DYALOG

The tool of thought for software solutions

Dyalog APL

Experimental Functionality

HTML Renderer

Version 16.0

Dyalog Limited

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

tel: +44(0)1256 830030

fax: +44 (0)1256 830031
email: support@dyalog.com
http://www.dyalog.com

Dyalog Version 16.0

Revision: 20170628_160

Contents

1 Introduction

This document is not yet complete – updates will be made available at http://www.dyalog.com/documentation-16-0.htm.

Dyalog Webinar #1 includes a discussion and demonstration of the HTMLRenderer; it can be viewed at https://video.dyalog.com/Webinar.

Dyalog 16.0 introduces a new object, HTMLRenderer, which is a cross-platform mechanism for producing Graphical User Interfaces (GUI), based on HyperText Markup Language (HTML). Our plan is to support the HTMLRenderer under Microsoft Windows, Apple macOS and Linux – including the Raspberry Pi – and that applications build using the HTMLRenderer will work in exactly the same way on all platforms.

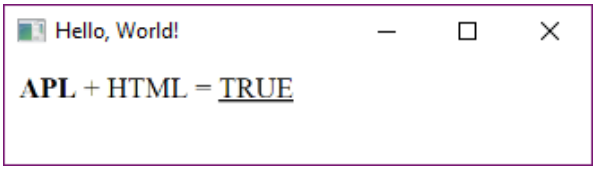
HTMLRenderer is a built-in class, instances of which are created using the Dyalog GUI framework functions ⎕WC/⎕WS/⎕WG/⎕NEW and ⎕DQ/⎕NQ. User interfaces are defined using HTML, which can, in turn, make references to data in a number of additional formats such as JavaScript to manage highly interactive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

1.1 Hello World

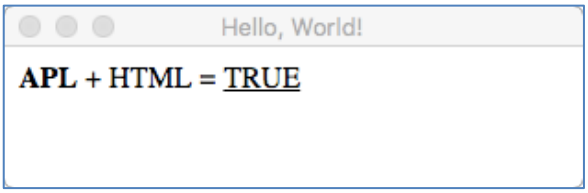
A minimal example of using an HTML renderer would be the following: The first line of code below creates an HTML header which includes a title tag – this will set the caption for the form that contains the HTML renderer. The second line defines the HTML body, and uses simple HTML tags to mark text up as bold, italic and underlined. Finally, an HTMLRenderer is created, using the header and body as the HTML and setting the size property as well:

```
head←'<head><title>Hello, World!</title><head>'
body←'<body><b>APL</b> + </i>HTML</i> = <u>TRUE</u></body>'
'hr'⎕WC'HTMLRenderer'(head,body)('Size' (10 20))
```

Under Microsoft Windows, the result will be:



And under macOS:



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), passing the HTML to the CEF for rendering.

1.2 Other Resources

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

If the HTML contains references to other documents, the CEF will retrieve each one by making an HTTP request. Each request will generate an `HTTPRequest` event on the instance of `HTMLRenderer`, which can be directed to a call-back function in APL. Thus, the APL application can inspect each HTTP request and decide whether to provide the data itself, or to decline and allow the CEF to push the request out to the network and see whether an external server is able to service it. This decision is typically based upon an inspection of the leading part of the URL; the application should only service requests for "internal" data and allow external requests to pass through.

This approach allows an APL application to decide how much content it wants to provide, and to what extend it wants to act as a portal for other services that will provide the rest of the data.

