DYALOG

The tool of thought for software solutions

# HTMLRenderer User Guide

## Dyalog APL Version 19.0


Dyalog Limited

Minchens Court, Minchens Lane
Bramley, Hampshire
RG26 5BH
United Kingdom

tel: +44 1256 830030

fax: +44 1256 830031
email: support@dyalog.com
http://www.dyalog.com

*HTMLRenderer User Guide*

*Dyalog Version 19.0*
*Document Revision: 20240522_190*

Unless stated otherwise, all examples in this document assume that `(⎕IO ⎕ML) ← 1`



*email: support@dyalog.com*
*https://www.dyalog.com*

# Contents

# 1 Introduction

HTMLRenderer is a built-in Dyalog object which provides a cross-platform mechanism for producing Graphical User Interfaces (GUIs), based on Hypertext Markup Language (HTML). As of Dyalog version 18.0 HTMLRenderer is available on Microsoft Windows, Apple macOS, and Linux (excluding the Raspberry Pi). Using HTMLRenderer, your application can use the same code to provide a consistent user interface across platforms.

HTMLRenderer is a built-in class, instances of which are created and managed using the Dyalog GUI framework functions `⎕WC`/`⎕WS`/`⎕WG`/`⎕NEW` and `⎕DQ`/`⎕NQ`. User interfaces are defined using HTML, which can, in turn, make references to code and data in a number of additional formats such as JavaScript to manage highly interactive content, Cascading Style Sheets (CSS) for both simple and sophisticated styling, and SVG, JPG or BMP for images.

On all platforms, the creation of an HTMLRenderer object causes APL to open a new window and run a copy of the Chromium Embedded Framework (CEF). HTMLRenderer manages the communication between your APL code and CEF.

The HTMLRenderer can be disabled by setting the ENABLE_CEF environment variable to 0; if ENABLE_CEF is not set or is set to 1 then the HTMLRenderer is enabled (the default).

On the Linux and MacOS platforms the use of HTMLRenderer and APs (Auxilliary Processors) is mutually exclusive. With ENABLE_CEF set to 1 (which is the default), you will be able to use HTMLRenderer but you cannot use APs. Attempting to using an AP with ENABLE_CEF set to 1 will cause the AP to hang. With ENABLE_CEF set to 0, you will be able to use APs but not HTMLRenderer.

Attempting to create an instance of HTMLRenderer on a platform where ENABLE_CEF when ENABLE_CEF is 0 will cause a "`LIMIT ERROR: The object could not be created`" error to be signalled. See section 4, *Technical Overview* for more information on enabling the HTMLRenderer on various platforms.

## 1.1 Hello World

Below is a simple HTMLRenderer example. The first line defines the HTML body, and uses the <h1> (heading level 1) tag create the large, bolded, "Hello World" followed by more text containing some APL characters. The second and third lines define the size and coordinate system. The last line creates an HTMLRenderer using the HTML, Size and Coord properties:

```
html←'<h1>Hello World</h1>This is an ⍺⍴⌊ page.'
size←150 250
coord←'ScaledPixel'
'hr' ⎕WC 'HTMLRenderer' ('HTML' html)('Size' size)('Coord' coord)
```

The resulting window on various platforms looks like:


**Linux (Ubuntu)**


**macOS (Catalina)**


**Windows 10**

Screenshots throughout this document will be captured from different platforms on different machines with different screen resolutions.

## 1.2 Other Resources

All HTML applications are based on an initial HTML document. Most modern HTML-based user interfaces will reference other resources, such as JavaScript and CSS files which contain code that can influence the way the base HTML is rendered, image files, and of course hyperlinks to other pages.

If the HTML contains references to other resources, the CEF will retrieve each one by making an HTTP request. Each request with a URL that matches a triggering pattern in `InterceptedURLs` will generate an `HTTPRequest` event on the instance of HTMLRenderer, which can be directed to a callback function in APL. Requests with URLs that do not match a pattern in `InterceptedURLs` or that match a pattern with a 0 in the second column will cause the CEF to push the request out to the network and see whether an external server is able to service it. `InterceptedURLs` allows an APL application to decide how which content it wants to provide, and to what extent it wants to act as a portal for other services that will provide the rest of the data.

## 1.3 User Events

When a user submits an HTML form for processing, or a user interface component which is being managed by JavaScript code wishes to make a server request, this is also done by making an HTTP request. These requests will also be directed through the same `InterceptedURLs` mechanism. This makes it possible to develop interactive applications where your APL code is responding to user input, as well as providing the content of resources needed to render the UI.

## 1.4 Platform Variations

Under Microsoft Windows, HTMLRenderer objects can be used as children of normal ⎕WC forms. Some properties such as `MinButton`, `MaxButton`, and `Sizeable` are not available on all platforms. See Properties Available by Platform.

## 1.5 UTF-8 Support

UTF-8 is the default character set for HTML5. Prior to version 18.0, Unicode code points greater than 127 would need to be converted to their equivalent HTML entities. For instance, ↊ would need to be encoded as `&#9025;`. Version 18.0 removes this requirement by prepending a UTF-8 byte order mark (BOM) to the HTML content sent to CEF. Content that should not have the BOM prepended (for example, an image) should be sent as integer datatype 83. This is a **breaking change** from previous versions of HTMLRenderer.

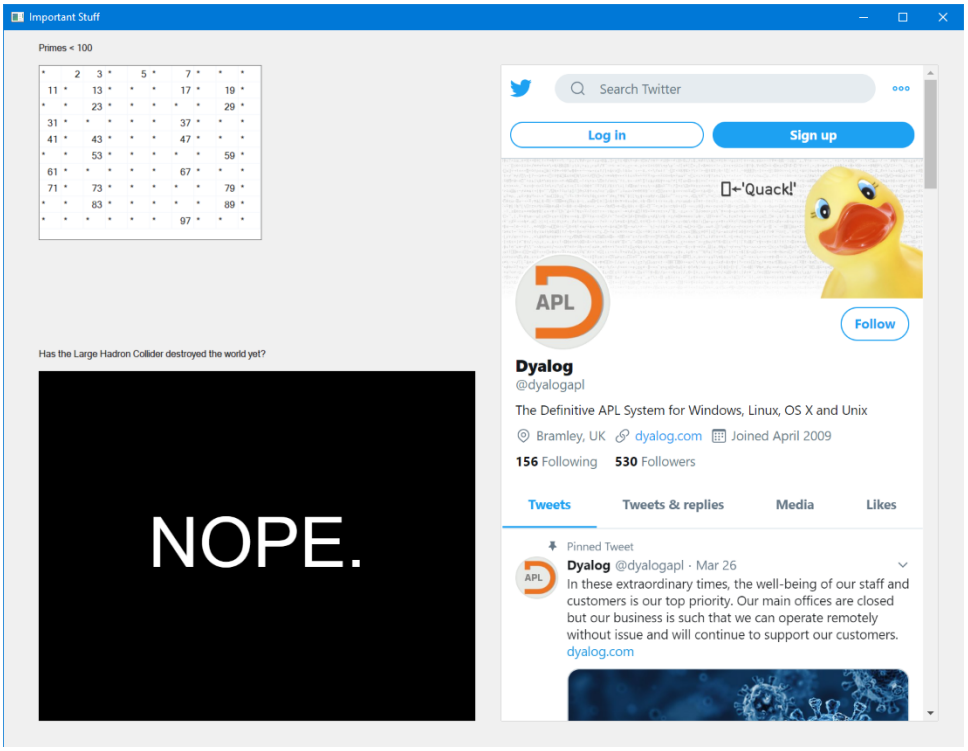
## 1.6 Integration with the Dyalog GUI on Windows

