Documentation Library

DataHub® Scripting

Version 7.3

Cogent Real-Time Systems, Inc.

July 24, 2013

DataHub® Scripting: Version 7.3

A user's guide to scripting in the DataHub.

Published July 24, 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 7.3-1 September 2007
Added 'What's Different' chapter.
Revision 6.2-1 September 2005
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

Chapter 1. Introduction

The Cogent DataHub has a powerful, built-in scripting language called *Gamma*. Using Gamma, you can interact with the DataHub and its data in various ways, such as:

- Run a script whenever a data point value changes.
- Build custom dashboards and summary displays,
- Create self contained DataHub applications.
- Create alarm condition scripts that display warning messages.
- Connect to ODBC compliant relational databases to extract data as well as create records from live data.
- Create server simulation programs to test production systems before you ’go live’.

Chapter 2. What's Different About DataHub Scripting?

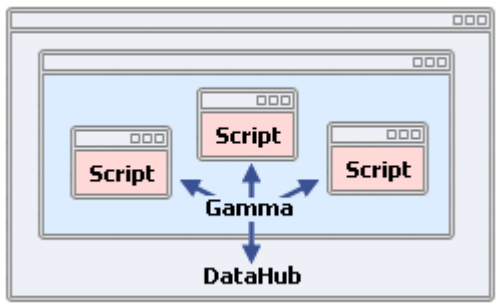
2.1. Scripts and their Environment

The DataHub scripting environment is different from most other programming paradigms because its primary purpose is to allow users to interact with the highly dynamic real-time process data that flows through the DataHub. Although syntactically similar to C, DataHub scripts offer unique capabilities, due in part to these features:

- Dynamic Environment
- Event Driven
- Object Oriented

2.1.1. Dynamic Environment

DataHub scripts run in a *dynamic environment*. The DataHub scripting engine, called Gamma, starts when the DataHub starts, and runs continuously until the DataHub shuts down. You can think of Gamma as a kind of processing power grid that's always switched on and running in the background. All of the data in the DataHub is available in this grid. DataHub users can tap into the power of the grid through scripts.



Scripts are like tools plugged into the grid. All of the live data in the DataHub is available to each script. A user script can be started manually or automatically, and typically runs until the DataHub shuts down. Scripts can range in complexity from the simple Hello World example in this manual to the entire Data Logging interface of the DataHub itself.

In addition to scripting, you can access Gamma interactively, from a command line in the Script Log.

2.1.2. Event Driven

DataHub scripts are *event-driven*, meaning that they respond to events as they occur. In our power grid analogy above, when you start a script, it's like switching on a tool that's plugged into the grid. The tool sits in standby mode, ready to respond when needed. It has a been programmed to respond in a particular way to certain events.

This behavior is difficult to achieve using a typical linear program that gets executed instruction by instruction. For example, a DataHub script has no mainline. Instead, a DataHub script contains two major elements:

1. **Event handlers** contain the code to be run when a DataHub point changes value.
2. **The `.OnChange` method** specifies the event (usually a data change) for which an event handler should be invoked. This change may be caused by an alarm condition or a timer firing or any other real-world event represented by a point in the DataHub. The `.OnChange` method binds a data change event in the DataHub to a specific event handler.

   Please refer to Partial Evaluation for an example.

When the script is run, Gamma loads all the bound event handlers into memory, and waits. Whenever a bound point in the DataHub changes value, Gamma executes the event handler code.

2.1.3. Object Oriented

DataHub user scripts are *object oriented*. Each user script creates a class derived from a base class called the `Application` class. This approach keeps the variables and methods of the script together in a tidy package, allowing them to be global in scope within the class, but separate from any other scripts running in Gamma. To make the scripts easy to write, the New Script File dialog from the Scripting option of the DataHub Properties window creates an instance of the `Application` class automatically, complete with templates for methods. Based on this template, a typical script contains the following:

1. A Gamma `require` function to load the `Application` class.
2. A class definition for the script's class. By default this class gets the same name as the script.
3. Method definitions for event handlers and other functions.
4. A `constructor` method, containing one or more calls to the `.OnChange` method of the `Application` class (see explanation above). This is as close to a "mainline" as a DataHub script gets, but in reality there is no linear flow to the program once all the code has been read in.
5. A `destructor` method, specifying any code that needs to be cleaned up when the class is destroyed or the script shuts down.
6. Class instantiation. Once the class has been instantiated, it just sits there and responds to events.

Example

Here is what a new, unedited template for the class `MyApp` would look like:

```
/* All user scripts should derive from the base "Application" class */

require ("Application");

/* Get the Gamma library functions and methods for ODBC and/or
 * Windows programming.  Uncomment either or both. */

//require ("WindowsSupport");
//require ("ODBCSupport");

/* Applications share the execution thread and the global name
 * space, so we create a class that contains all of the functions
 * and variables for the application.  This does two things:
 *   1) creates a private name space for the application, and
 *   2) allows you to re-load the application to create either
 *      a new unique instance or multiple instances without
 *      damaging an existing running instance.
 */
class MyApp Application
{
}
```

```
/* Use methods to create functions outside the 'main line'. */
method MyApp.samplemethod ()
{
}

/* Write the 'main line' of the program here. */
method MyApp.constructor ()
{
}

/* Any code to be run when the program gets shut down. */
method MyApp.destructor ()
{
}

/* Start the program by instantiating the class.  If your
 * constructor code does not create a persistent reference to
 * the instance (self), then it will be destroyed by the
 * garbage collector soon after creation.  If you do not want
 * this to happen, assign the instance to a global variable, or
 * create a static data member in your class to which you assign
 * 'self' during the construction process.  ApplicationSingleton()
 * does this for you automatically. */
ApplicationSingleton (MyApp);
```

2.2. Symbols, Variables, and Evaluation

Complementing their dynamic environment and event-driven behavior, DataHub scripts have a slightly different approach to symbols and variables compared to other programming languages like C or Java.

2.2.1. Symbols and Variables

One of the fundamental units of Gamma, the DataHub scripting language, is a symbol. Symbols are made up of one or more alphanumeric characters, as well as "_". A symbol gets created whenever a unique group of characters appears in a script for the first time; from that point on the symbol is a unique object in Gamma. When first created, symbols are variables. The value of a variable can be assigned and reassigned at any point in the script. Variables that are not assigned a value have a default value of `_undefined_`.

Variable Scoping and Dynamic Typing

Similar to many programming languages, variables in Gamma can be local or global in scope. At the same time, to provide maximum flexibility, variables in Gamma are dynamically typed. Each time a variable is assigned a value, Gamma assigns or reassigns the type for that variable, based on the new value. This facilitates rapid development and eliminates the need to type or even declare all variables before they are used. Of course, good programming principles must still be observed when writing scripts to ensure that the variables are of the correct type for the circumstances.

2.2.2. The Read/Evaluate Cycle

When a DataHub script is run, Gamma first parses the script, reading and evaluating each statement in turn. For functions or methods, first each argument gets evaluated, and then each statement gets evaluated. Variables get evaluated to their value. Literals like numbers, strings, arrays, and so on get evaluated to themselves. This read/evaluation cycle iterates through the progam on a recursive basis until the entire code gets read and evaluated. When the process is complete, Gamma executes the code.

Preventing Evaluation

Sometimes you might not want Gamma to evaluate a statement or variable when it parses the code. For example, when attaching an event handler to the `.OnChange` method, you don't want the code of the event handler to run until the event actually occurs. To prevent evaluation of a statement or variable, you can use the # character, a Gamma quote operator. Putting the # quote operator in front of any Gamma expression turns it into a literal, causing it to be evaluated to itself, as if it were a number or a string.

Example 1

This example shows an interactive session with Gamma in the Script Log. The `-->` symbols indicate where the user has input an expression, and the next line shows what Gamma has returned as the result of evaluation.

1. First we assign a value of `5` to the variable `myvar`:

   ```
   --> myvar = 5;
   5
   ```

   Gamma returns the value of `myvar`, which is `5`.

2. Then we pass the variable `myvar`, to Gamma:

   ```
   --> myvar;
   5
   ```

   Gamma evaluates it and returns the value, `5`.

3. Now we pass the variable `myvar`, to Gamma, this time quoted using the # symbol.

   ```
   --> #myvar;
   myvar
   ```

   And now Gamma evaluates `myvar` as its literal name, `myvar`, The expression itself has passed through the evaluator intact, without being evaluated.

Partial Evaluation

In some circumstances you might need Gamma to evaluate part of your statement, but not all of it. For this, there are two more quote operators. The `‘` quote operator indicates that this stament should not be evaluated, *except* for those places marked by the `@` quote operator.

Example 2

In this example, we use the Gamma `list` function to illustrate how partial evaluation works. The `list` function creates a space-separated list out of its arguments

1. First, let's define our variables and demonstrate the `list` function.

   ```
   --> myvar = 5;
   5
   --> yourvar = 9;
   9
   --> list(myvar, yourvar);
   (5 9)
   ```

   Gamma first evaluates the arguments of the `list` function to `5` and `9`, then applies the `list` function and puts them into a list: `(5 9)`.

2. Now let's use the # quote operator:

   ```
   --> list(#myvar, #yourvar);
   (myvar yourvar)
   --> #list(myvar, yourvar);
   (list myvar yourvar)
   ```

First we quoted the individual arguments, then the entire expression. Do you see the difference in the result? The second return value, `(list myvar yourvar)` illustrates the internal syntax of Gamma, which is Lisp. Lisp functions are always of this syntax: `(`*`function_name arg`*`1 arg`*`2 ...`*`)`.

3. Now, suppose we want to partially evaluate the expression. First, let's use the ‘ quote operator alone:

```
--> ‘list(myvar, yourvar);
(list myvar yourvar)
```

This gives the same result as the # operator, above. Now let's use the ‘ operator with the @ to allow partial evaluation, like this:

```
--> ‘list(@myvar, @yourvar);
(list 5 9)
--> ‘list(myvar, @yourvar);
(list myvar 9)
```

In the first line, we prvented the evaluation of the `list` function itself, but allowed Gamma to evaluate both of its arguments. In the second line, we allowed the evaluation of only one argument.

4. Here are a few more examples, incorporating the Gamma `string` function, which turns an expression into a string:

```
--> string(list(myvar, yourvar));
"(5 9)"
--> string(#list(myvar, yourvar));
"(list myvar yourvar)"
--> string(‘list(@myvar, @yourvar));
"(list 5 9)"
--> ‘string(list(@myvar, @yourvar));
(string (list 5 9))
--> ‘string(@(list(myvar, yourvar)));
(string (5 9))
```

In each case, the ‘ quote operator prevents the evaluation of the overall expression, while the @ operator allows the evaluation of the sub-expression that immediately follows it.

5. How does this apply to events in a typical DataHub script? Here is an example, using the `.OnChange` method in a class named `Example` with a method called `MethodA`:

This the most important example, because this syntax is commonly used with the `.OnChange` method for handling events.

```
class Example Application
{
...
}

method Example.MethodA (x, y)
{
...
}

method Example.constructor ()
{
     .OnChange (#V1, ‘(@self).MethodA (#V1, #V2));
}
```

In this example, we want to apply `MethodA` of our `Example` class to the values of variable `V1` and `V2` at the exact moment when `V1` changes its value. To do this we protect `V1` and `V2` from evaluation, using the # quote operator. We also do not want to evaluate the `MethodA` method, but we do have to evaluate the key variable indicating the class (`self`). So we use the ‘ operator to prevent the evaluation of the method, and the @ operator to allow the key variable `self` to be evaluated. This way Gamma knows which class the `.MethodA` belongs to.

Forcing Evaluation

In some cases you might need to force Gamma to evaluate an expression. For this, you can use the Gamma `eval` function. For example:

```
--> myvar = 5;
5
--> #myvar;
myvar
--> eval(#myvar);
5
```

2.3. Access to DataHub Points

The main purpose of writing DataHub scripts is to interact with the live data represented by DataHub points. Gamma, the DataHub scripting language, has a few special provisions for working with DataHub points. Understanding these will make your task much easier.

2.3.1. Point Names

First, a DataHub point name is not a legal symbol in Gamma. DataHub point names use a colon (`:`) to separate the point name from the domain name, and commonly use a dot (`.`) as well, like this: `DataSim:Sine` or `MyDomain:Plant1.Tank3.Valve2`. Colons and dots are not normally allowed in a Gamma variable name, so we use the Gamma symbol character operater `$` to turn the whole string of characters into a single, valid Gamma variable. For example, thess expressions:

```
$DataSim:Sine
$MyDomain:Plant1.Tank3.Valve2
```

are valid Gamma variables.

The `$` operator tells Gamma that all characters except white space (space, tab, newline, carriage return, form feed) and the the set: `[]()",';` are accepted in the symbol name. To get any of the above characters in a symbol name, you need to precede that character by a backslash. So, for example, this DataHub point::

```
default:plant.water level[1]
```

would be written in a script like this:

```
$default:plant.water\ level\[1\]
```

In brief, you need to put a `$` before any DataHub point name in a DataHub script, and use a backslash within the name before any whitespace or `[]()",';` character.

2.3.2. Point Values

Often when we use a DataHub point as a variable in a script, we don't want to evaluate it at the time that the code is being read. We want to simply quote it, and let it be evaluated when an event occurs. In these cases, we use the # quote operator when referring to a DataHub point, like this:

```
#$DataSim:Sine
```

Or, if the point name is within an expression that requires partial evaluation, the syntax would be like this:

```
@$DataSim:Sine
```

Please refer to Section 2.2.2, *The Read/Evaluate Cycle* for more details about evaluation, or to Section 4.3, *Accessing Data* for an example of this syntax in use.

2.3.3. Point Timestamps and Qualities

In addition to the value of a DataHub point, you might need to know its timestamp or quality at the moment of an event. This information can be accessed through two special Gamma scripts: `Time.g` and `Quality.g`. You can require these scripts by adding a require function at the beginning of your script, like this:

```
require ("Application");
...
require ("Time");
require ("Quality");
...
```

The `Time.g` script offers a number of time-related functions, while the `Quality.g` file has a single function that converts a numerical quality code into a human-readable text string.

2.4. ODBC and Windows Scripting

2.4.1. DataHub ODBC (Open Database Connectivity) Scripting

DataHub ODBC support allows the DataHub to interface with ODBC-compliant databases. Please refer to the DataHub ODBC Scripting manual for more information.

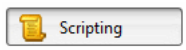
2.4.2. DataHub Windows Scripting

DataHub Windows scripting gives access to over 1700 of the most important classes used for programming in MS Windows, wrapped as Gamma classes. Please refer to the DataHub Windows Scripting manual for more information.

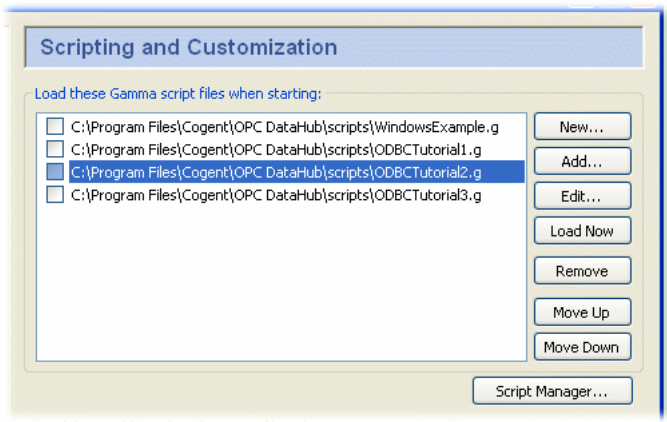
Chapter 3. Getting Started

3.1. How to Run a Script

DataHub scripts run on Gamma, the DataHub scripting engine, which starts up whenever the DataHub starts and runs continuously as long as the DataHub runs. You can access DataHub scripts and scripting capabilities by pressing the Scripting button in the Properties window.

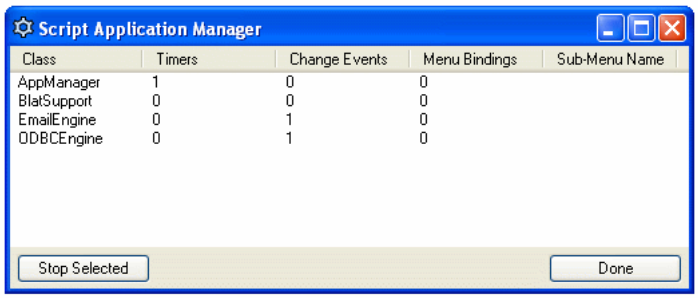


This will display the Scripting and Customization screen. The upper half of the screen shows the Gamma files currently configured in the DataHub:



To run an existing script for the first time, you will need to first add it to the list of scripts. To create a new script, please refer to Section 4.1, *Creating a Script*

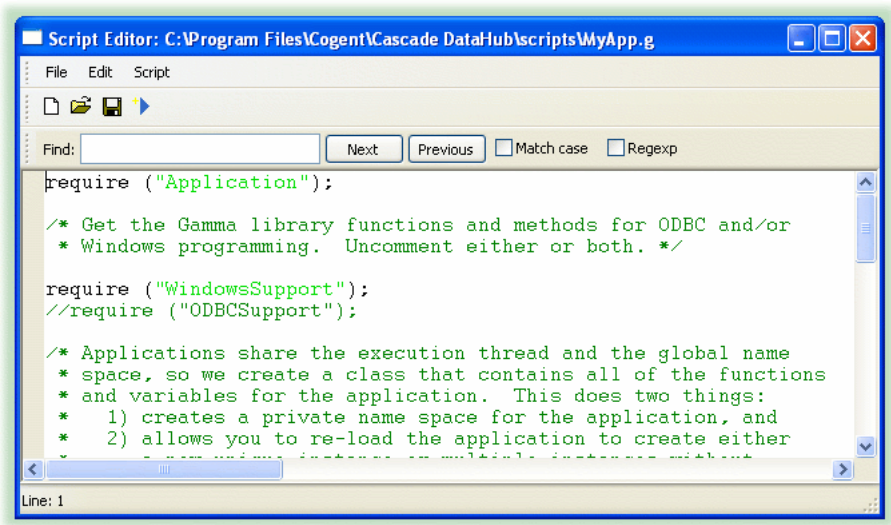
1. To add a script to the list, press the Add button and choose the script from the file selector. Scripts are normally kept in the DataHub's `scripts` subdirectory, such as `C:\Program Files\Cogent\DataHub\scripts\`*`myscript`*`.g`. Be sure to choose a Gamma script (with a `.g` extension).
2. If you need to edit the script before running it, press the Edit button to open the selected script in the Script Editor.
3. To run the script manually, press the Load Now button.
4. To see any script output or error messages, you can press the Script Log button near the bottom of the Properties window to open the Script Log.
5. To configure a script to run automatically at startup, check the checkbox next to it ☑. The next time you start the DataHub, this script will load and run automatically.
6. Once a script is started, it will continue running until the DataHub shuts down. To stop the script without shutting down the DataHub, press the Script Manager button to open the Script Application Manager.



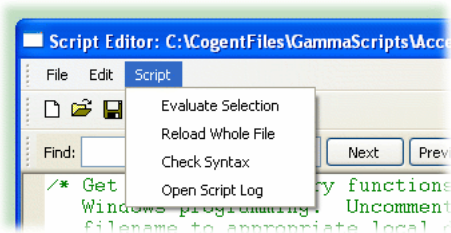
Highlight the script you want to stop, and press the Stop Selected button.

3.2. The Script Editor

The DataHub comes with a built-in Script Editor[1]. You can open the Script Editor from the Scripting option of the Properties window of the DataHub. Select a filename, press the Edit button, and the Script Editor will open:



The Script Editor offers basic script editing features such as context-sensitive highlighting, prompted fill-ins for functions and variable names, automatic indenting, text string searches, and so on. In addition to the normal menu and toolbar options, it has a Script menu that provides the following options:



- Evaluate Section sends whatever block of text you have selected to the Gamma interpreter for immediate processing.

- Reload Whole File sends the entire file to the Gamma interpreter for immediate processing, without you having to save the file first. This functionality is also activated by pressing the blue arrow icon on the toolbar.
- Check Syntax checks the syntax of the whole file without running the script. Any errors will be displayed in the Script Log.
- Open Script Log opens the Script Log.

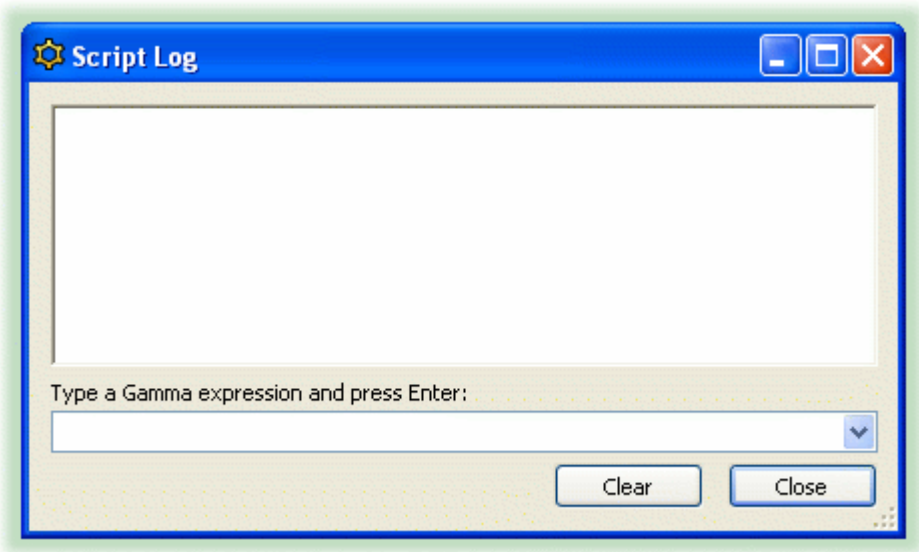
The Script Editor toolbar has a blue arrow icon  in addition to three standard icons for creating, opening, and saving files. Clicking the blue arrow saves the script as written and runs it.

3.3. The Script Log

The Script Log displays output from Gamma scripts, and can also be used to conduct interactive sessions, like a terminal. You can open this window in either of two ways:

- Click the Script Log button in the Properties window.

The Script Log should appear on your desktop:



You can use the text entry field at the bottom to send code to Gamma. Try the following:

1. Type: **`a = 5;`** and press **Enter**

   You should see the following on the screen:

   ```
   --> a = 5;
   5
   ```

   You have just created a *symbol* (a) and assigned it a value (5). That symbol is a variable, and is now available to Gamma until the DataHub shuts down. Notice that the Script Log inserts a prompt (`-->`) and shows your command to help you identify what you typed in.

2. Press the Clear button to clear the Script Log. Press the Close button to close the Script Log window, then reopen it.

3. Type: **`a;`** and press **Enter**

   You should see the following on the screen:

```
--> a;
5
```

Sure enough, the value of `a` is still in Gamma

4. Type: **`princ("Hello world.\n");`** and view the results:

```
--> princ("Hello world.\n");
Hello world.
t
```

Why the `t`? It is the return value from the `princ` function, a logically true value. Every Gamma function returns a value. The string '`Hello world.`' is the byproduct or result of running the function, but the actual return value is `t`. For more details on Gamma programming, please refer to the Gamma manual.

5. Now, let's see a value in the DataHub. Start DataSim, then type: **`$DataSim:Sine;`** and press **Enter**. You should see something like this:

```
--> $DataSim:Sine;
-0.47552825816976968
```

This was the value of the `Sine` point in the `DataSim` domain of the DataHub at the moment you pressed the **Enter** key.

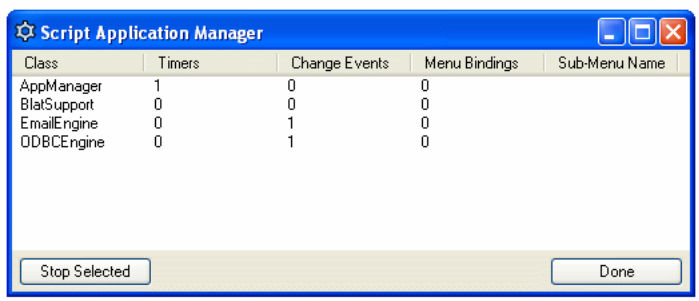
The colon character (`:`) is used to divide the domain name from the point name. The dollar sign character ($) tells Gamma that the colon is part of the variable name, not a sytactic element.

6. You can re-enter up to the last 10 commands by pressing the down arrow on your keyboard. Try it now. Press the down arrow until you see the last command, **`$DataSim:Sine;`**, and press **Enter**. Try it several times. You will get different values because the DataSim program is running.

