 70 MB plus any additional space for historical files, log files. The Cogent DataHub does not use substantial space beyond the minimum unless you configure it to use more.

Memory: minimum 50 MB, recommended 500 MB. The DataHub will consume more memory with more data points configured, and with more client connections. We recommend having enough spare memory to load a large data set. Swapping will reduce performance.

Number of processor cores: minimum 1, recommended 2. An extra core will allow busy connections and scripts to run on a separate core and will help to keep the GUI responsive. The DataHub can use as many cores as you provide.

Internet browser for WebView: One of Firefox, Internet Explorer, Chrome or Opera. Effectively any browser that will load Silverlight will work. Internet Explorer is recommended.

Network protocol between WebView and DataHub: DataHub WebView needs access via TCP/IP on three ports:

- The web server port can be any port number (default is `80`).

- The Silverlight policy server port must be port `943`.
- The DataHub plain-text tunnel port port must be in the range of `4502-4534` (default is `4502`).

UDP is not used.

Compatible Operating Systems and Browswers

Windows 8 Desktop: IE10*, FF3.6+, Chrome 12+

Windows Server 2012: IE10*, FF3.6+, Chrome 12+

Windows 7: IE8, IE9, FF3.6+, Chrome 12+

Windows 7 SP1: IE8, IE9*, FF3.6+, Chrome 12+

Windows Server 2008 SP2: IE7, FF3.6+, Chrome 12+

Windows Server 2008 R2 SP2: IE8*, IE9*, FF3.6+, Chrome 12+

Windows Vista: IE7, IE8, IE9, FF3.6+, Chrome 12+

Windows Server 2003: IE7, IE8, FF3.6+, Chrome 12+

Windows XP SP2, SP3: IE7, IE8, FF3.6+, Chrome 12+

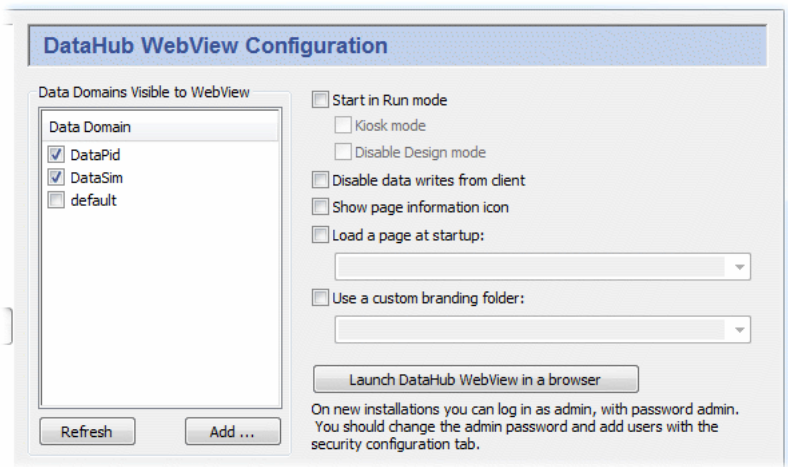
Macintosh OS 10.5.7+ (Intel-based): FF3.6+, Safari 4+

* Supports 64-bit mode.

1.2. Configuration

1.2.1. DataHub Configuration

Certain parameters of DataHub WebView can be configured from within the Cogent DataHub Properties window:



Data Domains Visible to DataHub WebView

Check the data domains that you want to access from DataHub WebView. Use the Add... button to create and add new domains.

Start in Run mode

Allows you to start in Run mode, rather than Design mode, with these options:

Kiosk mode

Presents just the working screen of the web browser, with no border, menus, URL entry field, etc. To escape from Kiosk mode (and close the browswer), press **Alt** + **F4**.

Disable Design mode

Prohibits any switch from Run mode to Design mode, whether running in Kiosk mode or normally.

Disable data writes from client

Prevents the web client from accessing DataHub point values.

Show page information icon

Shows or hides the page information icon.

Load a page at startup

Allows you to specify a page that will automatically load when DataHub WebView starts.

Launch DataHub WebView in a browser

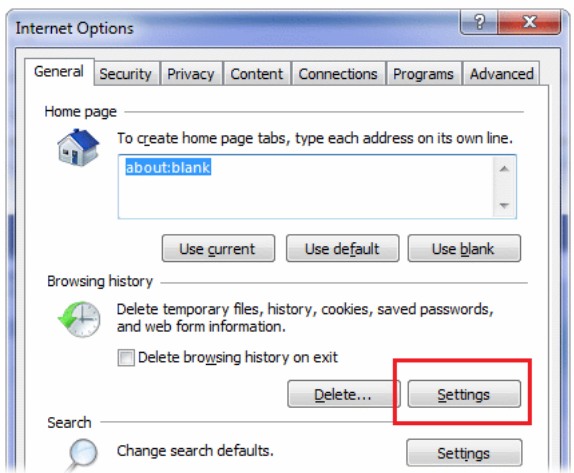
Provides a convenient way to start DataHub WebView to check this configuration.

DataHub WebView requires the DataHub to be configured as a tunnelling master. Please refer to Tunnel/Mirror Master in the DataHub manual for details.

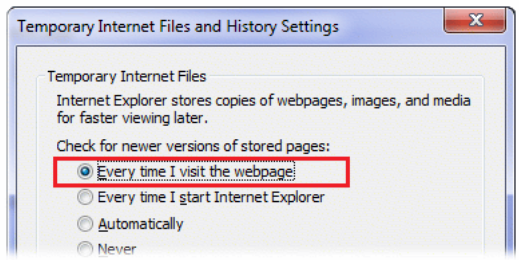
1.2.2. For Internet Explorer Users

For best results, anyone editing or viewing DataHub WebView pages in Internet Explorer should modify its settings as follows:

1. Open the Internet Explorer menu option Tools / Internet Options and select the Browsing History Settings button:



2. Change the cache update algorithm to check for new versions Every time I visit the webpage:



Do *not* select the option Automatically. This will cause Internet Explorer to try to guess whether to check for updates to its cache based on a heuristic algorithm that attempts to guess how frequently it should check. This algorithm will usually not check for updates, and the behaviour described below will occur.

Explanation

Internet Explorer maintains a local cache of pages that it has visited recently, and when you make a request for a page it will serve the page from the cache instead of from the web server. This can speed up browsing, but it makes the behaviour of DataHub WebView unpredictable.

Normally a web server can mark a page as "no-cache" in which case the browser will always re-load the page on each visit. That is not what we want in the case of DataHub WebView. We want the pages to be re-loaded only if they have changed on the server. The HTTP protocol allows for this, and this is the best method for ensuring that the page you are looking at is up to date. Essentially, the browser makes a request from the server, telling the server the timestamp of the cached version of the page. If the cached version is up to date, the web server just returns an indication that the browser should use its cached copy of the page. This is efficient since the amount of information transmitted is small.

Unfortunately, Internet Explorer does not default to the best behaviour. Instead, it contains a heuristic algorithm that uses the cached copy of a page without ever consulting the web server. This can result in the following condition in DataHub WebView:

1. You load a DataHub WebView page
2. You edit the page and then save it. The page is correctly stored on the DataHub Web Server.
3. You re-load the page you have just edited. Internet Explorer retrieves the page from its cache without consulting the DataHub Web Server.
4. DataHub WebView receives the old page from Internet Explorer’s cache and your changes appear to have vanished. At this point, the copy of the page stored in the DataHub Web Server is correct, but Internet Explorer never asks for it. To you, it appears as if the page was not saved and your edits were lost.

1.3. Advantages of DataHub WebView

- **Edit screens anywhere.** Just open your web browser, type the address of your DataHub Web Server, log in to the system and, based on your access permissions, you can build, edit and view DataHub WebView screens from anywhere with network/Internet access.
- **No coding required.** Build all your screens using the powerful built-in graphical environment. Complete point and click freedom, with no code in sight.

- **No development system required.** You do not need to install a development system on your computer in order to build DataHub WebView screens. The development interface is provided to you, in your browser, by the web server. No compiler necessary.
- **No deployment required.** Because the screens you build in DataHub WebView are saved on the web server, you never have to deploy your changes to other users. When you are editing a page and you want to show your colleagues, you simply save your changes, and tell your team to reload the page in their browser. No deployment, no complicated file updates.
- **Collaborate on development.** With no limit to the number of users accessing the system, you can have a team of developers building pages at the same time. Depending on the permissions you give to your team members you can have them edit everyone's pages, or restrict access so they can only edit the pages they build.
- **Build entire multipage HMI applications** with hyperlinks between pages, just like a web site or standard HMI system.
- **Security at every level.** The permissions based security model allows you to define very precise privaledges to each user. You can define certain users to have read only access, while other can view and make changes but not access the develoment interface. In WebView, the security model was one of the fundamental primary requirements, not an afterthought.
- **Specify sophisticated graphical interactions between controls,** using DataHub WebView's powerful property binding feature. Property binding allows you to associate the input values for one control, with the properties of other controls. Simpler to do than to describe, this saves you time and adds significant power to your DataHub WebView screen.
- **Based on Microsoft Silverlight,** which means it is a leading web technology that is widely supported by all the popular browsers. Silverlight allows us to produce browser based applications that are every bit as feature rich and powerful as desktop based applications and they leverage the new opportunities offered by remote development and web based presentation.
- **Comes with a standard set of built-in controls,** such as trends, gauages and sliders. In additon, DataHub WebView also ships with a complete set of Symbol Factory™[1] symbols, which offer thousands of industry standard symbols for a variety of industries. DataHub WebView is able to provide these and other future third party symbols.

Notes

1. Symbol Factory™ is a trademark of Reichard Software Corporation. Symbol Factory graphics are included with DataHub WebView under license from Software Toolbox and Reichard Software Corporation.

Chapter 2. Working With DataHub WebView

2.1. Quick Start

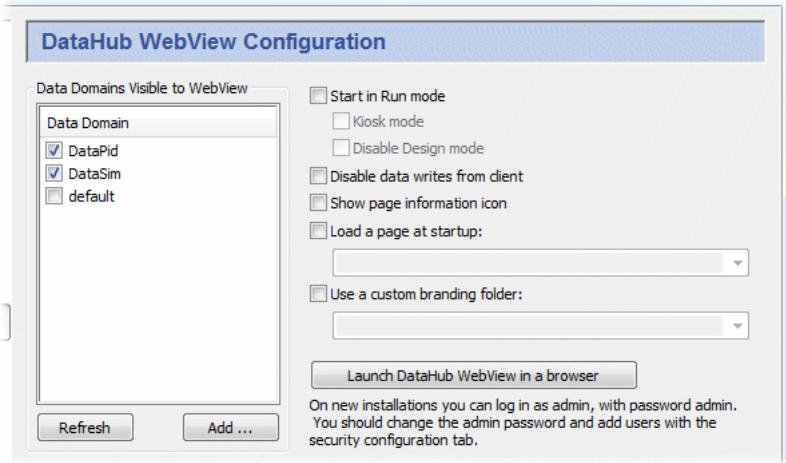
Here is a brief example of how to start DataHub WebView, add a control to a page, edit a property and animate the control with live data, and then save and view the resulting page.

If you are using Internet Explorer, please see Section 1.2.2, *For Internet Explorer Users* and configure Internet Explorer as shown there before continuing.

2.1.1. Start the Editor

1. In the Cogent DataHub Properties window, select WebView 

.
2. In the DataHub WebView configuration window, select the Data Domains that you wish to access data from, and press the Apply button.



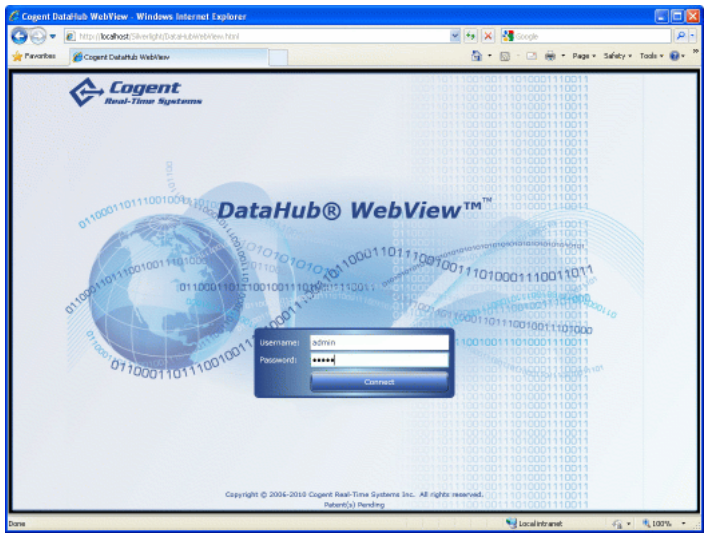
See also Section 1.2, *Configuration* for more details about configuration.

3. Once configured, there are two ways to start DataHub WebView:
   A. From here in the Properties window, press the Launch DataHub WebView in a browser button.
   B. From a web browser:
      1. Open a web browser like Internet Explorer, Firefox, or Chrome.
      2. Type in the DataHub WebView default URL:

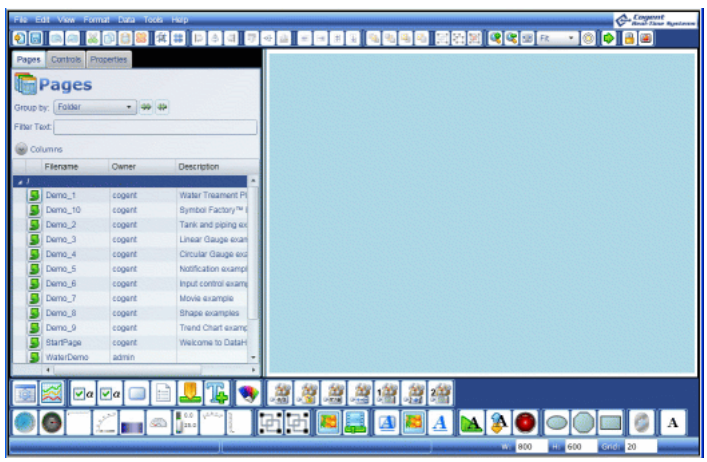
```
http://localhost/Silverlight/DataHubWebView.asp
```

Using `localhost` opens the DataHub WebView on your computer. If you need to connect to DataHub WebView on another computer, instead of `localhost` use that computer's IP address in the URL, and keep everything else the same.

Either of these will open the DataHub WebView application in your web browser.

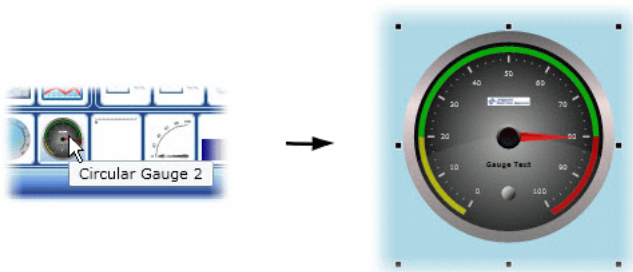


4. Enter the password **`admin`** to start the DataHub WebView Editor. See also Section 2.2.1, *Configure User Permissions* for more details about security, user names, and passwords.



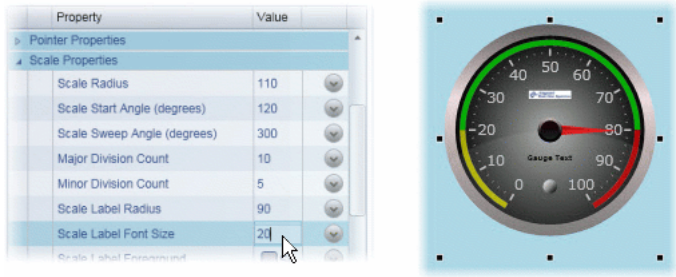
2.1.2. Add and Modify a Control

1. Find the Circular Gauge 2 control button at the bottom of the editor, and click it.



A copy of the Circular Gauge 2 control will appear in the blank page.

2. Now let's adjust a property of the gauge. In the Properties list on the left, find the Scale Properties, expand the list, and in the Scale Label Font Size, enter **`20`**.



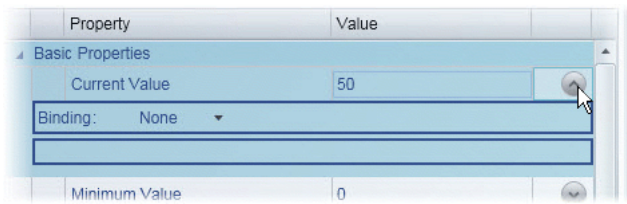
Press **Enter**, and the font size of the numbers on the guage will expand to 20 points.

See also Section 2.4.3, *Control Properties* for more details about working with control properties.

2.1.3. Bind a Control to a Data Point

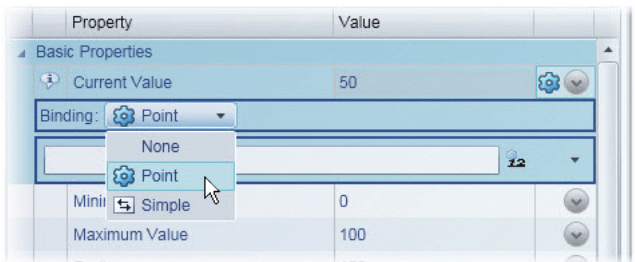
Many of the controls in DataHub WebView can be bound to certain variables, so that whenever the variable changes, the value or appearance of the control changes as well. For example, a gauge or meter can be bound to a DataHub point to display changes to the point in real time. In this example we bind the gauge we created above to display the value of the DataPid point Pv.

1. Start the DataPid program that is included in your Cogent DataHub archive to generate some test data.
2. Open the Basic Properties of your gauge, and click the arrow button on the right side of the Current Value row.



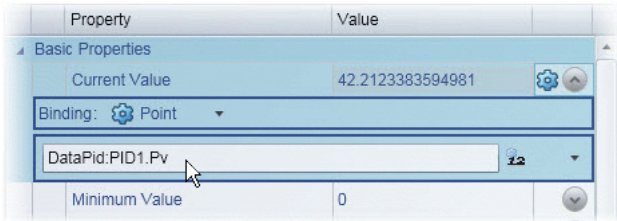
A Binding selection box will appear.

3. In the Binding selection box, click the down arrow to open the list, and select Point.



This will activate the point selection entry field.

4. We want to connect to the DataPid point DataPid:PID1:Pv, so enter just Pv. All the points that have "Pv" in their names will appear.

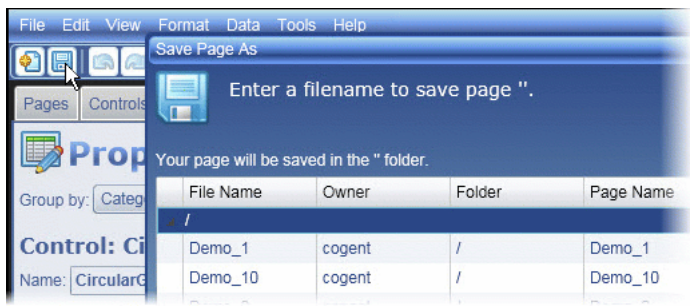


Select the point DataPid:PID1:Pv. Once selected, the data will start updating in the value entry, and the gauge needle will start to move.

