Introduction to HTMLRenderer

Brian Becker and Josh David
Related Materials

Available at:

https://github.com/Dyalog19/SA3

This includes demo files and the workshop handout
Agenda

- Goals
- Introductions
- Prerequisites and Setup
- HTMLRenderer Overview
- Break 1
- Diving Deeper
- Utilities and Frameworks
- Break 2
- Advanced Topics
- Q&A
Goals
- Teach you HTMLRenderer
  - What it is
  - What it's not
  - Properties, Methods, Events
- Tools and Frameworks
- Give you hands-on experience

Non Goals
- Teach you DUI
- Teach you HTML/CSS
Introductions

- Have you used...
  - ⎕wc
  - HTML/CSS/JavaScript?
  - MiServer

- Your goals
What is HTMLRenderer?

- A Dyalog object that provides an interface between Dyalog APL and CEF (Okay, so what is CEF?)
- CEF – Chromium Embedded Framework
  - An open-source software framework for embedding a Chromium web browser within another application
  - CEF is NOT Google Chrome, though Google Chrome uses the Chromium web browser as its core
- Web browsers render HTML, CSS, and JavaScript
  - Dyalog has utilities and frameworks that reduce your need to learn these
Why use HTMLRenderer?

- ⎕WC/Win32 GUI has been wonderful on Windows…
  - But what about macOS and Linux?
- HTMLRenderer is cross-platform
  - Write once, run everywhere
- Plethora of resources available
  - Syncfusion, jQuery, FontAwesome, DataTables, …
- HTML5/CSS/JavaScript enables more flexible formatting/interactivity-animation than ⎕WC
HTMLRenderer Properties

Just like most other Dyalog objects, HTMLRenderer has

- Properties

```
⍳9ρ((⊂∘⍋)⌷⊢)
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsChild</td>
<td>EventList</td>
</tr>
<tr>
<td>Attach</td>
<td>HTML</td>
</tr>
<tr>
<td>Border</td>
<td>IconObj</td>
</tr>
<tr>
<td>CEFVersion</td>
<td>InterceptedURLs</td>
</tr>
<tr>
<td>Caption</td>
<td>KeepOnClose</td>
</tr>
<tr>
<td>ChildList</td>
<td>MaxButton</td>
</tr>
<tr>
<td>Coord</td>
<td>MethodList</td>
</tr>
<tr>
<td>Data</td>
<td>MinButton</td>
</tr>
<tr>
<td>Event</td>
<td>Moveable</td>
</tr>
<tr>
<td>Posn</td>
<td>PropList</td>
</tr>
<tr>
<td>Size</td>
<td>Sizeable</td>
</tr>
<tr>
<td>SysMenu</td>
<td>Sizeable</td>
</tr>
<tr>
<td>Translate</td>
<td>Type</td>
</tr>
<tr>
<td>URL</td>
<td>Visible</td>
</tr>
</tbody>
</table>

HTMLRenderer Properties

Just like most other Dyalog objects, HTMLRenderer has

- Properties
- Events

```dlicher
3 3⍴((⊂∘⍋)⌷⊢)
hr.EventList

Close    HTTPRequest    WebSocketError
Create   SelectCertificate WebSocketReceive
DoPopup  WebSocketClose  WebSocketUpgrade
```
HTMLRenderer Properties

- Just like most other Dyalog objects, HTMLRenderer has
  - Properties
  - Events
  - Methods