This gives you a taste of working with Gamma, but to accomplish anything really useful and to save your work, you'll need a script. The following sections will explain how to access and edit scripts, and create your own.

3.4. The Script Application Manager

The Script Application Manager lets you view the scripts currently running in the DataHub, and stop selected scripts if desired. You can open the Script Application Manager by pressing the Script Manager button from the Scripting option of the DataHub Properties window.



To stop a script, highlight it, and press the Stop Selected button.

The columns display the following information:

- Class: the name of the instance of the Application class created in the script.
- Timers: the number of timers active in this script.

- Change Events: the number of change events active in this script.
- Menu Bindings: the number of menu entries that this script has placed in the system tray menu.
- Sub-Menu Name: the name of the submenu in the system tray menu into which this script has placed its menu entries.

Notes

1. This editor is based on the Scintilla and SciTE editor.

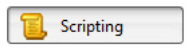
Chapter 4. Writing Scripts

Writing a script for the DataHub is not difficult, particularly when you follow a few basic principles. Working directly with DataHub points and the Gamma interpreter environment creates special opportunities, and you can take best advantage of them by using the suggestions offered here.

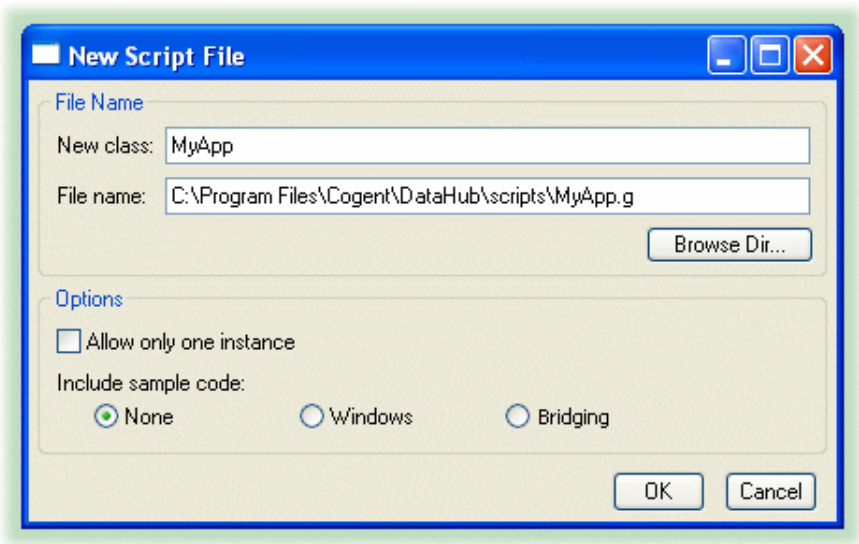
Most of the examples in this chapter use DataSim, which is installed with the DataHub. You should ensure that it is connected and sending data to the DataHub before attempting these examples. For more information, please refer to DataSim in the Cogent DataHub manual.

4.1. Creating a Script

The best way to write a DataHub script is to create a single class in which your entire script runs. This keeps all variables and functions (methods) local to the class and isolated from any other script running in Gamma. Rather than create the class from scratch, the DataHub writes a template for you that contains what you need. Here's how it works:

1. Open the DataHub Properties window and select the Scripting 

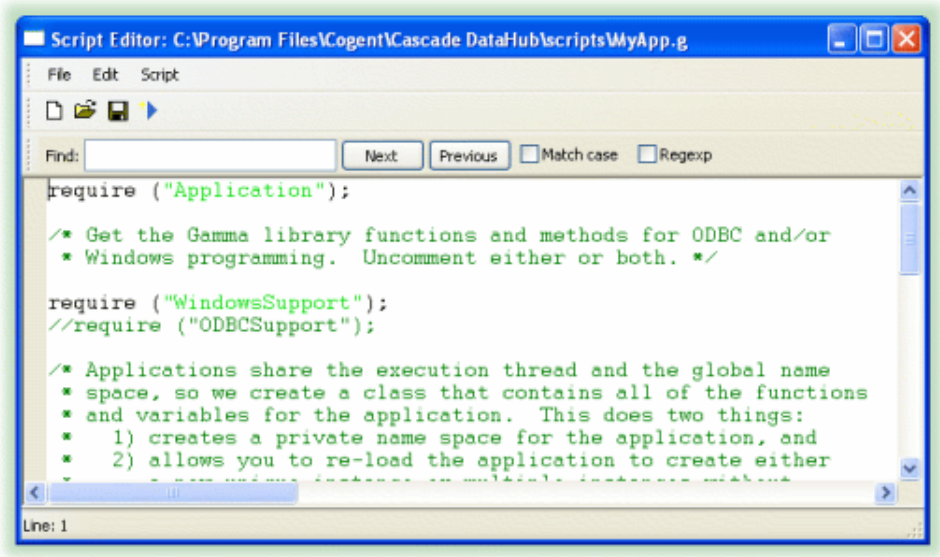
option.
2. Click the New button. This dialog will appear:



3. In the New class: field, enter a name of your choice. The default is `MyApp`.
4. Look at the File name: field to make sure this is where you want the script to be created. If not, you can browse your file system for a better location.
5. Checking the Allow only one instance box will ensure that each instance of the class gets destroyed whenever a new instance is created. Thus you will only ever have one instance. If you want your code to create multiple simultaneous instances of the class, don't check this box.
6. The Include sample code option lets you choose one of the following:
   - None will generate no sample code.
   - Windows will generate code to help you create windows.
   - Data Manipulation will generate code for doing linear transformations and other data manipulation.

These options will be illustrated and discussed in upcoming sections.

7. Click the OK button to create the file.



The Script Editor should open, displaying a script template ready for editing. The basic template looks like this:

```
/* All user scripts should derive from the base "Application" class */

require ("Application");

/* Get the Gamma library functions and methods for ODBC and/or
 * Windows programming.  Uncomment either or both. */

//require ("WindowsSupport");
//require ("ODBCSupport");

/* Applications share the execution thread and the global name
 * space, so we create a class that contains all of the functions
 * and variables for the application.  This does two things:
 *   1) creates a private name space for the application, and
 *   2) allows you to re-load the application to create either
 *      a new unique instance or multiple instances without
 *      damaging an existing running instance.
 */
class MyApp Application
{
}

/* Use methods to create functions outside the 'main line'. */
method MyApp.samplemethod ()
{
}

/* Write the 'main line' of the program here. */
method MyApp.constructor ()
{
}

/* Any code to be run when the program gets shut down. */
method MyApp.destructor ()
{
}

/* Start the program by instantiating the class.  If your
 * constructor code does not create a persistent reference to
 * the instance (self), then it will be destroyed by the
 * garbage collector soon after creation.  If you do not want
```

```
 * this to happen, assign the instance to a global variable, or
 * create a static data member in your class to which you assign
 * 'self' during the construction process.  ApplicationSingleton()
 * does this for you automatically. */
ApplicationSingleton (MyApp);
```

The following sections explain how to edit this template and run the finished script.

4.2. Hello World

This simple example demonstrates how to edit a new script by editing the `.constructor` method.

Edit the Constructor

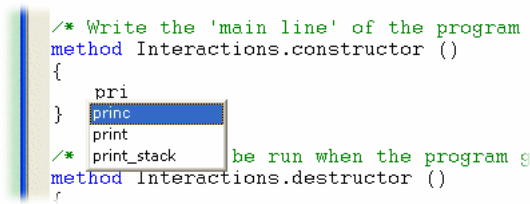
1. Create a new script whose main class is called '`HelloWorld`'. Here's how.
2. The body of a typical script is written as part of the `.constructor` method. Find the following lines to start editing:

   ```
   /* Write the 'main line' of the program here. */
   method HelloWorld.constructor ()
   {
   }
   ```

3. In between the brackets of the `.constructor` method, enter the following:

   ```
   pri
   ```

   Notice that a drop-down box appears:

   This box will appear any time you begin to write a function or variable name that is already available in Gamma.

4. Continue writing:

   ```
   princ(
   ```

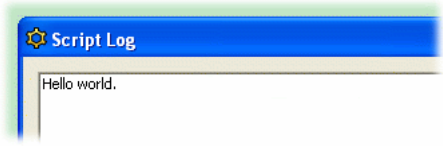
   The box now shows you the `princ` function syntax, in this case a symbolic expression (`s_exp`), commonly know as a symbol. Please refer to the Gamma manual for more information about function syntax and arguments.

5. Continue editing the `princ` function until your `.constructor` function looks like this:

   ```
   /* Write the 'main line' of the program here. */
   method HelloWorld.constructor ()
   {
       princ("Hello world.\n");
   }
   ```

Run the Script

Click the blue arrow icon  in the Script Editor toolbar to run the script, and then check the results in the Script Log window:



If you don't get a '`Hello world`' string in the Script Log, see Appendix A, *Basic Troubleshooting*.

Evaluate a Selection

Here's a way to evaluate just a part of your code:

1. In the Script Editor, use the cursor to select just the text:

   ```
   princ("Hello world.\n");
   ```

2. From the Script menu, select Evaluate Selection.

   You should see the string '`Hello world.`' appear in the Script Log.

This feature of the Script Editor lets you run any part of a script without running the whole thing.

4.3. Accessing Data

A DataHub script has complete access to all the data in the DataHub. Every point in the DataHub is available in Gamma as a global variable, with the syntax:

$*domain_name*:*point_name*

The dollar sign character ($) tells Gamma that the colon is part of the variable name. A colon in Gamma is normally a syntactic element and can't be used in a variable name. However, the DataHub uses a colon (:) in point names to separate the domain name from the rest of the point name. So we need to use the dollar sign. The dollar sign is not part of the variable name.

For these examples, ensure that the DataSim program is running and connected to the DataHub.

Access a Value Once

To start with, here's how to read a value one time.

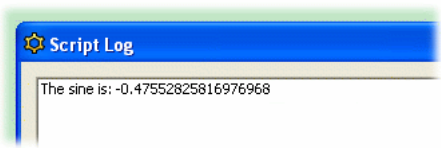
1. Create a new script with a class named `AccessData`, selecting Allow only one instance. Here's how.

2. Scroll down to the `.constructor` method, and enter the following:

   ```
   method AccessData.constructor ()
   {
       princ("The sine is: ", $DataSim:Sine, "\n");
   }
   ```

3. Make sure the DataSim program is running, and that the Script Log is open so you'll be able to see the script output.

4. Click the blue arrow icon  in the Script Editor toolbar. You should see a value appear in the Script Log window, like this:



5. Click the blue arrow icon several more times. Each time, a new value should appear in the Script Log. The values will differ because the value for the sine wave in DataSim is constantly changing.

Access Values Continuously

Getting a single value from the DataHub is fine, but we can do much better. Let's print each new value from a point every time it changes.

1. First you should move the `princ` statement out of the `.constructor`. This may seem trivial in this example, but it is a good habit to get into. Move the `princ` statement to the `AccessData`.sample method, and edit the method name and `princ` statement to look like this:

```
method AccessData.print_point (pt)
{
    princ("The sine is: ", pt, "\n");
}
```

2. Edit the `.constructor` method like this:

```
method AccessData.constructor ()
{
    .OnChange(#$DataSim:Sine,
              '(@self).print_point($DataSim:Sine));
}
```

   The `OnChange` method is inherited from the parent class, `Application`. This method is a wrapper for the `on_change` function, which tells Gamma to evaluate an expression when a given symbol changes value. In this case, the symbol is `$DataSim:Sine` and the expression is our `print_point` function. The first expression is protected from evaluation by the # quote operator.

   The second expression is also quoted so that it doesn't run until the point actually changes. However, the `self` that is usually understood must be explicitly written and evaluated so that Gamma knows what the `.print_point` method applies to. Therefore, we use the `'` and `@` quote operators.

3. Click the blue arrow icon  in the Script Editor toolbar.

   A new value gets written twice a second. To see it really fly, bring up the DataSim window, click the More button, and change the Update Frequency to 200 (don't forget to click Apply Changes). To make it stop, click DataSim's Pause button.

   But what if you can't stop the data flow? How do you get the `princ` function to stop?

Stop Accessing Values

Often you will want a script's change functions to stop when the class instance is destroyed. Conveniently, the Application class has a destructor that runs any time a child class gets destroyed, and which removes all `OnChange` functions. One way to destroy the instance of class is with a timer.

The timer is used here for demonstration purposes. Most Windows programs get destroyed by clicking a button or some other means.

1. Go to `.constructor` method and adding the following code:

```
after(3, 'destroy(@self));
```

   This uses Gamma's `after` timer function, which in this case will activate 3 seconds after the instance is constructed and cause the instance to destroy itself. Again, to prevent the `destroy` function from being evaluated and destroying our instance prematurely, we have to quote it. And again, we use the `'` and `@` quote operators to evaluate the `self` argument. After 3 seconds, the `destroy` function will be evaluated, and the instance gets destroyed.

2. Your two methods should now look like this:

```
method AccessData.print_point (pt)
{
    princ("The sine is: ", pt, "\n");
}

method AccessData.constructor ()
{
    .OnChange(#$DataSim:Sine,
              '(@self).print_point($DataSim:Sine));
    after(3, 'destroy(@self));
}
```

3. Run the script. You should see output in the Script Log for 3 seconds, then nothing.

Verify Results

1. To verify that the instance of the class has been destroyed, type **`instance_p(_AccessData_Singleton)`** in the text entry field at the bottom of the Script Log and press **Enter**. The output should look like this:

   ```
   --> instance_p(_AccessData_Singleton);
   nil
   ```

   The `instance_p` function is a Gamma predicate that checks to see if `accessdata` is an instance. Gamma returns `nil`, indicating that it is not an instance.

2. Since `accessdata` was an instance and is no more, it is probably a destroyed instance. To check, type **`destroyed_p(_AccessData_Singleton);`** and press **Enter**. The output should look like this:

   ```
   --> destroyed_p(_AccessData_Singleton);
   t
   ```

   Gamma returns `t`, indicating that it is a destroyed instance.

4.4. Modifying Data

DataHub scripting lets you modify data as it passes through the DataHub. This example script makes a linear transformation, writes the results to a different DataHub point, and incidentally prints the value of point and the conversion.

1. Create a new script whose main class is called '`ModifyData`'. Here's how.
2. Change the name of `.samplemethod` to `.convert` and edit it like this:

   ```
   method ModifyData.convert (pt1, !pt2, multiplier, adder)
   {
       local output;
       set(pt2, ((pt1 * multiplier) + adder));
       princ(format("The sine is: %2.3f    Converted it is: %2.3f\n",
                    pt1, eval(pt2)));
   }
   ```

   This method does the conversion and prints the output. Notice that the `pt2` argument has an exclamation point (`!`) in front of it. This protects the argument from being evaluated, because we only want the point name, not its value. Please refer to Function Arguments in the Gamma manual for more information.

   We need to assign a value to the point, but keep the variable name associated with the DataHub point, not the value. This requires the `set` function, because using the equals sign would just assign the value to the variable, and it would never reach the DataHub point.

3. Edit the `.constructor` method to look like this:

```
method ModifyData.constructor ()
{
    multiplier = 3;
    adder = 5;

    if (undefined_p($default:ConvertedSine))
        datahub_command ("(cset default:ConvertedSine \"\")", 1);

    .OnChange(#$DataSim:Sine,
              '(@self).convert($DataSim:Sine,
                               $default:ConvertedSine, multiplier, adder));

    after(3, 'destroy(@self));
}
```

We don't make the `multiplier` and `adder` variables local because they'll be used by the `Application class's .destructor` method. We use the `datahub_command` function call the `cset` function to have the DataHub create the `ConvertedSine` point and set its value to an empty string. We then use the inherited `.OnChange` method to set up the `.convert` method.

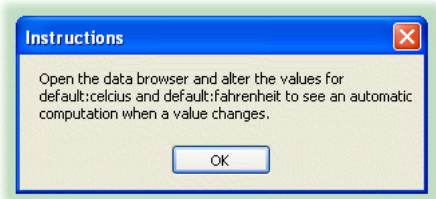
4. The script is ready to run. Open the Data Browser window right-clicking the DataHub icon in the System Tray and selecting View Data from the pop-up menu. Select the default domain. Also make sure the DataSim program is running and the Script Log is open.
5. Run the script. It should run for 3 seconds, writing data to the Script Log, and changing the values to the point `ConvertedSine` in the default domain.

   Of course, the data printed in the Script Log is just for convenience. You could comment out the `princ` statement, or remove it altogether. The main thing is the data updating in the Data Browser.

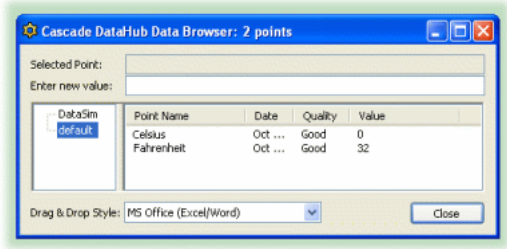
4.5. Two-Way Data Manipulation

The previous example, Section 4.4, *Modifying Data*, showed how to modify data from a source point to a destination point. But what if you need to modify data *bidirectionally*—such as keeping two readings of Celsius/Fahrenheit temperature data in synch? You can quickly set up a script for two-way data manipulation like this:

1. Open the Properties window, select the Scripting option, and click the New button to create a new script.
2. In the New Script File dialog, name the main class '`MyXform`' and select the Allow only one instance and Bridging options. More details.
3. Click OK and the script window will open, loaded with the newly-created script. If you'd like you can now add the file to your list of files in the Properties window. Here's how.
4. When you run the script the Data Browser will open, along with an Instructions dialog:



5. Click OK and open the Data Browser if it isn't open already:



In the `default` domain, you should see two new points: `Celsius` and `Fahrenheit`.

6. Click on point name `Celsius` and in the Enter new value: field, enter a new value. You should see the value for `Fahrenheit` change accordingly. Change the value of `Fahrenheit`, and `Celsius` will change as well.

This is an example of bi-directional linear transformation between two points. It is based on the `LinearXform` class, which is defined in the `LinearXForm.g` script that comes with your DataHub distribution. The class gets instantiated by the code in the script you just created and are now running. Here are the main parts of your `MyXform.g`script, with a bit of commentary:

```
class MyXform Application
{
    xform = new LinearXform ();
}
```

The `LinearXform` gets instantiated as an instance variable of the `MyXform` class.

```
/* This method configures all of the bridging transformation for this example.
   To add more bridging transformations, simply copy and modify the line
   starting with .xform.AddLinearXform. */
method MyXform.bridgeconfig ()
{
    .xform.verbose = t;
    .xform.AddLinearXform (self, #$default:Celsius, #$default:Fahrenheit, 9/5, 32, t);
}
```

As the comment suggests, here you can add points with transformation arguments. The syntax is explained below. All that is left to do now is construct the class. The constructor calls `.bridgeconfig` to build the data transformation bridges, and then sets initial values, opens the Data Browser, and displays the usage instructions:

```
method MyXform.constructor ()
{
    /* Create the transformations */
    .bridgeconfig();

    /* Set initial values */
    $default:Fahrenheit = 32;

    /* Display the data view window */
    datahub_command ("(show_data 1)", 1);

    /* Give instructions to the user */
    MessageBox (0, "Open the data browser and alter the values for
default:Celsius and default:Fahrenheit to see an automatic
computation when a value changes.", "Instructions", 0);
}
```

4.5.1. Adapt it for your needs

This code can easily be adapted to do any two-way linear transformation between two points. Just add lines to the `.bridgeconfig` method to meet your needs. Here is the syntax:

```
    .xform.AddLinearXform (self, #$domain:source, #$domain:target, multiplier, adder, bidirec-
tional);
```

The arguments you need to specify are:

`domain`:*`source`*

The domain and name of the data point that is the source of the data. Remember, the # symbol prevents evaluation, and the `$` escapes the colon (`:`) between the domain and point name. Also, you will need to use a forward slash (\) to escape blank spaces in a point name. Windows allows blank spaces in point names, but you have to escape them in DataHub scripts.

`domain`:*`target`*

The domain and name of the data point that is the target of the data.

`multiplier`

A multiplier (`m`) for the linear transformation, such as in the equation: `y = mx + b`

`adder`

An adder (`b`) for the linear transformation, such as in the equation: `y = mx + b`

`bidirectional`

Indicates whether the data can be written from either direction. Use t for yes, or nil for no. These are the only two values that may be used.

For example, for currency exchange you could add this line to the `.bridgeconfig` method:

```
.xform.AddLinearXform (self, #$default:euros, #$default:dollars, .82, 0, t);
```

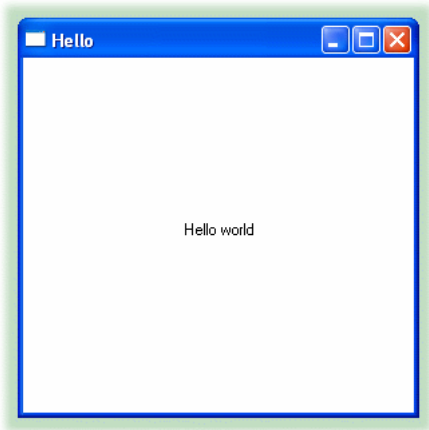
To set `dollars` to `100`, add this line to the `.constructor` method:

```
$default:dollars = 100;
```

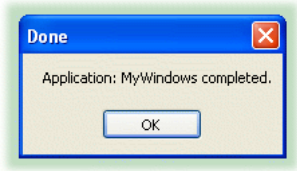
4.6. Making a Window

DataHub scripting offers many of the Windows classes wrapped as Gamma classes. Thus you can create windows, buttons, entry forms, tabs, dialogs, and so on. The DataHub Windows Scripting manual contains more information. This is how you create a basic window.

1. Open the Properties window, select the Scripting option, and click the New button to create a new script.
2. In the New Script File dialog, name the main class '`MyWindows`' and select the Windows option. More details.
3. Add the file to your list of files, and load it now. Here's how. A new window will open:



4. Click the close icon in the top right corner. The window will close, and you will see this dialog box:



This is an example of a basic window. Here are the main parts of your `MyWindows.g` script:

```
class MyWindows Application
{
    window;
}
```

The window itself is an instance variable of the `MyWindows` class.

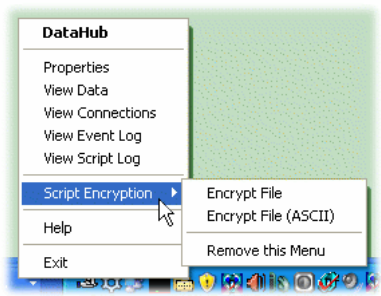
```
method MyWindows.constructor ()
{
    local    rect = CreateRect (0, 0, 300, 300), txt;
    .window = new GWindow();

    .window.Create (0, rect, "Hello", WS_OVERLAPPEDWINDOW, 0);
    .window.CenterWindow();
    txt = .window.CreateControl (GStatic, 0, 0, 280, 22, "Hello world", SS_CENTER);
    txt.CenterWindow();
    .window.MessageHandler (WM_DESTROY, 'destroy(@self));
    .window.ShowWindow (SW_SHOW);
}

method MyWindows.destructor ()
{
    // The WM_DESTROY message could come before or after this destructor depending
    // on whether the application instance is destroyed or the window is closed
    // first.  We protect against the case where the window is closed first.
    if (instance_p(.window) && .window.GetHwnd() != 0)
        .window.PostMessage (WM_CLOSE, 0, 0);
    MessageBox(0, string ("Application: ", class_name(self), " completed."), "Done", 0);
}
```

More information about Windows scripting and the Windows classes and methods can be found in the DataHub Windows Scripting manual.

4.7. Encrypting a Script

The DataHub archive contains a script called `EncryptScript.g` that makes an encrypted copy of any Gamma script. You can encrypt your scripts as follows:

1. Run the `EncryptScript.g` script. Here's how.
2. Right-click on the DataHub icon in the system tray to open the DataHub menu. You should see a new menu entry: Script Encryption.



There are two options for encrypting a file. The Encrypt File option uses 8-bit characters, which produces a smaller, faster loading file than an ASCII file, but it may not transfer properly through some mail or FTP servers. If that is an issue, you can use Encrypt File (ASCII).

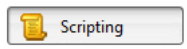
3. Choose either Encrypt File or Encrypt File (ASCII). This will open a file selection dialog.
4. Select the Gamma (`.g`) file that you want to encrypt. If the encryption succeeds, you will get a success message. The encrypted script will have the same name as the original script, except with a `.gmc` extension.
5. You can add the encrypted script to your list of scripts now, and run it. Here's how. The encrypted script should behave exactly the same as the `.g` file, but will be difficult for a user to examine.