See also Section 2.4, *Controls* for more details about using controls, or Section 2.5, *Property Binding* for binding data them to DataHub points.

2.1.4. Save and View a Page

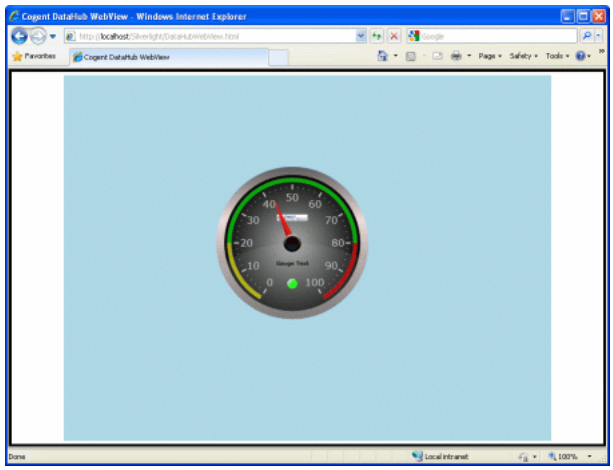
1. To save the page, you can click the Save button , or choose Save from the File menu, or press **Ctrl** + **Shift** + **S**.



   Specify a filename for the page.

2. To enter Run Mode and view your page, click on the Enter Run Mode button or press **Ctrl** + **Shift** + **R**.

   Your page will appear in the web browser as a user would see it, with all the controls fully animated and functional.

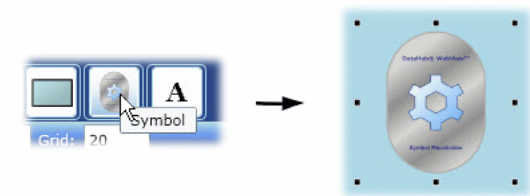


To exit Run Mode and return to Edit Mode, click on the Exit Run Mode button  or press **Ctrl** + **Shift** + **R**.

See also Section 2.3, *Pages* for more details about saving and viewing pages.

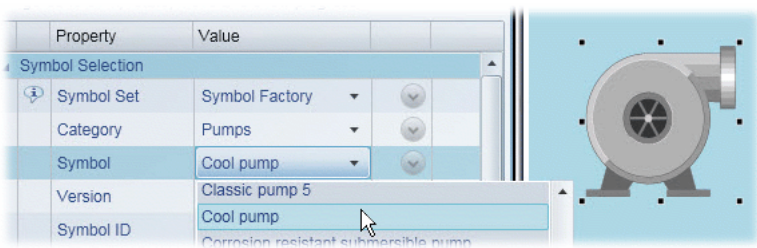
2.1.5. Add a Symbol

1. Find the Symbol control button at the bottom of the editor, and click it.



A copy of the Symbol control will appear in the page. This one control can be used to represent any symbol in the symbol library, which contains thousands of different symbols.

2. In the Properties list, for the Symbol Set, choose Symbol Factory. For Category choose Pumps, and for Symbol, choose Cool pump.



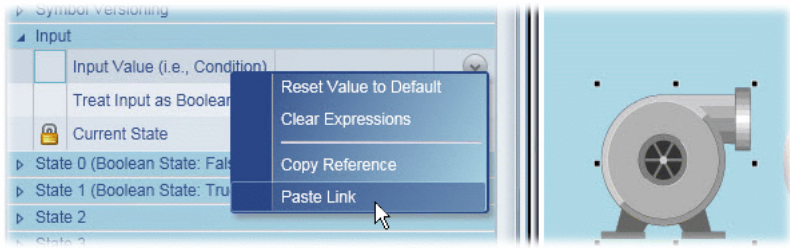
The generic symbol icon should change into a symbol of a pump.

2.1.6. Bind a Control to another Control

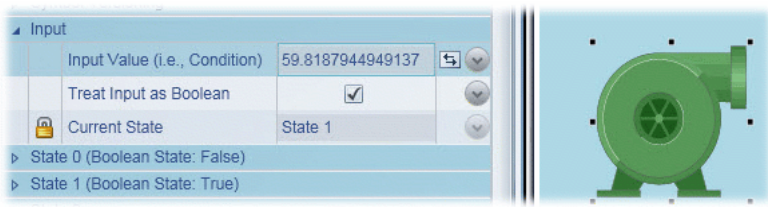
Most controls can bind their properties to other controls, so that when the first control is modified, the bound control gets modified automatically. Here's an example, binding the value of the pump we just created to the value of the gauge.

1. Click the gauge, and in the Basic properties, right-click the Current Value row, and select Copy Reference.

2. Click the pump, and in the Input properties, right-click the Input Value row, and select Paste Link.



The pump will take the same values as the gauge, and turn green, the default non-zero color for this symbol.



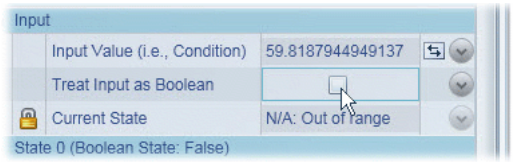
Notice that the Current State is State 1, for `True`. This default can be changed, as explained below.

See also Section 2.5, *Property Binding* for more details about binding control properties.

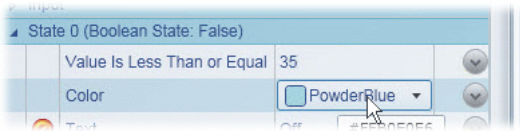
2.1.7. Set Symbol States

Most symbols can be set as booleans, to show on/off states, and many can also display multiple states. Here we'll change the default boolean settings and colors of the pump to display three different states.

1. Click on the pump, and uncheck the Treat as Boolean box.



2. In the State 0 properties, for Value Is Less Than or Equal enter a value of 35. Then change the Color to PowderBlue.



3. In the State 1 properties, enter a value of 65 and change the Color to MediumBlue.
4. In the State 2 properties, enter a value of 100 and change the Color to Navy.

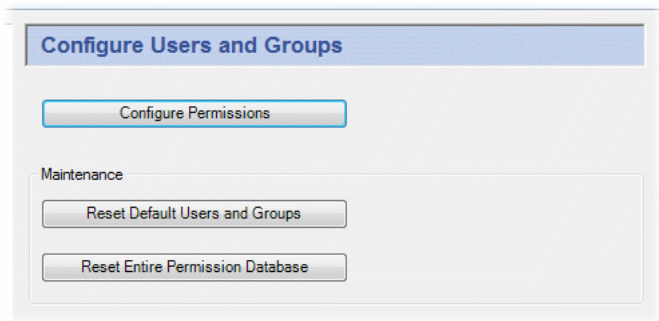
Now, whenever the gauge value is between 0 and 35, the pump color will be light blue, 36 to 65 medium blue, and 66 to 100 dark blue. Note that for each state, you enter the maximum value, while the minimum value is controlled by the previous state.

2.2. User Access

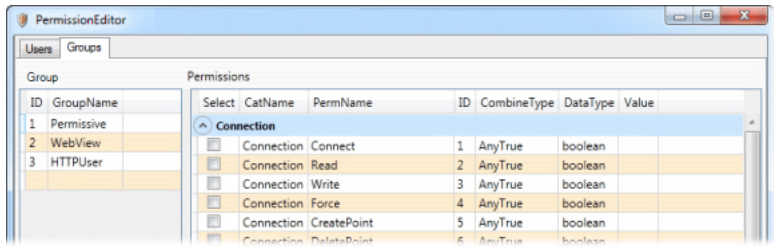
2.2.1. Configure User Permissions

User permissions and passwords are assigned according to groups in the Security option of the DataHub Properties window. Here is an example of how to configure two different groups: operators and page designers, and then assign users to them.

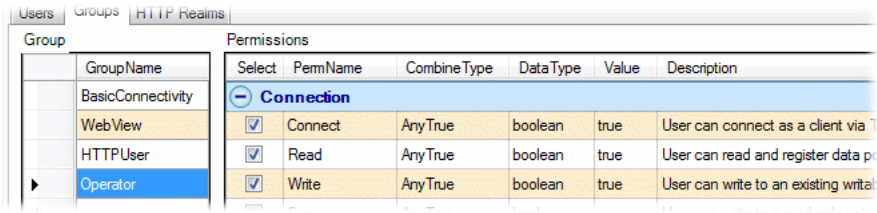
1. Open the Security option of the Properties window and click the Configure Permissions button.



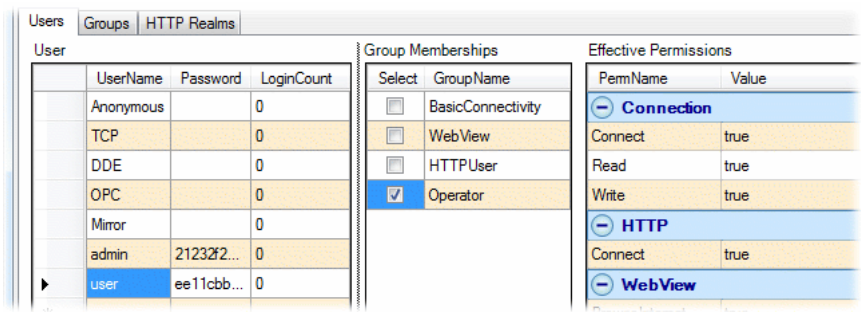
2. Click on the Groups tab. This will allow you to configure a group of users that will all inherit the same permissions.



3. Create a new group by typing in the name: **`Operator`**.



4. Select the following permissions:
   - In the Connectionssection, to give access to real-time data, select: Connect, Read, and Write.
   - In the HTTP section, select Connect to allow DataHub WebView to connect to the DataHub Web Server.
   - In the DataHub WebView section, to allow basic interaction with the DataHub WebView application, select: Connect, ViewPage, ViewOtherOwnerPage, ViewOnlineHelp, and BrowseInternet.
5. Click on the Users tab.



6. Add a user name, with no spaces in it, for example, **`JohnDoe`**, and enter a password for him.

   The password is not stored in the system. If a user forgets his password, you cannot retrieve it. You will need to assign him a new password.

7. Check the Operator box to make `JohnDoe` a member of the Operator group. Notice that all of the group's permissions are given to `JohnDoe`.
8. Another useful group to create would be Designers, who have permission to design pages. These users would need the same Connection and HTML permissions as above but could take more, even all, of the DataHub WebView options, as needed.
9. Once that group is created, you can assign designer users to the group.

10. When finished, press the Apply button to write your configuration to the DataHub permissions database.

Now, when you launch DataHub WebView and log in as **`JohnDoe`**, you should go straight to the Start page in Run Mode, and have no access to Edit Mode. If you log in as a designer, you will be place in Edit Mode.

2.2.2. Log in Remotely

Any user can very easily log in to DataHub WebView remotely from another computer on the network, or over the Internet. Just open your web browser and navigate to this URL:

```
http://IP_or_computer_name/Silverlight/DataHubWebView.asp
```

Where *`IP_or_computer_name`* is the network IP address or DNS computer name of the computer running the Cogent DataHub.

2.3. Pages

If you are using Internet Explorer, please ensure that it is configured properly for use with DataHub WebView.

2.3.1. Create, Open, Save, and Delete Pages

To create a page click on the Pages tab and then click on the New button  in the Toolbar. You can also create a page from the Edit menu, or by pressing **Ctrl** + **Shift** + **N**.

To open a page click on the Pages tab and then click on the Open button  next to the name of the page. You can also open a page by double-clicking on the name of the page in the Pages tab.

To save a page simply click on the Save button  . You can also save a page from the Edit menu, or by pressing **Ctrl** + **Shift** + **S**.

To remove a page from the Pages tab, you will need to manually delete the page from the DataHub WebView installation directory. Pages are stored in a user directory that matches your login name. If you log in as **`admin`**, the pages you save will be located here:

```
C:\Program Files\Cogent\Cogent DataHub\Plugin\WebServer\html\Silverlight\Pages\Users\admin
```

2.3.2. Page Size

To change page size enter a number of pixels for the width and heightin the W: and H: entry fields at the bottom right corner of the editing window.

2.3.3. The Grid

To show gridlines click the Show gridlines button  , or type **`True`** in the Tools menu, Options dialog, Design Mode list, Show gridlines entry.

To change the grid size enter a number of pixels in the Grid entry field at the bottom right corner of the editing window.

To snap controls to the grid click the Snap to grid button , or type `True` in the Tools menu, Options dialog, Design Mode list, Snap to grid entry.

2.3.4. View and Zoom

To view the page at a specific size, click the Page Zoom button Fit, which opens a list of zoom levels, and choose the level you need. There are several other ways to zoom in and out, to make resizing the page convenient.

To fit the page into the window, click the Fit button , use the View/Zoom menu, or press **Ctrl** + **Shift** + **Z**.

To zoom in and out click the Zoom In or Zoom Out buttons, use the View menu, or press **Ctrl** + **Shift** and spin the mouse wheel up or down.

To focus your zoom on a specific location in the page, click the Set Zoom Focal Point button , or click that option in the View/Zoom menu. Then click the page where you want to focus your zoom.

To zoom on a control click the control, and then from the View/Zoom menu check the Zoom on Selected Control option.

2.3.5. Edit and Run Modes

At log in DataHub WebView checks for user and editing permissions. If you log in with a user name that does not have editing permissions, then DataHub WebView will automatically open in Run Mode and you will not be able to switch to Edit Mode. If you do have permissions to edit and create new pages, then DataHub WebView will open in Edit Mode.

To enter Run Mode from Edit Mode, click on the Enter Run Mode button or press **Ctrl** + **Shift** + **R**.

To exit Run Mode and return to Edit Mode, click on the Exit Run Mode button or press **Ctrl** + **Shift** + **R**.

To display the Kiosk View, which removes the toolbars in Run Mode, go to Edit Mode, and from the Edit menu, select Run Mode Options and check Use Kiosk View.

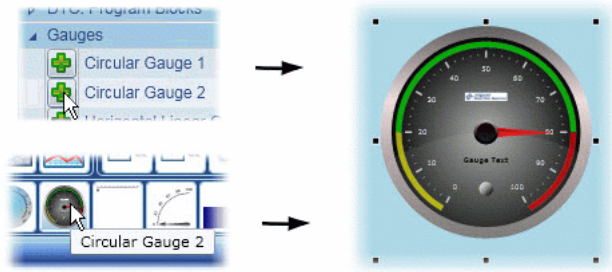
If you have configured Run Mode to display in Kiosk View, there will be no toolbar at the top of the page, so you will need to use **Ctrl** + **Shift** + **R** to exit Run Mode.

2.4. Controls

2.4.1. Add, Copy, Resize, and Move Controls

To add a new control to your page, you have two options:

1. In the Controls tab, find the control in its appropriate group and click the Add button .



2. Click that control's button in the Control Toolbar at the bottom of the page to put the control in the center of the page.

   Pressing the **Shift** key while you click on the control button lets you manually position the control on the page. While positioning the control, pressing **Ctrl** + **Shift** and turning the mouse wheel allows you to zoom in and out.

To copy a control select it, and from the Edit menu choose Copy, or press **Ctrl** + **Shift** + **C**.

To paste a control select it, and from the Edit menu choose Paste, or press **Ctrl** + **Shift** + **V**.

To resize a control select it and resize it with one of the black resize handles. Or, enter a width and height in the W: and H: entry fields at the bottom right corner of the editing window.

To move a control select it, and move it with the mouse. For precise movements, you can show gridlines, and snap controls to the grid. Or, enter X and Y coordinates (distance from the top-left corner) in the X: and Y: entry fields at the bottom right corner of the editing window. Alternatively, you can use the cursor keys as follows:

- move by 1 pixel: **Ctrl** + arrow keys
- move by 10 pixels: arrow keys
- move by 100 pixels: **Ctrl** + **Shift** + arrow keys

2.4.2. Grouping Controls

Controls can be grouped together, forming essentially a single large control.

To group controls click the controls that you want to be in the group, and then click the Group button . This is also available from the right-click pop-up menu.

To ungroup controls click the group that you want to ungroup, and then do choose one of the following:

- Click the Ungroup button to preserve the size and position changes you made while the controls were grouped. This is also available from the right-click pop-up menu.

- Click the Cancel Group button  to discard all changes that you made to the group. This is also available from the right-click pop-up menu.

To access and change the properties of any control in a group

1. Select the group.
2. Click the **Ctrl** key on your keyboard.
3. Click the control in the goup that you need to access.

If you have a group within a group, you can access a single cell like this:

1. Select the outer group.
2. Click the **Ctrl** key on your keyboard.
3. Click the inner group that you need to access.
4. Click the **Ctrl** key *twice* on your keyboard.
5. Click the control that you need to access.

2.4.3. Control Properties

Each control has a number of properties associated with it. There are common properties (see below) shared with each control, as well as other properties unique to that particular control. When you click on a control, the Properties tab opens, listing the properties by groups, with the Basic Properties group viewable. The Basic Properties are the ones that you'll probably use most often. Below these are groups of other properties unique to the control, followed by the common properties.

To change the value of a property click the control and type in or select the value.

To bind the properties of a control to a DataHub point or another control, please refer to Section 2.5, *Property Binding*.

2.4.4. Common Properties

The common properties shared by all controls include:

Background, Border and Margin

Background

A color for the background of the control.

Border

A color for the border of the background of the control.

Border Thickness

The thickness of the background border, in pixels, for the left, top, right, and bottom borders, respectively. Adding to this thickness will reduce the visible size of the control.

Border Corner Radius

The radius of each corner of the background border, in pixels, for the top-left, top-right, bottom-right, and bottom-left corners, respectively.

Content Margin

The width of the margins, in pixels, for the left, top, right, and bottom borders, respectively. Adding to this width will reduce the visible size of the control.

Background Image

Image File

A file to use for a background. To add your own images to the DataHub WebView library, you need to copy your image files to the following directory:

```
C:\Program Files\Cogent\Cogent DataHub\Plugin\WebServer\html\Silverlight\Images
```

If you copy your images to a subdirectory, then they will appear together in the image file selector within the editor.

Image Width

The width of the image, in pixels.

Image Height

The height of the image, in pixels.

Image Alignment

Aligns the background image with a corner, side, or middle of the control, or stretches it to fill the whole area.

Image Opacity

A number between 0 (transparent) and 1 (fully opaque).

Image Margin

The width of the margins, in pixels, for the left, top, right, and bottom sides of the image, respectively. Adding to this width will reduce the visible size of the image.

Image Rotation (degrees)

A number of degrees to rotate the image to the right.

Flip X-Axis

Flip the image top-to-bottom.

Flip Y-Axis

Flip the image right-to-left.

Content Visibility and Appearance

Visible in Run Mode

Will this control be visible in Run mode?

Content Opacity

A number between 0 (transparent) and 1 (fully opaque).

Static Rotation (degrees)

A number of degrees to rotate the control to the right.

Maintain Uniform Size for Static Rotation

Will the control change size to fit its container when rotated?

Clip Content

Not yet documented.

Flip X-Axis

Flip the content of the control top-to-bottom.

Flip Y-Axis

Flip the content of the control left-to-right.

Position and Size

Left

A number of pixels specifying the distance of the top left corner of the control from the left side of the page.