1.3 User Events

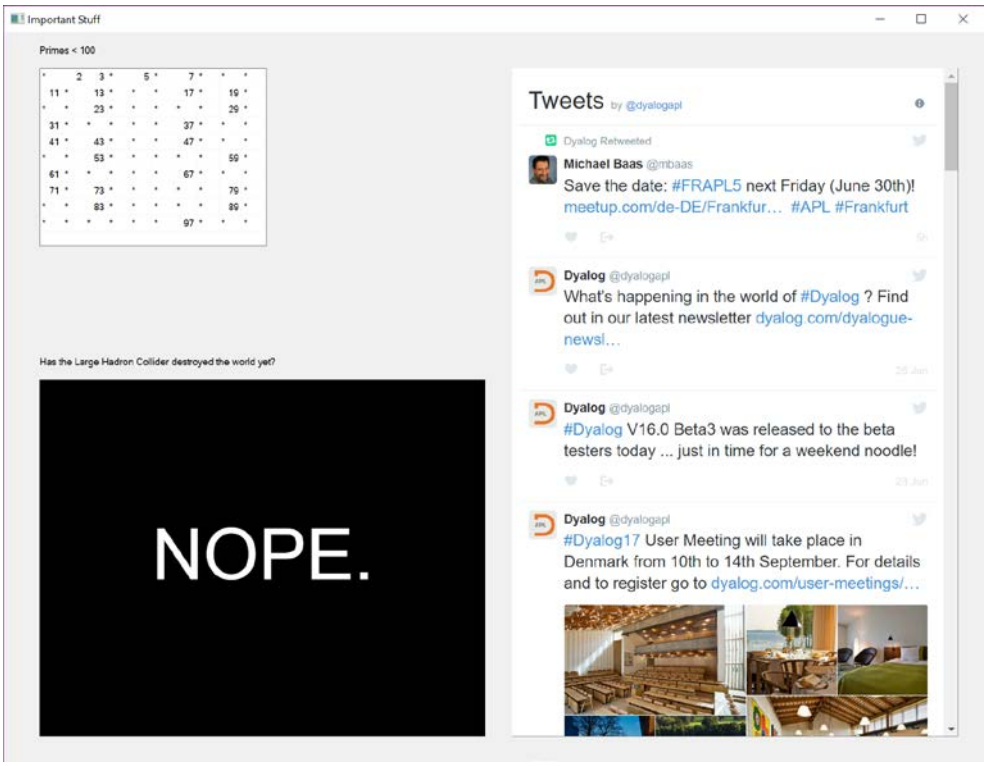
When a user submits an HTML form for processing, or a user interface component which is being managed by JavaScript code wishes to make a server request, this is also done by making an HTTP Request. These requests will also be directed to the same application call-back function. This makes it possible 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 An Example of a Portal

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 window be embedded as a subform of another window.

```
'f1'⎕WC'Form' 'Important Stuff' ('Coord' 'ScaledPixel')('Size' 820 1100)
)copy dfns pco
'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="twitter-timeline" 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 can be seen on the next page; 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.



1.1 Current Status

At the time of the first release of 16.0, only the Microsoft Windows version has seen significant testing. The macOS version is still very fresh and the Linux version is not expected to become available until the end of July. The HTMLRenderer should be viewed as experimental functionality: although it is unlikely that it will change significantly, Dyalog may decide to make adjustments to the design based on feedback from early adopters.

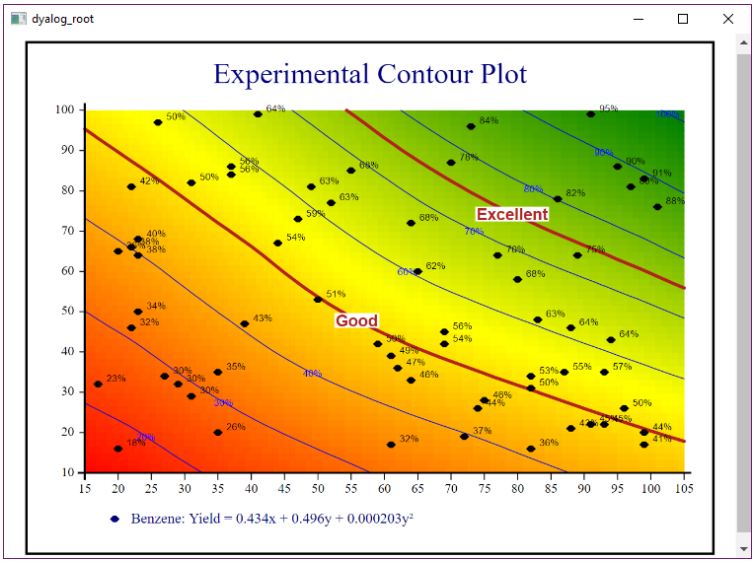
2 Simple Examples

2.1 Render a SharpPlot chart