4.8. Scripting Tips

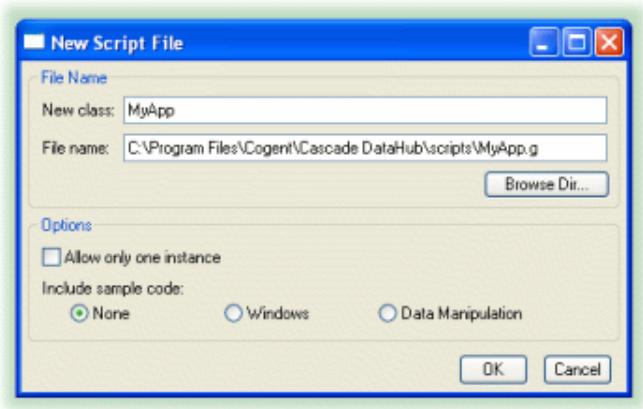
Here are some ways to facilitate scripting.

4.8.1. Copying a complete tutorial

The code from DataHub tutorials can be copied directly from the text into the Script Editor. Here's how to make a new script from a complete tutorial.

1. Open the DataHub Properties window and select the Scripting 

option.
2. Click the New button. This dialog will appear:



3. In the New class: field, enter a name of your choice.
4. Click the OK button to create the file. The Script Editor will open, displaying a script template.
5. Using the cursor or **Ctrl-A**, select all the text and delete it.
6. Copy a complete tutorial from the DataHub Windows Help manual or online documentation into the Script Editor.
7. Save and run the script using the blue arrow icon  .

4.8.2. Setting up a scripting environment

If you are working on one script regularly and need to restart the DataHub often, here's a way to automatically open the Properties window, the Script Log and the Script Editor with your file loaded.

1. Open the Properties window and Script Log.

2. Position them on the screen, then close them. They will come up in the same place the next time you open them. (The Script Editor does not yet have this feature.)
3. Write a script of these three lines:

```
datahub_command ("(show_properties 1)");
datahub_command ("(show_script_log 1)");
edit_file ("c:\\projects\\myfile.g");
```

   Two slash marks (`\\`) are necessary—the first slash mark escapes the second one.
4. Add this script to the list of scripts in the DataHub, and check the activation box beside it. Then next time you start the DataHub, the properties and script log windows will come up where they were before, and an editor will be opened on the file `c:\projects\myfile.g`.
5. When you don’t want the script to auto-load, just un-check the activation box.

Chapter 5. The `Application` class

The `Application` class is the parent class for all the applications you create with the New button in the Scripting option of the Properties window. It provides an environment that makes it easy for you to program for changes and events. It also allows you to set up timers, and to add menus for your scripts to the DataHub's pop-up system tray menu. This chapter contains an overview of the `Application.g` file, where the `Application` class and its methods are defined.

5.1. Class Definition

This code defines the base `Application` class:

```
/* A base application class that keeps track of change functions
   and removes them when the object is destroyed */

if (undefined_p(Application) || !class_p(Application))
{
    class Application
    {
        _ChangeFunctions;
        _TimerIDs;
        _MenuActions;
        _SubMenus;
    static:
        _Instances;
        _MenuItemID = 20000;
        _MenuItems;
        _TraySubmenu;
        _AllMenuActions;
        _TrayMenuPosition = 6;
    }
}
```

Before defining the class, we test to see if it already exists, using the Gamma predicates `undefined_p` and `class_p` functions in an `if` statement. We want only a single instance of the class so that there is only one copy of the `_Instances` static variable in the system. This `_Instances` variable tracks the currently loaded applications, and we do not want several different copies of it floating about.

The instance variable `_ChangeFunctions` is a list of all the change functions that are defined by the class's `.OnChange` method. The `_TimerIDs`, `_MenuActions`, and `_Submenus` instance variables are similar lists for any timer IDs, menu actions, and custom submenus that may be defined for the class.

5.2. Construction and Destruction

The `constructor` and `destructor` methods help with general class housekeeping.

```
method Application.constructor ()
{
    ._Instances = cons (self, ._Instances);
}
```

The `constructor` adds each instance of the class to the list of all the class instances currently defined in Gamma. The Gamma `cons` function is used to build the list.

```
method Application.destructor ()
{
    local   id;

    ._Instances = remove (self, ._Instances);
    .RemoveAllEventHandlers ();
    .RemoveAllMenus ();
```

```
    if (!._AllMenuActions)
    {
        .RemoveSystemMenu();
    }
}
```

When an instance of the class gets destroyed, this `destructor` method will be run first. It removes this instance of the class from the `_Instances` list using the Gamma `remove` function. Then it removes all event handlers and menus, using the `RemoveAllEventHandlers`, `RemoveAllMenus`, and `RemoveSystemMenu`, as applicable.

5.3. Handling Events

Event handlers are bound to events using the `OnChange` method.

```
method Application.OnChange (sym, fn)
{
    local    chfn = cons (sym, fn);
    ._ChangeFunctions = cons (chfn, ._ChangeFunctions);
    on_change (sym, fn);
    chfn;
}
```

The `OnChange` method is a wrapper for the `on_change` function that does the actual binding of the event handler. First the `OnChange` method creates a two-member list (`chfn`) consisting of a symbol (`sym`) and the function that should run (`fn`) when the symbol changes value. That short list is added to the class's `_ChangeFunctions` list. All lists are constructed using the Gamma `cons` function. Finally, the `on_change` function links the symbol to the event-handling function. What gets returned, `chfn`, is a two member list—exactly what the unwrapped `on_change` function would have returned.

One way to remove an event handler is with the `RemoveChange` method.

```
method Application.RemoveChange (chfn)
{
    ._ChangeFunctions = remove (chfn, ._ChangeFunctions);
    remove_change (car(chfn), cdr(chfn));
}
```

This is a wrapper on `remove_change`, to be used when you need to remove just a single function from `_ChangeFunctions` rather than all of them. See also `RemoveAllChanges` and `RemoveAllEventHandlers`.

5.4. Timers

There are three different methods for attaching a timer to an event-handling method or function: `TimerAt`, `TimerAfter`, and `TimerEvery`. They use the Gamma functions `at`, `after`, and `every` respectively. For example:

```
method Application.TimerEvery (seconds, fn)
{
    local    tid = every (seconds, fn);
    ._TimerIDs = cons (tid, ._TimerIDs);
    tid;
}
```

This is a wrapper on the Gamma `every` function. It creates a timer ID and adds it to the class's list of timer IDs (`_TimerIDs`). This way the timer can be identified and removed when necessary.

```
method Application.RemoveTimer(tid)
{
    if (find_equal (tid, ._TimerIDs))
    {
```

```
            cancel (tid);
            .droptimer (tid);
        }
    }
```

This method uses the Gamma `find_equal` function to locate a timer according to its ID number (`tid`), and then uses the Gamma `cancel` function to cancel it.

Timers created using `TimerAt`, `TimerAfter`, and `TimerEvery` are automatically cancelled when the Application instance is destroyed.

5.5. Menus

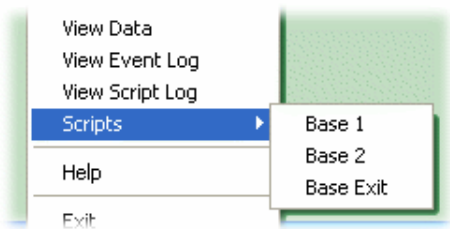
There are a number of methods in the `Application` class that you can use to create submenus and menu items on the DataHub pop-up system tray menu. Rather than examining the code, let us look at what kinds of menus can be generated. Each of the menus below has two regular items that simply print their name in the Script Log, as well as an "`Exit`" item that shuts down the script. A complete, working script containing all of these examples is given below. All of the methods used here are documented in Reference IV, *Methods and Functions from `Application.g`*.

Putting Menu Items in the Scripts Submenu

The methods used here cause a Scripts submenu to be created on the main DataHub pop-up menu, and attach items directly to that submenu. This code:

```
.AddCustomMenuItem("Base 1", 'princ("Base 1\n"));
.AddCustomMenuItem("Base 2", 'princ("Base 2\n"));
.AddStopMenuItem("Base Exit");
```

Would add these menu items.

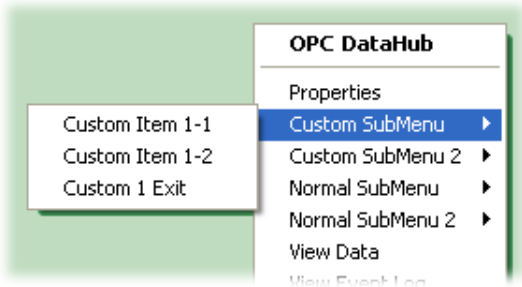


Creating a Submenu using Convenience Methods

The methods used here are convenience methods that wrap the standard methods used below. They create a submenu on the main DataHub menu, attaching menu items to the immediately preceding parent, in the order written in the code. This code:

```
.AddCustomSubMenu("Custom SubMenu", 3);
.AddCustomMenuItem("Custom Item 1-1", 'princ("Custom Item 1-1\n"));
.AddCustomMenuItem("Custom Item 1-2", 'princ("Custom Item 1-2\n"));
.AddStopMenuItem("Custom 1 Exit");
```

Would put up this submenu.

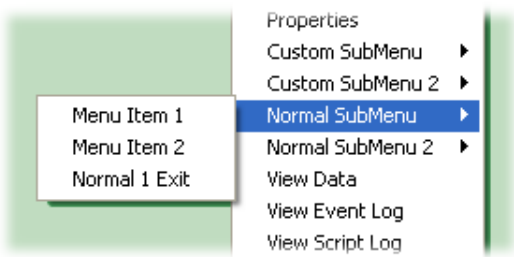


Creating Submenus on the Main Menu

The methods used here are the standard methods used to create submenus on the main DataHub menu. Each menu item is identified by its parent, so menu items can be added in any order. This code:

```
local    mymenu, mymenu2;
mymenu = .AddSubMenu(get_tray_menu(), 3, "Normal SubMenu");
mymenu2 = .AddSubMenu(get_tray_menu(), 4, "Normal SubMenu 2");
.AddMenuItem(mymenu, -1, "Menu Item 1", 'princ("Menu Item 1-1\n"));
.AddMenuItem(mymenu2, -1, "Menu Item 1", 'princ("Menu Item 2-1\n"));
.AddMenuItem(mymenu, -1, "Menu Item 2", 'princ("Menu Item 1-2\n"));
.AddMenuItem(mymenu2, -1, "Menu Item 2", 'princ("Menu Item 2-2\n"));
.AddMenuItem (mymenu, -1, "Normal 1 Exit", 'destroy (@self));
.AddMenuItem (mymenu2, -1, "Normal 2 Exit", 'destroy (@self));
```

Would put up two submenus, the first of which is open here.



The `menutest.g` script

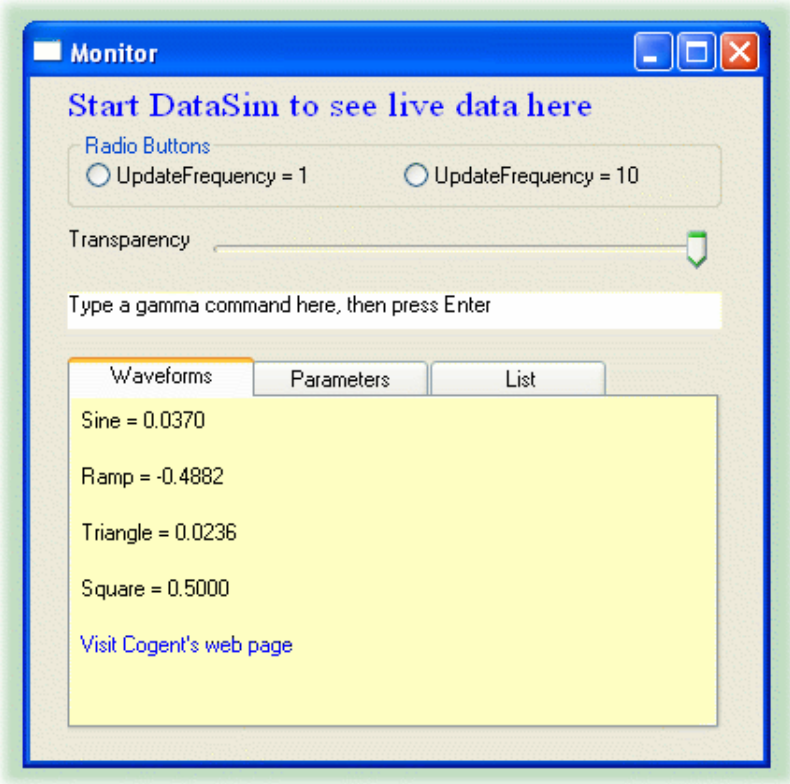
This script contains all of the above examples.

```
/* All user scripts should derive from the base "Application" class */

require ("Application");

/* Get the Gamma library functions and methods for ODBC and/or
 * Windows programming.  Uncomment either or both. */

//require ("WindowsSupport");
//require ("ODBCSupport");

/* Applications share the execution thread and the global name
 * space, so we create a class that contains all of the functions
 * and variables for the application.  This does two things:
 *   1) creates a private name space for the application, and
 *   2) allows you to re-load the application to create either
 *      a new unique instance or multiple instances without
 *      damaging an existing running instance.
 */
class menutest Application
{
}

/* Use methods to create functions outside the 'main line'. */
```

```
/* Create submenus and add items to them, using the AddCustom* convenience methods.
   Note that the items are added in sequential order for each menu. */
method menutest.custom()
{
    .AddCustomSubMenu("Custom SubMenu", 3);
    .AddCustomMenuItem("Custom Item 1-1", 'princ("Custom Item 1-1\n"));
    .AddCustomMenuItem("Custom Item 1-2", 'princ("Custom Item 1-2\n"));
    .AddStopMenuItem("Custom 1 Exit");

    .AddCustomSubMenu("Custom SubMenu 2", 4);
    .AddCustomMenuItem("Custom Item 2-1", 'princ("Custom Item 2-1\n"));
    .AddCustomMenuItem("Custom Item 2-2", 'princ("Custom Item 2-2\n"));
    .AddStopMenuItem("Custom 2 Exit");
}

/* Create regular submenus and add items to them, using the Add* methods.
   Note that the items can be added in any order. */
method menutest.direct()
{
    local    mymenu, mymenu2;
    mymenu = .AddSubMenu(get_tray_menu(), 3, "Normal SubMenu");
    mymenu2 = .AddSubMenu(get_tray_menu(), 4, "Normal SubMenu 2");
    .AddMenuItem(mymenu, -1, "Menu Item 1", 'princ("Menu Item 1-1\n"));
    .AddMenuItem(mymenu2, -1, "Menu Item 1", 'princ("Menu Item 2-1\n"));
    .AddMenuItem(mymenu, -1, "Menu Item 2", 'princ("Menu Item 1-2\n"));
    .AddMenuItem(mymenu2, -1, "Menu Item 2", 'princ("Menu Item 2-2\n"));
    .AddMenuItem (mymenu, -1, "Normal 1 Exit", 'destroy (@self));
    .AddMenuItem (mymenu2, -1, "Normal 2 Exit", 'destroy (@self));
}

/* Write the 'main line' of the program here. */
method menutest.constructor ()
{
    /* Add menu items to a "Scripts" submenu that gets created automatically. */
    .AddCustomMenuItem("Base 1", 'princ("Base 1\n"));
    .AddCustomMenuItem("Base 2", 'princ("Base 2\n"));
    .AddStopMenuItem("Base Exit");

    /* Create the normal menus. */
    .direct();

    /* Create the custom menus. */
    .custom();
}

/* Any code to be run when the program gets shut down. */
method menutest.destructor ()
{
}

/* Start the program by instantiating the class.  If your
 * constructor code does not create a persistent reference to
 * the instance (self), then it will be destroyed by the
 * garbage collector soon after creation.  If you do not want
 * this to happen, assign the instance to a global variable, or
 * create a static data member in your class to which you assign
 * 'self' during the construction process.  ApplicationSingleton()
 * does this for you automatically. */
ApplicationSingleton (menutest);
```

Appendix A. Basic Troubleshooting

Before doing anything else, you should ensure that the DataHub and Gamma are running properly. You can do this by running the `WindowsExample.g` file. Start the DataHub, and in the Scripting option of the Properties window, load the `WindowsExample.g` script. (How do I load a script?) When the script is properly loaded, it should display the Monitor window:



This is a demonstration script for DataHub MS Windows Support. If the DataSim program is running, you should be able to see its data and interact with it through this Monitor. If you cannot, you need to check your DataHub installation and configuration before troubleshooting scripts.

Troubleshooting Scripts

Did the script not run as expected? What happened instead? Here are some suggestions:

- The Script Log displays one or more error messages with relevant line numbers if there are any syntatical or scripting errors, and Gamma will stop executing the script at that point.
- Incorrect output or no output probably means that your script's logic is incorrect. Review your code, correct any errors, and load it again.
- Single expressions, whole lines, or entire blocks of code can be evaluated independently. Highlight the portion of the code text and select Evaluate Selection from the Script menu.
- You can trace the steps of execution by inserting `princ` statements at strategic points.

Appendix B. License Copyright Information

The following licenses are being used in this product:

License for Scintilla and SciTE

Copyright 1998-2003 by Neil Hodgson <neilh@scintilla.org>

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

NEIL HODGSON DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL NEIL HODGSON BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

I. Example Scripts

Table of Contents

LogFile.g

`LogFile.g` — logs data to a text file when a point changes value.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/* This script shows how to log data to a text file. It uses a trigger point
 * to signal an alarm condition (non-zero), which causes a value to be written
 * to the file.
 *
 * To use this script with your points, replace 'default:triggerpt' in the LogFile
 * class with your trigger point, and replace 'default:loggedpt' with the point
 * whose value you wish to log.  You can also change the name of the log file.
 */

/* All user scripts should derive from the base "Application" class */

require ("Application");

/* Get the Gamma library functions and methods for ODBC and/or
 * Windows programming.  Uncomment either or both.
 */

require ("WindowsSupport");
//require ("ODBCSupport");

/* Applications share the execution thread and the global name
 * space, so we create a class that contains all of the functions
 * and variables for the application.  This does two things:
 *   1) creates a private name space for the application, and
 *   2) allows you to re-load the application to create either
 *      a new unique instance or multiple instances without
 *      damaging an existing running instance.
 */

class LogFile Application
{
    // The trigger point whose value determines when logging takes place.
    trigger = #$default:triggerpt;

    // The point whose value gets logged.
    logged = #$default:loggedpt;

    // The name of the log file.
    log_file_name = "c:/tmp/logfile.txt";

    // The file handle to the open file
    log_file;
}

/*
 * This method writes the trigger value and the logged point value
 * for alarm conditions and non-alarm conditions.  The first argument
 * is the actual value of the trigger point, and the second is the
 * symbolic name of the point to be logged.
 */
method LogFile.AlarmOccurred(triggervalue, !logpoint)
{
    local    value = eval (logpoint);
    if (triggervalue != 0)
    {
        writec (.log_file, format ("Alarm:   %-20s = %10g\n", string(logpoint), value));
```

```
        princ ("Alarm condition: ", logpoint, ", ", value, "\n");
    }
    else
    {
        writec (.log_file, format ("Cleared: %-20s = %10g\n", string(logpoint), value));
        princ ("No alarm: ", logpoint, ", ", value, "\n");
    }
    flush (.log_file);
}

/* Write the 'main line' of the program here. */
method LogFile.constructor ()
{
    /* If the trigger and logged points don't exist in the DataHub, create them.*/
    datahub_command (string ("(create ", .trigger, " 1)"), 1);
    datahub_command (string ("(create ", .logged, " 1)"), 1);

    /* Attempt to open the log file. */
    .log_file = open (.log_file_name, "a");
    if (!.log_file)
    {
        MessageBox (0, string ("Could not open alarm log file: ", .log_file_name),
                            "Error opening file", 0);
    }
    else
    {
        /* Log the data whenever the trigger point changes. */
        .OnChange (.trigger, '(@self).AlarmOccurred (value, @.logged));
    }
}

/* Any code to be run when the program gets shut down. */
method LogFile.destructor ()
{
    if (.log_file)
        close (.log_file);
}

/* Start the program by instantiating the class.  If your
 * constructor code does not create a persistent reference to
 * the instance (self), then it will be destroyed by the
 * garbage collector soon after creation.  If you do not want
 * this to happen, assign the instance to a global variable, or
 * create a static data member in your class to which you assign
 * 'self' during the construction process.  ApplicationSingleton()
 * does this for you automatically. */
ApplicationSingleton (LogFile);
```

ReadCSV.g

`ReadCSV.g` — reads a CSV file and writes the points and values to the DataHub.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/*
 * This script reads a CSV file and writes the values found there
 * into a set of data points in the DataHub.  The format of the
 * file is:
 *
 * row 1:  name1, name2, name3, ...
 * row 2:  value1, value2, value3, ...
 * row 3:  value1, value2, value3, ...
 *  ...
 * row N:  value1, value2, value3, ...
 *
 * The script will read all rows in the file, but ignore all but
 * the first and last.  The first row contains the point names and
 * the last contains the most recent data.
 * If a name is left blank then that column is ignored.
 * If a point name does not contain a domain name, then the domain
 * set in the "domain" member of the application is used.
 *
 * e.g.,
 *    default:point1, default:point2, default:point3
 *    1, 2, 3
 *    4, 5, 6
 *
 * will result in:
 *    default:point1 = 4
 *    default:point2 = 5
 *    default:point3 = 6
 *
 * Strings containing ',' characters must be quoted within double quotes,
 * like this:
 *    "hello, friend"
 *
 * Double-quotes within strings must be escaped, like this:
 *    "He said, \"hello\"."
 *
 * This script will guess whether a value is a number or a string.  If the
 * value can be parsed to a number, it is treated as a number.  Otherwise it
 * is a string.
 *
 * This script looks for new data at a set time interval.
 *
 * This script will operate in one of two modes:
 *    In "reload" mode, the file is re-read from the beginning on each
 *    timer tick.
 *    In "append" mode, the file is kept open, and the file is read
 *    from the last read position on each timer tick.  This mode will
 *    not work if the writing application does not open the file
 *    as "shared".
 *
 * This script adds a menu item to the OPC DataHub system tray icon that
 * allows the user to re-load the file, change reade mode, and toggle logging
 * to the Script Log window.
 */

require ("Application");
```

```
class ReadCSV Application
{
    mode = #reload;  // set to #reload or #append
    domain = "default";
    filename = "c:/tmp/data.csv";
    verbose = t;
    update_secs = 5;
    separators = ","; // e.g, use " " for space separated, or "\t" for tab-separated

    /* --- No need to change these --- */
    columns;
    fptr;
    modemenu;
    verbosemenu;
}

/* Logging function that prepends the time to the output. */
method ReadCSV.Log (args...)
{
    if (.verbose)
    {
        funcall (princ, cons (date(), cons(": ", args)));
        princ("\n");
    }
}

/* Open the given file, if possible. */
method ReadCSV.OpenFile (filename)
{
    .fptr = open(filename, "r");
    if (!.fptr)
    {
        .Log ("Could not open file: ", filename);
    }
    else
    {
        .filename = filename;
        .Log ("File: ", filename, " opened");
        .ReadColumns();
    }
    .fptr;
}

method ReadCSV.CloseFile ()
{
    if (.fptr)
    {
        close(.fptr);
        .Log ("File: ", .filename, " closed");
        .fptr = nil;
    }
}

method ReadCSV.Trim(str)
{
    local  l = strlen(str), start, end;
    for (start=0; start<l && strchr(" \t",str[start]) != -1;)
        start++;
    for (end=l-1; end >= start && strchr(" \t",str[end]) != -1;)
        end--;
    if (start != 0 || end != l-1)
        substr(str,start,end-start+1);
    else
        str;
}

method ReadCSV.ReadColumns ()
{
```

```
    local  line = read_line (.fptr);
    local  i;

    if (line != _eof_)
    {
        line = list_to_array(string_split(line,.separators,0,t,"\"\"",t,"\\",nil));
        for (i=0; i<length(line); i++)
        {
            if (.Trim(line[i]) == "")
            {
                line[i] = nil;
            }
            else
            {
                if (strchr(line[i],':') == -1)
                    line[i] = string(.domain,":",line[i]);
                line[i] = symbol(line[i]);
                datahub_command(format("(create %s 1)", stringc(line[i])),1);
            }
        }
        .columns = line;
        .Log("Set columns to ", .columns);
    }
}

method ReadCSV.GuessTypeValue (str)
{
    local  value;

    try
    {
        value = parse_string(str,nil);
        if (!number_p(value))
            value = str;
    }
    catch
    {
        value = str;
    }
    value;
}

method ReadCSV.ApplyLine (line)
{
    local  i, value;

    .Log ("Applying line: ", line);
    if (line)
    {
        line = list_to_array(string_split(line,.separators,0,t,"\"\"",nil,"\\",nil));
        for (i=0; i<length(.columns); i++)
        {
            if (.columns[i])
            {
                value = .GuessTypeValue(line[i]);
                //.Log ("Set: ", .columns[i], " to ", stringc(value));
                if (value)
                    set(.columns[i], value);
            }
        }
    }
}

method ReadCSV.ReadLines ()
{
    local  line, input;

    .Log ("Looking for new data...");
```

```
    while ((input = read_line(.fptr)) != _eof_)
    {
        if (.Trim(input) != "")
            line = input;
    }
    if (line)
    {
        .ApplyLine(line);
    }
}

method ReadCSV.ReadFile (filename)
{
    if (!.fptr)
        .OpenFile(filename);
    if (.fptr)
    {
        .ReadLines();
        if (.mode == #reload)
            .CloseFile();
    }
}

method ReadCSV.SetMode (mode)
{
    .mode = mode;
    .Log("Set read mode to ", mode);
    .ChangeMenuItemLabel(.modemenu,
                         string("Set ", (mode == #append) ? "Reload" : "Append",
                                " Mode"));
    if (.filename)
        .Reload(.filename);
}

method ReadCSV.ToggleMode ()
{
    .SetMode((.mode == #append) ? (#reload) : (#append));
}

method ReadCSV.ToggleVerbose ()
{
    .SetVerbose(!.verbose);
}

method ReadCSV.SetVerbose (mode)
{
    .verbose = t;
    .Log("Set verbosity to ", (mode ? "verbose" : "quiet"));
    .verbose = mode;
    .ChangeMenuItemLabel(.verbosemenu,
                         string(mode ? "Quiet" : "Verbose", " Mode"));
}

method ReadCSV.Reload (filename)
{
    .CloseFile();
    .ReadFile(filename);
}

/* Write the 'main line' of the program here. */
method ReadCSV.constructor ()
{
    .TimerEvery(.update_secs, '(@self).ReadFile((@self).filename));
    .AddCustomSubMenu("CSV File Reader");
    .AddCustomMenuItem("Reload CSV File", '(@self).Reload((@self).filename));
    .modemenu = .AddCustomMenuItem("Set Append Mode", '(@self).ToggleMode());
    .verbosemenu = .AddCustomMenuItem("Verbose", '(@self).ToggleVerbose());
    .SetVerbose(.verbose);
```

```
    .SetMode(.mode);
}

method ReadCSV.ChangeMenuItemLabel (menuitemid, label)
{
    local parent = .CreateSystemMenu();
    local info = new MENUITEMINFO();

    if (cons_p(menuitemid))
        menuitemid = car(menuitemid);
    info.cbSize = 48;
    info.fMask = MIIM_STRING | MIIM_ID;
    info.fMask |= (WINVER < 0x0500 ? MIIM_TYPE : MIIM_FTYPE);
    info.fType = MFT_STRING;
    info.wID = menuitemid;
    info.dwTypeData = label;
    SetMenuItemInfo (parent, menuitemid, 0, info);
}

method ReadCSV.destructor ()
{
    .CloseFile();
}

ApplicationSingleton (ReadCSV);
```