Top

A number of pixels specifying the distance of the top left corner of the control from the top of the page.

Width

A number of pixels specifying the width of the control.

Height

A number of pixels specifying the height of the control.

Content Animation

Is Content Rotating

Specifies if the control content rotates.

Animated Rotation (rpm)

Specifies the speed, in rotations per minute, of the control content.

2.4.5. Controls Listed by Category

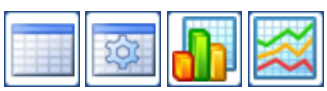
WebView controls are arranged in the following categories. You can click on the control icon for more information about the specific control.

-

Alarms

-

Charts



-

Common Input Controls

.

Configuration

.

DateTime

.

DTC: Palettes

.

DTC: Program Blocks

.

Gauges

.

Media Controls

.

Navigation

.

Notification

.

Shapes

.

Symbols

.

Text Controls

.

Tutorial Controls

2.5. Property Binding

DataHub WebView provides extensive support for property binding, allowing the properties of one control to be bound to other controls, or to DataHub point values. These three binding options are available on many control properties:

- **None** allows you to enter a static value, not bound to anything.
- **Point** lets you bind a control property to the value of any DataHub point. When the point changes value, the property changes value with it. For example, you can bind the current value of a gauge to a DataHub point, animating the gauge indicator with live data.
- **Simple** lets you bind a property of a *reference* control to the property of a *linked* control. Whenever you change the value of the property on the reference control, the property will change on any linked controls. For example, if you bind the color and size of several buttons to a single, reference button, whenever you change the color or size of the reference button, all the other buttons will change too. Any control with bindable properties can be used as a reference control, a linked control, or both.

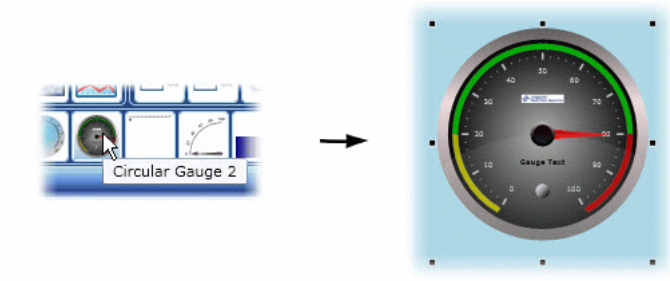
There are several features of property binding that facilitate page design, such as:

- **Simple bindings are universal** so that any control can be bound to any other control, as long as the bindings are compatible (eg. value-to-value, color-to-color, etc.).
- **The Property Picker shows compatible properties** for simple bindings.
- **For simple bindings, properties can be both references and links** which means that bindings can be chained, allowing controls to act simultaneously as a references to other controls or be linked to them, in any combination.
- **Simple and Point bindings can be combined** in a single control, which allows you to get your data from the DataHub, and the control appearance from another control.
- **Copying a control** will copy all property bindings. So, if you need a number of similar controls to share certain properties, you can make a reference control and one linked control, and then copy the linked control as many times as needed.

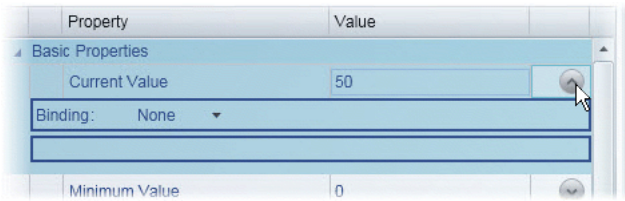
2.5.1. DataHub Point Binding

To create a Point binding you can follow this example, where we bind the indicator value of a gauge to a DataPid point:

1. Start the DataPid program that is included in your Cogent DataHub archive to generate some test data.
2. Open a blank page and add a Circular Guage 2 control.

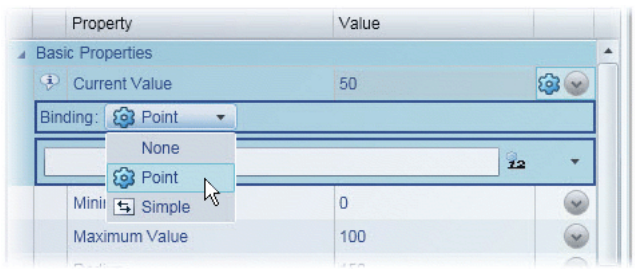


3. Open the Basic Properties of the gauge, and click the binding button on the right side of the Current Value row.



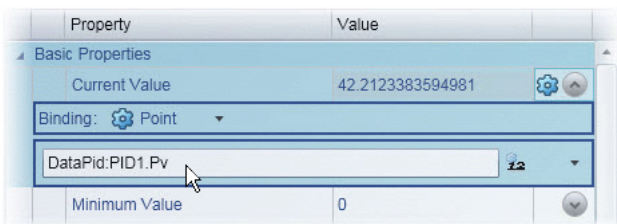
A Binding selection box will appear.

4. In the Binding selection box, click the down arrow to open the list, and select Point.



This will activate the point selection entry field.

5. To connect to the DataPid point DataPid:PID1:Pv, you can enter just Pv. All the points that have "Pv" in their names will appear.



Select the point DataPid:PID1:Pv. Once selected, the data will start updating in the value entry, and the gauge needle will start to move.

2.5.2. Point Attribute Selection

When binding a point, DataHub WebView provides a choice of point attributes that are available for the control you are working with. These attributes may include, in various formats, the point name, the quality of the connection, the timestamp of the most recent value change, and the value of the point itself. These are chosen through right-clicking on the small arrow to the right of the text-entry field, and selecting from the drop-down menu that appears:

Here are the possible choices, availability depends on the control you are working with:

Menu Item	Point Attribute	Example
n.ab	The full point name, no domain.	`PID1.Pv`
ab	The abbreviated point name.	`Pv`
d:n.ab	The full point name, with domain name.	`DataPid:PID1.Pv`
Q	The point quality, in plain text.	`Good`
Q#	The code for the point quality.	`192`
	The full date and time.	`09/02/2011 13:09:15`
12	The point value.	`71.33489150492`
VQT	The point value, quality, and timestamp.	`71.334 {Good,13:09:15.32}`

The point value (12) is the default, and used for most bindings, while the other options are available if necessary. Certain controls that need to reference timestamps, such as the Trend control, require using VQT.

2.5.3. Simple Binding - Property Picker

There are two ways to create a simple binding, either with the Property Picker, or by copy and paste. The following example shows how to use the Property Picker. In this example, we use a hexagon as the reference control, and a rectangle as the linked control.

1. On a blank page, add one hexagon and one rectangle, using the Simple Path and Simple Rectangle controls.

2. Select the hexagon (the reference control), and in the Basic Properties, Fill Color choose a color other than White, such as SandyBrown.

3. Select the rectangle (the linked control), and in the Basic Properties click the binding button on the right side of the Fill Color row.

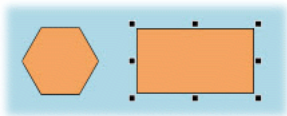
A Binding selection box will appear, above the color selection dialog.

4. In the Binding box, select Simple. A text-entry box will appear, and to the right, the Property Picker icon.

5. Click the Property Picker icon. The Properties area will turn gray, and when you move the mouse over the page, it will take the shape of the Property Picker.

6. Click the hexagon. This will open the Property Picker menu. From here you can choose which property of the hexagon you wish to give to the rectangle.
7. From the Basic Properties submenu, choose Fill Color.

The rectangle will change to SandyBrown. It is now bound to the color of the hexagon, and will change whenever that color gets changed.



There are two options on the Property Picker menu:

Filter by Matching Type

Reduces the list of properties in the reference control to only those that are also present in the linked control. This helps you quickly identify which properties you can actually bind to this particular control.

Include Common Properties

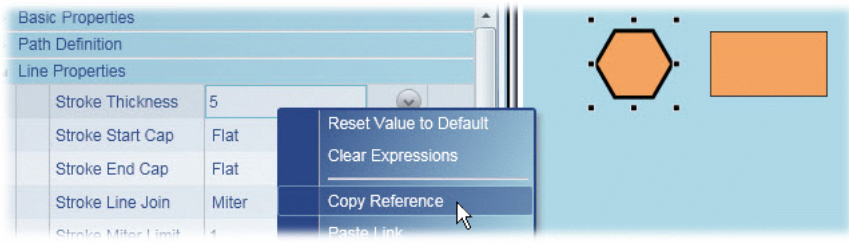
Hides or displays the Common Properties in the list.

2.5.4. Simple Binding - Copy and Paste

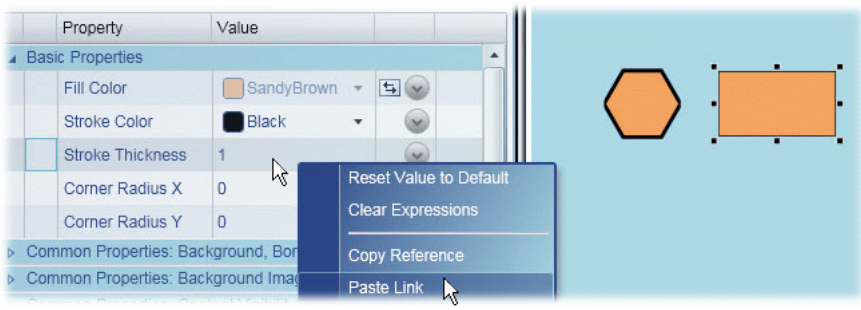
Another way to create a simple binding is by copying a reference for a binding and then pasting the link. This is expecially useful if you know what properties you need to bind, or if you need to link many controls to a single reference control. Below is an example, based on the controls created in the example above. In this example, the hexagon again is the reference control, and the rectangle is the linked control.

1. Select the hexagon (reference control), and in the Line Properties, change the Stroke Thickness to 5. You should see the border of the hexagon become 5 pixels thick.

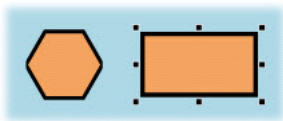
2. Right-click in the text box where you entered 5, and from the drop-down menu, select Copy Reference.



3. Select the rectangle (linked control), and in the Basic Properties, Stroke Thickness, Value entry field, right click, and from the drop-down menu, select Paste Link,



The border of the rectangle will change to 5 pixels. It is bound to the width of the hexagon's border, and will change whenever that gets changed.



2.6. Adding Images

To add your own images to the DataHub WebView library, you need to copy your image files to the following directory:

```
C:\Program Files\Cogent\Cogent DataHub\Plugin\WebServer\html\Silverlight\Images
```

If you copy your images to a subdirectory of that directory, then they will appear together in the image file selector within the editor.

Chapter 3. DataHub WebView Scripting

Please refer to DataHub WebView Scripting for an overview and reference for the scripting capabilities and features in DataHub WebView.

Chapter 4. Dynamic Binding

Using WebView Scripting, it is possible to dynamically bind DataHub points to symbols and controls, and use these capabilities to make template pages.

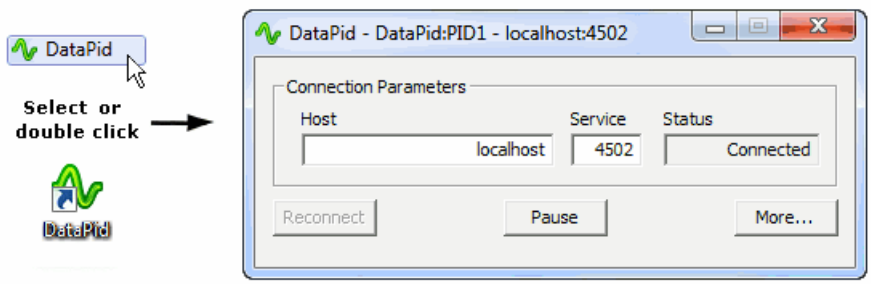
4.1. Dynamic Point Binding

Dynamic point binding allows you to change the DataHub points associated with a control at run time. This tutorial shows two different ways to do this (simple array and dynamic list), using two controls - ComboBox and ListBox.

4.1.1. Combo Box control

Binding a Combo Box to a Circular Gauge using a simple array

1. Start the DataPid program from the Windows Start menu, the command line, or by clicking on the desktop icon.



   As soon as DataPid starts, it attempts to connect to a DataHub and begins generating data.

2. Open DataHub WebView, open a new page, and add a ComboBox control to the page.

3. In the Basic Properties of the Combo Box, for the Items Source, select the Script binding type and edit the default script to read:

```
["Sp", "Mv", "Pv"];
```

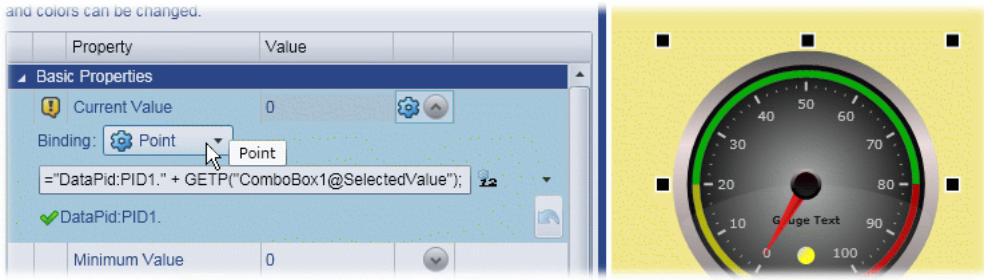
   In this line of script, the [and] characters tells the WebView scripting engine that this is an array of comma-separated strings, each of which should be assigned to one value in the Combo Box control.

   As an alternative to using an array, it is possible to use a script to list the items, as explained in the List Box control section, below.

4. Add a Circular Gauge 2 control to the page.

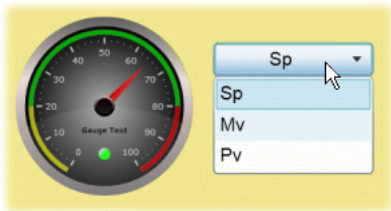
5. In the Basic Properties of the Circular Gauge 2, for the Current Value, select the Point binding type and make a script:

```
="DataPid:PID1." + GETP("ComboBox1@SelectedValue");
```



When a line of script is entered as a point binding as we see here, the = sign tells the WebView scripting engine that the point name will be assigned. In this example, this allows the name of the point to be constructed by concatenating strings. The `GETP` function gets the property of a control. The syntax for the `GETP` argument is a string consisting of the name of the control, an `@` symbol, and the name of the parameter that you need to get.

6. Switch to Run Mode and choose selections from the Combo Box and see the results in the Circular Gauge.

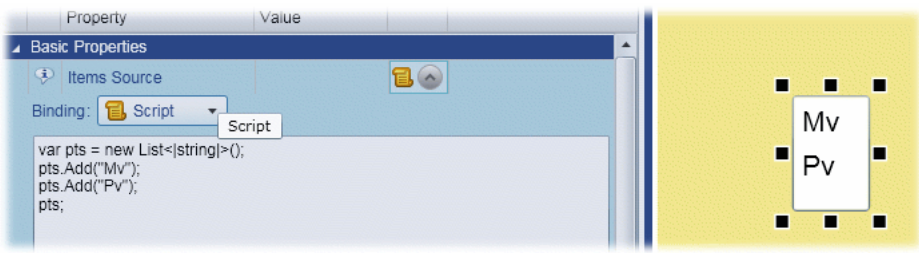


4.1.2. List Box control

Binding a List Box to a Trend Chart using a script

1. Ensure that the DataPid program is running, or start it from the Windows Start menu, the command line, or by clicking on the desktop icon.
2. Open DataHub WebView, open a new page, and add a List Box control to the page.
3. In the Basic Properties of the List Box, for the Items Source, select the Script binding type and edit the default script to read:

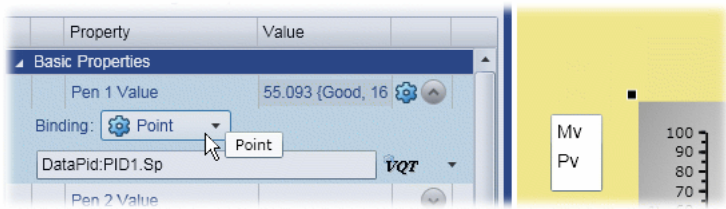
```
var pts = new List<|string|>();
pts.Add("Mv");
pts.Add("Pv");
/* More strings can be added ... */
pts;
```



This example script creates a list of strings, and then adds two strings (point names in this case) to the list. The list can hold any number of strings. The final line of the script calls the `pts;` variable, which causes the list of strings to be passed to the List Box as its source of items.

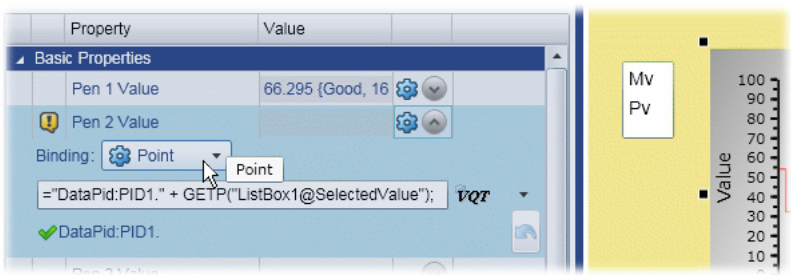
As an alternative to using a script, it is possible to use a an array to list the items, as explained in the Combo Box control section, above.

4. Add a Trend Chart (3 pens) control to the page.
5. In the Basic Properties of the Trend Chart control, for the Pen 1 Value, select the Point binding type enter the value **`DataPid:PID1.Sp`**.

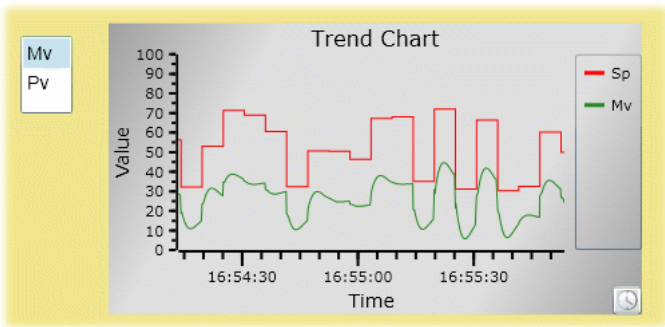


6. For the Pen 2 Value, select the Point binding type and make a script:

```
="DataPid:PID1." + GETP("ListBox1@SelectedValue");
```



7. Switch to Run Mode and choose selections from the List Box and see the results in the Trend Chart.



4.2. Dynamic Control and Symbol Binding

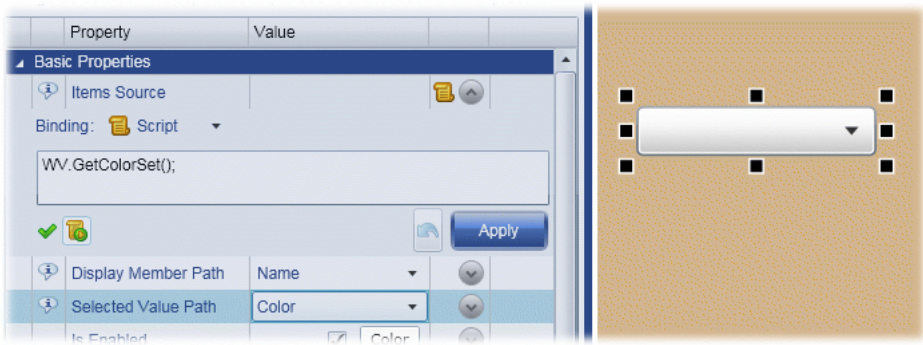
Dynamic control and symbol binding allow you to change bindings on a control or symbol at run time. This tutorial shows how to use a Combo Box to change the color of a Shining Light, and then how to change a light switch symbol to appear as if it is being switched on and off.