The following code illustrates how HTMLRenderer objects can be used as children of normal ⎕WC forms under Microsoft Windows. By setting the AsChild property of an HTMLRenderer object to 1, we request that the HTMLRenderer window be embedded as a sub-form of another window.

```
'pco' ⎕CY 'dfns'
'f1'⎕WC'Form' 'Important Stuff' ('Coord' 'ScaledPixel')('Size' 820 1100)
'f1.label1' ⎕WC 'Label' 'Primes < 100' (10 40)
'f1.primes' ⎕WC 'Grid' ('*' @ (0∘pco) 10 10⍴⍳100) ('Posn' 40 40)
f1.primes.(TitleHeight TitleWidth CellWidths Size)←0 0 25 (200 255)
```

```
'f1.label2' ⎕WC 'Label' 'Has the Large Hadron Collider destroyed the world yet?' (360 40)
'f1.areWeStillHere' ⎕WC 'HTMLRenderer' ('AsChild' 1) ('Posn' 390 40)('Size' 400 500)
f1.areWeStillHere.URL←'http://hasthelargehadroncolliderdestroyedtheworldyet.com'
twitter←'<a class="twittertimeline" href="https://twitter.com/dyalogapl">'
twitter,←'Tweets by dyalogapl</a>'
twitter,←'<script async src="//platform.twitter.com/widgets.js" charset="utf-8"></script>'
'f1.twitter' ⎕WC 'HTMLRenderer' ('AsChild' 1) ('Posn' 40 570)('Size' 750 500)
f1.twitter.HTML←twitter
```

The result after clicking on the "Tweets by dyalogapl" link can be seen below; a form that contains a Windows grid showing prime numbers between 1 and 100 as well as provides live feeds from two external sites. Note that no callbacks have been assigned; in this case the HTMLRenderer always goes to the network to satisfy requests for data.



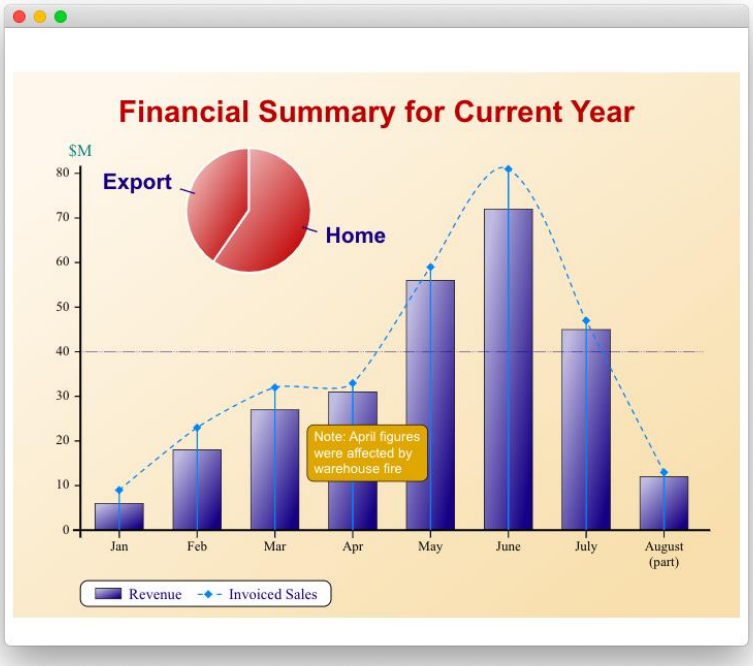
# 2 Simple Examples

## 2.1 Render a SharpPlot chart

SharpPlot charts can be rendered as SVG which in turn can be visually rendered with HTMLRenderer.

```
      )LOAD sharpplot
saved…

      'HR' ⎕WC 'HTMLRenderer' ('Size' 75 50)
      HR.HTML←#.Samples.Sample.RenderSvg #.SvgMode.FixedAspect
```

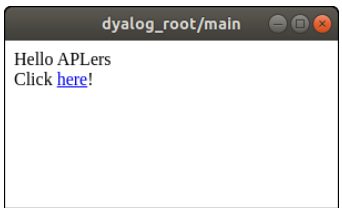


## 2.2 An application with 2 Pages

The function on the next page creates a very simple application with 2 pages: A home page called main and another page called clicked which is displayed if the user follows a link. Initialise the application by calling myapp with an empty right argument; this will cause it to create a namespace containing all the resources, and then create an HTMLRenderer and set the URL property so that it navigates to the first page – and itself as the callback function.

If called with a non-empty argument, the function handles callbacks. It extracts the page name from the URL, which corresponds to a variable in the namespace and returns the value of that variable as the response to the request.

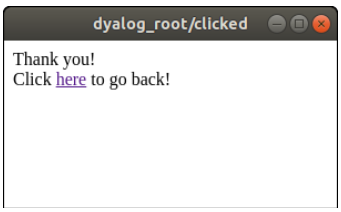
```
     ∇ r←myapp args;root;evt;url;size;coord;obj;op;int;sc;st;mime;hdr;data;meth;page
[1]    ⍝ Serve up a small 2 page application
[2]
[3]    ⍝ If you set the root to something other than http://dyalog_root/
[4]    ⍝  then you need to use InterceptedURLs to indicate APL is to handle the request
[5]    root←'http://dyalog_root/'  ⍝ set the root, requests from CEF will start with this
[6]
[7]    :If 0∈⍴args ⍝ empty args means we're doing Setup
[8]
[9]    ⍝ define the "app" in MyApp, 2 static HTML pages
[10]       #.MyApp←⎕NS''
[11]   ⍝ HTML for the "main" page
[12]       #.MyApp.main←'Hello APLers<br/>Click <a href="clicked">here</a>!'
[13]   ⍝ HTML for the "clicked" page
[14]       #.MyApp.clicked←'Thank you!<br/>Click <a href="main">here</a> to go back!'
[15]   ⍝ whenever we get a request for a resource, call myapp (this function)
[16]       evt←'Event' 'HTTPRequest' 'myapp'
[17]   ⍝ set the initial URL to the "main" page
[18]       url←'URL'(root,'main')
[19]   ⍝ set some window parameters
[20]       size←'Size'(150 300) ⋄ coord←'Coord' 'ScaledPixel'
[21]   ⍝ and off we go...
[22]       'hr'⎕WC'HTMLRenderer'url evt size coord
[23]
[24]   :Else ⍝ handle the HTTPRequest event
[25]
[26]       (obj evt op int sc st mime url hdr data meth)←11↑args
[27]   ⍝ extract the page name
[28]       page←(≢root)↓url
[29]   ⍝ does the page exist?
[30]       :If 2=#.MyApp.⎕NC page
[31]   ⍝ set the HTTP status and text for a successful request
[32]           (sc st)←200 'OK'
[33]   ⍝ set the response data to the new page's HTML
[34]           data←#.MyApp⍎page
[35]       :Else
[36]   ⍝ set the HTTP status and text for a failed (not found)
[37]           (sc st)←404 'Not Found'
[38]           data←'<h2>Page not found!</h2>'
[39]       :EndIf
[40]   ⍝ set the MIME type for the response
[41]       mime←'text/html'
[42]   ⍝ indicate that we've intercepted and handled this request
[43]       int←1
[44]       r←obj evt op int sc st mime url hdr data
[45]   :EndIf
     ∇
```