```
      )LOAD SharpPlot
saved…

      'HR' ⎕WC 'HTMLRenderer'

      HR.HTML←#.Samples.Contour.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 verifies whether the request is for a page within its own domain and that a variable by that name exists; if all is well it returns the value of that variable as the response to the request.

```
      ∇ r←myapp args;event;obj;operation;intercept;scode;stext
                ;mime;url;header;postdata;approot;page;requested
[1]   ⍝ Serve up a small application
[2]
[3]    approot←'http://myapp/'
[4]
[5]    :If 0=≢args⍝ Setup
[6]        #.MyApp←⎕NS''
[7]        #.MyApp.main←'Hello APL''ers<br/>Click <a href="clicked">here</a>!'
[8]        #.MyApp.clicked←'Thank You!<br/>Click <a href="main">
                                                    to go back...</a>!'
[9]        event←'Event' 'HTTPRequest' 'myapp'
[10]       'hr'⎕WC'HTMLrenderer'('Size' 90 90)('Coord' 'ScaledPixel')event
[11]       hr.URL←approot,'main' ⍝ Off we go!
[12]       :Return
[13]   :EndIf
[14]
[15]   (obj event operation intercept scode stext mime url header postdata)←args
[16]   requested←url ⍝ remember this
[17]
[18]   :If intercept←approot≡(≢approot)↑url ⍝ is the URL in our domain?
[19]       page←(≢approot)↓url
[20]       :If 2=MyApp.⎕NC page             ⍝ Do we have a variable with the name
of the requested page?
[21]           (scode stext mime)←200 'OK' 'text/html' ⍝ Yes
[22]           postdata←MyApp⍎page          ⍝ Return the value of the variable as
output
[23]       :Else
[24]           (scode stext)←404 'Page not found'  ⍝ Aww shucks
[25]           postdata←'Page not found'
[26]       :EndIf
[27]       url←''                           ⍝ Not doing anything clever
[28]       header←'Cache-control: no-cache' ⍝ Always refresh these pages
[29]       r←(obj event operation intercept scode stext mime url header postdata)
[30]   :EndIf
[31]
[32]   ⎕←((1+intercept)⊃'[pass]'(6↑⍕scode)),' ',requested
      ∇
```

2.3 A Form with a Button

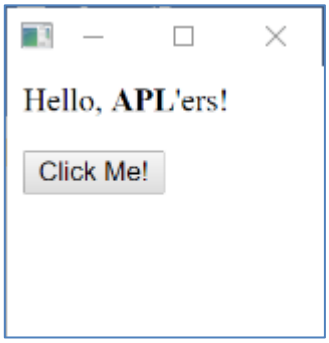
Define a callback function:

```
    ∇ r←my_first_callback args;obj;event;operation;scode;stext;
                              mime;url;header;postdata;intercept
[1]   ⍝ Our first HTTPRequest callback function
[2]
[3]    (obj event operation intercept scode stext mime url header postdata)←args
[4]
[5]    intercept←1                          ⍝ Intercept this call
[6]    (scode stext mime)←200 'OK' 'text/html' ⍝ HTTP success code
[7]    url←header←''                        ⍝ Nothing clever at all
[8]    postdata←'Thank you!'                ⍝ Data
[9]
[10]   r←(obj event operation intercept scode stext mime url header postdata)
    ∇
```