4.2.1. Control Binding

Bind a Combo Box to a Shining Light

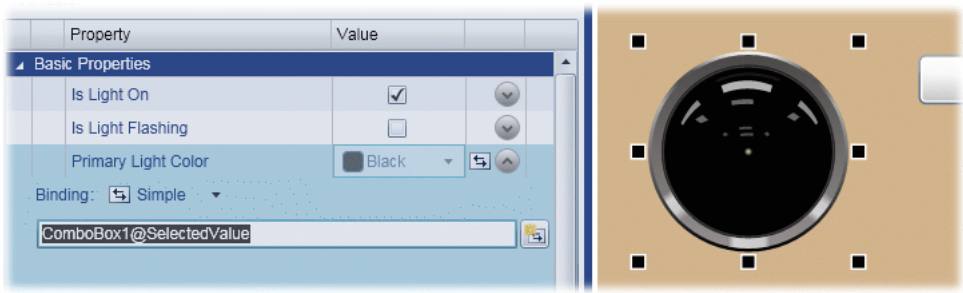
1. Open DataHub WebView, open a new page, and add a Combo Box control to the page.
2. In the Basic Properties of the Combo Box, for the Items Source, select the Script binding type and edit the default script to read:

```
WV.GetColorSet();
```

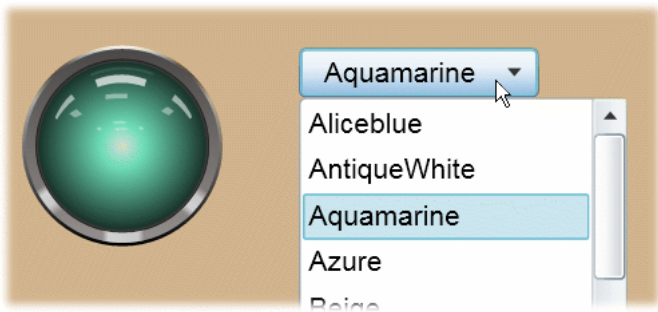


3. For Display Member Path, choose Name.
4. For Selected Value Path, choose Color.
5. Add a Shining Light control to the page.
6. In the Basic Properties of the Shining Light, for the Primary Light Color, select the Simple binding type and enter the value:

```
ComboBox1@SelectedValue
```



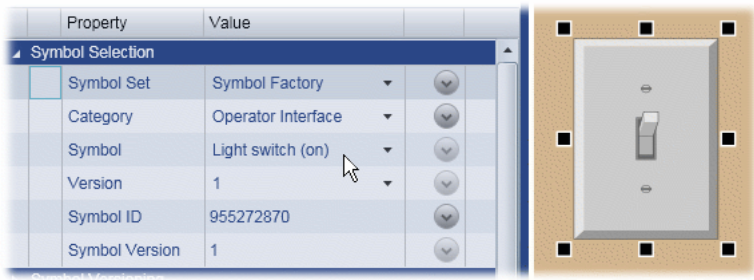
7. Switch to Run Mode and choose selections from the List Box and see the results in the Trend Chart.



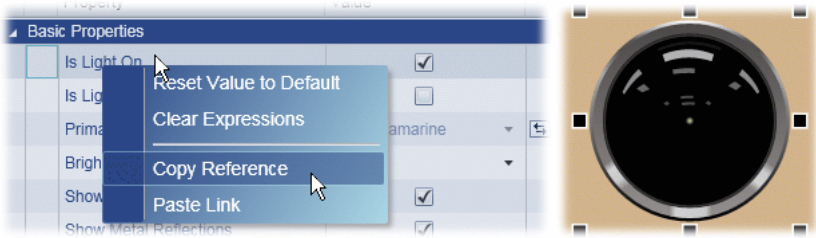
4.2.2. Symbol Binding

Change a Light Switch Symbol Binding to Simulate On and Off

1. Using the same page as above, add a Symbol control.
2. In the Symbol Selection for the Symbol Set, select Symbol Factory, for the Category, select Operator Interface, and for the Symbol, select Light switch (on).

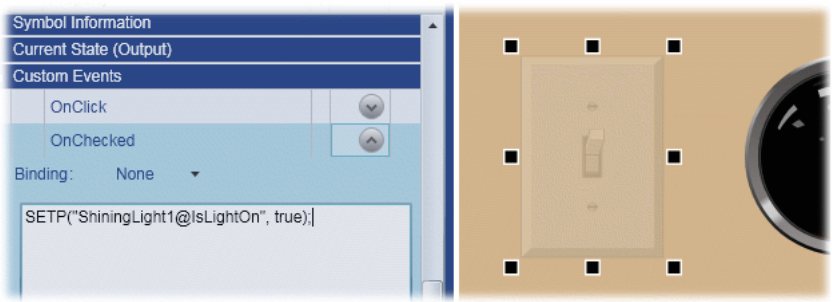


3. Select the ShiningLight control, and in the Basic Properties, right mouse click on Is Light On to copy the reference.



4. Select the Symbol (light switch) and in Custom Events, for OnChecked Event, enter:

```
SETP("ShiningLight1@IsLightOn", true);
```

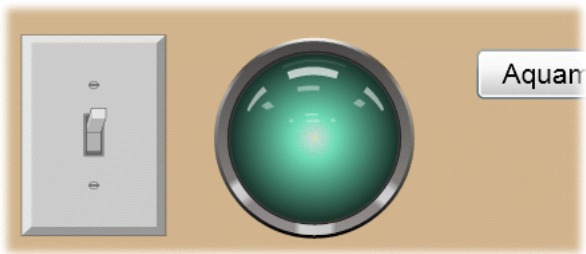


For this and the next step, the string `ShiningLight1@IsLightOn` is what you copied from the Shining Light control. You can simply paste it in.

5. In Custom Events, for OnUnChecked Event, enter:

```
SETP("ShiningLight1@IsLightOn", false);
```

6. Switch to Run Mode and click the light switch symbol to turn the Shining Light on and off.



This works OK, but it looks strange to have the light go off when the light switch is still in the ON position. We'll fix that next.

7. Select the Symbol (light switch) and in Custom Events, for OnChecked Event, add one more line:

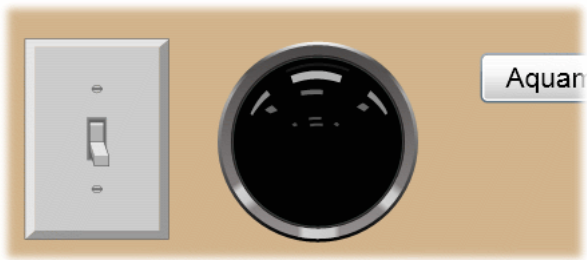
```
SETP("ShiningLight1@IsLightOn", true);
SETP("@SymbolID", 955272870);
```

For this and the next step, the string `@SymbolID` refers to the symbol itself.

8. In Custom Events, for OnUnChecked Event, add one more line:

```
SETP("ShiningLight1@IsLightOn", false);
SETP("@SymbolID", 1685125442);
```

9. Switch to Run Mode and click the light switch symbol to turn the Shining Light on and off.



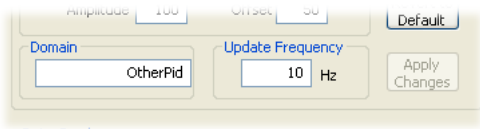
Now the symbol changes from Light switch (on) to Light switch (off) as the light goes on and off.

4.3. Creating a Template Page

This tutorial shows how to create a page where you can switch between data sources in a single display, at the click of a button.

Creating an Identical Data Source

1. With DataPid running, start another instance of DataPid. In the second DataPid instance, click the More... button to expose the DataPid Configurable Options
2. Change the Domain to **`OtherPid`** and click the Apply Changes button. Then press the Reconnect button.



If you look in the DataHub Data Browser, you should see a new data domain, `OtherPid` with data changing values. Now we need to add that data domain for DataHub WebView.

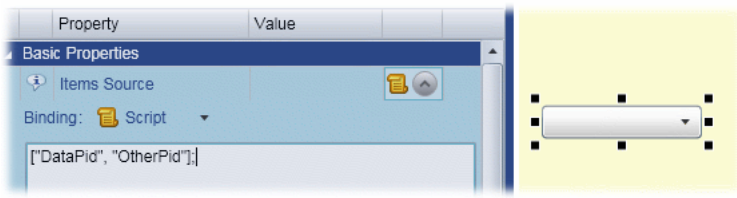
3. Go to the WebView option of the DataHub Properties window, and in the Data Domains Visible to WebView section, click the Refresh button.

4. The domain name `OtherPid` should appear in the list. Check the checkbox to make the OtherPid domain visible to DataHub WebView, and make its points accessible. Now we have two identical point sets with different data to demonstrate our template page.

Creating the Template Page

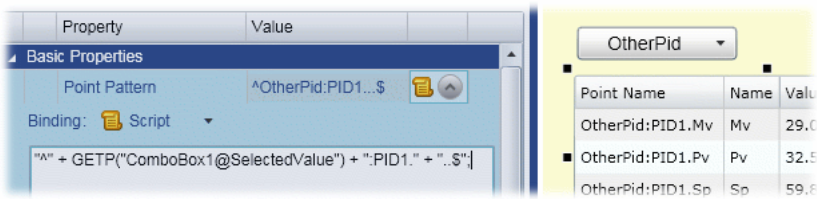
1. Start DataHub WebView, open new page, and add a Combo Box control  to the page
2. In the Basic Properties of the Combo Box, for the Items Source, select the Script binding type and edit the default script to read:

```
["DataPid", "OtherPid"];
```



3. Add a Point Data Table control to the page.
4. In the Basic Properties of the Point Data Table, for the Point Pattern, select the Script binding type and edit the default script to read:

```
"^" + GETP("ComboBox1@SelectedValue") + ":PID1." + "..$";
```



The ^ symbol allows any combination of prefix characters, while the GETP expression pulls in the value from the ComboBox. The `..$` string allows for any combination of suffix characters after the required `PID1.` string.

5. To make the table easier to read, in the Table Columns property, you can make the following columns in the table visible: Point Name, Display Name, Value, and Quality.
6. Switch to Run Mode and change the data domain in the Combo Box from DataPid to OtherPid to view the two different sets of data points.

Adding a Trend Chart to the Page

1. Add a Trend Chart (3 pens) control to the page.
2. In the Basic Properties of the Trend Chart, for the Pen 1 Value, select the Point binding type and enter:

```
=GETP("ComboBox1@SelectedValue") + ":PID1.Sp";
```

When a line of script is entered as a point binding as we see here, the = sign tells the WebView scripting engine that the point name will be assigned. In this example, this allows the name of the point to be constructed by concatenating strings. The GETP function gets the property of a control. The syntax for the GETP argument is a string consisting of the name of the control, an @ symbol, and the name of the parameter that you need to get.

3. To give a better shape to the trend line for the Sp point, in the Pen1 Properties check the boxes for the Pen 1 Is Square and the Pen 1 Auto Extend options.
4. Back in the Basic Properties, for the Pen 2 Value select the Point binding type and enter:

```
=GETP("ComboBox1@SelectedValue") + ":PID1.Mv";
```

And for the Pen 3 Value select the Point binding type and enter:

```
=GETP("ComboBox1@SelectedValue") + ":PID1.Pv";
```

You should now see all three trends.

5. Switch to Run Mode, and again change the data domain in the Combo Box from DataPid to OtherPid. Now both the table and the trend chart alternate between the two different sets of data points.

Following this example, you can build a page to display multiple identical sets of data in any group of controls. with the ability to switch between data sets at the click of a button.

Chapter 5. Customizing DataHub WebView

5.1. Simple Branding

Simple branding allows a DataHub WebView administrator to create a branded login page, to specify the application title, and to add company-specific links to the Help menu. Simple branding is intended for site administrators who want to provide their users with a general sense that the application is an integral part of the company's software solution for systems and processes.

In addition to simple branding options, OEM branding allows a distributor to replace Cogent-specific logos and references with icons and text of its own choosing. This level of branding is intended for Cogent partners and licensed distributors who need to re-brand the application to leverage their own corporate brand and to capitalize on specific market opportunities. For more information on OEM Branding, please contact http://www.cogentdatahub.com/Contact.htmlCogent.

Prerequisites

- An understanding of XML.
- Familiarity with XAML.
- Access to the Cogent DataHub installation directory and files.

5.1.1. Creating a custom login page

1. Using XAMLPad (or your favorite XAML editor), create a standalone XAML file that contains the details of your login page. The file must be named `LoginPage.xaml` and stored in a subdirectory of the `Branding` installation directory:

   ```
   C:\Program Files\Cogent DataHub\Plugin\WebServer\html\Silverlight\Branding\
   ```

   Your page may contain any XAML you wish, including graphics, controls and animation. For your convenience, you may wish to start by copying the sample `LoginPage.xaml` file from the `Template` subdirectory.

2. Your page must include a `DataHubConnection` element (and associated XML namespace). This element provides support to authenticate the user (i.e., username and password). Even if you want to bypass the login page with automatic credentials, this element must be included in `LoginPage.xaml`.

   To add the `DataHubConnection` element:

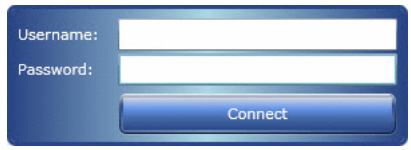
   a. At the top of `LoginPage.xaml`, on the root elememt, add an XML namespace reference to the `DataHubWebViewApplicationInfrastructure` assembly:

      ```
      <Grid
          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
          xmlns:dhc="clr-namespace:Cogent.DataHubWebView;assembly=DataHubWebViewApplicationInfrastructure"
          ...
      ```

   b. Insert the `DataHubConnection` element where you want to place the credential prompt, for example:

      ```
      <dhc:DataHubConnection Background="Transparent"/>
      ```

      When the user navigates to your published ASP/HTML URL to launch DataHub WebView, the `DataHubConnection` element will appear:



Simple branding does not support customizing the style of this element. You can set visibility, margin and other standard attributes, but you can not change the text, colors or button style.

5.1.2. Specifying text, icon, and URL targets

1. Using your favorite editor, create a standalone XML file to contain the branding settings for the application. The file must be named `Branding.xml` and stored in a subdirectory of the `Branding` installation directory:

   ```
   C:\Program Files\Cogent DataHub\Plugin\WebServer\html\Silverlight\Branding\
   ```

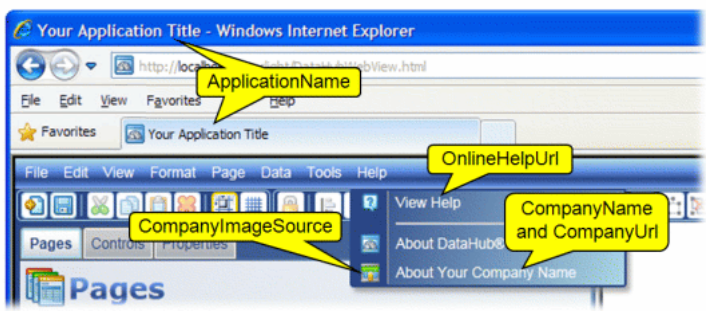
2. For your convenience, you may wish to start by copying the sample `Branding.xml` file from the `Template` subdirectory.

   ```
   <c:Branding
     xmlns="http://schemas.microsoft.com/client/2007"
     xmlns:c="clr-namespace:Cogent.DataHubWebView;assembly=DataHubWebViewApplicationInfrastructure"

     ApplicationName="Your Application Title"

     CompanyName="Your Company Name"
     CompanyImageSource="Branding/Template/CompanyIcon.png"
     CompanyUrl="http://www.yourcompany.com"

     OnlineHelpUrl="http://www.yourcompany.com/Docs/ProductOverview.html"

     Version="1.0" />
   ```

3. Do not change the XML namespace references (`xmlns` and `xmlns:c`) or the names of the `Branding` attributes (e.g., `ApplicationName`).

4. Replace the placeholder text and URLs with your branding information:

   - `ApplicationName`: incorporated into the browser title
   - `CompanyName`: incorporated into the Help menu's About item text.
   - `CompanyImageSource`: the icon for theHelp menu's About item text.
   - `CompanyUrl`: the target for the Help menu's About item text.
   - `OnlineHelpUrl`: the target for the Help menu's About item text.



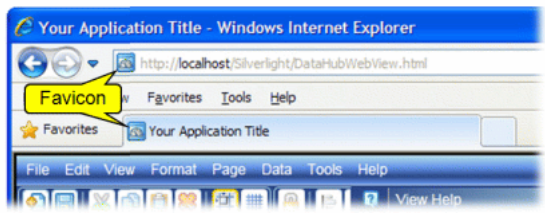
5. Do not change the `Version` attribute or its value.

5.1.3. Adding a favorite icon

1. When accessing website content, browsers look for a file named `favicon.ico` in the website root directory:

   ```
   C:\Program Files\Cogent DataHub\Plugin\WebServer\html\
   ```

   If found, this icon is used in several places to represent your website, such as the browser address bar, Favorites list, etc.



2. To use your own icon, replace the DataHub WebView default `favicon.ico` file deployed during a typical installation.

For more information on `favicon.ico`, see http://en.wikipedia.org/wiki/Favicon.

5.1.4. Testing the results

1. Ensure your `Branding.xml` and `LoginPage.xaml` files (and all local resources referenced by the login page) have been copied to a subdirectory of the `Branding` installation directory:

   ```
   C:\Program Files\Cogent DataHub\Plugin\WebServer\html\Silverlight\Branding\
   ```

2. Open the Cogent DataHub Properties window and select the WebView option. Ensure the Use a custom branding directory box is checked and select your branding directory from the dropdown list. If your directory does not appear, confirm you have completed the previous step.

   In most cases, an administrator sets the DataHub WebView launch configuration from the DataHub Properties window. However, if your users navigate to a custom ASP/HTML page to launch DataHub WebView, you will need to add a key-value pair to the `initParams` property of the Silverlight control in the custom ASP/HTML page. By default, this file is `DataHubWebView.asp` and located in the root:

   ```
   C:\Program Files\Cogent DataHub\Plugin\WebServer\html\Silverlight\Branding\
   ```

   The `initParams` key is **`brandingdirectory`** and the value is the name of the subdirectory. For example, if you save your `Branding.xml` and `LoginPage.xaml` files in a directory named ABC (under the `Branding` installation directory), your configuration would look something like this:

   ```
   <object id="Object1" data="data:application/x-silverlight-2,"
             type="application/x-silverlight-2" width="100%" height="100%">

     <param name="source" value="Bin/Debug/DataHubWebView.xap"/>
     <param name="initParams" value="domains=default;...,brandingdirectory=ABC" />
    ...
   </object>
   ```

3. Launch DataHub WebView. It should present your custom login page. After entering your credentials, you should see that the application references the settings you specified in the `Branding.xml` file.