WriteCSV.g

`WriteCSV.g` — writes data to CSV files.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/*
 * Write data to a number of different CSV files.  Each data set is written at
 * a different interval, either per second, per minute or per hour.  A new CSV
 * file is created every hour or every day.
 *
 * The points names in each data set are read from a file.  The file consists of each
 * point name, one per line.  The output file will contain a time stamp followed
 * by the value for each point in the order that the points are listed in the
 * point name file.
 *
 * To change the format of the file name, change the method "GenerateFileName".
 *
 * To change the extension of the file, change the member variable "filesuffix".
 *
 * To change the file names and timing, change the .NewDataSet calls in the
 * method WriteCSV.constructor.
 */

require ("Application");
require ("CSVSupport");

class WriteCSV Application
{
    datasets;
}

class MyCSVWriter CSVWriter
{
    pointfile;            // The name of a file containing the point list.
    sample_rate;          // one of #second, #minute, #hour or list(hour,minute,second)
    file_rotate_rate;     // one of #minute, #hour, #day or list(hour,minute,second)
    filesuffix = ".csv";  // normally .csv.  Override it to create .txt files.
}

/* This will produce a file name like:
 *     base_20090423_PM3.csv
 */

/*
method MyCSVWriter.GenerateFileName()
{
    local tm = localtime(nanoclock());
    format("%s_%d%02d%02d_%s%d%s", .filebase,
           tm.year+1900, tm.mon+1, tm.mday,
           tm.hour >= 12 ? "PM" : "AM",
           tm.hour > 12 ? tm.hour - 12 : (tm.hour == 0 ? 12 : tm.hour),
           .filesuffix);
}
*/

/* This will produce a file name like:
 *     base-20090423-1545.csv
 */
method MyCSVWriter.GenerateFileName()
{
    local tm = localtime(nanoclock());
```

```
    format("%s-%d%02d%02d-%02d%02d%s", .filebase,
           tm.year+1900, tm.mon+1, tm.mday,
           tm.hour, tm.min,
           .filesuffix);
}

/*
 * Create a new writer and start the timers to write new data and to
 * create a new log file.
 */
method WriteCSV.NewDataSet (output_file_base, pointfile, sample_rate, file_rotate_rate, separate_lines)
{
    local writer = new MyCSVWriter();
    if (writer.ReadPointsFromCSV(pointfile))
    {
        writer.sample_rate = sample_rate;
        writer.file_rotate_rate = file_rotate_rate;
        writer.SetFileBase(output_file_base);
        writer.SetSeparateLines(separate_lines);

        switch(writer.sample_rate)
        {
            case (#second):
                .TimerAt(nil,nil,nil,nil,nil,nil,'(@self).WriteData(@writer));
            case (#minute):
                .TimerAt(nil,nil,nil,nil,nil,0,'(@self).WriteData(@writer));
            case (#hour):
                .TimerAt(nil,nil,nil,nil,0,0,'(@self).WriteData(@writer));
            case (#day):
                .TimerAt(nil,nil,nil,0,0,0,'(@self).WriteData(@writer));
            default:
                if (list_p(writer.sample_rate))
                {
                    // List of hour, minute, second specification
                    local  times = list_to_array(writer.sample_rate);
                    .TimerAt(nil,nil,nil,times[0],times[1],times[2],'(@self).WriteData(@writer));
                }
                else // Default is hourly
                {
                    .TimerAt(nil,nil,nil,nil,0,0,'(@self).WriteData(@writer));
                }
        }
        switch(writer.file_rotate_rate)
        {
            case (#minute):
                .TimerAt(nil,nil,nil,nil,nil,0,'(@self).RotateFile(@writer));
            case (#hour):
                .TimerAt(nil,nil,nil,nil,0,0,'(@self).RotateFile(@writer));
            case (#day):
                .TimerAt(nil,nil,nil,0,0,0,'(@self).RotateFile(@writer));
            default:
                if (list_p(writer.file_rotate_rate))
                {
                    // List of hour, minute, second specification
                    local  times = list_to_array(writer.file_rotate_rate);
                    .TimerAt(nil,nil,nil,times[0],times[1],times[2],'(@self).RotateFile(@writer));
                }
                else // Default is hourly
                {
                    .TimerAt(nil,nil,nil,nil,0,0,'(@self).RotateFile(@writer));
                }
        }
        .datasets = cons(writer, .datasets);
    }
}

method WriteCSV.WriteData(writer)
{
```

```
    writer.WriteLine();
}

method WriteCSV.RotateFile(writer)
{
    .TimerAfter(0.1, '(@writer).IncrementFileName());
}

/* Write the 'main line' of the program here. */
method WriteCSV.constructor ()
{
    /* Specify the CSV files to write.  Modify the .NewDataSet lines to specify how
     * to write each CSV file.
     * Arguments are:
     *    output_file_base:  The first part of the file, before the date string
     *    pointfile:  The name of a file containing the point names for this data set
     *    sample_rate:  One of #second, #minute, #hour or list(hour,minute,second)
     *     telling how frequently to write a line to the CSV file
     *    file_rotate_rate:  One of #minute, #hour, #day or list(hour,minute,second)
     *     telling how frequently to create a new CSV file.
     *    separate_lines:  If this is nil, write all point values on one line.  If this
     *                  is t, write each point value on a separate line.
     *
     * This function will read the pointfile as a CSV file and treat the first field in each
     * row as the name of a point to be recorded into the output file.
     *
     * When specifying timing, the input is the list of (hour,minute,second) as accepted
     * by the "at" function (see documentation).  A value of nil for hour, minute or second
     * stands for all values for that parameter.  A list of values for hour, minute or second
     * specifies an event only when the hour, minute or second matches one of the values in
     * the list.
     * Examples:
     *    list(nil,nil,nil)    - every second
     *    list(nil,list(0,15,30,45),0) - every 15 minutes
     *   list(nil,list(0,15,30,45),30) - every 15 minutes, 30 seconds past the minute
     *    list(list(0,8,16), 0, 0)      - at midnight, 8AM and 4PM exactly
     *    list(list(0,8,16), 5, 0)      - at 12:05AM, 8:05AM and 4:05PM
     *
     * The symbols #second, #minute, #hour, #day are conveniences for:
     *    second:   list(nil,nil,nil)  - any hour, any minute, every second
     *    minute:   list(nil,nil,0)    - any hour, every minute at 0 seconds
     *    hour:     list(nil,0,0)      - every hour, on the hour
     *    day:      list(0,0,0)        - at midnight (0 hour, 0 minute, 0 second)
     */
    .NewDataSet("c:/tmp/Group1", "c:/tmp/Group1_Points.txt", #second, #hour, nil);
    .NewDataSet("c:/tmp/Group2", "c:/tmp/Group2_Points.txt", #second, list(nil,list(0, 15,30,45),0), nil);
    .NewDataSet("c:/tmp/Group3", "c:/tmp/Group3_Points.txt", #second, list(nil,nil,0), nil);
}

/* Any code to be run when the program gets shut down. */
method WriteCSV.destructor ()
{
    with writer in .datasets do
    {
        destroy(writer);
    }
}

/* Start the program by instantiating the class.  If your
 * constructor code does not create a persistent reference to
 * the instance (self), then it will be destroyed by the
 * garbage collector soon after creation.  If you do not want
 * this to happen, assign the instance to a global variable, or
 * create a static data member in your class to which you assign
 * 'self' during the construction process.  ApplicationSingleton()
 * does this for you automatically. */
ApplicationSingleton (WriteCSV);
```

XMLReader.g

`XMLReader.g` — reads an XML file from a URL.

Code

The script allows you to process the data you receive in any way you like, and gives an example of printing the data with the Gamma `princ` function. If instead you would like to write the data to a point in the DataHub, you can replace this code:

```
            with point in model._xml_children do
            {
                princ (point.name, " = ", point.value, "\n");
    }
```

with this

```
            with point in model._xml_children do
            {
                datahub_write (string("domain_name:", point.name), point.value);
    }
```

where *`domain`* is the name of the domain in which you want the point to be created. Be sure to include the colon (`:`) at the end of the domain name, and make sure there are no other colons in the point name. For more information, please refer to the Data Domains section of the Cogent DataHub manual, and the `datahub_write` function in this manual.

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:

`C:\Program Files\Cogent\Cogent DataHub\scripts\`

Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/*
This script reads an XML file from a URL and parses it into a class hierarchy that can be
used to run further scripts or to populate the DataHub.

The example in this case requires that DataPid is running and that the DataHub web server
is turned on, as the script is simply reading some data point values from the DataHub web
server in XML format.
*/

require ("Application");

/* Get the Gamma library functions and methods for XML parsing and wget command-line tool */
require ("XMLSupport");
require ("WgetSupport");

class XMLReader Application
{
    DocumentUrl = "http://localhost/points?name=DataPid:PID1.Mv|DataPid:PID1.Pv|DataPid:PID1.Sp";
    TickSeconds = 5;
}

method XMLReader.loadXml ()
{
    // Call wget asynchronously.  Protect against the possibility that the script is stopped
    // while a wget command is still outstanding.
    Wget(.DocumentUrl, 'progn {
        if (!destroyed_p(@self))
            (@self).processWgetResult();
    });
}

method XMLReader.processWgetResult()
{
    if (ExitCode == 0) // success
    {
```

```
        // Print the raw XML document
        princ ("------- Original Document -------\n");
        princ (ResultString);
        princ ("---------------------------------\n");

        // Parse the XML into a class hierarchy that we can easily manipulate
        local reader = scew_reader_buffer_create(string_to_buffer(ResultString));
        local parser = new scew_parser();
        local tree = parser.load(reader);
        local model = tree.create_model("myxml_");

        // Print the whole XML file model.
        // pretty_princ(model, "\n");

        // The XML model consists of a class instance for each XML Element.  Each
        // class has a member called _xml_children that is an array of the sub-elements.
        // In addition, each class has member variables that are named as the attributes
        // of the XML element.  Replace this with your own custom processing.
        with point in model._xml_children do
        {
            princ (point.name, " = ", point.value, "\n");
        }
    }
    else
    {
        princ ("XML read failed with exit code: ", ExitCode, "\n");
    }
}

/* Write the 'main line' of the program here. */
method XMLReader.constructor ()
{
    .TimerEvery(.TickSeconds, '(@self).loadXml());
}

/* Any code to be run when the program gets shut down. */
method XMLReader.destructor ()
{
}

/* Start the program by instantiating the class. */
ApplicationSingleton (XMLReader);
```

ParseExcel.g

`ParseExcel.g` — parses data from an Excel spreadsheet.

Description

This script shows how to read an array of data into a DataHub script from a source such as an Excel worksheet, and then parse and extract values as necessary. The values in this example are message strings that correspond with an alarm, but they could be any value. The results of the triggered alarm are written to a text file.

Required setup

1. In the Excel worksheet:

    a. Select a group of cells in the worksheet that is 2 columns wide by any number of rows deep.

    b. Give this region a name by entering the name `alarm_table` in the Excel Name Box located in the top-left corner of the worksheet.

    c. Select a cell outside of this range, and name it `time_stamp`.

    This example assumes that there is a data feed to this `time_stamp` cell updating it with the most recent time. You could set this up to be coming from a DataHub point if desired, or from any other source.

2. In the DataHub's DDE properties:

    a. Add a new entry in the DDE Client section defined as follows:

Connection Name:	**`alarms`**
Service:	**`Excel`**
Topic:	**`Alarms.xls`** (the name of the Excel file)

    Enter the name `alarm_table` in the Item Names entry field, and choose Add. If the Data Domain column does not say `default`, double-click the name and change it to `default`.

    b. Now create another item named `time_stamp` in the same way.

    c. Select OK to close the DDE Item Definition window.

    d. Press Apply in the main properties window. The status of the DDE Connection should change to Connected.

3. In the DataHub Scripting properties:

    a. Select Add... to add a DataHub script.

    b. Navigate to the file `ExcelAlarms.g`, and choose Open. The script name should now appear in the list of scripts in the Scripting property page.

    c. Select the checkbox to the left of the `ExcelAlarms.g` file name, then press the Apply button. This will cause the script to run whenever the DataHub starts.

    d. With the `ExcelAlarms.g` file selected, press the Edit... button to open the file for editing in the Script Editor.

4. In the Script Editor:

a. Go to line 35 and change the output file name to the name of a file that will receive the alarm log.

b. Press the blue arrow icon  or select Reload Whole File from the Script menu. The script is now running.

5. Close the Script Editor.

You can test the script by manually changing the alarm point values using the DataHub's Data Browser window. If you want to see output to the Script Log as well as the output file, un-comment lines 106, 107, 113, and 114.

This script assumes that the data for the alarm points comes from an outside data source connected to the DataHub via some other configuration.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/* All user scripts should derive from the base "Application" class */

require ("Application");

/* Get the Gamma library functions and methods for ODBC and/or
 * Windows programming.  Uncomment either or both. */

//require ("WindowsSupport");
//require ("ODBCSupport");

/* Applications share the execution thread and the global name
 * space, so we create a class that contains all of the functions
 * and variables for the application.  This does two things:
 *   1) creates a private name space for the application, and
 *   2) allows you to re-load the application to create either
 *      a new unique instance or multiple instances without
 *      damaging an existing running instance.
 */

class ParseExcel Application
{
    domain = "default";

    // The point containing the alarm table from Excel.  This
    // is chosen from the DDE configuration tab in the OPC
    // DataHub properties.
    pt_alarm_table;
    pt_time_stamp;

    // The alarm lookup table.  We parse the table we get from
    // Excel into a more efficient lookup table.
    alarm_lut = make_array(0);

    // The name of the log file.
    log_file_name = "c:/tmp/alarmlog.txt";

    // The file handle to the open file
    log_file;
}

/*
 * Hold information for one alarm entry in the Excel spreadsheet
```

```
 */
class AlarmSpec
{
    tagname;
    point;
    description;
    eventid;
}

method AlarmSpec.constructor (domain, tag, description)
{
    .tagname = tag;
    .description = description;
    .point = symbol(string(domain, ":", tag));
}

/*
 * Comparison function for sorting.  We don't really need to sort unless
 * we plan to write code that looks up an alarm by name in this table.  It
 * is more efficient to use the event handler (.OnChange) function to map
 * a point change to its alarm specification
 */
function CmpAlarmSpecs (alarm1, alarm2)
{
    symcmp (alarm1.point, alarm2.point);
}

method ParseExcel.NewAlarmTable(value)
{
    local    rows = string_split (string(value), "\r\n", 0);
    local    columns;
    local    tagsym;

    with alarm in .alarm_lut do
    {
        .RemoveChange (alarm.eventid);
    }

    .alarm_lut = make_array(0);
    with row in rows do
    {
        columns = list_to_array (string_split (row, "\t", 0));
        .alarm_lut[length(.alarm_lut)] = new AlarmSpec(.domain, columns[0], columns[1]);
    }
    .alarm_lut = sort (.alarm_lut, CmpAlarmSpecs);

    with alarm in .alarm_lut do
    {
        datahub_command (string ("(create ", stringc(alarm.point), " 1)"), 1);
        alarm.eventid = .OnChange (alarm.point, '(@self).AlarmOccurred (@alarm, value));
    }
}

/*
 * This method is called whenever the alarm condition point
 * changes in the OPC server.
 */
method ParseExcel.AlarmOccurred(alarm, value)
{
    if (value != 0)
    {
        writec (.log_file, format ("%-20s%-12s%s\n", $default:time_stamp,
                alarm.tagname, alarm.description));
        //princ (format ("%-20s%-12s%s\n", $default:time_stamp,
        //  alarm.tagname, alarm.description));
    }
    else
    {
```

```
        writec (.log_file, format ("%-20s%-12s%s cleared\n", $default:time_stamp,
                alarm.tagname, alarm.description));
        //princ (format ("%-20s%-12s%s cleared\n", $default:time_stamp,
        //  alarm.tagname, alarm.description));
    }
    flush (.log_file);
}

/* Write the 'main line' of the program here. */
method ParseExcel.constructor ()
{
    .log_file = open (.log_file_name, "a");
    if (!.log_file)
    {
        MessageBox (0, string ("Could not open alarm log file: ", .log_file_name),
                                "Error opening file", 0);
    }
    else
    {
        // Set the point name for the alarm table coming from Excel
        .pt_alarm_table = symbol (string (.domain, ":", "alarm_table"));
        .pt_time_stamp = symbol (string (.domain, ":", "time_stamp"));

        datahub_command (string ("(create ", stringc(.pt_alarm_table), " 1)"), 1);
        datahub_command (string ("(create ", stringc(.pt_time_stamp), " 1)"), 1);

        // Whenever somebody changes the Excel spreadsheet, update the
        // alarm table.
        .OnChange (.pt_alarm_table, '(@self).NewAlarmTable(value));

        // If we already have a value for the alarm table point, create the
        // alarm table.
        if (!undefined_p(eval(.pt_alarm_table)) &&
            string_p(eval(.pt_alarm_table)))
                .NewAlarmTable (eval(.pt_alarm_table));
    }
}

/* Any code to be run when the program gets shut down. */
method ParseExcel.destructor ()
{
    if (.log_file)
        close (.log_file);
}

/* Start the program by instantiating the class.  If your
 * constructor code does not create a persistent reference to
 * the instance (self), then it will be destroyed by the
 * garbage collector soon after creation.  If you do not want
 * this to happen, assign the instance to a global variable, or
 * create a static data member in your class to which you assign
 * 'self' during the construction process.  ApplicationSingleton()
 * does this for you automatically. */
ApplicationSingleton (ParseExcel);
```

LinearXform.g

`LinearXform.g` — performs linear transformation functions on points.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/* This script creates a class that performs linear
   transformation functions on DataHub points. */

class LinearXform
{
    verbose;
}

method LinearXform.cbLinearXform (dst, value, mult, add)
{
    if (.verbose)
        princ ("xform ", dst, " = ", value, " * ", mult, " + ", add, "\n");
    if (number_p(value))
        set (dst, value * mult + add);
    else
        set (dst, value);
}

method LinearXform.AddLinearXform (app, src, dst, mult, add, bidirectional_p?=nil)
{
    datahub_command (string ("(create \"", src, "\" 1)"));
    datahub_command (string ("(create \"", dst, "\" 1)"));
    app.OnChange (src, '(@self).cbLinearXform (#@dst, value, @mult, @add));
    if (bidirectional_p && mult != 0)
    {
        allow_self_reference (src, 1);
        allow_self_reference (dst, 1);
        app.OnChange (dst, '(@self).cbLinearXform (#@src, value, @(1/mult), @(-add/mult)));
    }
}
```

MakeArray.g

`MakeArray.g` — creates an array point from individual points.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/* This script creates an output array point from an input set of individual
 * points.  The array can be any reasonable length, though the algorithm is
 * not efficient for large arrays whose constituent points change quickly.
 *
 * The only part of the code that you need to alter to create your own arrays
 * is the MakeArray.constructor method.
 *
 * If a change is made to any individual point, the array will be updated
 * immediately.
 *
 * If a change is made to the array point, the change will not affect the
 * individual points that make up the array.
 */

require ("Application");

/* Applications share the execution thread and the global name
 * space, so we create a class that contains all of the functions
 * and variables for the application.  This does two things:
 *   1) creates a private name space for the application, and
 *   2) allows you to re-load the application to create either
 *      a new unique instance or multiple instances without
 *      damaging an existing running instance.
 */
class MakeArray Application
{
}

/* Ensure that a point exists in the DataHub data store.  We do this because the
 * startup order of the DataHub vs. the data source is not necessarily predictable.
 * By creating the point now, we are sure that it exists even if the data source
 * starts later.
 */
method MakeArray.CreatePoint(pointname, type?=nil)
{
    datahub_command(format("(create \"%s\" 1)", pointname), 1);
    if (type)
        datahub_command(format("(set_canonical \"%s\" \"%s\" 1)", pointname, type), 1);
}

/* Write the array based on the input point list.  There are more efficient ways to
 * do this if the array is large or the data updates very frequently.  It may also be
 * reasonable to do it on a timer rather than every time any constituent point changes
 */
method MakeArray.EmitArray(arraypoint, inputpoints)
{
    local val = make_array(0), i=0, tmp;
    with point in inputpoints do
    {
        val[i++] = undefined_p(tmp = eval(symbol(point))) ? 0 : number(tmp);
    }
    set (symbol(arraypoint), val);
}

/* This is a convenient function that declares an array point to create from a set of
 * input points.  You can create as many of these array points as you need, using any
```

```
 * number of input points.  Just put all of the input points into the argument list of
 * the call (see the call in the constructor below).
 * The arraytype is a VARIANT array type: I1, I2, I4, R4, R8, BSTR, UI1, UI2, UI4
 * followed by a space and the word "array".  E.g., "R8 array" or "BSTR array".AddCustomMenuItem
 * BSTR means "string".
 */
method MakeArray.DeclareArray(arraypoint, arraytype, inputpoints...)
{
    .CreatePoint (arraypoint, arraytype);
    with point in inputpoints do
    {
        .CreatePoint(point);
        .OnChange(symbol(point), '(@self).EmitArray(#@arraypoint, #@inputpoints));
    }