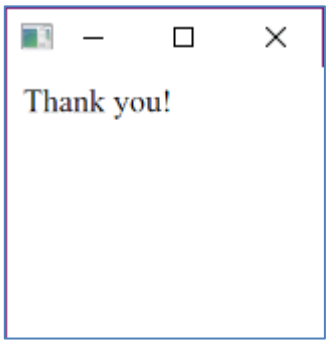
Now, define a form and set up the callback:

```
'hr' ⎕WC 'HTMLrenderer' '<p>Hello, <b>APL</b>''ers!</p>'
hr.(Coord Size Posn)←'Pixel'(300 300)(20 20)
hr.HTML,←'<form action="#"><button>Click Me!</button></form>'
hr.onHTTPRequest←'my_first_callback'
```

The form should look like this:



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



Note that, if your HTML references other resources such as CSS or JavaScript files, images etc., each one will cause a callback.

2.4 Using `HttpUtils` with HTMLRenderer

`HttpUtils` is a utility namespace provided with Dyalog APL v16.0. It contains classes and functions for processing and formatting HTTP request and response messages. `HttpUtils` is designed to work with HTMLRenderer and with Conga[1] 'HTTP' mode.

`HttpUtils` is 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 HttpUtils
⎕SE.SALT.Load 'HttpUtils'
```

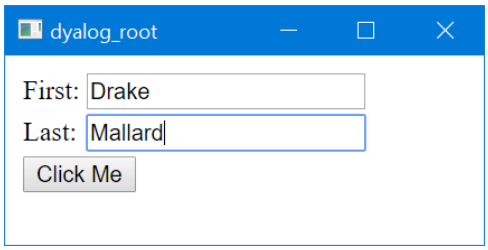
`HttpUtils` is maintained in the library-conga Dyalog GitHub repository found at https://github.com/Dyalog/library-conga. There you may see the revision history and participate in the development community by reporting issues and by posting questions and suggestions.

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

[1] Conga is the Dyalog TCP/IP utility library – HTTP mode was introduced in Conga version 3.0

```
     ∇ r←SimpleForm args;evt;html;req;resp;who
[1]    :If 0∈⍴args ⍝ setup
[2]        html←'<form method="post" action="SimpleForm"><table>'
[3]        html,←'<tr><td>First: </td><td><input name="first"/></td></tr>'
[4]        html,←'<tr><td>Last: </td><td><input name="last"/></td></tr>'
[5]        html,←'<tr><td colspan="2"><button type="submit">Click Me</button></td></tr>'
[6]        html,←'</table></form>'
[7]        evt←'Event' 'HTTPRequest' 'SimpleForm'
[8]        'hr'⎕WC'HTMLRenderer'('HTML'html)('Coord' 'ScaledPixel')('Size' 400 400)evt
[9]        :Return
[10]   :Else      ⍝ handle the callback
[11]       req←⎕NEW #.HttpUtils.HttpRequest args   ⍝ create a request from the callback data
[12]       resp←⎕NEW #.HttpUtils.HttpResponse args ⍝ create a response based on the request
[13]       who←req.(FormData∘Get)¨'first' 'last'   ⍝ req.FormData has the data from the form
[14]       who←∊' ',¨who
[15]       resp.Content←'<h2>Welcome',who,'!</h2>' ⍝ set the content for the response page
[16]       r←resp.ToHtmlRenderer                   ⍝ and send it back
[17]   :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 [11-16] are executed.




```
[11]       req←⎕NEW #.HttpUtils.HttpRequest args   ⍝ create a request from the callback data
```

The `HttpRequest` constructor accepts an argument of HTMLRenderer callback data and will parse and extract the various bits of the HTTP message into a more useful and accessible format.

```
[12]       resp←⎕NEW #.HttpUtils.HttpResponse args ⍝ create a response based on the request
```

We create a response object to send back to HTMLRenderer. Like `HttpRequest`, the `HttpResponse` constructor also accepts an argument of the HTMLRenderer callback data.

```
[13]       who←req.(FormData∘Get)¨'first' 'last'   ⍝ req.FormData has the data from the form
```

The `HttpRequest` class has extracted the HTML form field values into `FormData`. The values are retrievable by their field names in the HTML form, in this case 'first' and 'last'. Refer to lines [2-3] in `SimpleForm` to see where the field names were originally assigned.