5.2. Initialization Parameters

DataHub WebView is implemented as a Microsoft Silverlight object embedded in a web page. Consequently, it is possible to apply formatting and parameters to the <object> tag in the HTML of that web page. This allows the administrator of the DataHub WebView installation to customize the user experience, for example to set background color, size and position of the DataHub WebView object within the page. In addition it is possible to pass parameters to DataHub WebView from the web page to customize the behaviour of the DataHub WebView application. This document deals with these various initialization parameters.

The `initParams` Parameter

The Silverlight object recognizes a parameter on the <object> tag called *`initParams`*. In its simplest form, it is specified in HTML like this:

```
<object id="DataHubWebView" ...>
  <param name="initParams" value=""/>
</object>
```

The values of the initialization parameters can be inserted into the `value` attribute of the <param> tag as a comma-separated list of parameters. Each parameter consists of a parameter name, followed by the equal sign, (=), followed by the parameter value.

5.2.1. Accessing Parameters from the DataHub Properties Window

The Cogent DataHub Properties window allows you to specify some of the possible intialization parameters that are accepted by DataHub WebView. You can access these parameters in HTML as follows:

1. Ensure that the HTML page containing the DataHub WebView object has the extension `.asp` instead of `.html`. This will allow the Cogent DataHub to execute Gamma scripts specified in the page when the web client loads the page. The default URL for starting DataHub WebView is `http://localhost/Silverlight/DataHubWebView.asp`.
2. Add the following HTML to the page:

   ```
   <%
   require ("WebViewSupport.g");
   local initparams = WebViewInitString();
   %>
   ```

3. Modify the `initParams` in the DataHub WebView <object> tag as follows:

   ```
   <object id="DataHubWebView" ...>
     <param name="initParams" value="<%= initparams %>"/>
   </object>
   ```

Every time the page is loaded, the DataHub WebView `initParams` will be computed from the settings in the DataHub Properties window.

5.2.2. Adding Custom Parameters

To customize the initialization parameters, simply add your own definitions to the `initParams` <param> tag. For example:

```
<param name="initParams" value="Page=MyStartPage,StartInRunMode=true" />
```

You may combine your own custom parameters with those specified in the DataHub Properties window simply by adding your own parameters to those read from the Properties window. For example:

```
<param name="initParams" value="<%= initparams %>,Page=MyStartPage,StartInRunMode=true" />
```

The last value of a parameter will take precedence, so if you insert your parameters after the <%= initparams %> ASP code, as in the example above, then your parameters will override those specified in the Properties window.

5.2.3. Specifying Parameters in the Page URL

Some WebView initialization parameters can be specified in the URL of the web page, starting with a question mark (?). Multiple parameters are separated by ampersands (&). For example:

```
http://localhost/Silverlight/DataHubWebView.asp?StartInRunMode=true

http://localhost/Silverlight/DataHubWebView.asp?username=admin&password=admin
```

The **URL** columns in the tables below indicate which parameters can be specified in this way.

5.2.4. Parameter List

The following tables show all of the initialization parameters available in DataHub WebView. The parameter names are not case sensitive, but the parameter arguments may be, depending on the meaning of the argument. The **Properties** column indicates whether the parameter can be set via a selection in the Cogent DataHub Properties window. The **URL** column indicates whether the parameter can be set in the URL for the web page.

Table 5-1. Connection

Parameter	Value	Properties	URL	Description
TcpPort	Integer	Tunnel/ Mirror	No	This is the port number used to connect to the DataHub data feed. The default is `4502`.
WebPort	Integer	Web Server	No	This is the HTTP port number on which the DataHub Web Server is listening. The default is `80`.
Domains	Stringlist	DataHub WebView	No	This is a list of data domain names separated by semicolon (`;`) characters.
Host	String	No	No	This is determined automatically by DataHub WebView from the URL entered into the client's web browser. It should not normally be specified explicitly.

Table 5-2. Login

Parameter	Value	Properties	URL	Description
UserName	String	No	Yes	The user name. If this is specified then DataHub WebView will bypass the login screen and automatically log in using this user name.
Password	String	No	Yes	The password matching *UserName*.
FullLogin	True/False	No	No	If this is true, then the port, host and domain information will be requested in the DataHub WebView login dialog.

Table 5-3. Initial View

Parameter	Value	Properties	URL	Description
Page	String	DataHub WebView	Yes	The DataHub WebView page to show when the user first logs in. This page will be loaded even if the user starts in design mode. The page name must include any path components separated by / characters. The page name does not include an extension. Example: Users/admin/mypage
PageData	*name*=*value* pairs	DataHub WebView	Yes	These values become global variables in the script context. Each *name=value* pair is separated by a comma, as in: PageData=a=5,b=6. Please see Page Data in the DataHub WebView Scripting manual for more information.
StartInRunMode	True/False	DataHub WebView	Yes	If this parameter is true, DataHub WebView will enter Run mode when the user logs in, otherwise it will enter Design mode if the user has permission to do so.
UseKioskView	True/False	DataHub WebView	No	If this parameter is true, DataHub WebView will start in kiosk mode, removing the menu and icon bars.
FreezeScreenWhileLoadingPage	True/False	No	No	If this is true (the default) then the current page is frozen and the new page is loaded in the background. If this is false then the current page is erased and the new page is loaded in the foreground, with controls on the screen appearing as they load.
DisableDesignMode	True/False	DataHub WebView	No	If this parameter is true, DataHub WebView will not allow the user to enter Design mode, even if the user has permission to do so.
HideInitializationElements	True/False	No	Yes	If this parameter is true, no wallpaper is displayed during initialization of the DataHub WebView application.

Parameter	Value	Properties	URL	Description
EnableBrowserScripts	True/False	DataHub WebView	No	If this parameter is true, DataHub WebView will be allowed to execute Javascript code in the hosting browser. The default is `False`.

Table 5-4. Branding

Parameter	Value	Properties	URL	Description
BrandingFolder	String	DataHub WebView	No	The path to a folder to search for custom branding information. This folder is relative to the DataHub WebView installation's `Branding` directory. The path separator is the `/` character.

Table 5-5. Designer Controls

Parameter	Value	Properties	URL	Description
MakeDataPointsReadOnly	True/False	DataHub WebView	Yes	If this parameter is true, DataHub WebView will be unable to write data to the DataHub's data set, regardless of the configuration or user permissions.
DisablePageInformationButton	True/False	DataHub WebView	Yes	If this parameter is true, the page information button will not be displayed in Run mode.
ShowHiddenControls	True/False	No	No	If this parameter is true, controls that are normally hidden will be visible in Design mode. Hidden controls are controls that act as base classes for other controls, are deprecated, or are experimental. In any case, these controls should not be used by a page designer.
RunSilently	True/False	DataHub WebView	Yes	If this parameter is true, DataHub WebView will not display any error messages in Run mode.

Parameter	Value	Properties	URL	Description
HideLoadingPageMessage	True/False	DataHub WebView	Yes	If this parameter is true, DataHub WebView will not display a message each time a page is loaded.
ShowExitConfirmation	UnsavedChanges, Never, Always	None	Yes	This setting determines whether to prompt the user when he attempts to exit DataHub WebView. If set to `Never`, then the user will not be prompted on exit. If set to `Always` then the user will be prompted. If set to `UnsavedChanges` then the user will only be prompted if there are unsaved changed on the current page. The default is `Always`.

5.3. Adding Controls

This section outlines the steps to add a custom control, with custom behavior, that will be used inside the DataHub WebView Silverlight application.

This procedure is intended for control designers and power users who want to complement or extend the suite of controls that ship with DataHub WebView. For those who merely want to re-style an existing control, please refer to a soon-to-be-added section: Restyling a DataHub WebView Control.

Prerequisites

- An understanding of how to develop Microsoft .NET class libraries.
- Familiarity with Visual Studio 2010.
- An understanding of XML.
- Some familiarity with XAML.
- Access to the Cogent DataHub installation directories and files.

5.3.1. Preparing the Visual Studio project

The first step in adding a custom control is to create a Visual Studio project containing the default XAML style, `ControlTemplate`, and the control's code-behind class.

1. In Visual Studio 2010, create a new project:
   - `Template`: **`Silverlight Class Library`**
   - `Version`: **`Silverlight 4`**

- `Project Name:` ***`MyGauges`***

replacing *`MyGauges`* with the name of your project.

2. Delete `class1.cs` and add a new class to represent your control. For this documentation, we name the class *`GlossyGauge`*; which you can replace with your own class name.

```
public  class GlossyGauge
```

3. Now the XAML needs to be separated from the code-behind class. Create a new directory and add an XAML file to that directory, as follows:

   - New directory name: **`Themes`**
   - New XML file name: **`Generic.xaml`**

4. Replace the content of `Generic.xaml` with the following XAML code:

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:vsm="clr-namespace:System.Windows;assembly=System.Windows"
    xmlns:layout=
  "clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Layout.Toolkit"
    xmlns:toolkit="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Toolkit"
    xmlns:input="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Input.Toolkit"
    xmlns:controls="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:primitives="clr-namespace:System.Windows.Controls.Primitives;assembly=System.Windows"

    xmlns:local="clr-namespace:MyWebViewControls.MyGauges"
    >

    <Style TargetType="local:GlossyGauge">
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="local:GlossyGauge">
                    <Grid x:Name="ControlRoot" Background="Transparent">
                        <Viewbox>

                        </Viewbox>
                    </Grid>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
</ResourceDictionary>
```

5. Set the `local` XML namespace to match the namespace used for your control (i.e., where the *`GlossyGauge`* class resides).

```
xmlns:local="clr-namespace:MyWebViewControls.GlossyGauge"
```

6. Set the `TargetType` (for both the `Style` and `ControlTemplate` elements) to refer to your class:

```
TargetType="local:GlossyGauge"
```

7. Between the start and end `<Viewbox>` tags, paste the XAML code that represents your control (perhaps created using Microsoft Expression Blend).

8. Change the definition of your class to work with the `Style` and `ControlTemplate` (in `Generic.xaml`) by inheriting from `Control` and by using the `partial` modifier.

```
public partial class GlossyGauge : Control
```

9. Add a default constructor to your class and set the `DefaultStyleKey` property to refer to the class's type. This triggers the framework to invoke the `OnApplyTemplate` method and apply the matching XAML template in the project's `\Themes\Generic.xaml` file.

```
public GlossyGauge()
{
```

```
        this.DefaultStyleKey = typeof(GlossyGauge);
    }
```

10. For illustration, add a `Dependency Property`, which will manage the `BackgroundColor` of the control, and will be exposed to the user via Property Explorer.

    The `UpdateVisualState` method will be coded in an upcoming step.

```
public Color BackgroundColor
{
    get { return (Color)GetValue(BackgroundColorProperty); }
    set { SetValue(BackgroundColorProperty, value); }
}

public static readonly Dependency Property BackgroundColorProperty =
    Dependency Property.Register("BackgroundColor", typeof(Color), typeof(DHCommonGauge),
    new PropertyMetadata(Colors.Black, new PropertyChangedCallback(Dependency Property_Changed)));

private static void Dependency Property_Changed(
    DependencyObject sender, Dependency PropertyChangedEventArgs e)
{
    DHCommonGauge gauge = sender as DHCommonGauge;
    gauge.UpdateVisualState();
}
```

11. Define the `ControlTemplate` behavior, i.e., the interaction among the `Style` and `ControlTemplate` you defined in `Generic.xaml` and your control's properties and behavior.

```
private Grid ControlRoot;
public override void OnApplyTemplate()
{
    // Capture the Framework Elements from the ControlTemplate
    ControlRoot = GetTemplateChild("ControlRoot") as Grid;

    UpdateVisualState();
}

private void UpdateVisualState()
{
    // Logic to update the control, e.g., gauge range, current value, color, etc.
    // For example:
    ControlRoot.Background = new SolidColorBrush(this.GaugeBackgroundColor);
}
```

    For example, the above code:

    - Defines the `ControlRoot` element, which is set in the `OnApplyTemplate` method by referencing the named element in the `ControlTemplate`.
    - Updates the `Background` of the `ControlRoot` whenever the `UpdateVisualState` method is called, including when the `BackgroundColor Dependency Property` changes.

For more information about modifying custom control behavior, please refer to a soon-to-be-added section: Making your Custom DataHub WebView Control Host-Aware.

5.3.2. The XAML file to reference the control

You will need to create a standalone XAML file that references the Silverlight control.

1. Using your favorite editor, create a standalone XAML file that references the Silverlight control you created in your Visual Studio project. The file can be named anything you like. By convention, the file name refers to the Silverlight class (e.g., *`GlossyGauge`*`.xaml`).

   The root element is `<UserControl>` and four XML namespaces are required.:

- `xmlns` and `xmlns:x` are used by Silverlight to process XAML.
- `xmlns:live` is used by the DataHub WebView infrastructure to associate XAML elements with control Parameters (discussed in the next section).
- `xmlns:myControls` is used to reference your Visual Studio control.

```
<UserControl
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

    xmlns:live="clr-namespace:Cogent.DataHubWebView;assembly=DataHubWebViewApplicationInfrastructure"
    xmlns:myControls=" clr-namespace:MyWebViewControls.MyGauges;assembly=MyWebViewControls"
    >

  <myControls:GlossyGauge live:DynamicElement.ID="Gauge" />
</UserControl>
```

2. When ready to test, this file needs to reside in the DataHub installation directory hierarchy:

```
C:\Program Files\Cogent DataHub\Plugin\WebServer\html\Silverlight\Controls\
```

5.3.3. The XML file for public properties and behavior of the control

In this section, we will create a standalone XML file that encapsulates the public properties and behavior of the Silverlight control.

1. Using your favorite editor, you need to create a standalone XML file to define the public interface for your DataHub WebView control. The file can be named anything you like. By convention, the file name refers to the Silverlight class (e.g., *GlossyGauge*`.xml`). The file must end with the `.xml` file extension.

   The best way to begin is to use one of the existing control XML files as a template. Like the standalone XAML file (previous step), these files reside in the DataHub Controls installation directory:

   ```
   C:\Program Files\Cogent DataHub\Plugin\WebServer\html\Silverlight\Controls\
   ```

2. DataHub WebView parses control XML files to determine:

   - Identifying control information, including `ControlID` (unique), `ControlName`, `Description` and `ShortDescription` (the latter three are presented to the user in Property Explorer).
   - The base XAML used to render the control, `ControlFileName`, which in turn references your Silverlight control class.
   - The public data (i.e., Parameters) to make available to the user via Property Explorer. Parameters are grouped within `ParameterCategory` elements.

   For example, the XML file for the *GlossyGauge* control might look like this:

```
<c:EditableObjectConfig Owner="System" Category="Gauges"
 ControlID="MyControls:GlossyGauge:1" ControlName="My Glossy Gauge"
 ControlFileName="GlossyGauge.xaml"
 Description=" This gauge has glossy style and looks cool!"
 ShortDescription="Cool-lloking glossy gauge"
 Width="200" Height="200">

  <c:EditableObjectConfig.ParameterCategories>

    <c:ParameterCategory ParameterCategoryName="CAT_Basic" Label="Basic Properties">
      <c:ParameterCategory.Parameters>
        <c:Parameter ParameterName="CurrentValue" EditorType="general" Label="Current Value"
       Default="50" TargetPath="Gauge.CurrentValue" />
        <c:Parameter ParameterName="MinimumValue" EditorType="general" Label="Minimum Value"
       Default="0" TargetPath="Gauge.MinValue"/>
```

```
            <c:Parameter ParameterName="MaximumValue" EditorType="general" Label="Maximum Value"
           Default="100" TargetPath="Gauge.MaxValue"/>
            <c:Parameter ParameterName="GaugeBackgroundColor" EditorType="color" Label="Background Color"
           Default="#FF000000" TargetPath="Gauge.GaugeBackgroundColor"/>
          </c:ParameterCategory.Parameters>
        </c:ParameterCategory>

      </c:EditableObjectConfig.ParameterCategories>
    </c:EditableObjectConfig>
```

3. Notes for Parameters:
   - The `c:` namespace prefix is used by DataHub WebView and is required for the `EditableObjectConfig`, `ParameterCategory` and `Parameter` elements.
   - `ParameterName` is required and must be unique for the control.
   - Label values can be any text you want. They appear in Property Explorer.
   - DataHub WebView uses `EditorType` to determine how the user should interact with the `Parameter` value in Property Explorer. `EditorType` can be one of several values, including `bool`, `color`, `enum`, `togglebutton`, etc.
   - The `Default` value is used as the initial `Parameter` value.
   - `TargetPath` refers to the source object and source property path of the control to which this `Parameter` maps. For example, referring to:

     ```
     <c:Parameter ParameterName="BackgroundColor" ... Default="#80FF0000" TargetPath="Gauge.CurrentValue" />
     ```

     the `BackgroundColor Parameter` has a `TargetPath` of `Gauge.BackgroundColor` which means it maps to:

     - the XAML element defined with the `DynamicElement.ID` of `Gauge` i.e., the element defined in *`GlossyGauge`*`.xaml` (`ControlFileName`)

       ```
       <myControls:GlossyGauge live:DynamicElement.ID="Gauge" />
       ```

     - and the `"Backgroundcolor" Dependency Property` defined in the Silverlight control.

   - There are several more attributes that can be used to configure the `EditableObjectConfig` and `Parameter` objects. For a comprehensive list, please refer to "Configuring Control XML with EditableObjectConfig and Parameter Attributes."

5.3.4. Testing the results

1. Confirm your Visual Studio project (which includes the *`GlossyGauge`* class and `Generic.xaml`) builds successfully.
2. Copy the resulting assembly (`MyGauges.dll`) into the `ControlAssemblies` installation directory:

   ```
   C:\Program Files\Cogent DataHub\Plugin\WebServer\html\Silverlight\ControlAssemblies\
   ```

3. Confirm that *`GlossyGauge`*`.xaml` and *`GlossyGauge`*`.xml` reside in the `Controls` installation directory:

   ```
   C:\Program Files\Cogent DataHub\Plugin\WebServer\html\Silverlight\Controls\
   ```

4. Launch DataHub WebView. Your control should be listed in Control Explorer and on the Controls Toolbar. You should be able add an instance of your control to any page.

5. Now that you've confirmed you can build a control and get it working inside DataHub WebView, it;s time to enhance and extend your control's functionality. It's a simple, iterative cycle:
   - Define new `Dependency Properties` on your Silverlight control class.
   - Implement new behaviors with custom functionality in your Silverlight control class and with the control XAML in the `Generic.xaml` file.
   - Build your Silverlight class library and copy the `dll` to the `ControlAssemblies` installation directory.
   - In the control XML file (`Controls` installation directory), add new `Parameters` for each of the control properties you want to expose to the user.

Chapter 6. Creating Custom Symbols

DataHub WebView uses a single "Symbol Host" control to host all of the available symbols. Symbols are arranged in symbol sets, where each symbol set has one or more categories, and each category has one or more symbols. The symbols themselves are just drawings in their simplest form. Consequently, you can add different symbols to DataHub WebView by creating your own drawings and telling DataHub WebView where to find them.

What you'll need

1. A XAML Editor. Symbols are specified entirely in XAML. The XAML editor can be any one of the following:
   - Microsoft Expression Blend
   - Microsoft Visual Studio
   - Any text editor (e.g., Notepad++)

   Microsoft Expression Blend is the most powerful of these tools. Microsoft Visual Studio 2010 Express is free of charge, as are many text editors. We do not recommend writing XAML using a text editor until you are very experienced.

2. An XML Editor. Symbol maps are specified in XML. A symbol map tells DataHub WebView how to identify your symbols and how to associate the XAML files with the symbol names and identifiers. XML is a plain-text format that can be edited with any text editor, including the editors built into Expression Blend and Visual Studio.

6.1. Creating Your Symbol Library

6.1.1. Create a symbol map

The symbol map is a simple XML file that tells DataHub WebView about your symbol set. The first part details information about the whole symbol set, and the second part identifies the individual symbols. Below is an example of a symbol map file. A symbol map file can be formatted either as 7-bit ASCII or as UTF-8. For example, the following map file defines two shapes, an ellipse and a rectangle:

```
<?xml version="1.0" encoding="utf-8"?>
<wvsm:SymbolMap
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:wvsm="clr-namespace:Cogent.DataHubWebView;assembly=DataHubWebViewSymbolHelpers"

   SymbolSet="My Test Symbols"
   Manufacturer="Cogent Real-Time Systems Inc."
   Copyright="(C) 2011 Cogent Real-Time Systems Inc."
   License="Licensed for use with DataHub WebView only."
   Description="These Symbols illustrate how to incorporate custom objects into DataHub WebView."
   SymbolSetVersion="1.0"
   >
  <wvsm:SymbolMap.Symbols>
   <wvsm:Symbol Category="Simple Shapes" Name="Rectangle" SymbolID="201"
      FileName="SmipleShapes.xaml" Style="RectangleStyle" />
   <wvsm:Symbol Category="Simple Shapes" Name="Ellipse" SymbolID="202"
      FileName="SimpleShapes.xaml" Style="EllipseStyle" />
  </wvsm:SymbolMap.Symbols>
</wvsm:SymbolMap>
```

You can make a copy of this symbol map and alter it to your needs. Some of the SymbolMap fields deserve further description:

`SymbolSet`

The name that will appear in the DataHub WebView user interface when the user chooses to place one of your symbols on the screen.

`SymbolSetVersion`

A version number of the form *`N.N`*. The `Symbol Host` control can interpret this version number to allow you to offer multiple versions of the same symbol. The intention here is that you could create an equivalent visual representation of your symbols and make them available as an optional upgrade from a previous version. Different versions of symbols should perform the same role, such that a newer version of a symbol is a "drop-in" replacement of the previous version.

In addition to the symbol set definitions, this file contains a Symbols list, with a single entry for each symbol in the symbol set. Each symbol has the following fields:

`SymbolID`

A number that must be unique for each symbol in this symbol set. This is the most important identifying element of a symbol. Once you have released your symbol set, you must not change this number for a given symbol. If you release multiple versions of your symbol set, then the `SymbolID`s must match for different versions of the same symbol. `SymbolID` is a 32-bit signed integer number, and does not have to be in order or contiguous.

`Category`

A character string that identifies the symbol category. This category will be displayed to the user in the DataHub WebView interface. You may use any string you like. If two or more symbols have the same category name, they will be grouped within the DataHub WebView interface.

`Name`

The symbol name that will be displayed to the user in the DataHub WebView interface. This name does not have to be unique, as the `SymbolID` is the sole unique identifier for a symbol.

`FileName`

This is the name of the file that contains the symbol's XAML definition. When DataHub WebView encounters this symbol, the XAML file will be read from the server. DataHub WebView will only look for the file in the same directory as it originally found your symbol map file.

`Style`

This identifies the `<Style>` tag associated with this particular symbol within the XAML file. You may combine multiple symbol definitions into a single XAML file. The specific XAML definition of a symbol is identified by a `<Style>` tag within that file.

6.1.2. Create a symbol XAML file

The symbol file defines a single `UserControl`. The definition of that control consists of a header, one or more `ControlTemplate` resources, a matching number of `Style` resources, and an instance definition of a `DHSymbol` control. In its simplest form, it would look like this:

```
<UserControl
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   xmlns:ed="http://schemas.microsoft.com/expression/2010/drawing" mc:Ignorable="d"
```

```
    xmlns:sym="clr-namespace:Cogent.DataHubWebView.Controls;assembly=DataHubWebViewSymbol"
    >

    <UserControl.Resources>

       <ControlTemplate x:Key="RectangleTemplate" TargetType="sym:DHSymbol">
          <Grid>
             <Rectangle Fill="Transparent" Stroke="Black"/>
          </Grid>
       </ControlTemplate>

       <ControlTemplate x:Key="EllipseTemplate" TargetType="sym:DHSymbol">
          <Grid>
             <Ellipse Fill="Transparent" Stroke="Black"/>
          </Grid>
       </ControlTemplate>

       <Style x:Key="RectangleStyle" TargetType="sym:DHSymbol">
          <Setter Property="StyleName" Value="Rectangle" />
          <Setter Property="Template" Value="{StaticResource RectangleTemplate}" />
       </Style>

       <Style x:Key="EllipseStyle" TargetType="sym:DHSymbol">
          <Setter Property="StyleName" Value="Ellipse" />
          <Setter Property="Template" Value="{StaticResource EllipseTemplate}" />
       </Style>

    </UserControl.Resources>

    <sym:DHSymbol Style="{StaticResource RectangleStyle}" />

</UserControl>
```

A DataHub WebView symbol can contain any legal XAML definition. That means that a symbol is in fact more than just a drawing. It could contain buttons, dials, shapes, storyboards and virtually anything that XAML can describe. However, you may not provide C# code-behind class for your symbol. The intention is that the symbol definition contains the rendering definition of the symbol, while the behaviour of the symbol is implemented by the `Symbol Host` control.

The symbol XAML file can contain any number of matching pairs of <ControlTemplate> and <Style> tags. The <Style> tag for a particular symbol refers to its matching <ControlTemplate> in the Template property within the Style.

6.1.3. Deploy your symbol set

You can deploy your symbol set by simply copying the XAML files and symbol map file into a directory from which DataHub WebView can load them. When you install Cogent DataHub, the web server root directory is:

```
C:\Program Files\Cogent\Cogent DataHub\Plugin\WebServer\html
```

You can change this directory by modifying the Cogent DataHub Web Server properties.

This document will refer to the web root directory (above) as `[WebRoot]` henceforth.

By default, the base directory for all symbols is:

```
[WebRoot]\Silverlight\Symbols
```

You should create a sub-directory beneath the `Symbols` directory that is specific to your symbol set. Do not deploy your symbols into any directory that is installed with the Cogent DataHub, as your symbol files would be subject to deletion when you un-install the Cogent DataHub. For example, create a directory called:

```
[WebRoot]\Silverlight\Symbols\Test Symbols V1.0
```

Copy your symbol files into this directory. They may have any file extension, but normally they should be `.xaml` files. In the above examples, you should call your symbol file `SimpleShapes.xaml` to match the contents of the sample `.map` file.

Copy your symbol map file into this directory. It must have the extension `.map`. DataHub WebView searches for all `.map` files to determine which symbol sets are available. For example, you could create a `.map` file using the above example called `Test Symbols V1.0.map`.

You should now have created the following two files:

```
[WebRoot]\Silverlight\Symbols\Test Symbols V1.0\SimpleShapes.xaml
[WebRoot]\Silverlight\Symbols\Test Symbols V1.0\Test Symbols V1.0.map
```

Whenever you have changed or added files to your symbol set directory, you must inform DataHub WebView that it needs to update its resource definitions. You can do this by selecting the menu item Tools / Diagnostics / Refresh Application Objects. When you modify the symbol map file, you must reload DataHub WebView by saving your work and then refreshing the browser page.

6.2. Symbol Animation

In its simple form a symbol is not animated. That is, it does not blink, rotate or change color as its input data value changes. In order to animate your symbol you must indicate which parts of your symbol will change according to its data value. The Symbol Host control provides XAML markup attributes that you can use on the parts of your symbol to produce the animation of your choice. For example, if you include an attribute on a part of your symbol that indicates blinking behaviour, the `Symbol Host` control will enable the blinking properties for that symbol and will allow your user to configure the conditions under which the symbol will blink.

6.2.1. Animation attributes

An animation attribute is defined as an XML attribute attached to any `UIElement` tag within your symbol's XAML definition. For example, to make our ellipse blinkable, we can modify its template definition:

```
<ControlTemplate x:Key="EllipseTemplate" TargetType="sym:DHSymbol">
   <Grid sym:DHSymbol.BlinkTargetProperty="Opacity">
      <Ellipse Fill="Transparent" Stroke="Black" />
   </Grid>
</ControlTemplate>
```

In this case, we have attached the blink behaviour to the top-level `UIElement` (the `Grid`) in the symbol. This will cause the entire symbol to blink when it is configured to do so. We have chosen to blink the symbol by modifying the `Opacity` property of the `Grid`, which in practice is the only reasonable property to use for the purpose.

We may not want to blink the entire symbol. For example, the following XAML code modifies the ellipse to contain a rectangle within it:

```
<ControlTemplate x:Key="EllipseTemplate" TargetType="sym:DHSymbol">
   <Grid>
      <Ellipse Fill="Transparent" Stroke="Black" />
      <Rectangle Fill="Red" Stroke="Transparent" Width="20"
     sym:DHSymbol.BlinkTargetProperty="Opacity"/>
   </Grid>
</ControlTemplate>
```

Now the ellipse outline will remain solid, and only the rectangle within it will blink.

You may add more than one animation attribute to a single element within your XAML definition, and you may add the same animation attribute to more than one element. Thus, if you have a symbol consisting of many elements, you can choose on an individual basis which element would blink, change color etc.

For example, we can add a rotation attribute to our rectangle:

```
<ControlTemplate x:Key="EllipseTemplate" TargetType="sym:DHSymbol">
   <Grid>
      <Ellipse Fill="Transparent" Stroke="Black" />
      <Rectangle Fill="Red" Stroke="Transparent" Width="20"
         sym:DHSymbol.BlinkTargetProperty="Opacity"
         sym:DHSymbol.RotationType="Normal"
         />
   </Grid>
</ControlTemplate>
```

Note that the animation attributes do not immediately animate the symbol. They indicate to the `Symbol Host` control that this symbol will participate in that type of animation according to the configuration that the user has applied. The symbol designer might offer a rotation animation in the symbol, but the page developer may choose not to enable it for his particular configuration.

Your symbol may implement a number of user-definable colors. These can simply be added to any element of the symbol's XAML code. For example, to make the base color of the ellipse into a user-definable property, we can add the `BaseColorTargetProperty` attribute:

```
<ControlTemplate x:Key="EllipseTemplate" TargetType="sym:DHSymbol">
   <Grid>
      <Ellipse sym:DHSymbol.BaseColorTargetProperty="Fill.Color"
         Fill="Transparent" Stroke="Black"/>
      <Rectangle Fill="Red" Stroke="Transparent" Width="20"
         sym:DHSymbol.BlinkTargetProperty="Opacity"
         sym:DHSymbol.RotationType="Normal"
         />
   </Grid>
</ControlTemplate>
```

In this example, we add tell the `Symbol Host` control that the base color property should be applied to the `Fill.Color` property of the `Ellipse`. The `Fill.Color` refers to the `Color` of the `SolidColorBrush` assigned to the `Fill` property of the `Ellipse`. Since this reference requires `Fill` to be a `SolidColorBrush`, we must also assign `Fill` to be initially `Transparent` (or any solid color brush). Now when the user modifies the `Base Color` property of the `Symbol Host` in the Aesthetics property category, the fill color of the `Ellipse` within our symbol will change.

6.2.2. Animation types

The `Symbol Host` control offers several animation types. A symbol designer may specify any combination of these types within his symbol, or may choose to implement none at all. These types are:

Type: Constant

Property	Data Type	Description
`BaseColorTargetProperty`	String (a Property Name)	The name of the control property that will change color when the `Base Color` is altered. E.g., a `Rectangle's Fill` or a `Grid`'s `Background`.

Property	Data Type	Description
`AccentColor1TargetProperty`	String (a Property Name)	The name of the control property that will change color when the `Accent Color 1` is altered.
`AccentColor2TargetProperty`	String (a Property Name)	The name of the control property that will change color when the `Accent Color 2` is altered.
`ShadowColorTargetProperty`	String (a Property Name)	The name of the control property that will change color when the `Shadow Color` is altered.
`GlassinessTargetProperty`	String (a Property Name)	The name of the control property that will receive a value in the range `0.0 - 1.0` in response to the `Glassiness` symbol property. This is normally used to modify the opacity of a gradient overlay on the symbol that will provide the appearance of a glass cover.

Type: Condition

Condition specifies that part of the symbol will change its color, text or value based on the input value condition.

Property	Data Type	Description
`ConditionColorTargetProperty`	String (a Property Name)	The name of the control property that will change color when the `Condition Color` is altered.
`ConditionColorOpacityTargetProperty`	String (a Property Name)	The name of the control property that will receive a value in the range `0.0 - 1.0` in response to the `Condition Color Opacity` symbol property.
`ConditionTextTargetProperty`	String (a Property Name)	The name of the control property that will receive the condition text string when the symbol condition changes.

Type: Blink

Blink causes the symbol element to blink according to the condition or blink override settings.

Property	Data Type	Default	Description
`BlinkTargetProperty`	String (a Property Name)		The name of the property that will receive a value between `0.0 - 1.0` indicating the blink opacity.
`BlinkColorTargetProperty`	String (a Property Name)		The name of the property that will receive a color when color blinking occurs.

Property	Data Type	Default	Description
`BlinkFrom`	Double	`0.0`	The minimum value between `0.0` - `1.0` that will be sent to the property identified by `BlinkTargetProperty`.
`BlinkTo`	Double	`1.0`	The maximum value between `0.0` - `1.0` that will be sent to the property identified by `BlinkTargetProperty`.
`BlinkBy`	Double		The rate of change of the value sent to the property identified by `BlinkTargetProperty`.
`BlinkInverse`	Boolean	`False`	If set to `True`, blinking will be backward from the normal sense. This would allow the control to have two elements that blink in counterpoint.
`BlinkSpeedRatio`	Double	`1.0`	Specifies the rate of blinking, where `1.0` is normal speed, `<1.0` is slower and `>1.0` is faster than normal.
`BlinkFillBehavior`	One of: `HoldEnd`, `Stop`	`Stop`	Determines the state of the symbol when blinking stops. If set to `Stop`, the control will be reset to normal. If set to `HoldEnd`, the control will be left as it was when blinking stopped.
`BlinkEasingFunction`	One of: `None`, `Back`, `Bounce`, `Circle`, `Cubic`, `Elastic`, `Exponential`, `Power`, `Quadratic`, `Quartic`, `Quintic`, `Sine`	`None`	Indicates the type of easing function to apply when approaching the limit of a blink based on fading. Some easing functions require parameters.
`BlinkEasingMode`	One of: `EaseOut`, `EaseIn`, `EaseInOut`		Determines whether the easing function will be applied at the end of the blink animation, the beginning, or both.
`BlinkEasingArg1`	Integer	`0`	The first argument to the `BlinkEasingFunction` if applicable.
`BlinkEasingArg2`	Integer	`0`	The second argument to the `BlinkEasingFunction` if applicable.

Type: Rotation

Elements within the symbol may rotate clockwise or counter-clockwise according to the condition or rotation override settings.

Property	Data Type	Default	Description
`RotationType`	One of: `None`, `Normal`	`None`	If this attribute is "Normal" then the element can rotate.
`RotationFrom`	Double	`0`	The starting angle of roatation, in degrees.
`RotationTo`	Double	`360`	The finishing angle of roatation, in degrees.
`RotationBy`	Double		The number of degrees to rotate by in each animation frame.
`RotationInverse`	Boolean	`False`	If `False`, rotation will be in the clockwise direction. If `True`, rotation is counter-clockwise.
`RotationSpeedRatio`	Double	`1.0`	Specifies the rate of rotation, where `1.0` is normal speed, `<1.0` is slower and `>1.0` is faster than normal.
`RotationFillBehavior`	One of: `HoldEnd`, `Stop`	`Stop`	Determines the state of the symbol when rotation stops. If set to `Stop`, the control will be reset to its initial rotation. If set to `HoldEnd`, the control will be left as it was when rotation stopped.
`RotationEasingFunction`	One of: `None`, `Back`, `Bounce`, `Circle`, `Cubic`, `Elastic`, `Exponential`, `Power`, `Quadratic`, `Quartic`, `Quintic`, `Sine`	`None`	Indicates the type of easing function to apply when approaching the limit of a rotation. Some easing functions require parameters.
`RotationEasingMode`	One of: `EaseOut`, `EaseIn`, `EaseInOut`		Determines whether the easing function will be applied at the end of the rotation animation, the beginning, or both.
`RotationEasingArg1`	Integer		The first argument to the `RotationEasingFunction` if applicable.
`RotationEasingArg2`	Integer		The second argument to the `RotationEasingFunction` if applicable.

Type: Progress

Elements within the symbol may indicate progress in the range of `0.0` - `1.0` (representing 0 - 100%). Elements may also take on a *progress color*.

Property	Data Type	Description
`ProgressPercentageTargetProperty`	String (a Property Name)	The name of an element property that will receive the progress value between `0.0` and `1.0`.
`ProgressColorTargetProperty`	String (a Property Name)	The name of an element property that will be set to the progress color.

6.2.3. Conditional animation

The primary feature of the `Symbol Host` control is the ability to change the animation of the symbol based on an input condition. The `Symbol Host` defines 5 states, each of which can be individually configured for `Color`, `Text`, `Blink` and `Rotation`. All of the properties in the `Condition`, `Blink` and `Rotation` categories in the table above can be used to affect the behaviour of the symbol on a per-state basis.

6.3. Scaling and Other Considerations

6.3.1. Scaling

Your symbol will be resized when the `Symbol Host` control changes size. You will need to decide what kind of resizing behaviour you would like. Normally, resizing the control will simply scale the contents in the X and Y direction independently, allowing the user to stretch your symbol. This is not always the correct behaviour. Typical requirements for alternate scaling methods include:

1. If the symbol needs to maintain an aspect ratio, resizing should not cause flattening of the symbol, or
2. If the symbol contains controls such as text boxes or buttons then resizing should also cause the text and other contained controls to scale, or
3. If the symbol contains elements that have a fixed size, and therefore do not scale as the rest of the symbol scales.

Scaling issues can commonly be handled using `<Grid>` tags, which automatically resize all of their contained elements. When `<Grid>` tags are not sufficient, the best choice is to surround the entire symbol definition in a `<Viewbox>`:

```
<ControlTemplate x:Key="EllipseTemplate" TargetType="sym:DHSymbol">
  <Viewbox Stretch="Fill">
   <Grid Width="100" Height="100">
      <Ellipse sym:DHSymbol.BaseColorTargetProperty="Fill.Color"
         Fill="Transparent" Stroke="Black"/>
      <Rectangle Fill="Red" Stroke="Transparent" Width="20"
         sym:DHSymbol.BlinkTargetProperty="Opacity"
         sym:DHSymbol.RotationType="Normal"
         />
   </Grid>
  </Viewbox>
</ControlTemplate>
```

Notice that the content within the `Viewbox` must include a specific width and height. The `Viewbox` must know the nominal size of its contained element, and will scale the content relative to that size.

6.3.2. Binding element properties to the symbol host control

The `Symbol Host` control is an instance of the class `DHSymbol`. Since the symbol content is specified as a `ControlTemplate`, any of the publicly exposed members of the host control can be accessed by the symbol XAML using `{TemplateBinding property_path}`. This allows the symbol designer to reflect the host control state within the symbol if necessary. For example, the symbol could contain a text box that presents the progress percentage as a number.

It is not always a good idea to use `TemplateBinding` instead of the `DHSymbol` attached properties. The attached properties act as markup that is not specific to a particular `Symbol Host` control. If you chose to implement your own symbol host control or use a third-party symbol host control then the attached property markup would make your symbol relatively easy to support. If you choose instead to bind your symbol using `TemplateBinding` then your symbol will be much less likely to function with another symbol host control.

6.3.3. Limitations on XAML code within symbols

A symbol can contain virtually any XAML code with the exception of event handlers. Event handlers require C# code-behind, which is not available to a symbol. Fairly complex behaviours can be achieved with storyboards and bindings among elements in the XAML definition.

6.3.4. Compressing symbol files

The XAML file containing the symbol definitions can be compressed with the standard ZIP compression. This will reduce loading time and network bandwidth when the application starts. You may not specify a password on this ZIP file.

6.3.5. Best Practices

When symbols are intended to line up with one another, use integer dimensions and positions. Stay consistent when offsetting symbol contents from the control boundary. For example, a symbol set consisting of pumps and pipes would be much easier to work with if the pipe were easily scaled to match the size of the pump’s output, and if the pipes were offset within their bounding box by the size of the flange on the pump’s ouput.

Create symbols that consist of multiple layers. The bottom layer should be filled with the base color, the second layer with the condition color and the top layer with a translucent overlay as a lighting effect whose opacity is tied to the glassiness. ZIP your XAML files when you deploy them to save time and network bandwidth.

6.4. Complete Example

The following two listings provide a complete example that implements 4 symbols:

- Spinner: A circle containing a rotating rectangle.
- Spiral: A spiral path
- Info (Attached): A rectangle containing symbol information using attached properties to create its bindings.
- Info (Template): The same rectangle using template bindings.

File: `TestSymbolsV1.0.map`

```
<?xml version="1.0" encoding="utf-8"?>
<wvsm:SymbolMap
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wvsm="clr-namespace:Cogent.DataHubWebView;assembly=DataHubWebViewSymbolHelpers"

  SymbolSet="My Test Symbols"
  Manufacturer="Cogent Real-Time Systems Inc."
  Copyright="(C) 2011 Cogent Real-Time Systems Inc."
  License="Licensed for use with DataHub WebView only."
  Description="These Symbols illustrate how to incorporate custom objects into DataHub WebView."
  SymbolSetVersion="1.0"
  >
  <wvsm:SymbolMap.Symbols>
     <wvsm:Symbol Category="Simple Shapes" Name="Spinner" SymbolID="1"
        FileName="SimpleShapes.xaml" Style="SpinnerStyle" />
    <wvsm:Symbol Category="Simple Shapes" Name="Spiral" SymbolID="2"
      FileName="SimpleShapes.xaml" Style="SpiralStyle" />
    <wvsm:Symbol Category="Simple Shapes" Name="Info (Template)" SymbolID="3"
        FileName="SimpleShapes.xaml" Style="Info1Style" />
    <wvsm:Symbol Category="Simple Shapes" Name="Info (Attached)" SymbolID="4"
        FileName="SimpleShapes.xaml" Style="Info2Style" />
  </wvsm:SymbolMap.Symbols>
</wvsm:SymbolMap>
```

File: `SimpleShapes.xaml`

```
<UserControl
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   xmlns:ed="http://schemas.microsoft.com/expression/2010/drawing" mc:Ignorable="d"

   xmlns:sym="clr-namespace:Cogent.DataHubWebView.Controls;assembly=DataHubWebViewSymbol"
   xmlns:wvc="clr-namespace:Cogent.DataHubWebView.Controls;assembly=DataHubWebViewControlsNew"
   >

   <UserControl.Resources>

   <!-- This string converter is used with {Binding} definitions to convert values into
        strings where they are then sumbitted to another control property and converted
 to the appropriate type as part of the XAML parsing.  This is a quick way to
 perform a type conversion that isn't naturally supported, such as Color to Brush. -->
   <wvc:ObjectToStringConverter x:Key="stringConverter" />

      <ControlTemplate x:Key="SpinnerTemplate" TargetType="sym:DHSymbol">
        <Viewbox Stretch="Uniform">
         <Grid Width="100" Height="100">
           <Ellipse sym:DHSymbol.BaseColorTargetProperty="Fill.Color" Fill="White"
     Stroke="Black"/>
         <Rectangle Fill="Red" Stroke="Transparent" Width="10"
                 sym:DHSymbol.BlinkTargetProperty="Opacity"
                 sym:DHSymbol.RotationType="Normal"
                 />
         </Grid>
        </Viewbox>
      </ControlTemplate>

   <ControlTemplate x:Key="SpiralTemplate" TargetType="sym:DHSymbol">
       <Viewbox Stretch="Uniform" sym:DHSymbol.RotationType="Normal">
      <Grid Width="100" Height="100">
          <Path sym:DHSymbol.AccentColor1TargetProperty="Stroke.Color"
           sym:DHSymbol.BaseColorTargetProperty="Fill.Color" Fill="Transparent"
           StrokeThickness="2"
           Data="M45,50 A5,5 180 1 1 55,50 M40,50 A7,7 -180 1 0 55,50 M40,50
```

```
          A10,10 180 1 1 60,50 M35,50 A12,12 -180 1 0 60,50 M35,50 A15,15 180
          1 1 65,50 M30,50 A17,17 -180 1 0 65,50 M30,50 A20,20 180 1 1 70,50
          M25,50 A22,22 -180 1 0 70,50 M25,50 A25,25 180 1 1 75,50 M20,50 A27,
          27 -180 1 0 75,50 M20,50 A30,30 180 1 1 80,50 M15,50 A32,32 -180 1 0
          80,50 M15,50 A35,35 180 1 1 85,50 M10,50 A37,37 -180 1 0 85,50 M10,50
          A40,40 180 1 1 90,50 M5,50 A42,42 -180 1 0 90,50 M5,50 A45,45 180 1 1
          95,50 M0,50 A47,47 -180 1 0 95,50"
          >
        <Path.Stroke>
            <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
          <GradientStop Offset="0.0" Color="Green"/>
          <GradientStop Offset="0.0" Color="Green"/>
          <GradientStop Offset="1.0" Color="Yellow"/>
          <GradientStop Offset="1.0" Color="Yellow"/>
           </LinearGradientBrush>
        </Path.Stroke>
         </Path>
    </Grid>
     </Viewbox>
</ControlTemplate>

<!-- Create a rectangular information block using template binding. -->
<ControlTemplate x:Key="InfoTemplate1" TargetType="sym:DHSymbol">
    <Grid>
   <Grid.Resources>
       <SolidColorBrush x:Key="SymbolBaseColor"
 Color="{Binding RelativeSource={RelativeSource TemplatedParent}, Path=BaseColor}"/>
       <LinearGradientBrush x:Key="HorizontalAccentGradient"
 StartPoint="0 0" EndPoint="1 0">
      <GradientStop Offset="0.0"
Color="{Binding RelativeSource={RelativeSource TemplatedParent}, Path=AccentColor1}"/>
      <GradientStop Offset="1.0"
Color="{Binding RelativeSource={RelativeSource TemplatedParent}, Path=AccentColor2}"/>
       </LinearGradientBrush>
       <LinearGradientBrush x:Key="VerticalProgressGradient" StartPoint="0 1" EndPoint="0 0">
      <GradientStop Offset="0.0"
Color="{Binding RelativeSource={RelativeSource TemplatedParent}, Path=ProgressColor}"/>
      <GradientStop Offset="{Binding RelativeSource={RelativeSource TemplatedParent},
Path=ProgressPercentage}"
Color="{Binding RelativeSource={RelativeSource TemplatedParent}, Path=ProgressColor}"/>
      <GradientStop Offset="{Binding RelativeSource={RelativeSource TemplatedParent},
Path=ProgressPercentage}" Color="Transparent"/>
      <GradientStop Offset="1.0" Color="Transparent"/>
       </LinearGradientBrush>
   </Grid.Resources>
   <!-- When we use template binding instead of attached properties, WebView cannot tell
        that the properties are in use, so it will not correctly display supported behaviours
 in its property editor.  We create an invisible control that attaches the properties
 that we want to show up as supported in the property editor. -->
   <Border Background="Transparent"
      sym:DHSymbol.AccentColor1TargetProperty="Background.Color"
      sym:DHSymbol.AccentColor2TargetProperty="Background.Color"
      sym:DHSymbol.BaseColorTargetProperty="Background.Color"
      sym:DHSymbol.ProgressColorTargetProperty="Background.Color"
      sym:DHSymbol.ConditionColorTargetProperty="Background.Color"
      sym:DHSymbol.GlassinessTargetProperty="Opacity"
      sym:DHSymbol.ConditionColorOpacityTargetProperty="Opacity"
      sym:DHSymbol.RotationType="Normal"
         Visibility="Collapsed">
       <TextBlock Text=""
      sym:DHSymbol.ConditionTextTargetProperty="Text"
      sym:DHSymbol.ProgressPercentageTargetProperty="Text"/>
   </Border>

   <!-- Create a rectangle.  TemplateBinding does not support a Converter, but Binding does.-
->
   <Rectangle Opacity="{TemplateBinding Glassiness}" Stroke="Black"
              Fill="{StaticResource VerticalProgressGradient}"/>
```

```
      <!-- Create some other controls inside the rectangle that bind to other properties
           of the symbol host. -->
      <StackPanel Margin="1,1,1,1" Opacity="{TemplateBinding ConditionColorOpacity}">
          <CheckBox Content="{TemplateBinding ConditionText}"
    IsChecked="{TemplateBinding IsRotating}" IsEnabled="False"/>
          <StackPanel Orientation="Horizontal"
    Background="{StaticResource HorizontalAccentGradient}">
         <TextBlock Text="Progress: " Margin="0,0,5,0"
   Foreground="{StaticResource SymbolBaseColor}"/>
         <TextBlock Foreground="{StaticResource SymbolBaseColor}"
   Text="{Binding RelativeSource={RelativeSource TemplatedParent},
   Path=ProgressPercentage, Converter={StaticResource stringConverter}}"/>
          </StackPanel>
      </StackPanel>
       </Grid>
   </ControlTemplate>

   <!-- Create the same rectangular information block using attached properties. -->
   <ControlTemplate x:Key="InfoTemplate2" TargetType="sym:DHSymbol">
       <Grid>
      <Grid.Resources>
          <LinearGradientBrush x:Key="HorizontalAccentGradient" StartPoint="0 0" EndPoint="1 0">
         <GradientStop Offset="0.0" Color="White"/>
         <GradientStop Offset="1.0" Color="Black"/>
          </LinearGradientBrush>
          <LinearGradientBrush x:Key="VerticalProgressGradient" StartPoint="0 1" EndPoint="0 0">
         <GradientStop Offset="0.0" Color="White"/>
         <GradientStop Offset="0.0" Color="White"/>
         <GradientStop Offset="1.0" Color="Transparent"/>
         <GradientStop Offset="1.0" Color="Transparent"/>
          </LinearGradientBrush>
      </Grid.Resources>

      <!-- Create a rectangle.  TemplateBinding does not support a Converter, but Binding does.-
->
      <Rectangle sym:DHSymbol.GlassinessTargetProperty="Opacity" Stroke="Black"
            Fill="{StaticResource VerticalProgressGradient}"

            sym:DHSymbol.ProgressPercentageTargetProperty="Fill.(LinearGradientBrush.GradientStops)
     [1].Offset,Fill.(LinearGradientBrush.GradientStops)[2].Offset"

            sym:DHSymbol.ProgressColorTargetProperty="Fill.(LinearGradientBrush.GradientStops)
     [0].Color,Fill.(LinearGradientBrush.GradientStops)[1].Color"
            />

      <!-- Create some other controls inside the rectangle that bind to other properties
           of the symbol host. Note that the condition text will only show up if the
    "Show Condition Text on Symbol" checkbox is checked in the property category
    "Advanced Configuration: Condition Color and Text". -->
      <StackPanel Margin="1,1,1,1" sym:DHSymbol.ConditionColorOpacityTargetProperty="Opacity">

       <!-- We have no way to reach IsRotating through an attached property.  We must use
       TemplateBinding for that. -->
          <CheckBox IsChecked="{TemplateBinding IsRotating}" IsEnabled="False"
    sym:DHSymbol.ConditionTextTargetProperty="Content"/>
          <StackPanel Orientation="Horizontal"
    Background="{StaticResource HorizontalAccentGradient}"
           sym:DHSymbol.AccentColor1TargetProperty="Background.(LinearGradientBrush.GradientStops)
    [0].Color"
           sym:DHSymbol.AccentColor2TargetProperty="Background.(LinearGradientBrush.GradientStops)
    [1].Color"
            >
            <TextBlock Text="Progress: "
      Margin="0,0,5,0" sym:DHSymbol.BaseColorTargetProperty="Foreground.Color"
      Foreground="Black"/>
           <TextBlock sym:DHSymbol.BaseColorTargetProperty="Foreground.Color" Foreground="Black"
            sym:DHSymbol.ProgressPercentageTargetProperty="Text"/>
```

```
        </StackPanel>
      </StackPanel>
    </Grid>
   </ControlTemplate>

   <Style x:Key="SpinnerStyle" TargetType="sym:DHSymbol">
         <Setter Property="StyleName" Value="Spinner" />
         <Setter Property="Template" Value="{StaticResource SpinnerTemplate}" />
   </Style>

   <Style x:Key="SpiralStyle" TargetType="sym:DHSymbol">
       <Setter Property="StyleName" Value="Spiral" />
       <Setter Property="Template" Value="{StaticResource SpiralTemplate}" />
   </Style>

   <Style x:Key="Info1Style" TargetType="sym:DHSymbol">
       <Setter Property="StyleName" Value="InfoTemplated" />
       <Setter Property="Template" Value="{StaticResource InfoTemplate1}" />
   </Style>

   <Style x:Key="Info2Style" TargetType="sym:DHSymbol">
       <Setter Property="StyleName" Value="InfoAttached" />
       <Setter Property="Template" Value="{StaticResource InfoTemplate2}" />
   </Style>