After running

```
myapp ''
```

clicking the links will toggle between these two pages



## 2.3 A Form with a Button

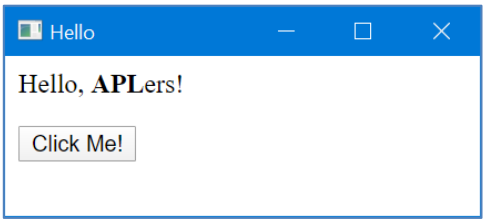
Define a callback function:

```
     ∇ r←my_callback args;obj;evt;op;sc;st;mime;url;hdr;data;int;meth
[1]   ⍝ Our first HTTPRequest callback function
[2]    (obj evt op int sc st mime url hdr data meth)←11↑args
[3]    int←1                              ⍝ indicate we've intercepted this call
[4]    (sc st mime)←200 'OK' 'text/html'  ⍝ HTTP success code
[5]    url←hdr←''                         ⍝ no url or headers
[6]    data←'<title>Thank You!</title><h2>Thank you!<h2>' ⍝ response Data
[7]    r←(obj evt op int sc st mime url hdr data)
     ∇
```

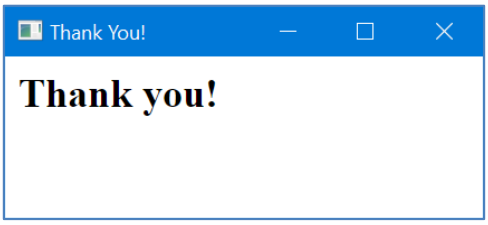
Now, define a form and set up the callback:

```
'hr' ⎕WC 'HTMLRenderer' '<title>Hello</title><p>Hello, <b>APL</b>ers!</p>'
hr.(Coord Size Posn)←'Pixel'(200 450)(20 20)
hr.HTML,←'<form action="#"><button>Click Me!</button></form>'
hr.onHTTPRequest←'my_callback'
```

The form should look like this:




If you click on the button, the content should be replaced:



## 2.4 Using HRUtils with HTMLRenderer

HRUtils is a utility namespace provided with Dyalog APL v18.0 and later. It contains classes and functions to streamline handling of HTMLRender's HTTPRequest events.

Another utility namespace, HttpUtils, which was released with Dyalog v16.0 was designed to provide a more consistent interface for managing HTTP requests whether using Conga or HTMLRenderer. Of the two, we recommend using HRUtils for HTMLRenderer applications.

Both HRUtils and HttpUtls are distributed in the /Library/Conga/ folder in your Dyalog installation and can be loaded using the SALT Load command. Both of the following statements will load HttpUtils, though the latter is suitable for running under program control.

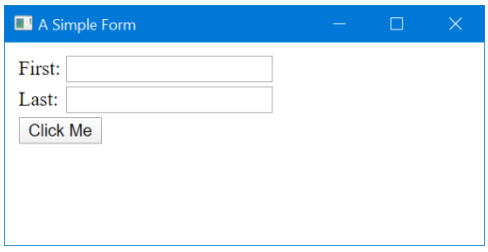
```
]load HRUtils
⎕SE.SALT.Load 'HRUtils'
```

HRUtils is maintained in the Dyalog GitHub repository found at https://github.com/Dyalog/library-conga. There you can see the revision history and you may participate in the development community by reporting issues and by posting questions and suggestions.

The following example shows a simple HTML form with 2 input fields and a submit button. The callback is processed using the HttpRequest class found in HRUtils.

```
    ∇ r←SimpleForm args;evt;html;req;resp;who
[1]   :If 0∈⍴args ⍝ Setup
[2]       html←'<title>A Simple Form</title>'
[3]       html,←'<form method="post" action="SimpleForm"><table>'
[4]       html,←'<tr><td>First: </td><td><input name="first"/></td></tr>'
[5]       html,←'<tr><td>Last: </td><td><input name="last"/></td></tr>'
[6]       html,←'<tr><td colspan="2"><button>Click Me</button></td></tr>'
[7]       html,←'</table></form>'
[8]       evt←'Event' 'HTTPRequest' 'SimpleForm'
[9]       'hr'⎕WC'HTMLRenderer'('HTML'html)('Coord' 'ScaledPixel')('Size'(200 400))evt
[10]  :Else ⍝ handle the callback
[11]      req←#.HRUtils.Request args          ⍝ create a request from the callback args
[12]      who←req.Get¨'first' 'last'          ⍝ retrieve from the form data
[13]      who←∊' ',¨who
[14]      req.Response.Content←'<h2>Welcome',who,'!</h2>' ⍝ set the content for the response
[15]      r←req.Respond ⍝ return the formatted response
[16]  :EndIf
    ∇
```

Running SimpleForm '' displays the form. After filling in the form and clicking the button, SimpleForm is called again as the callback function for the HTTPRequest event, but this time args is non-empty and the callback portion lines [11-15] are executed.




```
[11]      req←#.HttpUtils.Request args   ⍝ create a request from the callback
args
```

The Request function creates an instance of the HttpRequest class from the event message, parsing the message data and extracting the various elements into a more useful and accessible format.

```
[12]      who←req.Get¨'first' 'last'          ⍝ retrieve from the form data
```

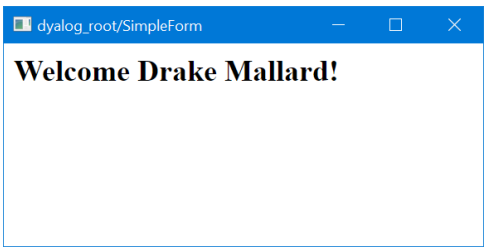
The `HttpRequest` class has extracted the HTML form field values into `FormData` which can be accessed using the `HttpRequest`'s `Get` method  The values are retrievable by their field names in the HTML form, in this case `'first'` and `'last'`. Refer to lines `[4-5]` in `SimpleForm` to see where the field names were originally assigned.

```
[13]      who←∊' ',¨who
[14]      req.Response.Content←'<h2>Welcome',who,'!</h2>' ⍝ set the content for
the response
```

We now set `Content` in the response to be our new content for the page.  The default content type is 'text/html', but other content types can be specified as appropriate for your application.

```
[15]      r←req.Respond ⍝ return the formatted response
```

Finally, the `Respond` method formats and populates a result appropriate for the callback and our friendly message is displayed.



# 3 Generating HTML

Dyalog provides a number of tools to help you generate HTML.

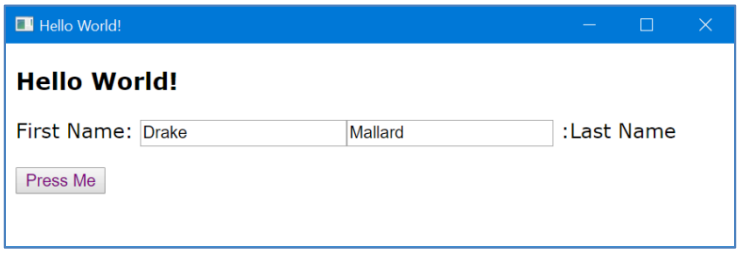
**SharpPlot**

The SVG data produced by the RenderSVG method can be assigned directly to the HTML property of an HTMLRenderer object. The CEF accepts SVG in place of HTML and is able to render it without further intervention. You can also use the various Save… functions in SharpPlot to save graphs in SVG or other formats, and link to them using an HTML `img` tag.

**DUI – Dyalog User Interface Utility Library**

DUI is an evolving library to assist in creating HTML content. Originally a part of MiServer, DUI is designed to enable you to create HTML that can be run locally with HTMLRenderer or on the net with MiServer – without changing your code. DUI contains APL code that is able to generate HTML, CSS and JavaScript based widgets based on the HTML5 widget set, Syncfusion controls (which are bundled with Dyalog), jQueryUI and other third-party widgets. DUI is currently available from the Dyalog GitHub repository at https://github.com/Dyalog/DUI. To use DUI, you will need to download or clone the repository. To illustrate the DUI style of coding, the following code should produce a form with two input fields and a button:

```
]load /path_to_DUI/DUI
DUI.Initialize
page←⎕NEW Page
page.Add _.title 'Hello World!'
page.Add _.Style 'body' ('font-family' 'Verdana')
page.Add _.h3 'Hello World!'
form←page.Add _.Form
'fn' form.Add _.Input 'text' 'Drake' 'First Name: '
'ln' form.Add _.Input 'text' 'Mallard' ' :Last Name' 'right'
p1←'p1' form.Add _.p ''
b1←'b1' form.Add _.Button 'Press Me'
b1.style←'color:purple'
page.Size←200 600
page.Run
```



# 4 Technical Overview

The HTML Renderer is implemented using the Chromium Embedded Framework (CEF); for more information on CEF visit https://en.wikipedia.org/wiki/Chromium_Embedded_Framework.