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

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.

```
[16]       r←resp.ToHtmlRenderer                   ⍝ and send it back
```

Finally, the response's `ToHtmlRenderer` method formats and populates a result appropriate for the callback and our friendly message is displayed.


dyalog_root/Simple...
Welcome Drake Mallard!

3 Generating HTML

To use the HTMLRenderer, you either need to be able to produce HTML and associated documents, or allow HTTP requests to pass through, as demonstrated in the example in chapter 1.

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.

MiServer HTML Engine

MiServer is Dyalog's Web Server framework. It 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 a few other third-party widgets. Dyalog is working on extracting the HTML generation code from MiServer, with a goal of providing tools to create HTML-based UI, and also process the callbacks generated by the widgets. To illustrate the style of coding that we expect to enable, the following code should produce a form with two input fields and a button:

```
page←⎕NEW HtmlRenderer
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' 'Morten' 'First Name: '
'ln' form.Add _.Input 'text' 'Kromberg' ' :Last Name' 'right'
p1←'p1' form.Add _.p ''
b1←'b1' form.Add _.Button 'Press Me'
b1.style←'color:purple'
```

We will make announcements when this code is ready for testing; contact Dyalog if you would like to participate in the design and testing of these components.

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.

5 Dyalog GUI Implementation for HTMLRenderer

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	Data
Attach	Translate	KeepOnClose
AsChild	MethodList	ChildList
EventList	PropList	

5.2 Properties Specific to HTMLRenderer

HTML

The HTML property is a character vector of the content rendered in the object. The interpreter does not perform any preprocessing of the text. As such, it must be properly formed HTML using single-byte (⎕DR 80) character data, including any necessary escaping and encoding.

URL

The URL property is a character vector representing the "root" URL of the object. If not specified, 'dyalog_root' is the default value of URL. If subsequent requests for resources are received via the HTTPRequest event, the URL element of the event's 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.

AsChild

This property only has an effect 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.

5.3 Events

The events for HTMLRenderer are found in table 2, with events specific to HTMLRenderer highlighted in red.

***Table 2**. HTMLRenderer events*

`Close`	`Create`	`HTTPRequest`

5.4 Events Specific to HTMLRenderer

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 10 element vector as described in table 3.

***Table 3**. Explanation of the 10-element vector HTTPRequest event message*

`[1]`	`HTMLRenderer object name or reference`
`[2]`	`Event name 'HTTPRequest'`
`[3]`	`Constant 'ProcessRequest'`
`[4]`	`0`
`[5]`	`0`
`[6]`	`''`
`[7]`	`''`
`[8]`	`Requested URL`
`[9]`	`HTTP Request Headers`
`[10]`	`HTTP Request Body`

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

- `[4]` : set to `1` if you will handle the request by updating other elements of the event message.

  In a typical scenario, you will check whether the requested URL in [8] begins with the "root" URL:
  - If it does, then your application will supply the content of the response. In this situation, update the appropriate elements of the event message, setting element `[4]` to 1 and return.

  - If it does not, then the request is for some external resource. Return without changing any elements of the event message and HTMLRenderer will attempt to retrieve the requested resource.

- `[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. For sending HTML, the MIME type is `'text/html'`.
- `[9]` : set to any HTTP message headers necessary for the response.
- `[10]` : set to the body of the response. Typically this will be HTML.

6 Debugging HTMLRenderer

Chromium's developer tools can be used to inspect and debug the rendered HTML content.

To use Chromium's developer tools

1. Start the Dyalog interpreter with a command line parameter of
   `-remote-debugging-port="xxxxx"`
   where `xxxxx` is the port number to use to connect to HTMLRenderer.
2. Start the HTMLRenderer application.
3. Open a Google Chrome browser and navigate to
   http://127.0.0.1:xxxxx
   where xxxxx is the port number you specified in step 1.