    // After creating everything, call the EmitArray method once to initialize the
    // array.  We do this in case the constituent points are already present in the
    // data set when the script starts.  Otherwise we would have to wait until one
    // of the points changes before the array gets initialized.
    .EmitArray (arraypoint, inputpoints);
}

/* Write the 'main line' of the program here.  Call the .DeclareArray method one or
 * more times to set up the event handling to construct an array from individual points
 */
method MakeArray.constructor ()
{
    .DeclareArray ("default:pointarray", "R8 array", "default:pointname1", "default:pointname2",
                   "default:pointname3");
}

/* Any code to be run when the program gets shut down. */
method MakeArray.destructor ()
{
}

/* Start the program by instantiating the class.  If your
 * constructor code does not create a persistent reference to
 * the instance (self), then it will be destroyed by the
 * garbage collector soon after creation.  If you do not want
 * this to happen, assign the instance to a global variable, or
 * create a static data member in your class to which you assign
 * 'self' during the construction process.  ApplicationSingleton()
 * does this for you automatically. */
ApplicationSingleton (MakeArray);
```

IntToBit.g

`IntToBit.g` — converts an integer data point into a set of single-bit points.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/*
 * Given an integer value, convert it to a set of data points where each point represents
 * a single bit in the original integer.
 *
 * Usage:
 * Modify the IntToBit.constructor to reflect the data points that need to be broken into
 * individual bits.  This consists of one or more calls to
 *      .initPoint(point, format, nbits)
 *  where
 *      point - a symbol or string representing the input integer data point
 *              specify symbols with #$domain:symbol_name or as a string
 *      format - a format string defining how to create the point names for the
 *               individual bits, given the symbol name and a bit number (starting at 0)
 *               use %a for the format specifier for input point name
 *      nbits - the number of bits to extract from the input, starting at the LSB
 */

require ("Application");

class IntToBit Application
{
}

/* Add any data points that you want to split into bits here */
method IntToBit.constructor ()
{
    .initPoint(#$default:test, "%a_%d", 8);
    .initPoint("default:test2", "%a_%02d", 16);
    .initPoint(format("default:test%d", 3), "%a_%02d", 32);
}

/* ------- Implementation: No need to change beyond here ----------- */

/* A callback that runs whenever the input integer changes value */
method IntToBit.processNewValue (inputsym, outputformat, value, nbits)
{
    if (string_p(inputsym))
        inputsym = symbol(inputsym);

    local bitsym, bitvalue, i;
    local valueinfo = PointMetadata(inputsym);

    if (valueinfo)
    {
        for (i=0; i<nbits; i++)
        {
            bitsym = format(outputformat, inputsym, i);
            bitvalue = (value >> i) % 2;
            datahub_write(bitsym, bitvalue, nil, valueinfo.quality, valueinfo.timestamp);
        }
    }
}

/* Set up the event handler and initial state for the output points */
method IntToBit.initPoint (inputsym, outputformat, nbits)
{
```

```
    local bitsym, i, curvalue;

    if (string_p(inputsym))
        inputsym = symbol(inputsym);

    for (i=0; i<nbits; i++)
    {
        bitsym = format(outputformat, inputsym, i);
        datahub_command (format("(create %s 1)", stringc(bitsym)), 1);
        datahub_command (format("(set_canonical %s BOOL)", stringc(bitsym)), 1);
    }
    .OnChange(inputsym, '(@self).processNewValue(this, @outputformat, value, @nbits));

    if (!undefined_p(curvalue = eval(inputsym)) && number_p(curvalue))
    {
        .processNewValue(inputsym, outputformat, curvalue, nbits);
    }
}

/* Any code to be run when the program gets shut down. */
method IntToBit.destructor ()
{
}

/* Start the program by instantiating the class. */
ApplicationSingleton (IntToBit);
```

MaskedBridge.g

`MaskedBridge.g` — copies a data point, applying a mask and shift operation.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/*
 * Copy a data point to another data point, applying a mask and shift operation
 * during the copy.  The mask is applied before the shift, allowing you to
 * mask any bits and then shift them right or left.  A negative shift will
 * shift left, and a positive shift will shift right.  The shift indicates the
 * number of bits to shift, where 0 indicates no shift.
 *
 * You can create any number of operations simply by reproducing the .OnChange
 * method call with different pairs of points.  The datahub_command call is simply
 * to ensure that the destination point exists, which may not be necessary depending
 * on your application.
 */

require ("Application");

class MaskedBridge Application
{
}

method MaskedBridge.WriteBits (value, pointname, mask, shift)
{
    if (shift < 0)
        set(pointname, (value & mask) << -shift);
    else
        set(pointname, (value & mask) >> shift);
}

method MaskedBridge.constructor ()
{
    datahub_command("(create DataPid:TwoBits 1)", 1);
    .OnChange(#$DataPid:PID1.Mv, '(@self).WriteBits(value, #$DataPid:TwoBits, 0x03, 0));
}

/* Start the program by instantiating the class. */
ApplicationSingleton (MaskedBridge);
```

ConnectionTrack.g

`ConnectionTrack.g` — changes a point when a connection is made or broken.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/*
 * This script watches the quality on an indicator point and sets
 * an output to 0 if the quality is NOT CONNECTED and 1 otherwise.
 * This effectively produces a synthetic point that changes according
 * to whether a data connection to the source of the indicator point
 * is made or broken.
 *
 * To use this script:
 * 1) Adjust the poll_rate to the desired number of seconds.  This can
 *    be fractional, such as 0.25.
 * 2) Set track_time to t or nil.  If set to t, the time stamp of the
 *    indicator point will be updated on each poll, causing a value
 *    change event to all attached clients, OPC servers, etc.
 * 3) In the constructor, make one or more calls to .BeginTracking
 *    to set up a mapping between the point being watched and the
 *    indicator point.  The watched_point
 *
 * Once this script is running, you can use the output point as a
 * trigger to send email, write to a database, update a PLC, write
 * to Excel or perform some other custom action through scripting.
 */

require ("Application");

class ConnectionTrack Application
{
    poll_rate = 1;  // polling rate in seconds
    track_time = nil; // set to t to adjust the output time stamp on each poll
}

method ConnectionTrack.BeginTracking(indicator_point, output_point)
{
    datahub_command(format("(create %a 1)", indicator_point), 1);
    datahub_command(format("(create %a 1)", output_point), 1);
    .TimerEvery(.poll_rate, '(@self).CheckQuality(@indicator_point, @output_point));
}

method ConnectionTrack.CheckQuality(!indicator_point, !output_point)
{
    local quality = PointMetadata(indicator_point).quality;
    local active = 1;
    if (quality == OPC_QUALITY_NOT_CONNECTED)
        active = 0;
    datahub_write(string(output_point), active, .track_time);
}

method ConnectionTrack.constructor ()
{
    .BeginTracking(#$default:indicator, #$default:active);
    // default:indicator = the point you want to monitor
    // default:active = the point used to trigger the notification
}

ApplicationSingleton (ConnectionTrack);
```

QualityTrack.g

`QualityTrack.g` — writes the quality of a point as the value of another point.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/*
 * This script polls the quality of an "input" point and writes the quality
 * as the value of another "output" point.
 * The input point is typically an actual point from an OPC server.  The output
 * point is a synthetic point that can be used to trigger email and ODBC events.
 *
 * To alter this script, just edit the QualityTrack.constructor to change the
 * parameters of the .Track() call.  The parameters are:
 *   poll_secs - the polling rate.  This can be fractional, as in 0.5
 *   input - the name of the input point as a symbol
 *   output - the name of the output point as a symbol
 *
 * You can add as many .Track() calls as you need to monitor multiple points.
 *
 * In an email or ODBC Condition configuration, the quality of a point can be
 * compared to the OPC quality constants:
 *   OPC_QUALITY_BAD
 *   OPC_QUALITY_COMM_FAILURE
 *   OPC_QUALITY_CONFIG_ERROR
 *   OPC_QUALITY_DEVICE_FAILURE
 *   OPC_QUALITY_EGU_EXCEEDED
 *   OPC_QUALITY_GOOD
 *   OPC_QUALITY_LAST_KNOWN
 *   OPC_QUALITY_LAST_USABLE
 *   OPC_QUALITY_LOCAL_OVERRIDE
 *   OPC_QUALITY_MASK
 *   OPC_QUALITY_NOT_CONNECTED
 *   OPC_QUALITY_OUT_OF_SERVICE
 *   OPC_QUALITY_SENSOR_CAL
 *   OPC_QUALITY_SENSOR_FAILURE
 *   OPC_QUALITY_SUB_NORMAL
 *   OPC_QUALITY_UNCERTAIN
 */

require ("Application");
require ("Time");

class QualityTrack Application
{
}

method QualityTrack.CheckQuality(input, output)
{
    local info = PointMetadata(input);
    set (output, info.quality);
}

method QualityTrack.Track(poll_secs, input, output)
{
    datahub_command(format("(create %a 1)", output));
    .TimerEvery(poll_secs, '(@self).CheckQuality(#@input, #@output));
}

method QualityTrack.constructor ()
{
    // Change the poll_secs and the two point names here.  The # in front
```

```
    //of the point name is necessary.
    .Track(1, #$default:my_real_opc_point, #$default:my_synthetic_trigger);

    // Add more calls to .Track() here
}

ApplicationSingleton (QualityTrack);
```

TagMonitor.g

`TagMonitor.g` — monitors DataHub points for changes in quality or failure to change value.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/*
 * This script monitors tags (or DataHub points) for two conditions:
 *
 * 1) change of quality
 * 2) failure to change within a time period
 *
 * For each of these, the script creates synthetic points (the "target")
 * that hold the result of monitoring the watch points.  Email and
 * database events can then be triggered from the synthetic points.
 *
 * You can modify the constructor function in this script to change the
 * point names, or to add additional watch conditions as needed.
 */

require ("Application");

class TagMonitor Application
{
}

/*
 * Set up a watch tag and a target tag such that the quality of the
 * watch tag is copied into the value of the target tag.
 */
method TagMonitor.copyQuality(poll_seconds, watch, target)
{
    // Ensure that the input and output tags exist.
    datahub_command(format("(create %s 1)", stringc(watch)), 1);
    datahub_command(format("(create %s 1)", stringc(target)), 1);

    // Periodically copy the quality of the watch point into the value of the target
    .TimerEvery(poll_seconds, ‘set(#@target, PointMetadata(#@watch).quality));
}

/*
 * Set up a watch tag, a target tag and a time such that the target tag
 * will be set to 1 if the watch tag has changed within the timer period,
 * or zero if the watch tag has not changed within the time period.  The
 * time period is specified by dead_seconds, which may be fractional.
 */
method TagMonitor.copyChangeStatus(dead_seconds, watch, target)
{
    // Ensure that the input and output tags exist.
    datahub_command(format("(create %s 1)", stringc(watch)), 1);
    datahub_command(format("(create %s 1)", stringc(target)), 1);

    // Start the watch point off as having changed
    setprop(watch, #has_changed, t);

    // Whenever the watch point changes, set its property "has_changed" to t.
    .OnChange(watch, ‘(@self).watchHasChanged(#@watch, #@target));
    .TimerEvery(dead_seconds, ‘(@self).checkChange(#@watch, #@target));
}

/*
 * A callback function that checks for a change in a point and puts a 1 or 0
```

```
 * into the target.  Reset the changed flag to zero.
 */
method TagMonitor.checkChange(watch, target)
{
    set(target, getprop(watch, #has_changed) ? 1 : 0);
    setprop(watch, #has_changed, nil);
}

/*
 * A callback whenever a change watch point changes.  We use this to change from
 * 0 to 1 as soon as we see a change in a watch point instead of waiting for the
 * poll delay.
 */
method TagMonitor.watchHasChanged(watch, target)
{
    setprop(watch, #has_changed, t);
    set(target, 1);
}

/* Write the 'main line' of the program here.
 *
 * As written, the points to watch for quality and change, as well as the
 * target points to modify, are all in the "default" domain, as follows:
 *
 * Point to watch for quality:  default:quality_watch
 * Point to watch for change:   default:change_watch
 * Target point for quality:    default:quality_target
 * Target point for change:     default:change_target
 *
 * You can change these to different domain and point names.  Also, you can
 * add any number of other points to monitor quality and change, following
 * the same syntax.
 *
 * The first argument of .copyQuality is the poll rate in seconds on the
 * quality of the point.  The first argument of .copyChangeStatus is the
 * number of "dead" seconds to wait for a change, before notifiying of
 * a failure.
 */
method TagMonitor.constructor ()
{
    .copyQuality(1, #$default:quality_watch, #$default:quality_target);
    .copyChangeStatus(5, #$default:change_watch, #$default:change_target);
}

/* Any code to be run when the program gets shut down. */
method TagMonitor.destructor ()
{
}

/* Start the program by instantiating the class.  If your
 * constructor code does not create a persistent reference to
 * the instance (self), then it will be destroyed by the
 * garbage collector soon after creation.  If you do not want
 * this to happen, assign the instance to a global variable, or
 * create a static data member in your class to which you assign
 * 'self' during the construction process.  ApplicationSingleton()
 * does this for you automatically. */
ApplicationSingleton (TagMonitor);
```

TimedUpdate.g

`TimedUpdate.g` — periodically updates the timestamp on a set of DataHub points without changing their values.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/*
 * This script runs a timer that periodically updates the timestamp on a
 * set of data points without changing their values.
 */

require ("Application");

/*
 * Modify the list of point names here to change which points will be
 * written.
 * Modify the update time (in seconds) to change the write frequency
 */

class TimedUpdate Application
{
 points = [ "default:test1", "default:test2" ];
 updateSecs = 5;
}

/* This is the callback that runs when the timer fires */
method TimedUpdate.doUpdate ()
{
 local current;
 with point in .points do
 {
  current = datahub_read(point);
  if (current[0])
  {
   // preserve the current value and quality, but let the time
   // change to the current system clock time by not specifying
   // the time argument.  The "1" for the "force" argument indicates
   // that the DataHub should emit a change even if the settings
   // would normally cause this change to be ignored.
   datahub_write(point, current[0].value, 1, current[0].quality);
  }
 }
}

/* Write the 'main line' of the program here.  This is where we start the timer. */
method TimedUpdate.constructor ()
{
 // The .TimerEvery function will start counting when the script starts
 // running, so it will not be synchronized with a particular time of day.
 .TimerEvery (.updateSecs, '(@self).doUpdate());

 // If you want to synchronize the update with the time of day, say exactly on
 // each half-hour, you would use the .TimerAt function.  The arguments
 // are .TimerAt(day, month, year, hour, minute, second, callback_function)
 // Specify nil to mean "any".  Specify a list to give multiple values.
 // For example, this will perform the update at (0 and 30 minutes) and 0
 // seconds past every hour.
 // .TimerAt (nil, nil, nil, nil, list(0, 30), 0, '(@self).doUpdate());
}
```

```
/* Any code to be run when the program gets shut down. */
method TimedUpdate.destructor ()
{
}

/* Start the program by instantiating the class. */
ApplicationSingleton (TimedUpdate);
```

FixQuality.g

`FixQuality.g` — changes point quality for OPC clients that treat bad quality as a disconnection.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/* This application monitors a set of points in the "input domain" and copies
 * them to the "output domain".  If the quality of the input point is not
 * GOOD, then the script modifies the point value to -1 and the quality to
 * GOOD.  This is to handle an OPC client that treats bad quality as a
 * disconnection.
 *
 * To configure this application, modify the class variables:
 *    domain_in = the name of the input domain
 *    domain_out = the name of the output domain
 *    value_if_bad = the value to substitute on the point if the quality is bad
 *                   If this is nil, then do no value substitution.
 */
require ("Application");

class FixQuality Application
{
    domain_in = "test";
    domain_out = "test2";
    value_if_bad = -1;
}

/* Monitor a point.  This includes creating the point if it does not exist,
 * and then creating the output domain's mirror of the point.  This function
 * also sets up an event handler to map any future changes of the point into
 * the output domain. */
method FixQuality.Monitor (ptname)
{
    local  outname, ptsym;

    outname = string(.domain_out, ":", ptname);
    ptname = string(.domain_in, ":", ptname);
    .OnChange(symbol(ptname), '(@self).BridgeQuality(#@ptname, #@outname));
    datahub_command(format("(create %s 1)", stringc(ptname)), 1);
    ptsym = symbol(ptname);
    if (!undefined_p(eval(ptsym)))
        .BridgeQuality(ptname, outname);
}

/* This is the function that does the work of mapping from the input domain
 * to the output domain. */
method FixQuality.BridgeQuality(ptname, outname)
{
    local  info = PointMetadata(symbol(ptname));
    local  value = info.value;
    if (info.quality != OPC_QUALITY_GOOD && .value_if_bad)
        value = .value_if_bad;
    datahub_write(outname, value, 1, OPC_QUALITY_GOOD, info.timestamp);
}

method FixQuality.MonitorDomain(domain)
{
    local  points = datahub_points(domain, nil);
    with point in points do
    {
        .Monitor(point.name);
    }
```

```
}

/* This is the mainline of the program.  You can either call .Monitor("pointname") for
 * each point, or you can call .MonitorDomain(.domain_in) to monitor all existing points
 * in the input domain.  If you choose to monitor the whole domain, you must re-run
 * this application whenever new points are added to the domain. */
method FixQuality.constructor ()
{
    .Monitor("point001");
    .Monitor("point002");
    .MonitorDomain(.domain_in);
}

/* Any code to be run when the program gets shut down. */
method FixQuality.destructor ()
{
}

/* Start the program by instantiating the class.  If your
 * constructor code does not create a persistent reference to
 * the instance (self), then it will be destroyed by the
 * garbage collector soon after creation.  If you do not want
 * this to happen, assign the instance to a global variable, or
 * create a static data member in your class to which you assign
 * 'self' during the construction process.  ApplicationSingleton()
 * does this for you automatically. */
ApplicationSingleton (FixQuality);
```

OPCItemLoader.g

`OPCItemLoader.g` — reads a list of OPC tags from a CSV file and configures DataHub points for them.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:

`C:\Program Files\Cogent\Cogent DataHub\scripts\`

Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/* This script reads a set of OPC item names and point names from a CSV file and updates
 * the Manually Selected Items configuration for an OPC connection.  The CSV file format can
 * be one of:
 *     OPC_ITEM_ID
 * or
 *     OPC_ITEM_ID, OPC_DATAHUB_POINT_NAME
 *
 * If the OPC_DATAHUB_POINT_NAME is absent (the line contains only an OPC item ID), then the
 * script will create an OPC DataHub point with the same name as the OPC item ID.  The OPC item
 * ID is the item ID defined by the OPC server.
 *
 * To use this script, follow these steps:
 *
 * 1) Create the OPC connection that you want to add items to.  Select "Manually Select Items"
 *    only in the "Item Selection" section.  You do not need to configure any items.  Press
 *    OK to save the OPC configuration, the press Apply on the OPC DataHub properties dialog.
 * 2) Open this script in the editor and change the parameters in the OPCItemLoader class
 *    definition.  These are documented below.
 * 3) Close the OPC DataHub Properties window.
 * 4) Run this script by pressing the run button in the toolbar of the editor (the right-facing
 *    blue arrow).
 * 5) Open the OPC DataHub properties dialog, open the OPC configuration for your server and
 *    verify that the items were added.
 *
 * If the script has problems, you should see error messages in the "Script Log" window.
 *
 * You must close the OPC DataHub Properties window before running this script or the
 * changes made by this script may be lost.
 *
 * You do not need to run this script each time the OPC DataHub is started.  The configuration
 * produced by this script will be saved in the OPC DataHub's configuration file permanently.
 *
 * Editable fields:
 *   connection_name = the name of the OPC connection to be adjusted.  This is the name
 *  entered into the box marked "Connection Name:" in the "Define OPC Server" dialog
 *  of the OPC DataHub properties dialog.
 *   file_name = the file name containing the OPC item names and point names.  The file
 *  can either either one or two strings per line separated by commas.  If only a single
 *  string appears, it is taken to be the OPC server item name.  The point name is
 *  computed from the item name.  If two strings appear then the first string is the
 *  OPC item and the second string is the OPC DataHub point name.  The point name will
 *  automatically be broken into components on a "." and a tree hierarchy will be
 *  created as if each component but the last is a tree branch.
 *   trim_spaces = t if you want the item and point names to be trimmed of all leading and
 *  trailing white space and tabs, otherwise nil.
 *   path_separator = nil if you do not want to split point names to produce a tree
 *  hierarchy, otherwise a string containing separator characters.  Normally this will
 *  be a "." character.
 */

require ("Application");
require ("WindowsSupport");
require ("OPCSupport");
```

```
class OPCItemLoader Application
{
    connection_name = "OPC000"; // Change this to the connection name of the connection to edit
    file_name = "my_items.csv"; // Change this to the file containing the item and point names
    trim_spaces = t;            // set to nil to preserve leading and trailing spaces in item names
    path_separator = ".";       // Characters used to split point names into tree components
    file_is_8bit_ansi = t;      // Set to t if file uses 8-bit extended ANSI, nil if 7-bit ASCII or UTF-
8
    opc_connection;
}

method OPCItemLoader.Trim (str)
{
    local i=0, j, len;
    len = strlen(str);
    while (i < len && (str[i] == ' ' || str[i] == '\t'))
        i++;

    j = len - 1;
    while (j >= i && (str[j] == ' ' || str[j] == '\t'))
        j--;

    if (j>=i)
        str = substr(str, i, j-i+1);
    else
        str = "";
    str;
}

method OPCItemLoader.ReadCSVFile (filename)
{
    local  fptr = open (filename, "r", nil);
    local  line, i;
    if (fptr)
    {
        while ((line = read_line(fptr)) != _eof_)
        {
            if (.file_is_8bit_ansi)
                line = strcvt(line);

            if (line != "")
            {
                line = list_to_array(string_split(line, ",", 0, nil, nil, nil, "\\", nil));
                if (.trim_spaces)
                {
                    for (i=0; i<length(line); i++)
                    {
                        line[i] = .Trim(line[i]);
                    }
                }
                if (length(line) == 1)
                    .AddOPCItem(line[0], line[0]);
                else if (length(line) > 1)
                    .AddOPCItem(line[0], line[1]);
            }
        }
        close (fptr);
    }
    else
    {
        local s = strerror(errno());
        princ (class_name(self), ": Could not open file: ", filename, ": ", s, "\n");
    }
}

method OPCItemLoader.AddOPCItem(itemname, pointname)
{
    .opc_connection.addItem(pointname, itemname, OPC_NODE_LEAF, .path_separator);
```

```
}

method OPCItemLoader.LoadFromCSV(opc_conn_name, filename)
{
    local opc = new OPCConnection();
    opc.setServer(opc_conn_name);
    .opc_connection = opc;
    .ReadCSVFile(filename);
}

/* Write the 'main line' of the program here. */
method OPCItemLoader.constructor ()
{
    .LoadFromCSV(.connection_name, .file_name);
}

/* Any code to be run when the program gets shut down. */
method OPCItemLoader.destructor ()
{
}

ApplicationSingleton (OPCItemLoader);
```

OPCReconnect.g

`OPCReconnect.g` — disconnects and reconnects an OPC server.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/* Sample script that disables the OPC connection labeled "OPC000", waits 2
 * seconds, then re-enables the connection.  The result will be that the
 * OPC DataHub disconnects from the server for 2 seconds and then reconnects.
 */

require ("Application");
require ("OPCSupport");

class OPCReconnect Application
{
}

/* Create an object that references an existing OPC connection that we have
 * configured through the user interface.  Its label is the first argument to
 * setServer below.  Once we have the OPC connection object, we can call
 * enable() with t or nil in the argument to enable or disable this connection.
 */
method OPCReconnect.enable(enabled)
{
    local opcclient = new OPCConnection();
    opcclient.setVerbose(t);
    opcclient.setServer("OPC000");
    opcclient.enable(enabled);
}

method OPCReconnect.constructor ()
{
    .enable(nil);
    .TimerAfter(2, '(@self).enable(t));
}

ApplicationSingleton (OPCReconnect);
```

OPCReload.g

`OPCReload.g` — requests a reload of OPC server data with no disconnect.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/* This script requests a reload of data from a connected OPC server
 * without disconnecting from the server.  It is particularly useful
 * for automatically adding new points, and can poll for new points
 * based on a timer or on a trigger point.
 *
 * You will need to edit the following three variables to run the
 * script on your system:
 *
 *  connection_label: The name given to your OPC connection when you
 *                    defined your OPC server.  This appears in the
 *                   "Connection" column in your list of OPC Client
 *                    connections in the Properties window.
 *
 *  poll_time:        The poll rate to check for new points, in seconds.
 *
 *  trigger_point:    The name of the point you will use as a trigger.
 *
 */

require ("Application");
require ("OPCSupport");

class OPCReload Application
{
    connection_label = "OPC002";              // connection label in "Connection" column of OPC properties
    poll_time = 30;                           // poll rate for new points, in seconds
    trigger_point = #$domain_name:point_name;  // name of point used as a trigger
}