## 4.1 HTMLRenderer on non-Windows platforms

The HTMLRenderer on non-Microsoft Windows platforms is an X-Windows application. As such there are a set of pre-requisites that are needed on the operating system instance on which the Dyalog interpreter is running (this in X-Windows terms is the server) and a set of pre-requisites that are needed on the operating system instance where the output will be displayed (in X-Windows terms the client). In most cases these two sets of functionality run in the same operating system instance. However, this means that a typical non-GUI installation of a Linux distribution is unlikely to allow you to create an HTMLRenderer object even if you are trying to display it elsewhere by setting the DISPLAY variable appropriately.

For Linux, we have tried creating the HTMLRenderer on a number of common distributions and versions. See https://forums.dyalog.com/viewtopic.php?f=20&t=1505 which details what pre-requisites are needed for the HTMLRenderer on those distributions.

If you get a LIMIT ERROR when attempting to create an HTMLRenderer object and you are either using a distribution/version that is not in the list below, or have ensured that you have met the pre-requisites mentioned below, then run the following expression from within Dyalog APL:

```
)sh ldd $DYALOG/lib/htmlrenderer.so | grep found
```

This should list any missing pre-reqs. Please let Dyalog know so that we can update the supported versions matrix.

## 4.2 CEF/Chromium Command Line Switches

There are very many command line switches that can be used with CEF to alter behavior, help debugging or aid in experimenting. These switches need to be set using the `-cef` or `-cef_all` options when Dyalog APL is started. If you're setting a single CEF command line switch, you can use:

```
-cef --command-line-switch
```

If you're setting more than one CEF command line switch, then you either need to prefix each on with `-cef` or place them at the end the Dyalog command line preceded by `-cef_all` as in:

```
-cef --command-line-switch1 -cef --command-line-switch2
-cef_all --command-line-switch1 --command-line-switch2
```

Note that the command line switches begin with a double dash (`--`).

One common command line switch is to enable a remote debugging port for CEF so that you can attach a browser to CEF and open the Developer Tools Console.

```
-cef --remote-debugging-port=12345
```

Then open a browser and navigate to the address of the computer where HTMLRenderer is running and the port indicated by `--remote-debugging-port`.

A fairly comprensive list of command line switches can be found at https://peter.sh/experiments/chromium-command-line-switches/.

# 5 HTMLRenderer Reference

This section highlights specific aspects the HTMLRenderer. For a complete description of the Properties, Events and Methods for the HTMLRenderer object, please refer to the object reference guide at http://help.dyalog.com/18.0/Content/GUI/Objects/HTMLRenderer.htm.

## 5.1 Properties

As HTMLRenderer is an object in the Dyalog GUI framework, it has many of the expected properties for a ⎕WC GUI control. The properties for HTMLRenderer are found in table 1, with properties specific to HTMLRenderer highlighted in red.

***Table 1**. HTMLRenderer properties*

| | | |
|---|---|---|
| Type | HTML | Posn |
| Size | URL | Coord |
| Border | Visible | Event |
| Sizeable | Moveable | SysMenu |
| MaxButton | MinButton | IconObj |
| Data | Attach | Translate |
| KeepOnClose | AsChild | InterceptedURLs |
| CEFVersion | Caption | AllowContextMenu |
| IsLoading | MethodList | ChildList |
| EventList | PropList | |

## 5.2 Properties Available by Platform

Not all properties are available on all platforms. The table below illustrates the properties that vary by platform. Properties listed as N/A are specific specific to Windows and do not have equivalent counterparts under macOS or Linux.

***Table 2**. HTMLRenderer properties by platform*

| Property | Windows | macOS | Linux |
|---|---|---|---|
| Type | Y | Y | Y |
| HTML | Y | Y | Y |
| Posn | Y | Y | Y |
| Size | Y | Y | Y |
| Coord | Y | Y | Y |
| Border | Only when using AsChild | N/A | N/A |
| Visible | Y | Y | Y |
| Sizeable | Y | Y | N |
| Moveable | N | N | N |

| | | | |
|---|---|---|---|
| SysMenu | Y | N/A | N |
| MaxButton | Y | N | N |
| MinButton | Y | Y | N |
| IconObj | Y | N/A | N/A |
| Attach | Only when using AsChild | N/A | N/A |
| Translate | Classic Only | N/A | N/A |
| KeepOnClose | Y | Y | Y |
| AsChild | Y | N/A | N/A |
| CEFVersion | Y | Y | Y |
| Caption | Y | Y | Y |
| AllowContextMenu | Y | Y | Y |
| IsLoading | Y | Y | Y |

## 5.3 Properties With Behavior Specific to HTMLRenderer

The properties are presented here in an order intended provide context for how certain properties interrelate.

**HTML**

The HTML property is the payload to be sent to CEF from APL. In general, it will be the HTML content that will be rendered in the HTMLRenderer window. But it could also be other content like an image file, a JavaScript file, or a CSS stylesheet. When sending HTML, the data is assumed to be UTF-8 and you can simply assign your character vector to the property. When the content is not UTF-8, like an image, you will need to send the data as single-byte integer (datatype 83).

**URL**

The URL property is a character vector representing the "root" URL of the object. If not specified, 'http://dyalog_root/' is the implied value of URL. If subsequent requests for resources are received via the HTTPRequest event, the URL element of the callback arguments can be examined to see if it begins with the "root". If so, the content is intended to be provided locally by your application, otherwise, it should be retrieved from the URL element of the argument.

**Relationship between the HTML and URL properties**

In general, either the HTML or URL property will be specified, but not both. If URL is non-empty, it will take precedence over HTML, even if the resource specified by the URL is not found. If neither HTML or URL is specified, HTMLRenderer will trigger an HTTPRequest event for the URL `'http://dyalog_root/'`.

**InterceptedURLs**

The InterceptedURLs property is a 2-column matrix what action HTMLRenderer will take on an HTTPRequest or a WebSocketUpgrade request. The first column contains wild-carded character vectors containing URL patterns to match. The second column is an integer where:

- 0 indicates HTMLRenderer should attempt to retrieve the resource over the net.
- 1 indicates HTMLRenderer should trigger an HTTPRequest event for a URL matching the corresponding pattern

- 2 indicates that a WebSocketUpgrade event triggered a matching URL should be manually verified.

`InterceptedURLs` may contain any number of rows and the first matching pattern for a requested URL will determine how the request is routed. URLs matching the pattern `'*://dyalog_root/*'` or that have a 1 in the second column will trigger an HTTPRequest event; all other URLs will be attempted to be retrieved over the net. The default value for `InterceptedURLs` is `0 2⍴'' 0`.

**Examples:**
The following will trigger an HTTPRequest event for all requested URLs

```
InterceptedURLs ← 1 2⍴'*' 1
```

The following will attempt to retrieve from the net URLs containing '.dyalog.com' and trigger an HTTPRequest event for all other requested URLS

```
InterceptedURLs ← 2 2⍴'*.dyalog.com*' 0 '*' 1
```

### AsChild

This property has an effect only on Microsoft Windows platforms.