  </UserControl.Resources>

  <sym:DHSymbol Style="{StaticResource SpinnerStyle}" />

</UserControl>
```

I. Controls

Table of Contents

Advanced Check Box

`Advanced Check Box` — toggles between two states, each with advanced properties.

Description

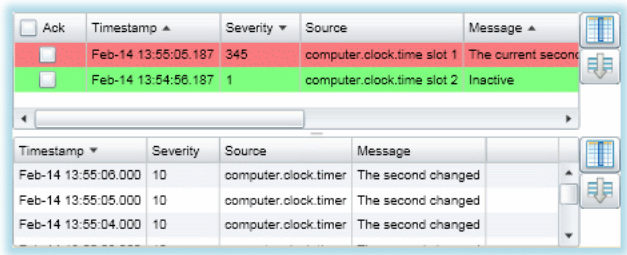
Changes between two states: checked and unchecked. Each state has additional user properties: a numeric value, a string, a DateTime, and a color.

Alarm List

`Alarm List` — a table that displays alarms and events.

Description

The Alarm control lets you view the most recent alarms at the top and events at the bottom. The alarms data comes from a data domain in the Cogent DataHub that has been configured for OPC Alarms and Events (OPC A&E).



Use

The Select Columns button for each section lets you choose which columns to display. The Default Sort button for each section reverts the sort order back to the default setting. The Ack button allows you to acknowledge and remove all the alarms, while the buttons in that column let you acknowledge and remove individual alarms.

Boolean Converter

`Boolean Converter` — a program block that selects between two states.

Description

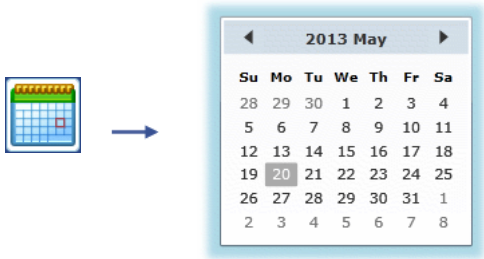
Selects an output value, an output DateTime and an output color based on an input boolean value (True/False). This is a design-time control and is not visible in run mode.

Calendar

`Calendar` — displays a calendar.

Description

Displays a calendar, enabling the user to select a date.

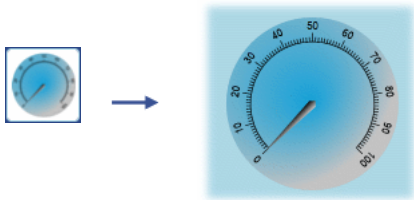


Circular Gauge 1

`Circular Gauge 1` — simple circular gauge.

Description

Used to graphically represent real-time data. In its default configuration, this gauge is a simple circle with a needle. With Edit Mode set to Drag, the user can interact with the gauge in run mode. This is a highly-configurable control. Properties like value, value range, editing ability, and angle range are all modifiable.

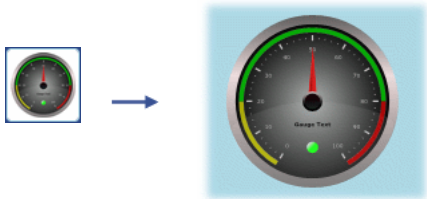


Circular Gauge 2

`Circular Gauge 2` — circular gauge with varying value ranges and an indicator light.

Description

Used to graphically represent real-time data. This gauge has an indicator light that shows whether the needle is in the optimal range, below optimal, or above optimal. This gauge does not allow user interaction in run mode. All ranges, sizes, values, and colors can be changed.



Color Selector

`Color Selector` — a palette used to store specific colors and to access application theme colors.

Description

Used to set a variety of colors as Custom Colors and to access application Theme Colors. These colors can then be used as binding sources for other controls on the page. This makes it easy to create and maintain custom color themes. This is a design-time control and is not visible in run mode.

Color Selector

`Color Selector` — a program block that uses ARGB values to produce a color.

Description

Uses four numeric inputs (alpha, red, green, blue), each in the range 0-255, to build a color. Alternatively, references a color from a Color Palette. This is a design-time control and is not visible in run mode.

ComboBox

`ComboBox` — a simple dropdown list used for item selection.

Description

Enables the user to select from among available items. The list of items can be configured as a comma-separated list, or bound to the result of an expression.

Comparator

`Comparator` — a program block that compares two values and outputs the results.

Description

Compares two inputs (DataPoints or values) and produces various output values. Optionally, accepts a numeric tolerance to stabilize comparisons based on rapidly-changing inputs. This is a design-time control and is not visible in run mode.

Condition Selector

`Condition Selector` — a program block used to select among five different states.

Description

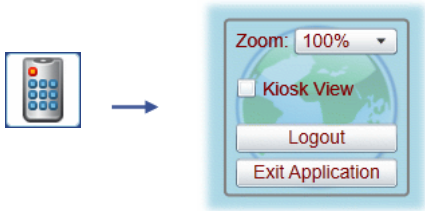
Selects output text and values based on a condition input value. The input can be treated as a boolean to produce a two-state result, or can be compared to each state's value range. If states have overlapping ranges, the first match is used. Each state can be associated with text, colors and values. Typically, these configurable state settings feed other controls and processes. This is a design-time control and is not visible in run mode.

Control Panel

`Control Panel` — supports Run Mode option changes.

Description

Enables the user to change various options while in Run Mode.



Filtered Data Table

`Filtered Data Table` — row/column results from a database query.

Description

Presents the result of a row/column data set in a table. Columns can be reordered and pinned. Rows can be filtered and grouped.

ID	PointName	PointValue	TimeStamp
4	Register-5	65	3/15/2013 12:00:00
5	Collectors	15	3/15/2013 12:00:00
6	Fan.021a	52	5/20/2013 12:00:00
7	Meter.581	71	5/14/2013 12:00:00

Page 1 of 1

Hi/Low Indicator

`Hi/Low Indicator` — changes color to indicate high and low values.

Description

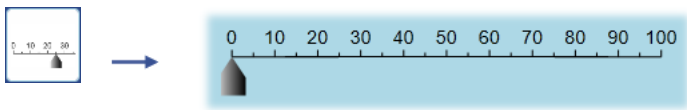
Responds to real-time data updates by changing color. The color corresponds to a matching value range. Five ranges can be configured: Low Low, Low, Normal, High, and High High. It has modifiable text, limits, colors, and transition time.

Horizontal Linear Gauge

`Horizontal Linear Gauge` — a linear horizontal gauge with a slider,

Description

Used to graphically represent real-time data. In its default configuration, this linear gauge is oriented horizonally. With Edit Mode set to Drag, the user can interact with the gauge in run mode. This is a highly-configurable control. Properties like the value, value range, editing ability, and angle range are all modifiable.



Hyperlink Button

`Hyperlink Button` — acts as a hyperlink to another page or a URL.

Description

Uses a button to link the user to another page or to an external URL. This control is often used to provide navigation support among a collection of related pages.

Hyperlink Image

`Hyperlink Image` — acts as a hyperlink to another page or a URL.

Description

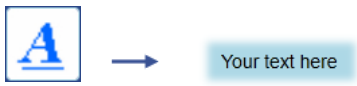
Uses an image to link the user to another webView page or to an external URL. This control is often used to provide navigation support among a collection of related pages.

Hyperlink Text

`Hyperlink Text` — acts as a hyperlink to another page or a URL.

Description

Uses a text label to link the user to another page or to an external URL. This control is often used to provide navigation support among a collection of related pages.



Image

`Image` — an image file container.

Description

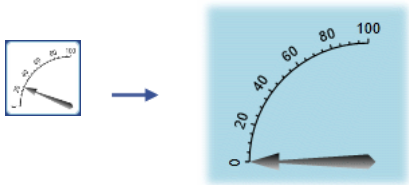
Displays an image located in the images file directory on the web server.

Left 90 Degree Gauge

`Left 90 Degree Gauge` — a quarter-circle gauge.

Description

Used to graphically represent real-time data. In its default configuration, this gauge is a quarter-circle with a needle. With Edit Mode set to Drag, the user can interact with the gauge in run mode. This is a highly-configurable control. Properties like the value, value range, editing abilitiy, and angle range are all modifiable.



List Box

`List Box` — a simple list used for item selection.

Description

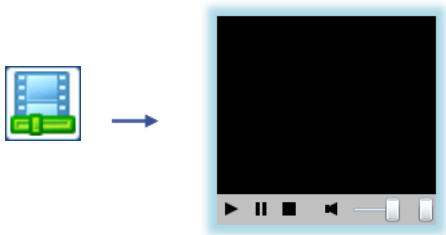
Enables the user to select from among available items. The list of items can be configured as a comma-separated list, or bound to the result of an expression.

Media Player

`Media Player` — plays audio and videos.

Description

Displays media located in the media file directory on the web server. Typical playback controls are available. Media can be configured to auto play and auto repeat. Audio controls are also available.



Numeric Gauge

`Numeric Gauge` — a digital gauge.

Description

A tutorial-oriented gauge that illustrates how to build custom controls. Developed using RadControls by Telerik Corporation.

One Input Calculator

`One Input Calculator` — a program block that performs calculations on a single input value.

Description

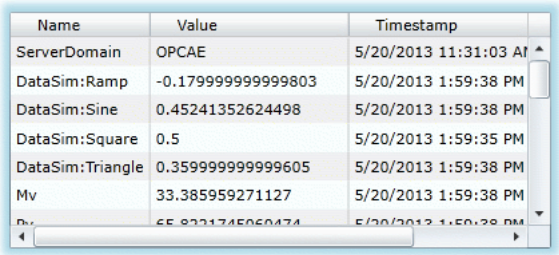
Calculates a variety of values based on a single input. Calculations include Boolean and Duration conversions, mathematical operations (Abs, Sign, Ceiling, Floor, Exponent, Log, Square, SquareRoot), and trigonometric functions (Sin, Cos, Tan, etc). This is a design-time control and is not visible in run mode.

Point Data Table

`Point Data Table` — a table consisting of all available data points.

Description

Presents all available data points in the DataHub in a table format. The columns available are Point Name, Display Name, Value, Timestamp, Quality Name, and Quality.



Name	Value	Timestamp
ServerDomain	OPCAE	5/20/2013 11:31:03 AM
DataSim:Ramp	-0.179999999999803	5/20/2013 1:59:38 PM
DataSim:Sine	0.45241352624498	5/20/2013 1:59:38 PM
DataSim:Square	0.5	5/20/2013 1:59:35 PM
DataSim:Triangle	0.359999999999605	5/20/2013 1:59:38 PM
Mv	33.385959271127	5/20/2013 1:59:38 PM

Polynomial Calculator

`Polynomial Calculator` — a program block that calculates the result of a polynomial expression.

Description

Calculates the result of a polynomial expression (up to 5th order). This is a design-time control and is not visible in run mode.

Progress Bar

`Progress Bar` — an expanding/shrinking progress bar.

Description

Used to graphically represent real-time data. This control shows the input value by expanding or shrinking in size. The bar's orientation, range, and size can all be modified. The progress bar also accommodates color change based on five ranges: Low Low, Low, Normal, High, and High High.

Radio Button

`Radio Button` — a button that offers a choice of mutually exclusive options.

Description

Used in groups where only one of the buttons in the group can be checked at a time (i.e., mutually exclusive). The Control Value for each radio button in a group should be bound to a single source (e.g., data point), which will be treated as the group's input. The radio button whose value matches the Control Value will be selected.

Range Mapper

`Range Mapper` — a program block that maps an input to an output, using ranges.

Description

Maps a single input (DataPoint or value) to a value in a corresponding output range. Ranges are specified with an input minimum and maximum, and an output minimum and maximum. The input can be clamped for limited, linear interpolation, or unclamped for extrapolation. This is a design-time control and is not visible in run mode.

Rising/Falling Indicator

`Rising/Falling Indicator` — a display that changes according to the rise or fall of a value.

Description

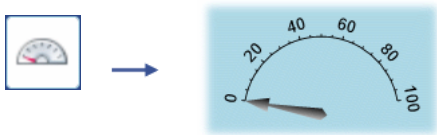
Responds to real-time data updates by changing color. The color reflects how quickly the input value is rising or falling. It has modifiable text, colors, transition time, and steady time.

Semi-circular Gauge

`Semi-circular Gauge` — a semi-circular gauge.

Description

Used to graphically represent real-time data. In its default configuration, this gauge is a semi-circle with a needle. With Edit Mode set to Drag, the user can interact with the gauge in run mode. This is a highly-configurable control. Properties like the value, value range, editing abilitiy, and angle range are all modifiable.

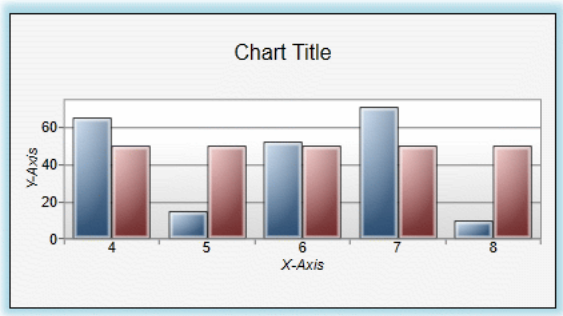


Series Chart

`Series Chart` — displays data in chart format - bars, lines, pie, etc.

Description

Provides a number of different chart formats for displaying related data values.

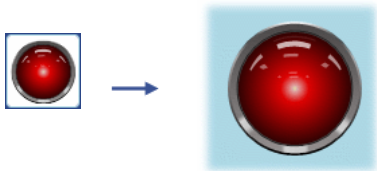


Shining Light

`Shining Light` — an indicator light that can flash and change color.

Description

Displays a light which responds to boolean triggers and color changes. This control is typically used for notification. Boolean inputs control whether the light is on or flashing. The light color can be set with a single input, or configured using gradient colors and offsets. Duration, auto reverse, and repeat behavior are also modifiable.



Simple Button

`Simple Button` — a simple, clickable button.

Description

Provides a simple way to attach script code to a button click event.

Simple Check Box

`Simple Check Box` — toggles between two states.

Description

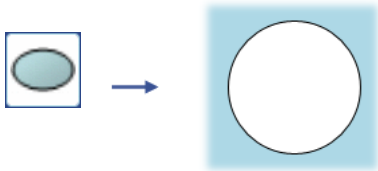
Changes between two states: checked and unchecked. Typically, a DataPoint is bound to the input value and used to toggle between states. Each state has an associated value which can be used to feed another control or process.

Simple Ellipse

`Simple Ellipse` — a simple ellipse with editable appearance and properties.

Description

A simple ellipse with modifiable fill color, stroke color, and stroke thickness.

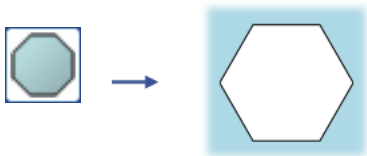


Simple Path

`Simple Path` — a path that can create any shape.

Description

A simple path that can be used to create any shape through mapping out each point using XAML path notation. Fill color, stroke color, canvas size, stroke thickness, stroke caps, stroke joins, and stroke miter limit are all modifiable. Please see the XAML path notation for Path Markup Syntax in the online Microsoft reference library for more information.

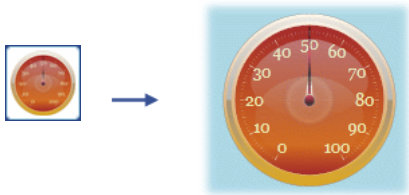


Simple Radial Gauge

`Simple Radial Gauge` — a simple, tutorial-oriented circular gauge.

Description

A simple, tutorial-oriented circular gauge that illustrates how to build custom controls. This customizable radial gauge has a scale and a needle. Developed using RadControls by Telerik Corporation.

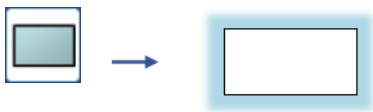


Simple Rectangle

`Simple Rectangle` — a simple rectangle with editable appearance and properties.

Description

A simple rectangle with modifiable fill color, stroke color, stroke thickness and corner radius.

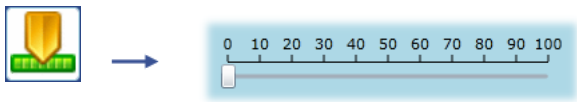


Slider

`Slider` — a scale with an adjustable slider to control or view values.

Description

Enables the user to select a value along a configurable scale by dragging the slider. The slider can be configured either horizontally or vertically. Axis labels can be displayed before or after the scale.

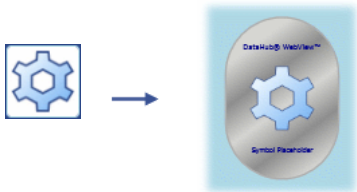


Symbol

`Symbol` — a container for over 4000 symbols.

Description

A common container for over 4000 industry standard symbols. Symbols can be configured to blink, rotate and show progress. Output states are selected based on a condition input value. The input can be treated as a boolean to produce a two-state result, or can be compared to each state's value range. If states have overlapping ranges, the first match is used. Each state is associated with a value range, color, text, blinking, blink rate, rotating, and rotation rate. Symbols can be automatically updated when new versions of the symbol are available.



System Information

`System Information` — a program block that can access system, user, and page information.

Description

Accesses information about the system and page. Outputs include local time, user, page name, file name, description, and owner. Typically, these values are used for page titles and footers. This is a design-time control and is not visible in run mode.

Text Entry Field

`Text Entry Field` — a simple textbox used for data entry.

Description

Enables the user to input a text string or numeric value while in run mode.

Use

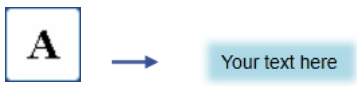
Values, including dates and times, can be formatted using common FormatString syntax. For example, to format a numeric value to two decimal places, specify a format of `0.00`. January 14, 2011 will be displayed as '`14-Jan-11`' when formatted as '`dd-MMM-yy`'.

Text Label

`Text Label` — a textual display with no entry field.

Description

A text label that displays text, but does not have an entry field. Color is modifiable.

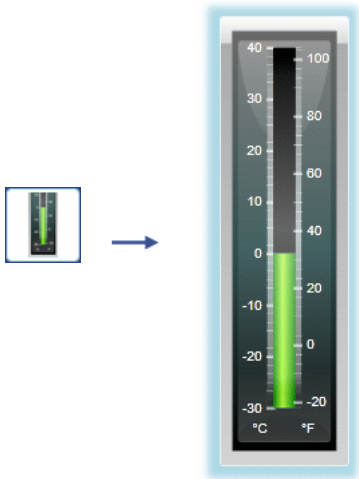


Thermometer

`Thermometer` — a tutorial-oriented linear gauge.

Description

A tutorial-oriented linear gauge that illustrates how to build custom controls. This highly customizable linear gauge has two linear scales, adjustable offsets and the ability to measure in either Celsius or Farenheit. Developed using RadControls by Telerik Corporation.



Three Indicator Radial Gauge

`Three Indicator Radial Gauge` — a tutorial-oriented, multi-indicator, circular gauge.

Description

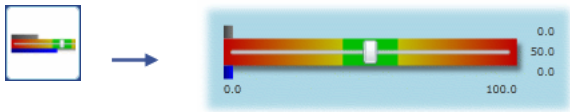
A tutorial-oriented, multi-indicator, circular gauge that illustrates how to build custom controls. This highly customizable radial gauge has one scale, three indicators, and a three-part range list. Developed using RadControls by Telerik Corporation.

Three Point Slider

`Three Point Slider` — a horizontal gauge that shows up to three data points on a slider.

Description

Used to graphically represent real-time data. This slider shows up to three values and has optimal, below optimal, and above optimal ranges. It has a slider for the primary value, which the user can drag in run mode, and a progress bar for each of the other values. It also has an error indicator which flashes when the primary value is outside the optimal range.



Timer

`Timer` — a program block executes that provides timer and counter behavior.

Description

Provides timer behavior to control process execution. Counter functions include Increment, Decrement and Toggle. Supports common repeat behavior. This is a design-time control and is not visible in run mode.

Toggle Button

`Toggle Button` — a push button with an optional two-state toggle.

Description

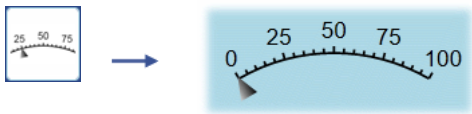
Has a normal and a pushed state that can have toggle behavior or a press-and-release behavior. Each state has associated text, text color, and a numeric value.

Top Sweep Gauge

`Top Sweep Gauge` — a horizontal curved gauge.

Description

Used to graphically represent real-time data. In its default configuration, this gauge is a curved upper-portion of a circle with a needle. With Edit Mode set to Drag, the user can interact with the gauge in run mode. This is a highly-configurable control. Properties like the value, value range, editing abilitiy, and angle range are all modifiable.

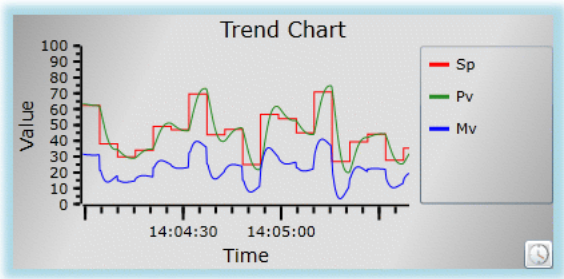


Trend

`Trend` — two chart controls can track up to 3 or 8 data points.

Description

These two charts (3-pen and 8-pen) allow a user to assign values and data points to three or eight trend lines. They track the values and variations of each point as they change over time. These Trend Charts also leverage the power of the Data Historian.



Two Input Calculator

`Two Input Calculator` — a program block that performs calculations on two input values.

Description

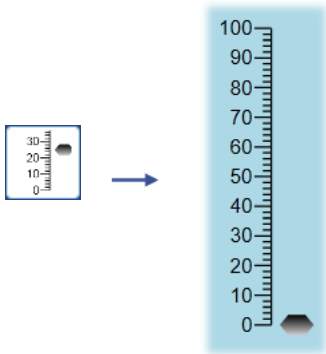
Calculates various mathematical and logical output values using two inputs (DataPoints or values). Functions include: Sum, Difference, Product, Quotient, Modulo, Minimum, Maximum, Round, etc. This is a design-time control and is not visible in run mode.

Vertical Linear Gauge

`Vertical Linear Gauge` — a linear vertical gauge with a slider.

Description

Used to graphically represent real-time data. In its default configuration, this linear gauge is oriented vertically. With Edit Mode set to Drag, the user can interact with the gauge in run mode. This is a highly-configurable control. Properties like the value, value range, editing abilitiy, and angle range are all modifiable.



Index

U

V

Colophon

This book was produced by Cogent Real-Time Systems, Inc. from a single-source group of SGML files. Gnu Emacs was used to edit the SGML files. The DocBook DTD and related DSSSL stylesheets were used to transform the SGML source into HTML, PDF, and QNX Helpviewer output formats. This processing was accomplished with the help of OpenJade, JadeTeX, Tex, and various scripts and makefiles. Details of the process are described in our book: Preparing Cogent Documentation, which is published on-line at
`http://developers.cogentrts.com/cogent/prepdoc/book1.html`.

Text and illustrations by Bob McIlvride and Paul Benford.