/* Transmit a "reload" command to the server, without disconnecting.
 * To disconnect and reload, change the paramter to opc.reload to t
 * instead of nil.
 */
method OPCReload.reload ()
{
    local   opc = new OPCConnection();
    opc.setVerbose(t);
    opc.setServer(.connection_label);
    opc.reload(nil);
}

/* Start the polling timer */
method OPCReload.constructor ()
{
    // To poll for new points periodically, use this line
    .TimerEvery (.poll_time, '(@self).reload());

    // To poll for new points only when a specific data point changes, use this line
    .OnChange (.trigger_point, '(@self).reload());
}

/* Instantiate the class.  This calls the constructor. */
ApplicationSingleton (OPCReload);
```

AutoCalculation.g

`AutoCalculation.g` — automatically calculates formulas based on data points.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/*
 * Automatically calculate formulas based on data points.  Create data points where
 * necessary to store the results back to the DataHub's data set.
 *
 * To use:
 *  - create a file, like c:/tmp/calculations.txt
 *  - in the file, create any number of calculations of the form
 *      $default:test = $DataPid:PID1.Mv * 10;
 *      $default:test2 = $DataPid:PID1.Mv / 10;
 *  - change the fileName member in the AutoCalculation class below to refer to your file
 *  - run the script
 *
 * Any symbol in the calculation of the form $domain:name is assumed to be a DataHub point,
 * and will be created if necessary.  The point on the left hand side of the equal sign will
 * be automatically updated whenever any point on the right hand side of the equal sign
 * changes.
 *
 * You can include complex Gamma expressions, like:
 *   $default:output = progn {
 *      local a = $DataPid:PID1.Mv;
 *      local b = $DataPid:PID1.Pv;
 *      if (b != 0)
 *              a / b;
 *      else
 *              a;
 *      };
 *
 * You can define functions within this file, though it is better to put functions into a
 * separate script file:
 *
 *      function average(a,b)
 *      {
 *          a + b / 2;
 *      }
 *
 *      $default:output = average($DataPid:PID1.Mv, $DataPid:PID1.Pv);
 */

require ("Application");

class AutoCalculation Application
{
    fileName = "C:/tmp/calculations.txt";
    fp;
}

class Computation
{
    app;        // the Application instance defining this computation
    output;     // a symbol
    inputs;     // a list of symbols
    code;       // the code to execute
}

method Computation.constructor(app, expression)
{
```

```
    .app = app;
    .code = expression;
    .output = .findOutput(expression);
    .inputs = .findInputs(expression);

    if (.output)
    {
        datahub_command(format("(create %s 1)", stringc(.output)), 1);
        with input in .inputs do
        {
            datahub_command(format("(create %s 1)", stringc(input)), 1);

            .app.OnChange(input, '(@self).compute());
        }
    }
    .compute();
}

method Computation.compute()
{
    try
    {
        eval(.code);
    }
    catch
    {
        princ ("Assignment to ", .output, " failed: ", _last_error_, "\n");
    }
}

method Computation.findOutput (expr)
{
    if (car(expr) == #setq)
        cadr(expr);
    else
        nil;
}

method Computation.findInputsRecursive (expr)
{
    local   inputs, partial;
    if (list_p(expr))
    {
        with part in expr do
        {
            partial = .findInputsRecursive(part);
            inputs = nappend(inputs, partial);
        }
    }
    else if (symbol_p(expr) && strchr(string(expr), ":") != -1)
    {
        inputs = cons(expr, inputs);
    }
    inputs;
}

method Computation.findInputs (expr)
{
    .findInputsRecursive(cddr(expr));
}

method AutoCalculation.readFile()
{
    if ((.fp = open(.fileName, "r", t)) != nil)
    {
        //create some local variables
        local line;
```

```
        //loop until we have read the entire file.
        while((line = read(.fp)) != _eof_)
        {
            local comp = new Computation(self, line);
        }

        //once we have read the entire file we need to close it.
        close(.fp);
    }
    else
    {
        princ ("AutoCalculation: Could not open file: ",
        .fileName, ": ", strerror(errno()), "\n");
    }
}

/* Write the 'main line' of the program here. */
method AutoCalculation.constructor ()
{
    .readFile();
}

/* Any code to be run when the program gets shut down. */
method AutoCalculation.destructor ()
{
}

/* Start the program by instantiating the class. */
ApplicationSingleton (AutoCalculation);
```

II. Built-in Classes

Table of Contents

DH_Domain

DH_Domain — the structure of a DataHub domain.

Syntax

```
class DH_Domain
{
    name;         // the name of the domain, as a string
    auto_created; // 1 if the domain was created implicitly due to a point reference,
                  // or 0 if the domain was configured in the General tab
    n_points;     // the number of points in this domain, including points with values
                  // and branches in the hierarchy tree
    n_assemblies; // Not implemented
    n_attributes; // Not implemented
    n_bridges;    // The number of domain bridges or redundancy pairs
                  // in which this domain is participating
}
```

DH_Item

DH_Item — the structure of a DataHub point.

Syntax

```
class DH_Item
{
    canonical_type;     // number: The canonical VARIANT type
    conf;               // number: confidence (0-100)
    domain;             // string: The domain name
    flags;              // number: flags describing the point
    n_aliases;          // number of aliases for this point
    n_attributes;       // number of attributes this point has
    n_pending;          // number of clients with a change pending
    n_properties;       // number of properties this point has
    n_registered;       // number of clients registered for changes
    n_subassemblies;    // number of subassemblies this point has
    name;               // string: name of the point, without the domain
    opcaccessrights;    // number: read/write flags
    propid;             // number: property id if applicable
    quality;            // number: OPC item quality
    security;           // number: security level (not used)
    timestamp;          // number: time stamp in Windows epoch time
    value;              // variant: the point value
}
```

Instance Variables

flags

This instance variable may be a combination of zero or more of the following flags.

Hex	Constant	Description
0x00001	READABLE	Item is readable.
0x00002	WRITABLE	Item is writable.
0x00004	LOCKED	Item is locked (not used).
0x00008	PROPERTY	Item is a property of another point.
0x00010	SUBASSEMBLY	Item is a subassembly of another point.
0x00020	ASSEMBLY	Item is an assembly.
0x00040	ATTRIBUTE	Item is an attribute of another point.
0x00080	TYPE	Item is an attribute type.
0x00100	ACTIVE	Item is active.
0x00200	PRIVATE_ATTRIBUTE	Item is a private attribute of a type.
0x00800	HIDDEN	Item is hidden.
0x01000	AUTO_ID	Item was assigned a `propid` automatically.
0x40000	TEMP_VALUE	Item's value is temporary, probably `Not Connected` assigned by an interface becoming disconnected.

III. Special Gamma Functions for DataHub Scripting

Table of Contents

_

_ — looks up a translation text.

Syntax

`_(string)`

Arguments

`string`

A string that should be translated.

Returns

The translation of the string.

Description

This function looks up a pre-configured translation text. It is used internally by the DataHub interface programmer to specify which text strings in the interface should be translated when the user selects a language in the Properties window of the DataHub. Language selection itself cannot be done within Gamma; it can only be done through the Properties window.

If your language is not available in the DataHub, you can contact Cogent for instructions on how to add a translation to the DataHub source code.

The unusual name for this function is based on a convention of GNU's `gettext` function, in which the underscore symbol (_) can be used in place of the `gettext` function name.

add_menu_action

`add_menu_action` — adds a menu action.

Syntax

`add_menu_action (`*`menu_id`*`, `*`s_exp`*`)`

Arguments

`menu_id`

A unique ID number for this item.

`s_exp`

A Gamma expression that is the action to be taken when a menu item is selected.

Returns

A list:

(*`menu_id s_exp`*)

Where the *`s_exp`* is in Lisp format.

Description

This function lets you specify what code gets run when a menu item is selected. The code can be removed using `remove_menu_action`

Example

This example is part of the `WindowsExample.g` example program:

```
method WindowsExample.AddSubMenu (parent, pos, label)
{
 local submenu = CreatePopupMenu();
 InsertMenu (traymenu, pos, MF_BYPOSITION | MF_POPUP,
    submenu, label);
 .menu_items = cons (cons (submenu, t), .menu_items);
 submenu;
}

method WindowsExample.AddMenuItem (parent, pos, label, code)
{
 local info = new MENUITEMINFO();

 info.cbSize = 48;
 info.fMask = MIIM_STRING | MIIM_FTYPE | MIIM_ID;
 info.fType = MFT_STRING;
 info.wID = ++MenuItemID;
 info.dwTypeData = label;
 InsertMenuItem (parent, pos, 1, info);
 local action = add_menu_action (MenuItemID, code);
 .menu_actions = cons (action, .menu_actions);
}

method WindowsExample.AddMenus ()
{
 local  traymenu = get_tray_menu ();

 if (traymenu != 0)
 {
```

```
  local submenu = .AddSubMenu (traymenu, 5, "Monitor Functions");

  .AddMenuItem (submenu, -1, "Select Color",
       '(@self).SelectTrayMenuItem (@MenuItemID+1));
  .AddMenuItem (submenu, -1, "Select File",
       'new GFileDialog (1).DoModal(0));
  .AddMenuItem (submenu, -1, "Select Folder",
       'new GFolderDialog ().DoModal(0));
 }
 else
 {
 }
}
```

allow_self_reference

`allow_self_reference` — permits changes to be written back to the point of origin.

Syntax

```
allow_self_reference (symbol, allow)
```

Arguments

`symbol`

Any valid Gamma symbol.

`allow`

`1` or any other non-zero value allows self-referential behavior; `0` disallows it.

Returns

The value of the *`allow`* parameter.

Description

This function tells Gamma not to generate a warning if a change function like `on_change` causes a sequence of events that changes the original point again.

Example

This example is from LinearXform.g.

```
method LinearXform.AddLinearXform (app, src, dst, mult, add, bidirectional_p?=nil)
{
    app.OnChange (src, '(@self).cbLinearXform (#@dst, value, @mult, @add));
    if (bidirectional_p && mult != 0)
    {
        allow_self_reference (src, 1);
        allow_self_reference (dst, 1);
        app.OnChange (dst, '(@self).cbLinearXform (#@src, value, 1/@mult, (@-add)/(@mult)));
    }
}
```

datahub_command

`datahub_command` — sends commands to the DataHub.

Syntax

```
datahub_command (command, sync?)
```

Arguments

`command`

A string containing a DataHub configuration command.

`sync`

`1` sends the command synchronously; `0` sends the command asynchronously.

Returns

The result of the command, as a string.

Description

This function sends configuration commands to the DataHub, in Lisp format. It returns the result of the command as a string. For more information on DataHub commands, please refer to the Using DataHub Commands chapter of the Cogent DataHub manual.

Setting the *`sync`* parameter to `1` (synchronous) may slow the execution a tiny bit, but it ensures that the DataHub will complete the command before your script continues. Setting the *`sync`* parameter to `0` will allow your script to continue whether or not the DataHub command has executed. For most cases, we recommend setting it to `1`, just to be on the safe side. This ensures that your script will execute commands in the order expected. The tiny delay for synchronous execution is in micro-milliseconds—insignificant for most applications.

Escaping Characters

Since commands are sent within strings, you probably need to escape certain characters, using the backslash (\). For example, to escape the quote marks delineating the string `"my_string"`, you would enter the string as: `\"my_string\"`.

Windows File Paths

Normally if a command is in a configuration file, you need to escape the backslash in file names, like this:

```
c:\\tmp\\datahub.log
```

But, if you are issuing the command from gamma using **datahub_command** you need to escape each of those two backslashes, like this:

```
c:\\\\tmp\\\\datahub.log
```

So the command would be:

```
datahub_command ("(log_file \"c:\\\\tmp\\\\datahub.log\")")
```

Since the quotes are optional (though recommended) in DataHub commands you could do this:

```
datahub_command ("(log_file c:\\\\tmp\\\\datahub.log)")
```

Fortunately, recent Microsoft operating systems will accept forward slashes in file names, so for those versions you can do this:

```
datahub_command ("(log_file c:/tmp/datahub.log)")
```

Example

This example creates a new point in the DataHub, and sets its value to an empty string.

```
if (undefined_p($default:MyNewPoint))
{
    datahub_command ("(create default:MyNewPoint)", 1);
    datahub_command ("(set default:MyNewPoint\"\")", 1);
}
```

datahub_domaininfo

`datahub_domaininfo` — gives information about data domains.

Syntax

```
datahub_domaininfo(pattern?)
```

Arguments

pattern

A character string which specifies a search pattern

Returns

An array of instances of the `DH_Domain` class.

Description

This function provides information about all data domains whose names match the *pattern*, or all domains if the *pattern* is left blank. The *pattern* can contain the following special characters:

- ***** matches any number of characters, including zero.
- **[*c*]** matches a single character which is a member of the set contained within the square brackets.
- **[^*c*]** matches any single character which is not a member of the set contained within the square brackets.
- **?** matches a single character.
- **{*xx*,*yy*}** matches either of the simple strings contained within the braces.
- ***c*** (a backslash followed by a character) - matches that character.

Example

```
--> datahub_domaininfo();
[{DH_Domain (auto_created . 1) (n_assemblies . 0) (n_attributes . 0)
     (n_bridges . 0) (n_points . 8) (name . "DataSim")}
 {DH_Domain (auto_created . 0) (n_assemblies . 0) (n_attributes . 0)
     (n_bridges . 0) (n_points . 6) (name . "MySource")}
 {DH_Domain (auto_created . 0) (n_assemblies . 0) (n_attributes . 0)
     (n_bridges . 0) (n_points . 3) (name . "default")}]

--> datahub_domaininfo("*Sim");
[{DH_Domain (auto_created . 1) (n_assemblies . 0) (n_attributes . 0)
     (n_bridges . 0) (n_points . 8) (name . "DataSim")}
```

datahub_domains

`datahub_domains` — creates a list of all domains.

Syntax

```
datahub_domains()
```

Arguments

none

Returns

A list of domain names.

Description

This function creates a list of all domains currently in the DataHub, except the default domain.

Example

```
--> datahub_domains();
(DataSim PlantA PlantB)
```

datahub_log

`datahub_log` — writes a string to the Event Log window.

Syntax

```
datahub_log (trace_level, message)
```

Arguments

`trace_level`

One of: `DHTL_DEBUG`, `DHTL_INFO`, `DHTL_WARNING`, `DHTL_ERROR`.

`message`

Any string.

Returns

`t` on success, otherwise an error message.

Description

This function will write a string to the DataHub Event Log window. The *`trace_level`* parameter corresponds to the level of Severity in the Event Log, and the relevant filtering will apply. The Facility is always General. Messages written with this function will be written to disk if the Event Log is being saved.

datahub_points

`datahub_points` — shows the points in a data domain.

Syntax

```
datahub_points (parent, top_level_only? = nil, pattern?)
```

Arguments

parent

A string containing the name of the parent domain or point name whose child points you wish to list.

top_level_only

If `t`, returns only the direct children of the parent. Otherwise, it returns all descendents of the parent.

pattern

Returns only those points whose full name (the entire point path without the `domain:` prefix) matches the pattern.

Returns

An array of instances of the `DH_Item` class.

Description

This function provides an array containing the DataHub points in a given domain. The *pattern* can contain the following special characters:

- ***** matches any number of characters, including zero.
- **[*c*]** matches a single character which is a member of the set contained within the square brackets.
- **[^*c*]** matches any single character which is not a member of the set contained within the square brackets.
- **?** matches a single character.
- **{*xx*,*yy*}** matches either of the simple strings contained within the braces.
- ***c*** (a backslash followed by a character) - matches that character.

Example

```
--> datahub_points("MySource", t);
[{DH_Item (canonical_type . 0) (conf . 0) (domain . "MySource") (flags . 307)
   (n_aliases . 0) (n_attributes . 0) (n_pending . 0)
   (n_properties . 0) (n_registered . 1) (n_subassemblies . 1)
   (name . "A_Branch") (opcaccessrights . 3) (propid . 0)
   (quality . 0) (security . 0) (timestamp . 0) (value)}]

--> datahub_points("MySource", nil, "*em2");
[{DH_Item (canonical_type . 5) (conf . 0) (domain . "MySource") (flags . 579)
   (n_aliases . 0) (n_attributes . 0) (n_pending . 0)
   (n_properties . 0) (n_registered . 1) (n_subassemblies . 0)
   (name . "A_Branch.Ramp.X_Branch.Item2") (opcaccessrights . 3)
   (propid . 0) (quality . 192) (security . 0)
   (timestamp . 40200.75880280092) (value . 0)}]
```

datahub_read

`datahub_read` — creates a list of all points for a domain.

Syntax

```
datahub_read (pointnames ...)
```

Arguments

`pointnames`

A DataHub point name as a string, or a list of such DataHub point names as strings. If the first argument is a list, then it is used for pointnames, and all following arguments are ignored. If not, multiple point names can be passed as individual arguments.

Returns

An array in the same order as the specified point names.

Description

This function reads the point identified by each string in *`pointnames`*, and returns a `DH_Item` structure for each point, or `nil` if the point does not exist.

datahub_write

`datahub_write` — assigns a value to a DataHub point.

Syntax

`datahub_write (`*`pointname`*`, `*`value`*` [, `*`force`*`, `*`quality`*`, `*`timestamp`*`])`

Arguments

`pointname`

The name of the point, as a string.

`value`

The value to be assigned to the point—a number, string, or array.

`force`

Causes the write to occur even if the value and quality are the same as the value currently in the DataHub. This is useful if you are changing only the time stamp of the point. Enter `t` to force, or `nil` to not force.

`quality`

The quality of a point, as a number. You can use the constants `OPC_QUALITY_*` found in `GetQualityName` by requiring that file like this: `require("Quality.g");`.

`timestamp`

The time stamp in Windows time. Windows time is a floating point number that can be obtained in Gamma by requiring `Time.g` like this: `require("Time.g");`, then calling the functions `GetCurrentWindowsTime`, `UnixTimeToWindowsTime`, or `PointGetWindowsTime`.

Returns

`t` on success, otherwise an error message.

Description

This function is the same as the Gamma `force` function, but the *`pointname`* is a string, so the symbol never needs to be created in the Gamma engine. This can be used to reduce Gamma engine load when using a large number of points.

edit_file

`edit_file` — opens a file in the Script Editor.

Syntax

```
edit_file (filename)
```

Arguments

`filename`

The name of the file you wish to edit, as a string.

Returns

`t` on success, otherwise an error message.

Description

This function opens the Script Editor and loads a requested file. If the requested file cannot be found, it creates a new file with the requested name.

get_point_queue_count

`get_point_queue_count` — counts the number of DataHub points queued for Gamma.

Syntax

```
get_point_queue_count ()
```

Arguments

none

Returns

On success, the number of points currently queued, as an integer. Otherwise an error message.

Description

This function queries the DataHub for the total number of point changes currently queued to be passed to Gamma. Please refer to `_point_queue_depth` for a description of the queue. See also `get_point_queue_depth`.

get_point_queue_depth

`get_point_queue_depth` — gets the depth of the DataHub's per-point queue for Gamma.

Syntax

```
get_point_queue_depth ()
```

Arguments

none

Returns

On success, the depth of the queue, as an integer. Otherwise an error message.

Description

This function queries the depth of the queue for DataHub changes that get passed to Gamma. Please refer to `set_point_queue_depth` for a description of the queue. See also `get_point_queue_count`.

get_tray_menu

`get_tray_menu` — returns a pointer to the tray menu.

Syntax

```
get_tray_menu ()
```

Arguments

none

Returns

The ID number for the tray menu.

Description

This function returns a pointer to the tray menu.

on_change

on_change — evaluates an expression when a variable changes value or quality.

Syntax

```
on_change (symbol, s_exp)
```

Arguments

symbol

Any Gamma symbol.

s_exp

Any Gamma expression, usually a function or method call.

Returns

A list of the following items:

```
({class_name } fn_name fn_args...)
```

Where *fn_name* and *fn_args* correspond to the *s_exp* parameter.

Description

This function causes an expression (*s_exp*) to be evaluated when a variable (*symbol*) changes value or quality. It can be undone with a call to remove_change.

Example

```
method AccessData.constructor ()
{
    on_change(#$DataSim:Sine,
              '(@self).print_point($DataSim:Sine));
    after(3, 'destroy(@self));
}
```

remove_change

remove_change — removes an on_change function.

Syntax

```
remove_change (symbol, s_exp)
```

Arguments

symbol

Any Gamma symbol.

s_exp

Any Gamma expression, usually a function or method call.

Returns

A list of the following items:

```
(class fn_name fn_args ...)
```

Where *fn_name* and *fn_args* correspond to the *s_exp* parameter.

Description

This function reverses a call to on_change, ending the evaluation of the *s_exp* whenever the *symbol* changes.

Example

```
method AccessData.destructor ()
{
    remove_change(#$DataSim:Sine,
                  '(@self).print_point($DataSim:Sine));
}
```

remove_menu_action

`remove_menu_action` — removes a menu action.

Syntax

```
remove_menu_action (menu_id, s_exp)
```

Arguments

`menu_id`

A unique ID number for this item.

`s_exp`

A Gamma expression that is the action to be removed.

Returns

A list:

```
(menu_id s_exp)
```

Where the *`s_exp`* is in Lisp format.

Description

This function removes a menu action, usually one that was added by `add_menu_action`.

Example

This example is part of the `WindowsExample.g` example program:

```
method MonitorWindow.destructor ()
{
...
    try
    {
        with x in .menu_actions do
            remove_menu_action (car(x), cdr(x));
    }
    catch
    {
        princ ("Error: ", _last_error_, "\n");
        print_stack();
    }
...
```

set_point_flush_flags

`set_point_flush_flags` — determines which data types are not buffered.

Syntax

```
set_point_flush_flags (flags)
```

Arguments

`flags`

Any combination of:

`0x01`	Turn on un-buffered checking. Always include this with any other of these flags
`0x02`	Flush on boolean data. If a boolean value changes, transmit all queued values.
`0x04`	Flush on integer data. If a boolean value changes, transmit all queued values.
`0x08`	Flush on floating point data. If a floating point value changes, transmit all queued values.
`0x10`	Flush on string data. If a string value changes, transmit all queued values.

Returns

`t` on success, otherwise an error message.

Description

This function sets which data types will cause the point buffer to immediately be transmitted to the Gamma engine. If *`flags`* is `0`, then the transmission is always buffered. For example, to flush all queued data whenever a boolean or string value changes, make this call:

```
set_point_flush_flags(0x01 | 0x02 | 0x10);
```

To return the Gamma engine to fully buffered transmission, use:

```
set_point_flush_flags(0);
```

set_point_queue_depth

`set_point_queue_depth` — sets the depth of the DataHub's per-point queue for Gamma.

Syntax

```
set_point_queue_depth (N)
```

Arguments

N

An integer defining the depth of the queue.

Returns

`t` on success, otherwise an error message.

Description

The DataHub has a special queueing mechanism for point changes that get sent to Gamma. If the DataHub receives incoming point changes faster than Gamma can process them, it puts the new values into a queue for that point. When Gamma is ready to process the next value of that point, it receives the earliest value from that queue, then the next value, and so on, in time-sequential order.

This function allows you to set the depth of this queue, in other words, the number of values per point that the DataHub will store in the queue. The default depth is `3`. We recommend setting the queue depth as small as possible, since a deep queue will decrease Gamma's response time. If you want to minimize the response time and always use the very latest value, you can set *N* to `0` or `1`, which effectively eliminates the queueing behavior altogether.

See also `get_point_queue_depth` and `get_point_queue_count`.

show_log

`show_log` — displays the Script Log.

Syntax

```
show_log ()
```

Arguments

none

Returns

`t` on success, otherwise an error message.

Description

This function displays the Script Log if it is not already open.

symcmp

`symcmp` — compares symbols to see if they are equal.

Syntax

`symcmp (`*`symbol`*`, `*`symbol`*`)`

Arguments

`symbol`

Any valid Gamma symbol.

Returns

A negative number if the first *`symbol`* is ordinally less than the second *`symbol`* according to the ASCII character set, a positive number if the first *`symbol`* is greater than the second, and zero if the two symbol strings are exactly equal.

Description

This function compares symbols to see if they are equal. Neither of the symbols are evaluated when passed to the function. This function can be used as a comparison function for `sort` and `insert`.