The `AsChild` property is a Boolean indicating how the HTMLRenderer object should be treated. Possible values are:

- 1 – the HTMLRenderer object should be treated as a child of its parent object.
- 0 – the HTMLRenderer object should be treated as a top level object similar to how a Form object is treated.

The default is 0.

### CEFVersion

Returns version information about the CEF. This is used primarily for support and debugging purposes.

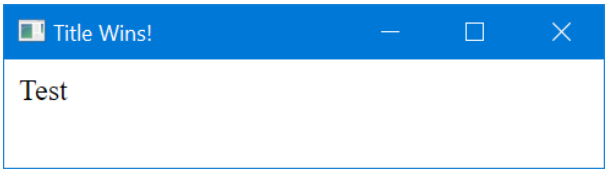
***Table 3**. Elements of* **CEFVersion**

| | |
|---|---|
| [1] | Formatted CEF release number. This is the primary identifier for a version of CEF. |
| [2] | CEF major version |
| [3] | Commit number |
| [4] | Chromium version number |
| [5] | Chromium version number |
| [6] | Chromium version number |
| [7] | Chromium version number |
| [8] | GIT hash |
| [9] | GIT hash |
| [10] | GIT hash |

### Caption

Note that the caption appearing in the title bar of the HTMLRenderer window can be set either with the `Caption` property or by a <title> element within the HTML for the page. If both are set, the <title> element takes priority. For example:

```
html←'<title>Title Wins!</title>Test'
'hr' ⎕WC 'HTMLRenderer'('HTML' html)('Caption' 'Caption Wins!')('Size' (10 20))
```



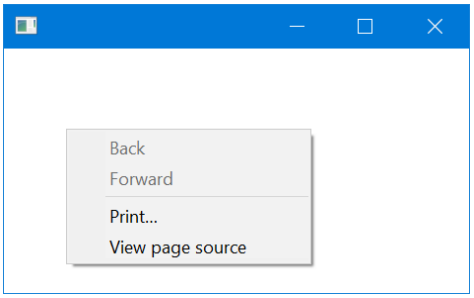

It is recommended that you use <title> to control the caption on the title bar element because changes to the HTML <title> will be reflected in the `Caption` property. However, the converse is not true – changes to the `Caption` property are not reflected in the document's <title> element.

**`AllowContextMenu`**

Controls whether right-clicking will display the context menu. Possible values are:

- 1 – the context menu will be displayed by right-clicking on the HTMLRenderer window as shown below
- 0 – the context menu will not be displayed by right-clicking on the HTMLRenderer window

The default is 1.




**`IsLoading`**

Returns 1 if HTMLRenderer is currently loading content into a frame and 0 otherwise. Note that for pages with multiple frames, **`IsLoading`** only indicates that there is no frame currently loading, not that all frames have been loaded. A **`LoadEnd`** event is signalled whenever a frame has finished loading.

## 5.4 Events

The events for HTMLRenderer are found in table 4, with events specific to HTMLRenderer highlighted in red.

***Table 4**. HTMLRenderer events*

| | | |
|---|---|---|
| `Close` | `Create` | `LoadEnd` |
| `HTTPRequest` | `WebSocketUpgrade` | `WebSocketReceive` |
| `WebSocketClose` | `WebSocketError` | `DoPopup` |
| `SelectCertificate` | | |

## 5.5 Events Specific to HTMLRenderer

**`HTTPRequest`**
An `HTTPRequest` event is raised whenever content is required that is not provided by the HTML property. This could be generated by a form submission, clicking on a hyperlink, an AJAX request or a link to a resource like a stylesheet, image or JavaScript file.

The event message reported as the result of □DQ or supplied as the right argument to your callback function, is a 11-element vector as described in table 5.

**NOTE:** the event message only had 10 elements in version 16.0. Application code should not assume a specific length for this, or indeed any other event messages.

***Table 5**. Explanation of the 11-element vector HTTPRequest event message*

| | |
|---|---|
| `[1]` | `HTMLRenderer object name or reference` |
| `[2]` | `Event name 'HTTPRequest' or 840` |
| `[3]` | `Constant 'ProcessRequest'` |
| `[4]` | `0` |
| `[5]` | `0` |
| `[6]` | `''` |
| `[7]` | `''` |
| `[8]` | `Requested URL` |
| `[9]` | `HTTP Request Headers` |
| `[10]` | `HTTP Request Body` |
| `[11]` | `HTTP Method - Typically this will be 'GET' or 'POST'.` |

When preparing a response, certain elements of the event message need to be updated. Specifically:

| | |
|---|---|
| `[4]` | Set to 1 to update the rendering window based on the updated elements of the event arguments. |
| `[5]` | Set to the HTTP status code for the response. Success is indicated by code 200. |
| `[6]` | Set to the HTTP status message for the response. Success is indicated by the message `'OK'`. |
| `[7]` | Set to the MIME type of the response. If not specified, `'text/html'` is assumed. |
| `[9]` | Set to any HTTP response headers necessary for the response. |
| `[10]` | Set to the body of the response. If the body is not UTF-8 text, convert it to single-byte integer (datatype 83). |

**`WebSocketUpgrade, WebSocketReceive, WebSocketClose, WebSocketEnd`**
Please refer to Section 6, *WebSocket support* for more information.

**`DoPopup`**
A `DoPopup` event is raised whenever the CEF client executes a request for a new window to be opened. This would typically be when a link element **<a>** specifies a target attribute of "_blank" or a framename, or when a name is specified when a new window is opened using the JavaScript **window.open()** method:

```
<a href="http://www.dyalog.com" target="_blank">
<a href="http://www.dyalog.com" target="dyalog_window">
```
**or**
```
window.open("http://www.dyalog.com")
window.open("http:..www.dyalog.com","dyalog_window")
```

When a `DoPopup` event occurs, the application should inspect the request and open another HTMLRenderer as appropriate.

*Elements of the DoPopup event message*

| | |
|---|---|
| [1] | HTMLRenderer object name or reference |
| [2] | Event 'DoPopup' or 846 |
| [3] | The requested URL |
| [4] | 7-element vector of requested window attributes |
| [5] | Character vector framename |

*Attributes vector in element[4] of the DoPopup event message*

| | |
|---|---|
| [1] | 2-element vector of top, left positions<br>positions not specified are 0 |
| [2] | 2-element vector of height,width<br>positions not specified are 0 |
| [3] | Integer "WindowDisposition". See https://magpcss.org/ceforum/apidocs3/projects/(default)/cef_window_open_disposition_t.html |
| [4] | Boolean menubar (default=1) |
| [5] | Boolean scrollbar (default=1) |
| [6] | Boolean statusbar (default=1) |
| [7] | Boolean location/toolbar (default=1) |

The attributes vector contains requested attributes for the new window. These are typically specified as paramters in a JavaScript **window.open()** method. The two attributes of most interest are the position element [1], and size element [2]. You can use these attributes to set the **`posn`** and **`size`** attributes of the new HTMLRenderer window. Note that these parameters are specified in pixels so you will need to convert to if you are using **`'coord'`** **`'prop'`**. HTMLRenderer currently has no mechanism to make use of attribute elements [3-7].

If a framename other than '_blank', '_top', '_self' or '_parent' was specified in client, it is passed as the fifth element of the **DoPopup** callback arguments. This can be used to identify a specific window that is being opened in the case where a page might open multiple windows.

**`LoadEnd`**

A **`LoadEnd`** event is raised when a particular frame has finished loading. Multiple frames may be loading at the same time. Sub-frames may start or continue to load even after the main frame has finished loading. A common technique is to wait for the main frame to finish loading before further interaction with the HTMLRenderer instance. In this case, you should set up an event handler on the **`LoadEnd`** event and check the 4th element which indicates if the loaded frame is the main frame.