```dyalog
⍪((⊂∘⍋)⌷⊢)hr.MethodList
Detach
PrintToPDF
ShowDevTools
Wait
WebSocketSend
```
Properties

- **Coord** – Prop, Pixel, ScaledPixel, RealPixel
- **Size, Posn** - (y,x) not (x,y), Top Left is 0 0
- Some properties are implemented only on platforms where they're allowed – e.g. `AsChild` is only valid on Windows
  - If a property is not allowed, setting it should have no effect
Try this...

'hr' ⎕WC 'HTMLRenderer' ('HTML' 'Hello World!')

OR

hr ← ⎕NEW 'HTMLRenderer'.,⊂'HTML' 'Hello World!'

hr.Caption←'My HTMLRenderer'
hr.HTML←'<h1>Hi!</h1>'
hr.Size←100 100
hr.(Size Posn)←(25 25)(25 25)
hr.Coord
hr.Coord←'ScaledPixel'
hr.(Size Posn)
hr.Posn←25 25
URL and HTML Properties

- **URL** sets the "root" for the HTMLRenderer
  Requests for resources will be relative to URL unless the resource specifies an absolute path
  - Relative - /uploads/css/jquery.fancybox.css
  - Absolute - https://platform.twitter.com/js/moment~timeline~tweet.059.js

- **HTML** specifies the content for the HTMLRenderer window

- **URL** supercedes **HTML**

- `'http://dyalog_root/'` is the "default" URL

- In general, you will set either URL or HTML, but not both
Try this...

hr 'hr' ⎕WC 'HTMLRenderer' ('URL' 'www.google.com') ('HTML' 'Hi!')
hr.URL←'www.dyalog.com'
hr.URL←'dyalog_root'
hr.URL←'www.dyalog.com'
hr.URL←''
HTTPRequest event

An HTTPRequest event is signaled whenever a request for a local resource is made. To react to this event, you define a handler.

'Event' ('onHTTPRequest' 'function_name')

OR

hr.onHTTPRequest←'function_name'
Try this...

)clear
]load HttpUtils
]load [SA3]/Demos/SimpleForm
SimpleForm ''
SYNTAX ERROR
SimpleForm[10] ⋯ ⋯ A comment this line to run without stopping
^
HTTPRequest event argument and result

HTTPRequest Argument Elements

[1] Object ref or character vector
[2] Event 'HTTPRequest' or 840
[8] URL Character vector containing the requested URL
[9] Headers Character vector containing the HTTP Request headers
[10] Body Character vector containing the HTTP Request body
[11] Method Character vector containing the HTTP method e.g. 'GET' or 'POST'.

HTTPRequest Result Elements

[4] Handle 1
[6] Message Success is indicated by 'OK'.
[7] MIME Defaults to 'text/html' and need be specified only if the response is not HTML.
[9] Response Headers (not normally required)
[10] Body Typically this will contain HTML.
Tools, Utilities, and Frameworks

- **HttpUtils** - helps manage HTTPRequest event arguments and results
- **MsgBox** - syntactically similar to Win32 MsgBox
- **EasyGUI** - utilities to implement relatively simple interactions
- **DUI** - Cross-platform framework to develop user interfaces that run locally or over the net
Try this...

]load [SA3]/Utilities/MsgBox
mb←⎕NEW MsgBox
mb.Caption←'Are you sure?'
mb.Style←'query'
mb.Text←'Engage ludricrous speed Captain?'
btnClicked←mb.Run
EasyGUI

- Create GUIs at a higher level of abstraction
- Cross platform
- Simple, recurring tasks
  - Minimal styling imposed, but styling options available
EasyGUI - Hosted on git

- [SA3]/Utilities/EasyGUI
  - Forked from https://github.com/JoshDavid/EasyGUI
- ]link or acre_desktop to bring into workspace
Layout of the EasyGUI library

- Functions
  - Queries
  - Notifications
  - Graphics
- All take one optional left arg
  - specifyParams
    - Key-value pairs or dot notation
DUI – Dyalog User Interface

- Web Content Creation (WC2)
  - Page class for building stand-alone HTMLRenderer pages
- HTML Server
  - MiServer – TCP/IP over the net
  - HRServer – local desktop using HTMLRenderer
- Used in APL Contest Website, miserver.dyalog.com, TryAPL.org, Conference Registration system, TamStat
Client-side Debugging

- `ShowDevTools` method
- `--remote-debugging-port` command line parameter
- Both bring up Chrome DevTools
Try this...

)clear
]load [SA3]/DUI/DUI
]load [SA3]/Demos/I*
DUI.Initialize
InputDemo
InputDemo2 ''
Try this...

)clear
]load [SA3]/DUI/DUI
DUI.Run ':[SA3]/Demos/2048/'
WebSockets

- Before WebSockets, servers could only respond to requests from clients.
- WebSockets enable bi-directional, asynchronous between client and server.
- Client must request upgrade of HTTP connection which the server will accept or decline.
- Once the WebSocket has been established, either side can send a message, no response is required.
- Either side can close the WebSocket
WebSocket Methods and Events

<table>
<thead>
<tr>
<th>JavaScript in the CEF client</th>
<th>HTMLRenderer in the workspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ws = new websocket(&quot;ws://dyalog_root/&quot;); Initiate the request</td>
<td>WebSocketUpgrade event</td>
</tr>
<tr>
<td></td>
<td>The websocket is established</td>
</tr>
<tr>
<td>ws.send(&quot;message&quot;);</td>
<td>WebSocketReceive event</td>
</tr>
<tr>
<td>ws.onmessage event</td>
<td>WebSocketSend method</td>
</tr>
<tr>
<td>ws.close()</td>
<td>WebSocketClose event</td>
</tr>
<tr>
<td>ws.onclose event</td>
<td>WebSocketClose method</td>
</tr>
<tr>
<td>ws.onerror event</td>
<td>WebSocketError event</td>
</tr>
<tr>
<td>is triggered when there is some error like the connection going down</td>
<td>occurs when there is some error like the connection going down</td>
</tr>
</tbody>
</table>
Try this...

)clear
]load [SA3]/Demos/Web*
WebSocketDemo''
InterceptedURLS property

- InterceptedURLs property
- Controls whether a request for a resource will be passed back to APL, or over the net
- 2-column matrix of [;1] patterns to match, [;2] 0 – net, 1 – APL
  All "local" resources will be passed to APL, non-local to the net
  `<img src="duck.jpg"/>` A local
  `<script src="https://www.google.com/analytics.js"/>` A non-local
- The default pattern is http[s]://dyalog_root/
- In general, you will not need to set InterceptedURLs
DoPopup Event

- When the client attempts to open a new window, a DoPopup event is signaled
- When this happens, you'll need to open another HTMLRenderer
- Event argument[3] is the requested URL which you use as the URL parameter to the new HTMLRenderer
Try this...

```dyalog
]load [SA3]/Demos/DoPop*
DoPopupDemo ''
DoPopupDemo2 ''
```
Coming Soon to a DUI Near You...

- WebSockets are an integral part of the data-binding model in DUI
  - Data-binding – keeping data in the workspace in sync with data in the GUI
  - DUI's MiPage class will have a built-in WebSocket capability to facilitate this
- In addition, we are developing a WebSocket widget that will use the same APL Jax protocol as DUI's event handling.
  - Hides all of the JavaScript
- Similarly, we are extending DUI to use multiple HTMLRenderers in support of the DoPopup event
HTMLRenderer To Do's (Right JD? 😊)

- If a page tries to initialize a WebSocket immediately upon the first time HTMLRenderer is loaded, the connection may fail.
- Extend InterceptedURLs to recognize protocols in addition to HTTP[S]. For example, WS[S] and possibly FTP[S].
- Allow references to file:// to read files directly without issuing a callback.
Questions?

A couple other demos:

```
)clear
]load [SA3]/Demos/cube/cubeDemo
cubeDemo '[SA3]'
```