Example

This example was done in the Script Log.

```
--> symcmp(a,b);
-1
--> symcmp(a,a);
0
--> symcmp(b,a);
1
-->  y = array(#c, #d, #a);
[c d a]
--> y = sort(y, symcmp);
[a c d]
--> insert(y, symcmp, #b);
b
--> y;
[a b c d]
```

IV. Methods and Functions from `Application.g`

Table of Contents

AddCustomMenuItem

`AddCustomMenuItem` — a convenience method for creating a menu item.

Syntax

```
AddCustomMenuItem (label, code)
```

Arguments

`label`

The name of the item, as a text string, that will actually appear on the menu.

`code`

Any piece of code that should be run when the menu item is selected.

Returns

A list containing two members: the ID number for this action, followed by a Gamma expression that is the action to be taken when a menu item is selected.

Description

This method is a convenience method for adding a menu item. It is a wrapper on `AddMenuItem`. It creates an item for a menu created by `AddCustomSubMenu`, allowing any arbitrary code to be attached. This method can be used for modifying the application as it runs. Items to be created by this command must be coded immediately after `AddCustomSubMenu`, and will appear in sequence on that submenu.

Example

```
.AddCustomSubMenu("My Menu", 3);
.AddCustomMenuItem("My First Item, ‘princ("First Item Activated\n"));
.AddCustomMenuItem("My Second Item, ‘princ("Second Item Activated\n"));
.AddCustomMenuItem("My Third Item, ‘princ("Third Item Activated\n"));
...
```

AddCustomSubMenu

`AddCustomSubMenu` — a convenience method for adding a menu.

Syntax

```
AddCustomSubMenu (label, pos?=._TrayMenuPosition)
```

Arguments

`label`

The name of the menu, as a text string, that will actually appear on the menu.

`pos`

An integer desinating the position of this item on the menu. This is an optional argument. If not specified, the value of the `_TrayMenu` class variable will be used.

Returns

The value of the `._CustomSubmenu` class variable, a positive integer.

Description

This method is a convenience method for adding a menu. It is a wrapper on `AddSubMenu` that also automatically assigns the parent menu and arranges its items sequentially as they coded, using the `AddCustomMenuItem` method.

AddMenuItem

`AddMenuItem` — adds a menu item and attaches code to it.

Syntax

```
AddMenuItem (parent, pos, label, code)
```

Arguments

parent

The parent menu for this item, such as that returned by a call to `CreateSystemMenu` or `AddSubMenu`.

pos

An integer desinating the position of this item on the menu.

label

The name of the item, as a text string, that will actually appear on the menu.

code

Any piece of code that should be run when the menu item is selected.

Returns

A list containing two members: the ID number for this action, followed by a Gamma expression that is the action to be taken when a menu item is selected.

Description

This method is used to populate a submenu of the DataHub's system tray with a selectable item that runs user-specified code when selected.

AddPermanentMenuItem

`AddPermanentMenuItem` — should not be used.

Syntax

`AddPermanentMenuItem (`*`parent`*`, `*`pos`*`, `*`label`*`, `*`code`*`)`

Description

This method should not be used.

AddStartMenuItem

`AddStartMenuItem` — should not be used.

Syntax

```
AddStartMenuItem (label)
```

Description

This method is for internal use only.

AddStopMenuItem

`AddStopMenuItem` — creates a menu item that destroys the running application.

Syntax

`AddStopMenuItem (`*`label`*`)`

Arguments

`label`

The name of the item, as a text string, that will actually appear on the menu.

Returns

A list containing two members: the ID number for this action, followed by a Gamma expression that is the action to be taken when a menu item is selected.

Description

This method lets you create an item on a DataHub system tray menu that will exit the application and remove all event handlers and menu items.

AddSubMenu

`AddSubMenu` — creates a submenu on a parent menu.

Syntax

```
AddSubMenu (parent, pos, label)
```

Arguments

`parent`

The parent menu for this submenu, such as that returned by a call to `CreateSystemMenu` or `AddSubMenu`.

`pos`

An integer designating the position of this item on the menu.

`label`

The name of the submenu, as a text string, that will actually appear.

Returns

The value of the `._CustomSubmenu` class variable, a positive integer.

Description

This method adds a submenu to the DataHub's system tray menu. The submenu can then be used for adding menu items or other submenus.

ApplicationMultiple

`ApplicationMultiple` — allows creation of multiple instances of the class.

Syntax

```
ApplicationMultiple (klass)
```

Arguments

`klass`

Returns

Description

This function does the opposite of `ApplicationSingleton`. It allows the creation of multiple instances of the `Application` class, giving each new instance the same name but not the exact same definition.

ApplicationSingleton

`ApplicationSingleton` — allows creation of only one instance of the class.

Syntax

```
ApplicationSingleton (klass)
```

Arguments

`klass`

The name of a class.

Returns

The instance of the class named by *`klass`*.

Description

This function creates a singleton based on the name of a class, which in most cases is desirable. This is useful because a class is routinely redefined by reloading its corresponding file, The alternative, `ApplicationMultiple` gives instances of the class the same name but not the same definition.

CreateSystemMenu

`CreateSystemMenu` — adds a submenu to the system tray menu.

Syntax

```
CreateSystemMenu ()
```

Arguments

none

Returns

The value of the `._CustomSubmenu` class variable, a positive integer.

Description

This method adds a Scripts submenu to the DataHub's system tray menu, which can then be used for adding submenus or menu items for DataHub scripts.

droptimer

`droptimer` — for internal use only.

Syntax

```
droptimer (tid)
```

Description

This method is for internal use only.

OnChange

`OnChange` — attaches an event handler to a point change event.

Syntax

```
OnChange (sym, fn)
```

Arguments

sym

Any Gamma symbol.

fn

A function or method call.

Returns

A list containing the *sym* and *fn*.

Description

This method is a wrapper for the `on_change` function. In addition to applying that function, it adds this change to the class's `_ChangeFunctions` list. For more information, please refer to Section 5.3, *Handling Events*.

`RemoveAllChanges`

`RemoveAllChanges` — removes event handlers from all point change events.

Syntax

```
RemoveAllChanges ()
```

Arguments

none

Returns

`nil` on success, otherwise an error.

Description

This method removes all changes from the class's `_ChangeFunctions` list, and reassigns the `_ChangeFunctions` variable to `nil`. For more information, please refer to Section 5.3, *Handling Events*.

RemoveAllEventHandlers

`RemoveAllEventHandlers` — removes all point change events and all timers.

Syntax

```
RemoveAllEventHandlers ()
```

Arguments

none

Returns

`nil` on success, otherwise an error.

Description

This method removes all change events and all timers by calling `RemoveAllChanges` and `RemoveAllTimers`.

RemoveAllMenus

`RemoveAllMenus` — removes all script menus, submenus, and menu actions.

Syntax

```
RemoveAllMenus ()
```

Arguments

none

Returns

`nil` on success, or an error.

Description

This method removes all script menus, submenus, menu items, and actions.

`RemoveAllTimers`

`RemoveAllTimers` — cancels all timers.

Syntax

```
RemoveAllTimers ()
```

Arguments

none

Returns

`nil` on success, or an error.

Description

This method cancels all timers and sets the `._TimerIDs` list to `nil`.

RemoveChange

`RemoveChange` — removes an event handler from a point change event.

Syntax

```
RemoveChange (chfn)
```

Arguments

chfn

A change function, as created by `OnChange`.

Returns

A list of the following items:

```
(class fn_name fn_args ...)
```

Where *fn_name* and *fn_args* are the name and arguments of the removed function.

Description

This method is a wrapper for the `remove_change` function. In addition to applying that function, it removes this change from the class's `_ChangeFunctions` list. For more information, please refer to Section 5.3, *Handling Events*.

RemoveSystemMenu

`RemoveSystemMenu` — for internal use only.

Syntax

```
RemoveSystemMenu ()
```

Description

This method is for internal use only.

RemoveTimer

`RemoveTimer` — cancels a timer.

Syntax

`RemoveTimer(`*`tid`*`)`

Arguments

`tid`

A timer ID number, as assigned by `TimerAt`, `TimerAfter`, or `TimerEvery`.

Returns

The `._TimerIDs` list with the *`tid`* removed.

Description

This method cancels the timer corresponding to *`tid`*, and removes that timer ID number from the `._TimerIDs` list.

TimerAfter

`TimerAfter` — attaches an event handler to an "after" timer.

Syntax

```
TimerAfter (seconds, fn)
```

Arguments

`seconds`

A number of seconds

`fn`

A function or method call.

Returns

An integer that is a timer ID number.

Description

This method sets an `after` timer that fires after the number of *`seconds`* specified, causing the *`fn`* function or method to execute. This method also creates a unique, sequential ID number for the timer, appends that number to the class's `._TimerIDs` list, and returns that same timer ID number.

Timers created using `TimerAt`, `TimerAfter`, and `TimerEvery` are automatically cancelled when the Application instance is destroyed.

TimerAt

`TimerAt` — attaches an event handler to an "at" timer.

Syntax

```
TimerAt (day, month, year, hour, minute, second, fn)
```

Arguments

day

Restriction on the day of the month (`1-31`), or `nil` for none.

month

Restriction on the month of the year (`1-12`), or `nil` for none.

year

Restriction on the year (`1994-2026`), or `nil` for none.

hour

Restriction on the hour of the day (`0-23`), or `nil` for none.

minute

Restriction on the minute in the hour (`0-59`), or `nil` for none.

second

Restriction on the second in the minute (`0-59`), or `nil` for none.

fn

A function or method call.

Returns

Description

This method sets an `at` timer that causes the *fn* function or method to execute at a specific time, or to occur regularly at certain times of the minute, hour, day, month or year. A restriction on a particular attribute of the time will cause the timer to fire only if that restriction is true.

A restriction may be any number in the legal range of that attribute, or a list of numbers in that range. Illegal values for the time will be normalized. For example, a time specified as `July 0, 2008 00:00:00` will be treated as `June 30, 2008 00:00:00`. If `nil` is specified for any attribute of the time, this implies no restriction and the timer will fire cyclically at every legal value for that attribute.

This method also creates a unique, sequential ID number for the timer, appends that number to the class's `._TimerIDs` list, and returns that same timer ID number.

Timers created using `TimerAt`, `TimerAfter`, and `TimerEvery` are automatically cancelled when the Application instance is destroyed.

TimerEvery

`TimerEvery` — attaches an event handler to an "every" timer.

Syntax

```
TimerEvery (seconds, fn)
```

Arguments

`seconds`

A number of seconds.

`fn`

A function or method call.

Returns

An integer that is a timer ID number.

Description

This method sets an `every` timer that that fires periodically every number of *`seconds`*, causing the *`fn`* function or method to execute. This method also creates a unique, sequential ID number for the timer, appends that number to the class's `._TimerIDs` list, and returns that same timer ID number.

Timers created using `TimerAt`, `TimerAfter`, and `TimerEvery` are automatically cancelled when the Application instance is destroyed.

V. Time Conversion Functions from `Time.g`

Table of Contents

The DataHub works in two different time standards:

- Windows `DATE` (`VT_DATE`) is the number of days, as an 8-byte float since December 30, 1899. The fractional part is unsigned, so 6 a.m. December 29, 1899 would be represented as -1.25.
- Unix time is the number of seconds since 12 a.m. January 1, 1970. It is a signed 4-byte integer. Negative times have no meaning.

In these functions, Unix time uses the traditional Unix time plus a number of nanoseconds. Since this cannot be expressed as a single 4-byte integer, we express it as an 8-byte floating point number where the fractional part adds accuracy. This accuracy is ideally measured down to the nanosecond, but in practise relies on the accuracy of the operating system clock. In Windows, this means milliseconds.

Unix time is important because it is the time format used in the Gamma functions `time`, `mktime`, `localtime` and `date`. It is also the time format used when explicitly specifying a time stamp on a point in Unix.

Windows time is the internal time representation used by DataHub points.

The functions listed here are created in the script `Time.g`, and are available to any DataHub script that requires or includes the `Time.g` file, using one of these statements:

- `require ("Time");`
- `include ("Time");`

GetCurrentWindowsTime

`GetCurrentWindowsTime` — returns the current clock time in Windows time format.

Syntax

```
GetCurrentWindowsTime ()
```

Arguments

none

Returns

The current clock time, in Windows time format.

PointGetUnixTime

`PointGetUnixTime` — gets the Unix time stamp from a point.

Syntax

```
PointGetUnixTime (point)
```

Arguments

`point`

The fully-qualified symbolic point name of a DataHub point.

Returns

The Unix time stamp for the point.

PointGetWindowsTime

`PointGetWindowsTime` — gets the Windows time stamp from a point.

Syntax

```
PointGetWindowsTime (point)
```

Arguments

`point`

The fully-qualified symbolic point name of a DataHub point.

Returns

The Windows time stamp.

PointMetadata

`PointMetadata` — queries a point for its metadata structure.

Syntax

`PointMetadata (`*`point`*`)`

Arguments

`point`

The fully-qualified symbolic point name of a DataHub point.

Returns

Either a `DH_Item` structure, or `nil`.

Description

This function queries a point for its metadata structure, which is a `DH_Item` containing the following fields:

`item.canonical_type`

A number representing the canonical type of the point. See the possible values of VARTYPE in Windows:

```
VT_EMPTY    = 0
VT_I2       = 2
VT_I4       = 3
VT_R4       = 4
VT_R8       = 5
VT_CY       = 6
VT_DATE     = 7
VT_BSTR     = 8
VT_BOOL     = 11
VT_I1       = 16
VT_UI1      = 17
VT_UI2      = 18
VT_UI4      = 19
VT_I8       = 20
VT_UI8      = 21
VT_INT      = 22
VT_UINT     = 23
```

`item.conf`

A number from `0` to `100` indicating a confidence factor.

`item.flags`

Any bitwise combination of the following numeric set of flags:

```
    DH_ITEM_READABLE 0x0001
    DH_ITEM_WRITABLE 0x0002
    DH_ITEM_LOCKED 0x0004
    DH_ITEM_PROPERTY 0x0008
    DH_ITEM_SUBASSEMBLY 0x0010
    DH_ITEM_ASSEMBLY 0x0020
    DH_ITEM_ATTRIBUTE 0x0040
    DH_ITEM_TYPE 0x0080
    DH_ITEM_ACTIVE 0x0100
    DH_ITEM_PRIVATE_ATTRIBUTE
    DH_ITEM_PROCESSED 0x0400
    DH_ITEM_HIDDEN 0x0800
    DH_ITEM_AUTO_XID 0x1000
```

`item.opcaccessrights`

Any bitwise combination of `1` for `READABLE` and `2` for `WRITABLE`. The same as the first two bits of `item.flags`.

`item.quality`

The OPC quality value. You can get the name associated with the value by using the `GetQualityName` function in the `Quality.g` file. Put the statement:

```
require ("Quality");
```

at the beginning of your script to gain access to this function.

`item.scan_max`

Maximum scan rate on this point. Not in use.

`item.security`

The security level on this point.

`item.timeoffset`

Offset in seconds from local clock time for the originator of this point value. Not in use.

`item.timestamp`

The Windows time stamp of this point.

The point metadata is likely to change in the future, though the actual fields available will remain the same. It would be good practice to wrap access to the metadata within wrapper functions.

`UnixLocalToUTC`

`UnixLocalToUTC` — converts a local Unix time into UTC.

Syntax

```
UnixLocalToUTC (secs, offsetsecs)
```

Arguments

`secs`

Unix time, in seconds.

`offsetsecs`

The offset between local time and UTC, in seconds.

Returns

The UTC time.

UnixTimeToWindowsTime

`UnixTimeToWindowsTime` — converts from Unix time to Windows time.

Syntax

```
UnixTimeToWindowsTime (secs)
```

Arguments

`secs`

Unix time, in seconds.

Returns

A single number representing Windows time.

UnixUTCToLocal

`UnixUTCToLocal` — converts a UTC Unix time into local time.

Syntax

```
UnixUTCToLocal (secs, offsetsecs)
```

Arguments

`secs`

Unix time, in seconds.

`offsetsecs`

The offset between local time and UTC, in seconds.

Returns

A local time.

`WindowsLocalToUTC`

`WindowsLocalToUTC` — converts a local Windows time into UTC.

Syntax

```
WindowsLocalToUTC (wdate, offsetsecs)
```

Arguments

`wdate`

The date, in Windows time.

`offsetsecs`

The offset between local time and UTC, in seconds.

Returns

A UTC time.

`WindowsTimeToUnixTime`

`WindowsTimeToUnixTime` — converts from Windows time to Unix time.

Syntax

```
WindowsTimeToUnixTime (wdate)
```

Arguments

`wdate`

The date, in Windows time.

Returns

The date, in Unix time.

Description

This function converts from Windows time to Unix time. If the *`wdate`* is prior to January 1, 1970, the function returns `nil`. If *`wdate`* is after the end of the Unix epoch (03:14:08 UTC on January 19, 2038) then the returned value will be greater than the acceptable input range for Unix date-based functions.

`WindowsUTCToLocal`

`WindowsUTCToLocal` — converts a UTC Windows time into local time.

Syntax

```
WindowsUTCToLocal (wdate, offsetsecs)
```

Arguments

`wdate`

The date, in Windows time.

`offsetsecs`

The offset between local time and UTC, in seconds.

Returns

A local Windows time.

VI. Regular Expression Methods from `RegexSupport.g`

Table of Contents

The `RegexSupport.g` file provides support for the Perl-Compatible Regular Expression (PCRE) library (http://www.pcre.org/) and includes the PCRS substitution extensions to PCRE.

You must require the Regex support methods before you can use them in a script. At the top of your script, include the following line:

```
require (RegexSupport);
```

The `RegexSupport.g` file provides three classes:

- `Regex`, a compiled regular expression pattern. All but one of the methods in this reference are for this class.
- `pcrs_job`, a compiled substitution expression. There is one method for this class, `pcrs_job.Exec`.
- `pcre_extra`, which contains hints for the regular expression Exec function to speed up execution.

Please see the `Match` and `Subst` entries for examples.

Compile

`Compile` — compiles a regular expression to a form that is more efficient to evaluate.

Syntax

```
static Regex.Compile(pattern, options)
```

Arguments

`pattern`

A string specifying the regular expression pattern.

`options`

A bitwise OR of zero or more of the following:

```
PCRE_ANCHORED
PCRE_BSR_ANYCRLF
PCRE_BSR_UNICODE
PCRE_CASELESS
PCRE_DOLLAR_ENDONLY
PCRE_DOTALL
PCRE_DUPNAMES
PCRE_EXTENDED
PCRE_EXTRA
PCRE_FIRSTLINE
PCRE_JAVASCRIPT_COMPAT
PCRE_MULTILINE
PCRE_NEWLINE_CR
PCRE_NEWLINE_LF
PCRE_NEWLINE_CRLF
PCRE_NEWLINE_ANYCRLF
PCRE_NEWLINE_ANY
PCRE_NO_AUTO_CAPTURE
PCRE_UNGREEDY
PCRE_UTF8
PCRE_NO_UTF8_CHECK
```

Returns

On success, an instance of `Regex`. On failure, an error list with three elements:

- `car(errlist)` = a numeric return code from the failing function
- `cadr(errlist)` = a character position in the pattern where the error occurred, or 0.
- `caddr(errlist)` = an error message, as a human-readable string

See Also

`Match`

CompileSubst

`CompileSubst` — compiles a substitution pattern.

Syntax

```
static Regex.CompileSubst(pattern)
```

Arguments

`pattern`

A pattern of the form `s/pattern/substitution/flags`.

Returns

On success, an instance of `pcrs_jobs`. On failure, an error list with three elements:

- `car(errlist)` = a numeric return code from the failing function
- `cadr(errlist)` = a character position in the pattern where the error occurred, or 0.
- `caddr(errlist)` = an error message, as a human-readable string

Description

This method compiles a pattern of the form `s/pattern/substitution/flags`.

CompileSubstEx

`CompileSubstEx` — an alternate interface to CompileSubst.

Syntax

```
static Regex.CompileSubstEx(pattern, substitution, flags?=0)
```

Arguments

`pattern`

Normally, a string, but optionally a buffer. If a buffer, it may contain `NUL` characters.

`substitution`

Normally, a string, but optionally a buffer. If a buffer, it may contain `NUL` characters.

`flags`

A string.

Returns

On success, an instance of `pcrs_job`. On failure, an error list with three elements:

- `car(errlist)` = a numeric return code from the failing function
- `cadr(errlist)` = a character position in the pattern where the error occurred, or 0.
- `caddr(errlist)` = an error message, as a human-readable string

`Config`

`Config` — queries the PCRE library for compiled-in options.

Syntax

```
static Regex.Config(what)
```

Arguments

`what`

The option to query.

Returns

A number.

Description

See the documentation for pcre_config for details.

Exec

`Exec` — applies a regular expression match to a subject string.

Syntax

```
Regex.Exec(extra, subject, length, startoffset, options, substrings)
```

Arguments

`extra`

An instance of `pcre_extra` returned from `Regex.Study`, or nil.

`subject`

The subject string on which to apply the pattern. This may be either a string or a buffer. If the subject is a buffer it may contain `NUL` characters.

`length`

The length of the subject, in bytes. If length is `-1`, the length of the subject string is computed automatically .

`startoffset`

A byte offset into the subject specifying at which point to start.

`options`

Option flags - a bitwise OR of zero or more of these flags:
`PCRE_ANCHORED`
`PCRE_NEWLINE_CR`
`PCRE_NEWLINE_LF`
`PCRE_NEWLINE_CRLF`
`PCRE_NEWLINE_ANYCRLF`
`PCRE_NEWLINE_ANY`
`PCRE_NOTBOL`
`PCRE_NOTEOL`
`PCRE_NOTEMPTY`
`PCRE_NOTEMPTY_ATSTART`
`PCRE_NO_START_OPTIMIZE`
`PCRE_NO_UTF8_CHECK`
`PCRE_PARTIAL_SOFT`
`PCRE_PARTIAL_HARD`

`substrings`

An array into which the match substrings are copied. The substrings array cannot be nil. The match substrings will be written into the substrings array starting at position `0`. If the array is not large enough to hold all of the substrings it will be automatically expanded.

Each element of the substrings array is itself an array of 3 elements:
`[0]` - the zero-based starting character of the match
`[1]` - the character position immediately after the match
`[2]` - the string that matched.
This will be returned as a buffer if the subject is a buffer, otherwise it will be a string.

The first element in the substrings array always contains the complete matching string within the subject. Each element after the first corresponds to a positional substring specified within the pattern.

Returns

On success, an array of match specifications as described in the *substrings* parameter (above). On failure, an error list with three elements:

- `car(errlist)` = a numeric return code from the failing function
- `cadr(errlist)` = a character position in the pattern where the error occurred, or 0.
- `caddr(errlist)` = an error message, as a human-readable string

See Also

Match

pcrs_job.Exec

`pcrs_job.Exec` — applies a compiled pattern substitution to a subject string.

Syntax

```
pcrs_job.Exec(subject, length?=-1)
```

Arguments

`subject`

A subject string for substitution.

`length`

The length of the subject string. Entering `-1` (the default) for this parameter will cause the length of the subject string to be automatically computed.

Returns

On success, the new string with the substitutions applied. On failure, an error list with three elements:

- `car(errlist)` = a numeric return code from the failing function
- `cadr(errlist)` = a character position in the pattern where the error occurred, or 0.
- `caddr(errlist)` = an error message, as a human-readable string

Description

This method of the `pcrs_job` class applies a previously compiled pattern substitution to a subject string.

GetStringNumber

`GetStringNumber` — retrieves the index number of named substrings.

Syntax

```
Regex.GetStringNumber(name)
```

Arguments

`name`

The name of a substring.

Returns

On success, a number greater than `1`. On failure, a numeric error code less than `0`.

- `car(errlist)` = a numeric return code from the failing function
- `cadr(errlist)` = a character position in the pattern where the error occurred, or 0.
- `caddr(errlist)` = an error message, as a human-readable string

Description

If a pattern contains named substring matches, this method will retrieve the index number (starting at `1`) of the substring corresponding to this name.

Match

`Match` — a convenience function that combines calls to Compile, Study, and Exec.

Syntax

```
static Regex.Match(pattern, subject, options?=0, length?=-1, startoffset?=0)
```

Arguments

`pattern`

A string specifying the regular expression pattern.

`subject`

The subject string on which to apply the pattern. This may be either a string or a buffer. If the subject is a buffer it may contain `NUL` characters.