*Elements of the DoPopup event message*

| | |
|---|---|
| [1] | HTMLRenderer object name or reference |

| | |
|---|---|
| [2] | Event 'LoadEnd' or 836 |
| [3] | The URL of the loaded frame |
| [4] | 1 if the loaded frame is the "main" frame, 0 otherwise |
| [5] | The HTTP status code as a result of loading the frame |

**SelectCertificate**
A SelectCertificate event is raised whenever a resource is requested from a server that requires a certificate for security. The available certificates are in element [7] of the callback arguments. The application should select one of the certificates and set element [3] to its origin-0 index in the Certificates element.

*Elements of the SelectCertificate event message*

| | |
|---|---|
| [1] | HTMLRenderer object name or reference |
| [2] | Event 'SelectCertificate' or 848 |
| [3] | Certificate index (result only) |
| [4] | Host address |
| [5] | Host port |
| [6] | 'is proxy' |
| [7] | Certificates (see below) |

Certificates is a vector of namespaces, each of which represents a certificate and contains the following variables:

| Name | Description |
|---|---|
| DER | The DER-encoded certificate |
| Subject | A namespace containing variables CommonName, CountryName and DisplayName for the certificate subject. |
| Issuer | A namespace containing variables CommonName, CountryName and DisplayName for the certificate issuer. |
| SerialNumber | Character vector certificate serial number |

## 5.6 Methods

The methods for HTMLRenderer are found in table 6, with events specific to HTMLRenderer highlighted in red.

***Table 6**. HTMLRenderer methods*

| | | |
|---|---|---|
| Detach | PrintToPDF | WebSocketSend |
| WebSocketClose | ShowDevTools | Wait |
| ExecuteJavaScript | GetZoomLevel | SetZoomLevel |

## 5.7 Methods Specific to HTMLRenderer

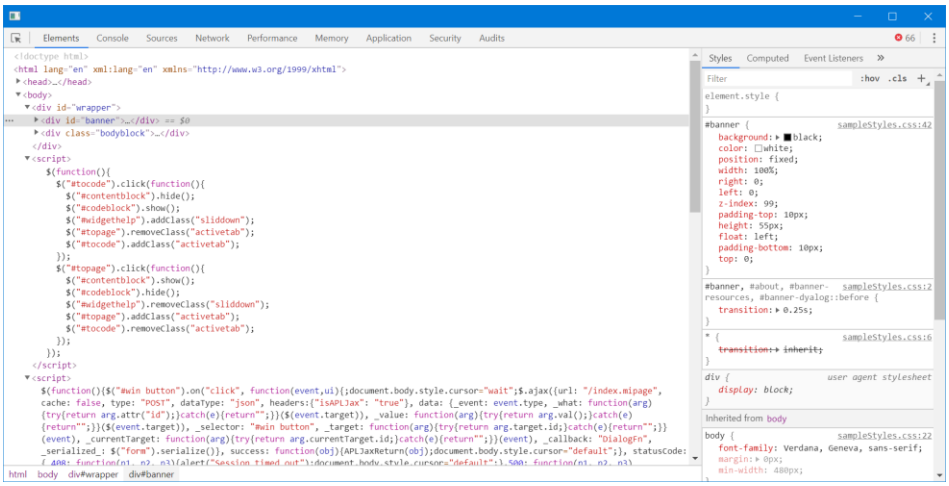
**WebSocketSend, WebSocketClose**
Please refer to section 6, WebSocket Support, later in this document.

### ShowDevTools

The `ShowDevTools` method is used to open or close the Chromium Developer Tools console. Its only argument is a Boolean as in:

```
'hr' ⎕WC 'HTMLRenderer'
hr.ShowDevTools 1  ⍝ open developer tools
hr.ShowDevTools 0  ⍝ close developer tools
```

Chromium Developer Tools is a very useful debugging and inspection utility.



### ExecuteJavaScript

The `ExecuteJavaScript` method is used send a character vector containing JavaScript code to CEF to be executed. At present this is a one-way communication and the shy result is always 1. No assumptions should be made about the result.

**Example:**

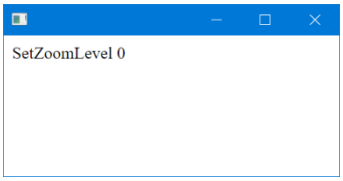
```
'hr' ⎕WC 'HTMLRenderer'
hr.ExecuteJavaScript 'alert("Hello")'
```

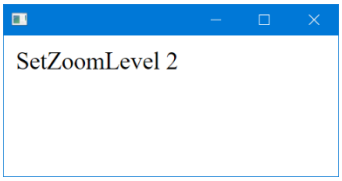
### GetZoomLevel

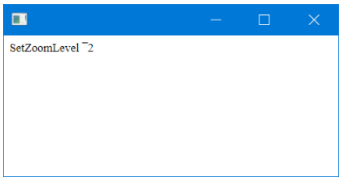
Returns the current CEF ZoomLevel. The default is 0.

### SetZoomLevel

Sets the CEF ZoomLevel. The default (unzoomed) level is 0. Setting a positive value will increase the zoom, whereas setting a negative will decrease the zoom. The zoom scale is not linear – increasing ZoomLevel from 1 to 2 will not result in a doubling effect. Rather the effective scaling is approximately **`1.2*level`** – so, setting the ZoomLevel to 1 will result in an approximate 20% size increase. ZoomLevel affects all instances of HTMLRenderer windows; it is not possible to have different ZoomLevels for individual windows. The images below show the relative sizes at ZoomLevels of ¯2, 0, and 2.




SetZoomLevel ¯2

# 6 WebSocket support

## 6.1 WebSocket Overview

In a typical HTTP application, all communication originates from the client which sends requests to the server which in turn sends back a response. When an application wanted to "push" information from the server to the client, the typical way to fake this was to have the client periodically poll the server so the server could send back any information that it had to offer. With the use of websockets, true asynchronous, bi-directional transmission between the client and server is possible.

HTMLRenderer presents a straightforward API to use websockets. A typical scenario would look something like this:

1. The client initiates an HTTP "upgrade" request to the server. After some validation and handshaking with the server, the websocket is established. With HTMLRenderer, the validation and handshaking are currently done behind the scenes and by the time you receive a WebSocketUpgrade event, the websocket already established.

2. Once the websocket is established, either the client or the server can send information which will trigger a "receive" event on the other end. No response is expected as a part of the websocket protocol. Whatever response you send (or don't) is up to your specific application.

3. Either side can close the websocket.

4. WebSocket error events may be triggered when an unexpected error, like disruption in the connection, occurs.

| **JavaScript in the CEF client** | | **HTMLRenderer in the workspace** |
|---|---|---|
| ws = new websocket(url);<br>Initiate the request | → | WebSocketUpgrade event<br>The websocket is established |
| ws.send("message"); | → | WebSocketReceive event |
| ws.onmessage event | ← | WebSocketSend method |
| ws.close() | → | WebSocketClose event |
| ws.onclose event | ← | WebSocketClose method |
| ws.onerror event<br>is triggered when there is some error like the connection going down | | WebSocketError event<br>occurs when there is some error like the connection going down |

The client may request multiple upgrades resulting in multiple websockets, each with its own unique id.

WebSockets require JavaScript in the client to function.

## 6.2 WebSocket Events

**`WebSocketUpgrade`**

This event is triggered when the client attempts to upgrade the HTTP connection to use the WebSocket protocol. The event message is a 6-element vector containing:

*Elements of the WebSocketUpgrade event message*

| | |
|---|---|
| [1] | HTMLRenderer object name or reference |
| [2] | Event 'WebSocketUpgrade' or 841 |
| [3] | Character vector WebSocket ID |
| [4] | Requested URL |
| [5] | Request/Response headers |
| [6] | 'auto' or 'manual' depending on InterceptedURLs setting of 1 or 2 respectively for the pattern that matched the URL in element [4] |

The protocol for establishing the connection is defined by InterceptedURLs and is reported by the 6th element of the event message.

If the element matches `'auto'`, the handshake is handled internally and this event is reported when the connection has already been made. In this case the result, if any, of the callback function is ignored.

If the element matches `'manual'`, a callback function for `WebSocketUpgrade` is mandatory and is responsible for completing (or denying) the connection. This is achieved by setting the 5th element (headers) of the event message to indicate an appropriate positive or negative response to the request and returning the entire event message as its result. Each header must be followed by CRLF. If a valid response is not generated in this way, the connection will time-out causing a `WebSocketError` event.

The WebSocket ID is used when sending data to the client using the `WebSocketSend` method or when closing the WebSocket with the `WebSocketClose` method.

**`WebSocketReceive`**

This event is triggered when the client sends data over the WebSocket. The result, if any, of the callback function is ignored. `ShowDevTools` method is used to open or close the Chromium Developer Tools console. Its only argument is a Boolean as in:

*Elements of the WebSocketReceive event message*

| | |
|---|---|
| [1] | HTMLRenderer object name or reference |
| [2] | Event 'WebSocketReceive' or 842 |
| [3] | Character vector WebSocket ID |
| [4] | Data from the client |
| [5] | Boolean - 1 indicates the entire message has been received, 0 indicates there is more data to follow. |
| [6] | Datatype - 1 indicates character (UTF-8), 2 indicates numeric in the range ¯128-127 |

**WebSocketError**
This event is triggered when an error occurs on the WebSocket. The result, if any, from the callback function is ignored.

*Elements of the WebSocketError event message*

| | |
|---|---|
| [1] | HTMLRenderer object name or reference |
| [2] | Event 'WebSocketError' or 844 |
| [3] | Character vector WebSocket ID |
| [4] | Character vector error message |

**WebSocketClose**
This event is triggered when the client closes the WebSocket. The result, if any, from the callback function is ignored.

*Elements of the WebSocketClose event message*

| | |
|---|---|
| [1] | HTMLRenderer object name or reference |
| [2] | Event 'WebSocketClose' or 843 |
| [3] | Character vector WebSocket ID |
| [4] | Integer status code. 1000 indicates normal closure |
| [5] | Character vector reason |

## 6.3 WebSocket Methods

**WebSocketSend (Method 847)**
Use this method to send data to the client over the WebSocket. The argument can be a 2, 3 or 4-element vector.

*Elements of the WebSocketUpgrade method argument*

| | |
|---|---|
| [1] | Character vector WebSocket ID |
| [2] | Data to send - either character (UTF-8) or integer vector |
| [3] | FIN flag - 1 indicates the message is complete, 0 indicates there is more data to come. This is not currently supported by CEF and should always be 1. |
| [4] | Datatype - 1 for character (UTF-8) data, 2 for binary (numeric) data in the range ¯128-255 which maps to 128-255,0-255 in the client, or 0 to indicate continuation of previous message fragment. |

The WebSocket protocol provides for messages to be sent in multiple fragments where the FIN flag is set to 0 for all but the last fragment. Currently CEF does not support fragmented messaging, but we include the FIN flag for possible future expansion.

The integer datatype range may seem a bit strange. It was implemented this way so that the user could conserve space by using single-byte integers (datatype 83) in the range ¯128-127, whereas some might find it more convenient to use the range 0-255. In either case both ranges translate to value 0-255 in the client.

**`WebSocketClose (Event 843)`**
This event is used to close the WebSocket from APL triggered when the client closes the WebSocket.

*Elements of the WebSocketClose method arguments*

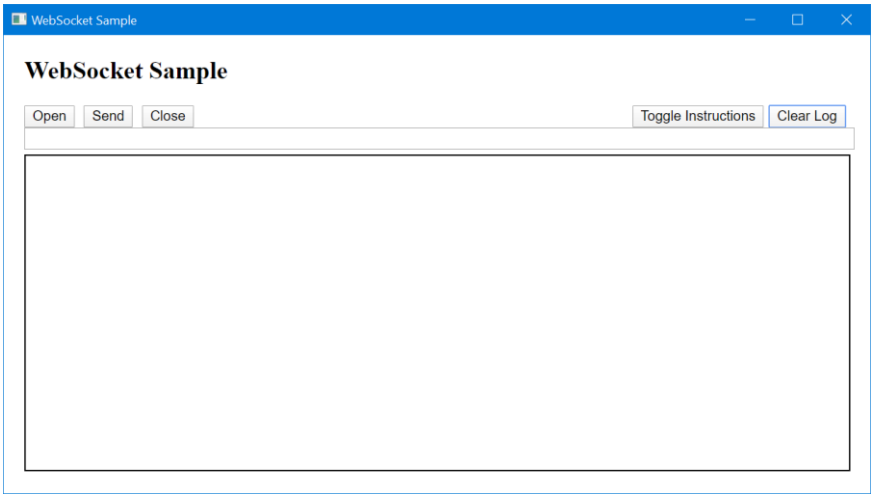
| | |
|---|---|
| `[1]` | `Character vector WebSocket ID` |
| `[2]` | `Integer status code. 1000 indicates normal closure` |
| `[3]` | `Character vector reason (max length 123 bytes)` |

## 6.4 WebSocketSample function

The code necessary to demostrate HTMLRenderer's WebSockets is larger than can be presented conveniently here. We have provided a sample function, WebSocketSample.dyalog found in the Dyalog Samples repository at https://raw.githubusercontent.com/Dyalog/Samples/master/HTMLRenderer/WebSocketSample.dyalog.

Once you've loaded WebSocketSample, you can start it by running

```
WebSocketSample ''
```




From there you can click "Toggle Instructions" to see instructions.

Clicking "Open" will create a WebSocket. Once a WebSocket is created, you can enter text in the input area and click "Send". When you create a WebSocket using "Open", WebSocketSample defines a `Send` function in your workspace that you can then use to send data from APL to CEF as in

```
Send 'Hello from APL!'
```

The event message elements for every handled WebSocket event are displayed in your APL session. Every event handled by CEF is displayed in the output (log) area of the page.

# 7 HRUtils

## 7.1 Overview

HRUtils is a namespace that contains utilities for working with HTMLRenderer's HTTPRequest event. The steps to use HRUtils are as follows:

Within your callback function:

1. Initialize
   When you receive an HTTPRequest event, call `HRUtils.Request`, passing the 11-element event message as its right argument
   ```
   req←#.HRUtils.Request evtMsg
   ```
   This creates an instance of a class called `HttpRequest` which in turn parses the elements of the event message and creates several conveniently accessible fields.

2. Process
   Examine/use the data in the fields of `req` to process the request as appropriate.
   For example, use the `Get` method to retrieve data elements passed in the request:
   ```
   who←req.Get¨'fname' 'lname' ⍝ retrive 2 fields
   ```
   Or check the `Uri` field to see what resource is being requested.

3. Compose
   If the requested resource is a file, you can set the `FileName` field of the response:
   ```
   req.Response.FileName←'c:/images/duck.png'
   'content-type' req.SetHeader 'image/png'
   ```
   Or, if appropriate, compose the HTML for your response:
   ```
   req.Response.Content←'Hi',⊂' ',¨who
   ```

4. Respond
   Then return the result of `req.Respond` as the result of your callback function.
   ```
   r←req.Respond
   ```
   HRUtils will take care of properly formatting and assigning the appropriate elements of the response.

## 7.2 HRUtils.`HttpRequest` class

The fundamental unit of work for HRUtils is the `HttpRequest`.