`options`

Option flags - a bitwise OR of zero or more of these flags:

```
PCRE_ANCHORED
PCRE_BSR_ANYCRLF
PCRE_BSR_UNICODE
PCRE_CASELESS
PCRE_DOLLAR_ENDONLY
PCRE_DOTALL
PCRE_DUPNAMES
PCRE_EXTENDED
PCRE_EXTRA
PCRE_FIRSTLINE
PCRE_JAVASCRIPT_COMPAT
PCRE_MULTILINE
PCRE_NEWLINE_ANY
PCRE_NEWLINE_ANYCRLF
PCRE_NEWLINE_CR
PCRE_NEWLINE_CRLF
PCRE_NEWLINE_LF
PCRE_NOTBOL
PCRE_NOTEMPTY
PCRE_NOTEMPTY_ATSTART
PCRE_NOTEOL
PCRE_NO_AUTO_CAPTURE
PCRE_NO_START_OPTIMIZE
PCRE_NO_UTF8_CHECK
PCRE_PARTIAL_HARD
PCRE_PARTIAL_SOFT
PCRE_UNGREEDY
PCRE_UTF8
```

`length`

The length of the subject, in bytes. If length is `-1` (the default), the length of the subject string is computed automatically.

`startoffset`

A byte offset into the subject specifying at which point to start.

Returns

On success, an array of match specifications equivalent to the substrings argument to `Regex.Exec`. On failure, an error list with three elements:

- `car(errlist)` = a numeric return code from the failing function
- `cadr(errlist)` = a character position in the pattern where the error occurred, or 0.
- `caddr(errlist)` = an error message, as a human-readable string

Examples

Find and return all matches of hello, case sensitive:

```
Gamma> Regex.Match ("(hello)", "I say Hello and you say Goodbye");
(-1 0 "No match found")
```

Find and return all matches of hello, case insensitive:

```
Gamma> Regex.Match ("(hello)", "I say Hello and you say Goodbye", PCRE_CASELESS);
[[0 31 #{I say Hello and you say Goodbye}] [6 11 #{Hello}]]
```

See Also

`Compile`, `Study`, `Exec`, and `Subst`

Study

`Study` — helpful for speeding up repeated applications of a regular expression.

Syntax

```
Regex.Study(options?=0)
```

Arguments

`options`

Must be zero in this implementation.

Returns

On success, an instance of `pcre_extra`. On failure, an error list with three elements:

- `car(errlist)` = a numeric return code from the failing function
- `cadr(errlist)` = a character position in the pattern where the error occurred, or 0.
- `caddr(errlist)` = an error message, as a human-readable string

Description

This method evaluates a regular expression to produce hints that can speed up the application of the regular expression in future. This method does not need to be called, but is useful to speed up repeated application of the same regular epxression.

See Also

`Match`

Subst

`Subst` — applies a substitution pattern to a string of a given length.

Syntax

```
static Regex.Subst(pattern, subject, length?=-1)
```

Arguments

`pattern`

A substitution pattern, in the form `s/pattern/substitution/flags`.

`subject`

The subject string on which the substitution will be made.

`length`

The length of the subject string. Entering `-1` (the default) for this parameter will cause the length of the subject string to be automatically computed.

Returns

On success, returns the new string with the substitutions applied. On failure, an error list with three elements:

- `car(errlist)` = a numeric return code from the failing function
- `cadr(errlist)` = a character position in the pattern where the error occurred, or 0.
- `caddr(errlist)` = an error message, as a human-readable string

Examples

String replacement:

```
Gamma> x = "This is a test of my test system"
"This is a test of my test system"
Gamma> Regex.Subst("s/test/build/g", x)
"This is a build of my build system"
```

To surround all words with parentheses:

```
Gamma> Regex.Subst ("s/(\\w+)/\\($1\\)/g", "This is a test!");
"(This) (is) (a) (test)!"
```

See Also

`Match`

VII. Quality Name Function from `Quality.g`

Table of Contents

GetQualityName

`GetQualityName` — converts the quality value of a point to a text string.

Syntax

`GetQualityName (`*`quality`*`)`

Arguments

`quality`

A value that indicates the quality of the point, as returned by the function `PointMetaData`.

Returns

A text string corresponding to the *`quality`*.

Description

This function converts the quality value of a point to a text string. The function is created in the script `Quality.g`, and is available to any DataHub script that requires or includes the `Quality.g` file, using one of these statements:

- `require ("Quality");`
- `include ("Quality");`

DataHub point information includes a data quality indication that is consistent with the OPC specification. Even if you are not working with OPC data, you may still wish to use the quality in your custom applications and DataHub scripts.

The possible quality strings are:

Value	String
OPC_QUALITY_BAD	Bad
OPC_QUALITY_COMM_FAILURE	Comm Failure
OPC_QUALITY_CONFIG_ERROR	Config Error
OPC_QUALITY_DEVICE_FAILURE	Device Failure
OPC_QUALITY_EGU_EXCEEDED	EGU Exceeded
OPC_QUALITY_GOOD	Good
OPC_QUALITY_LAST_KNOWN	Last Known
OPC_QUALITY_LAST_USABLE	Last Usable
OPC_QUALITY_LOCAL_OVERRIDE	Local Override
OPC_QUALITY_MASK	Mask
OPC_QUALITY_NOT_CONNECTED	Not Connected
OPC_QUALITY_OUT_OF_SERVICE	Out Of Service
OPC_QUALITY_SENSOR_CAL	Sensor Cal
OPC_QUALITY_SENSOR_FAILURE	Sensor Failure
OPC_QUALITY_SUB_NORMAL	Sub Normal
OPC_QUALITY_UNCERTAIN	Uncertain

VIII. Classes from `HistorianSupport.g`

Table of Contents

The functionality of the DataHub Historian features is made available through a `Historian` class provided in the `HistorianSupport.g` script shipped with the DataHub.

You must require the Historian support methods before you can use them in a script. At the top of your script, include the following line:

```
require (HistorianSupport);
```

The `HistorianSupport.g` file provides three classes: `Historian`, `HistoryBuffer`, and `HistoryValue`. The `HistTest.g` script provides an example of how these classes are used to query the DataHub Historian.

Historian

`Historian` — encapsulates the Historian facility in the DataHub.

Syntax

```
class Historian
{
}
```

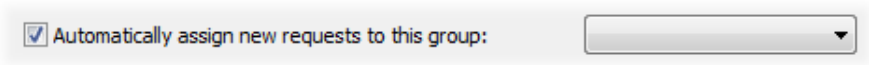
Instance Variables

None.

Methods

`allowAutoHistoryAdds`(*`state`*, *`defaultgroup`*)

Configures automatic history addition. The *`state`* should be `nil` to turn off the function and `t` to turn it on. The *`defaultgroup`* is a string naming the History Group to which automatic histories are added. This method corresponds to this setting in the Historian option of the Properties window:



`addGroup`(*`label`*, *`basedir`*, *`cachesize`*)

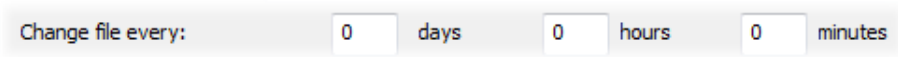
Creates a new History Group with the provided *`label`*, *`basedir`* (base directory) and *`cachesize`* (cache size). This corresponds approximately to pressing the Add ... button in the Historian option of the Properties window.

`removeGroup`(*`label`*)

Removes the History Group with the given *`label`*.

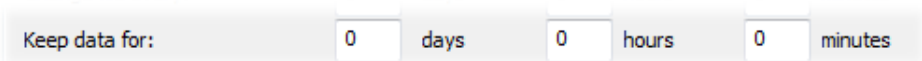
`setFileTimes`(*`labelpattern`*, *`days`*, *`hours`*, *`minutes`*)

Assigns the file times for all History Groups whose labels match the *`labelpattern`*. The pattern uses the pattern matching rules documented in the `shell_match` function. This method corresponds to this setting in the Properties window Historian configuration:



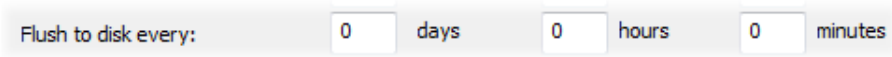
`setRentionTimes`(*`labelpattern`*, *`days`*, *`hours`*, *`minutes`*)

Assigns the data retention times for all History Groups whose labels match the *`labelpattern`*. The pattern uses the pattern matching rules documented in the `shell_match` function. This method corresponds to this setting in the Properties window Historian configuration:



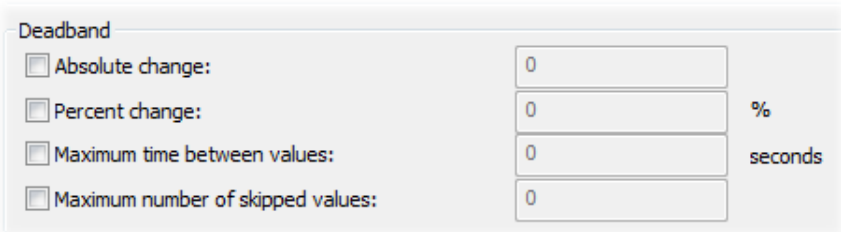
`setFlushTimes`(*`labelpattern`*, *`days`*, *`hours`*, *`minutes`*)
()

Assigns the file flush times for all History Groups whose labels match the *`labelpattern`*. The pattern uses the pattern matching rules documented in the `shell_match` function. This method corresponds to this setting in the Properties window Historian configuration:



`setDeadband`(*`labelpattern`*, *`flags`*, *`absolute`*, *`percent`*, *`maxsecs`*, *`maxcount`*)

Assigns the deadband settings for all History Groups whose labels match the *`labelpattern`*. The pattern uses the pattern matching rules documented in the `shell_match` function. This method corresponds to this section of the Properties window Historian configuration:



The *`flags`* argument is `0` to turn off all deadbands, or any combination of:

`0x0008` - Turn on the percent deadband
`0x0020` - Turn on the absolute deadband
`0x0040` - Turn on the maximum time limit
`0x0080` - Turn on the maximum skip count

`addPoint`(*`pointname`*, *`grouplabel`*`?=nil`)

Adds a point to the named History Group. If *`grouplabel`* is `nil` or not specified then the point will be added to the automatic group. If the automatic group is not set and the *`grouplabel`* is not specified, this method will fail silently.

The *`pointname`* is a string specifying the fully qualified point name. That is, the point name must contain the data domain and the full path of the point, for example, `DataPid:PID1.Mv`

`removePoint`(*`pointpattern`*, *`grouppattern`*`?="*"`)

Removes all points matching the *`pointpattern`* from all groups matching the *`grouppattern`*.

`saveConfig`()

Causes the Historian to save its configuration to disk.

`startConfiguring`()

Tells the Historian to enter configuring mode. In this mode, changes to group configurations are accepted but not applied. This allows a number of configuration commands to be sent without affecting the current state of the Historian. This method must eventually be followed by a call to `endConfiguring` for the configuration changes to be applied.

endConfiguring()

Tells the Historian to exit configuring mode, and to apply any changes to the configuration that have been made since the call to `startConfiguring`.

queryRawData(*pointname*, *start_unixtime*, *duration_secs*)

Queries the history for a given point, starting at the specified time (*start_unixtime*) and running for the specified number of seconds (*duration_secs*). The *pointname* must be a string containing the fully qualified name of the point. The *start_unixtime* must be specified as number of seconds since midnight January 1, 1970 UTC. Both the *start_unixtime* and *duration_secs* are floating point values and can contain a fractional component.

The return value from this function is either `nil` or an instance of `HistoryBuffer` containing the values that fall within the specified range.

The data returned from this method is the exact data set as stored on disk without modification.

queryStatistic(*pointname*, *start_unixtime*, *duration_secs*, *aggregation_period_secs*, *stat_name*)

Queries a statistical measure of the history for a given point, starting at the specified time (*start_unixtime*) and running for the specified number of seconds (*duration_secs*). The *pointname* must be a string containing the fully qualified name of the point. The *start_unixtime* must be specified as number of seconds since midnight January 1, 1970 UTC. Both the *start_unixtime* and *duration_secs* are floating point values and can contain a fractional component.

The return value from this function is either `nil` or an instance of `HistoryBuffer` containing the values that fall within the specified range.

The *aggregation_period_secs* specifies the number of seconds in each time interval within the overall query on which to produce statistics. For example, a query may request the average value in each 5-second period for a minute. The aggregation period would be 5 seconds, and the duration would be 60 seconds. The result set will contain 12 values, one for each 5-second interval within the duration of the query.

The *stat_name* is a string that specifies which statistical measure to query. The statistical measures available are:

Statistic Name	Description
`Total`	Retrieve the totalized value (time integral) of the data over the sample interval.
`Average`	Retrieve the average data over the sample interval.
`TimeAverage`	Retrieve the time weighted average data over the sample interval.
`Count`	Retrieve the number of good quality raw values over the sample interval.
`RawCount`	Retrieve the number of raw values over the sample interval.
`StDev`	Retrieve the standard deviation over the sample interval.
`MinimumActualTime`	Retrieve the minimum value in the sample interval and the timestamp of the minimum value.
`Minimum`	Retrieve the minimum value in the sample interval.

Statistic Name	Description
MaximumActualTime	Retrieve the maximum value in the sample interval and the timestamp of the maximum value.
Maximum	Retrieve the maximum value in the sample interval.
Start	Retrieve the value at the beginning of the sample interval. The time stamp is the time stamp of the beginning of the interval.
End	Retrieve the value at the end of the sample interval. The time stamp is the time stamp of the end of the interval.
First	Retrieve the first value change within the interval. The time stamp is the time stamp of the first value change.
Last	Retrieve the last value change within the interval. The time stamp is the time stamp of the last value change.
Delta	Retrieve the difference between the first and last value in the sample interval.
RegSlope	Retrieve the slope of the regression line over the sample interval.
RegConst	Retrieve the starting point of the regression line over the sample interval. This is the value of the regression line at the start of the interval.
RegPearsonR	Retrieve Pearson R measure of correlation between Y and Time over the sample interval. This is a number between -1.0 and 1.0.
RegRSquared	Retrieve R-squared measure of regression fitness over the sample interval. This is a number between 0 and 1.0.
Variance	Retrieve the variance over the sample interval .
Range	Retrieve the difference between the minimum and maximum value over the sample interval.
DurationGood	Retrieve the duration (in seconds) of time in the interval during which the data is good.
DurationBad	Retrieve the duration (in seconds) of time in the interval during which the data is bad.
PercentGood	Retrieve the percent of data (1 equals 100 percent) in the interval which has good quality.
PercentBad	Retrieve the percent of data (1 equals 100 percent) in the interval which has bad quality.
WorstQuality	Retrieve the worst quality of data in the interval.
ValueSum	Retrieve the sum of all raw values over the sample interval.

HistoryBuffer

`HistoryBuffer` — a set of `HistoryValue`s returned from a Historian query.

Syntax

```
class HistoryBuffer
{
    npoints;
}
```

Instance Variables

`npoints`

The number of values (of type `HistoryValue`) in this data set.

Methods

Result sets from historical queries can be large. Consequently the access methods for a result set come in two types. The first type, named `get`*`Something`*, will return a reference to the particular data array or item, without making a copy. This reference is only valid until the `HistoryBuffer` instance is destroyed, which may happen at any time after all references to the `HistoryBuffer` instance go out of scope.

If the `HistoryValue` or array returned from the `get` method is used after the `HistoryBuffer` is destroyed, the DataHub could crash.

Alternately, you can use the `copy`*`Something`* methods to create duplicates of the data requested. These instances will persist after the `HistoryBuffer` is destroyed. The `copy` methods are more robust, but increase the CPU usage and memory usage of large queries. In general you should use the copy methods when you are unsure whether the lifetimes of the `HistoryValues` will exceed the lifetime of the `HistoryBuffer`.

`copyToArray`()

Copy the internal data representation to an array of `HistoryValue` instances. This method returns the new array. This method is similar to `toArray`, except that it makes a copy of each `HistoryValue` instance.

`copyValue`(*`index`*)

Returns a copy of the `HistoryValue` at the given *`index`* within the data set. The *`index`* is a number starting at 0 and less then `npoints`. This is the safe version of the `getValue` method.

`getCount`()

Returns the number of values in this data set.

`getValue(`*`index`*`)`

Returns the `HistoryValue` at the given index in the data set. This is the unsafe version of `copyValue`. It will return a reference to the `HistoryValue` within the `HistoryBuffer`. This reference will only be valid until the `HistoryBuffer` is destroyed.

`setValue(`*`index`*`,` *`histvalue`*`)`

Modifies the original data in the `HistoryBuffer` at the given *`index`*. This method will result in changes to the original data, and to any `HistoryValue` instances that have been acquired by reference using `getValue` or `toArray`. The *`histvalue`* argument is an instance of `HistoryValue`.

`toArray()`

Returns an array of `HistoryValue` instances as references to the original data. This is equivalent to creating an array by repeatedly calling `getValue`. The resulting data is only valid until the `HistoryBuffer` is destroyed.

HistoryValue

`HistoryValue` — a single value from the Historian, consisting of value, x-axis and quality.

Syntax

```
class HistoryValue
{
    xaxis;
    value;
    quality;
}
```

Instance Variables

`xaxis`

The X axis value, generally the time stamp in UNIX time for this value. In some cases, a query can return an Y-vs-X result set instead of a Y-vs-Time result set. Consequently the X axis may not be time in all cases. This is a floating point number that can represent fractional seconds.

`value`

The data value. This is either a calculated value or a raw value, depending on the type of query. quality.

`quality`

The quality of this sample. For raw samples this is an OPC DA2 item quality. For computed samples this is `GOOD`, `BAD` or another more precise `BAD` quality (such as `DEVICE FAILURE`) taken from the qualities of the raw values used in the computation.

Methods

None.

HistTest.g

`HistTest.g` — an example script that demonstrates how to access `Historian` data.

Code

The code for this and other example scripts can be found in the DataHub distribution archive, typically at this location:
`C:\Program Files\Cogent\Cogent DataHub\scripts\`
Please refer to Section 3.1, *How to Run a Script* for more information on using scripts.

```
/*
 * This script demonstrates how to access the DataHub Historian data
 * both in raw and processed forms.  Generally, a script simply needs
 * to create an instance of the Historian class and then make calls
 * to the methods of that object.  See the documentation for the complete
 * list of methods provided.
 *
 * When you query data from the historian the result is a HistoryBuffer
 * instance.  This encapsulates the result data in an efficient internal
 * representation that is not visible directly to the script.  In order
 * to use the data you must retrieve the data from the HistoryBuffer
 * to produce one or more HistoryValue instances.  Each HistoryValue
 * contains a value, a timestamp and a quality.  The timestamp
 * is called "xaxis" because in some cases a query could return an
 * y-vs-x result instead of a y-vs-time result.  The Historian object
 * does not expose a method for a y-vs-x query, but the underlying
 * implementation will support it.
 *
 * A HistoryBuffer holds its data until there are no more references
 * to the HistoryBuffer object.  This means that a HistoryBuffer object
 * that is declared as a local variable in a function will only be valid
 * until the function exits (unless the function returns the HistoryBuffer
 * object).  If you need to hold one or more HistoryValue beyond the
 * lifetime of the HistoryBuffer, use copyToArray or copyValue to make a
 * copy of the data.
 *
 * If a history query produces a large amount of data it will be more
 * memory efficient to avoid making a copy of the data.  You can access
 * individual HistoryValue instances within the HistoryBuffer using the
 * getValue method.  Generally this will take more processing time, but
 * can substantially reduce memory use.
 *
 * Bear in mind that all scripts run in a single thread.  If you spend a
 * long time processing large historical requests, that will disrupt the
 * timing of other scripts.
 *
 * Statistical queries take substantially longer than raw data queries.
 */

require ("Application");

/* Load the functions for Historian access */

require ("HistorianSupport");

/* Applications share the execution thread and the global name
 * space, so we create a class that contains all of the functions
 * and variables for the application.  This does two things:
 *   1) creates a private name space for the application, and
 *   2) allows you to re-load the application to create either
 *      a new unique instance or multiple instances without
 *      damaging an existing running instance.
 */
class HistTest Application
{
    historian;
```

```
}

method HistTest.rawQuery(pointname, starttime, duration)
{
    local histbuffer, data, datacopy;
    local t1, t2, t3, t4;

    t1 = nanoclock();
    histbuffer = .historian.queryRawData(pointname, starttime, duration);
    t2 = nanoclock();
    data = histbuffer.toArray();
    t3 = nanoclock();
    datacopy = histbuffer.copyToArray();
    t4 = nanoclock();

    pretty_princ("Raw query produced ", histbuffer.npoints, " data values\n");
    pretty_princ("   Query took ", t2 - t1, " seconds\n");
    pretty_princ("   Data access took ", t3 - t2, " seconds\n");
    pretty_princ("   Data copy took ", t4 - t3, " seconds\n");

    // If we want to use the values outside this method, we need to use a copy, not the raw data.
    // The raw data will be cleaned up when the histbuffer object is no longer referenced.
    // We could also return the histbuffer object from this method, from which we could later query
    // the data using toArray().
    datacopy;
}

method HistTest.statisticalQuery(pointname, starttime, duration, aggregation_period_secs, statistic)
{
    local histbuffer, data, datacopy;
    local t1, t2, t3, t4, t5, t6, t7;
    local i, numbad, numbad2;

    t1 = nanoclock();
    histbuffer = .historian.queryStatistic(pointname, starttime, duration, aggregation_period_secs, statistic
    t2 = nanoclock();
    data = histbuffer.toArray();
    t3 = nanoclock();
    datacopy = histbuffer.copyToArray();
    t4 = nanoclock();

    pretty_princ("Statistical query, ", statistic, ", produced ", histbuffer.npoints, " data values\n");
    pretty_princ("   Query took ", t2 - t1, " seconds\n");
    pretty_princ("   Data access took ", t3 - t2, " seconds\n");
    pretty_princ("   Data copy took ", t4 - t3, " seconds\n");

    // There are a few ways to process the data.  We can convert the data to an array, as we do with
    // data and datacopy.  We can also walk the data buffer without converting it to an array.  The
    // next two examples both do the same thing, walking the data looking for BAD quality.
    numbad = numbad2 = 0;
    t5 = nanoclock();

    // Loop through the data set, having previously converted the buffer data to an array
    with value in data do
    {
        if (value.quality == OPC_QUALITY_BAD)
            numbad++;
    }
    t6 = nanoclock();

    // Loop through the data set without converting to an array.  If the query produces a large amount
    // of data, this will certainly be more memory-efficient since a large array does not have to be
    // created, but the processing cost of querying each value in the data set will be higher.
    for (i=histbuffer.npoints-1; i>=0; i--)
    {
        if (histbuffer.getValue(i).quality == OPC_QUALITY_BAD)
            numbad2++;
    }
```

```
    t7 = nanoclock();

    pretty_princ("   Loop through array took ", t6 - t5, " seconds\n");
    pretty_princ("   Loop through buffer took ", t7 - t6, " seconds\n");
    pretty_princ("      Found ", numbad, " (", numbad2, ") Bad quality values\n");

    // Print the last value in the data array, just to show some results.
    if(array_p(data) && length(data) > 0)
    {
        local result = data[length(data) -1];
        pretty_princ("     Query results for most recent value: ",
        date_of(result.xaxis), " : ", result.value,  "\n\n");
    }
}

/* Write the 'main line' of the program here. */
method HistTest.constructor ()
{
    .historian = new Historian();
    princ ("---------------------------------------\n");
    .rawQuery("DataPid:PID1.Mv", nanoclock() - 10, 10);
    .rawQuery("DataPid:PID1.Mv", nanoclock() - 100, 100);
    .rawQuery("DataPid:PID1.Mv", nanoclock() - 1000, 1000);
    .rawQuery("DataPid:PID1.Mv", nanoclock() - 10000, 10000);
    princ ("------\n");
    .statisticalQuery("DataPid:PID1.Mv", nanoclock() - 10, 10, 1, "Average");
    .statisticalQuery("DataPid:PID1.Mv", nanoclock() - 100, 100, 1, "StDev");
    .statisticalQuery("DataPid:PID1.Mv", nanoclock() - 1000, 1000, 1, "Minimum");
    .statisticalQuery("DataPid:PID1.Mv", nanoclock() - 10000, 10000, 1, "RegRSquared");
}

/* Any code to be run when the program gets shut down. */
method HistTest.destructor ()
{
}

/* Start the program by instantiating the class. */
ApplicationSingleton (HistTest);
```

Index

Symbols

A

C

D

E

G

H

I

L

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 written by Bob McIlvride and Andrew Thomas. Illustrations by Bob McIlvride and Paul Benford.