*HttpRequest Fields*

| | |
|---|---|
| `Method` | `The HTTP method for the request (generally 'get' or 'post')` |
| `Uri` | `The URI (URL) for the requested resource` |
| `Headers` | `2-column matrix of [;1] header names, [;2] header values` |
| `QueryData` | `2-column matrix of [;1] names, [;2] values of parameters (if any) passed in the query string of the request` |

| | |
|---|---|
| FormData | 2-column matrix of [;1] names, [;2] values of form fields either passed in the body of the request, or in the case of a 'get' Method, this is a copy of QueryData |
| Cookies | 2-column matrix of [;1] names, [;2] values of cookies passed with the request |
| Response | An instance of the HttpResponse class created using elements of the event message |
| HTMLRendererArgs | A namespace containing the original event message elements |

*HttpRequest Methods*

| | |
|---|---|
| GetHeader | value←req.GetHeader 'name'<br>return the value of the header named 'name' or '' if such a header doesn't exist. |
| Get | value←table req.Get 'name'<br>return the corresponding value for 'name' or '' if name does not exist.<br>table can be any of Headers, QueryData, FormData, or Cookies |
| Respond | callbackResult←req.Respond<br>convert the Response instance into a format acceptable for the result of the HTTPRequent event callback |

## 7.3 HRUtils.HttpResponse class

The HttpResponse contains what is sent back to CEF in response to the request.

*HttpResponse Fields*

| | |
|---|---|
| HttpStatus | The integer HTTP status for the response<br>200 is success |
| HttpStatusText | The character HTTP status message for the response<br>'OK' is success |
| HTMLRenderArgs | A namespace containing the original event message elements (same as HttpRequest) |
| FileName | If the requested resource is a file, assign FileName to the actual file name. |
| Content | If the requested resource is not a file, compose your response data and assign it to Content. |
| Headers | 2-column matrix of [;1] names, [;2] values of cookies to be sent with the response |
| Cookies | A vector of instances (if any) of the Cookie class representing cookies that are to be set in the client. |

*HttpResponse Methods*

| | |
|---|---|
| AddHeader | 'name' req.Response.AddGetHeader 'value'<br>Add a name/value pair to the response headers unless a header of the same name already exists. |

| | |
|---|---|
| AddCookie | req.Response.AddCookie arg<br>arg is either<br>- A character vector representing a formatted cookie string<br>- A 2-7 element vector of<br>Name - cookie name<br>Value - cookie value<br>Expires - cookie expiration datetime<br>Domain - hosts allowed to see the cookie<br>Path - path that must exist for cookie to be sent<br>Secure - cookie may only be sent using HTTPS<br>HttpOnly - cookie cannot be read by JavaScript |
| Respond | callbackResult←req.Response.Respond<br>convert the Response instance into a format acceptable for the result of the HTTPRequent event callback<br>req.Respond is essentially the same thing, but more convenient (less typing) to call |

## 7.4 HRUtils.Cookie class

The fundamental unit of work for HRUtils is the HttpRequest.
See https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies for more information on cookies

*Cookie Fields*

| | |
|---|---|
| Name | The name of the cookie (required) |
| Value | The value of the cookie (required) |
| Expires | The ⎕TS format expiration (optional) |
| Domain | Hosts allowed to see this cookie. (optional) |
| Path | Path that must exist for this cookie to be sent. (optional) |
| Secure | Boolean indicating this cookie may only be sent using HTTPs (optional) |
| HttpOnly | Boolean indicating that this cookie cannot be read by JavaScript. (optional) |

## 7.5 HRUtils Utility Functions

There are several utility functions HRUtils to aid in parsing and formatting response data.

```
r←{cpo} Base64Decode vec     - decode a Base64 encoded string
r←{cpo} Base64Encode vec     - Base64 encode a character vector, or
                                 an integer (⎕DR=83) vector
```

cpo - optional left argument (for code points only), is useful for encoding raw data like images.

Both Base64Decode and Base64Encode assume that the data is UTF-8. (setting cpo defeats this) This is useful for exchanging APL code and foreign characters.

**Examples:**

```
Base64Encode 'αpl'              ⍝ use default UTF-8
1 Base64Encode ⎕NREAD ¯1 83 ¯1 ⍝ where a .png file is tied to ¯1
```

```
   r←UrlDecode vec        - decodes a URL-encoded character vector
   r←{name} UrlEncode arg - URL-encodes string(s)
```

`arg` can be one of
- a simple character vector (name may be supplied as left argument)
- a vector of character vectors of name/value pairs
- a 2-column matrix of name/value pairs
- a namespace containing named variables

`name` - optional left argument name

Both UrlDecode and UrlEncode assume that the data is UTF-8

**Examples:**

```
      UrlEncode 'Hello World!'
Hello%20World%21

      UrlEncode 'phrase' 'Hello World!'
phrase=Hello%20World%21

      UrlEncode 'company' 'dyalog' 'language' 'APL'
company=dyalog&language=APL

      UrlEncode 2 2⍴'company' 'dyalog' 'language' 'APL'
company=dyalog&language=APL
```

# 8 Running HTMLRenderer under a Windows Runtime Application

To run HTMLRenderer under a Windows runtime interpreter (dyalogrt.exe) you should:

1. Create your runtime environment as described in the Dyalog for Microsoft Windows Installation and Configuration Guide
2. Copy the following items from the Dyalog installation folder into the same folder as the dyalogrt.exe:

**Files:**
chrome_100_percent.pak
chrome_200_percent.pak
chrome_elf.dll
d3dcompiler_47.dll
htmlrenderer.dll
icudtl.dat
libcef.dll
libEGL.dll
libGLESv2.dll
resources.pak
snapshot_blob.bin
v8_context_snapshot.bin

**Folders:**
locales
swiftshader

# 9 Resources and References

The Dyalog webinar "*Something Old, Something New & Something Experimental*" includes a discussion and demonstration of the HTMLRenderer; it can be viewed at https://dyalog.tv/webinar.

Code samples can be copied-and-pasted from an HTML version of this document at http://docs.dyalog.com/19.0/HTMLRenderer User Guide.htm.

# 10 Change History

This section details the changes made to HTMLRenderer by release of Dyalog APL.

## Version 19.0

This version provides:

- A new property, **`IsLoading`**, which returns **`1`** while page content is loading and 0 if no loading if presently taking place.
- A new method, **`LoadEnd`**, which is triggered when the page content is completely loaded.
- Improved stability and bug fixes.

## Version 18.2

This version provides:

- A new fifth element in the callback argument for the **`DoPopup`** event which is the name of the window, if any, assigned by the client code.
- Two new methods, **`SetZoomLevel`** and **`GetZoomLevel`** used to respectively set and query the
- Improved stability and bug fixes.

## Version 18.0

This version provides:

- More convenient UTF-8 support. The HTML property can now contain Unicode code points greater than 127 without additional translation or formatting. However, this is a potentially **breaking change** for applications written using earlier versions of HTMLRenderer. See UTF-8 Support.
- Improved WebSocket support.
- An **`ExecuteJavaScript`** method which allows you to send JavaScript statements to be executed in the rendering window.
- The default behavior for the **`InterceptedURLs`** property has been changed such that, in many cases, it will be unnecessary to set **`InterceptedURLs`**.

## Version 17.1

This version provides:

- Support for websockets, allowing asynchronous, bi-directional communication between the APL session and the CEF client window.
- A **`DoPopup`** event that is triggered when the CEF client issues a request for a new window.
- A **`SelectCertificate`** event that is triggered when the CEF client issues a request for a resource that requires a certificate.

- A **ShowDevTools** method that will toggle the visibility of the Chromium Developer Tools to inspect and debug from the CEF client.
- Support for several WC properties including Caption, SysMenu, MinButton, MaxButton, Sizeable and Moveable. Some properties may not be available on a particular platform because that platform does not have underlying support for the property; setting such a property will have no effect, nor will it cause an error.