Dyalog Release Notes

Dyalog version 17.1

DYALOG

The tool of thought for software solutions

Dyalog Release Notes

Dyalog version 17.1
Document Revision: 20200114_171

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

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

Contents

Chapter 1:

Highlights

For version 17.1, the focus of the efforts of the Dyalog development team has been to make Dyalog APL easier to install, update, integrate and deploy. Significant work has also been done on interpreter performance and potential new language features, but this work will not become available until version 18.0, scheduled for released in 2020.

Highlights of version 17.1 include:

Ease of Installation

Dyalog APL no longer requires pre-approval of a licence application and it can now be downloaded on demand from www.dyalog.com with or without registration. Note that the use of Dyalog APL is subject to a licence agreement and, in particular, Dyalog APL is not free for commercial use. To use Dyalog for commercial purposes you must contact Dyalog to obtain a commercial licence.

Under Microsoft Windows, Dyalog-produced binaries including the installer and executables are digitally signed, making them easier to install in corporate and other secure environments.

Dyalog APL is "Cloud Ready"

APL-based components are now easier to install and manage on servers in the cloud or in corporate server environments:

Under Microsoft Windows, the **RunAsService** configuration option has been extended to allow Dyalog APL for Windows to be started with the GUI disabled. In this mode, Dyalog APL consumes significantly fewer system resources, allowing a much larger number of service processes to be started simultaneously on a Windows Server.

Non-Windows versions which are started without a terminal attached ("headless") can now always be debugged using RIDE. Output from headless instances of APL can produce redirected output which is free of terminal control sequences.

Dyalog has published Docker Containers on https://dockerhub.com, making it straightforward to launch applications based on Dyalog APL version 17.1 for Linux on any real or virtual Linux machine. This includes containers that provide pre-configured web servers (based on MiServer) or web services (based on JSONServer) – including ODBC driver support.

Integration

For decades, Dyalog APL has provided tight integration with standard application interfaces such as COM and the Microsoft.NET framework – under Microsoft Windows. In version 17.0, a prototype of a new wrapper was provided, making it possible to wrap APL applications in standard C-style libraries (Dynamic Link Library or Shared Object files).

Version 17.1 provides a significantly enhanced version of this mechanism, with sample applications including a framework which can be used to easily turn any APL workspace into a library from which APL functions can be called with arguments and results passed in the form of JSON strings.

Cross Platform UI

The HTMLRenderer is a new mechanism for creating identical graphical user interfaces (GUI) on multiple platforms. It was first made available with Dyalog APL version 17.0 – for macOS and Microsoft Windows - and allows APL applications to make use of the Chromium Embedded Framework (CEF) to render HTML/JavaScript based user interfaces.

Version 17.1 provides significantly enhanced support for the CEF and adds GNU Linux to the list of supported platforms. In addition to being more robust, the v17.1 HTMLRenderer adds support for WebSockets, and a Certificate Selector which facilitates the embedding of components which require encrypted communications into an HTMLRenderer-based GUI.

A new user command `]html` makes it easy to render the results of functions which generate HTML (including SVG) under both the Windows IDE and RIDE. The]plot command makes use of the same technology to generate and display SharpPlot graphics from Dyalog APL running on any platform.

Source Code Management

Version 17.1 includes the "Link" system, which can be used to establish (optionally bi-directional) synchronisation between code in the workspace and text files. Link makes it simple to store APL source code in Unicode text files, which are typically managed by Git or SVN. Functions can either be edited in the workspace using the internal Dyalog editor, or using external editors to edit the text files – in both cases the source file and the version in the workspace are kept in synch.

The version of Link included with v17.1 includes experimental support for an Array Notation for representing APL array constants in text-based source files.

Open Source Projects

An increasing number of the tools which are written in APL and included with Dyalog APL are provided as open-source projects on GitHub. Users can elect to use the fully supported versions of the code which are installed alongside Dyalog APL, or they can download or clone the GitHub repositories, if they want to use the latest versions of the tools – or contributed to the projects. This includes the Link project for source code management.

Serial Number

If you have registered your copy of Dyalog or have a commercial licence then you will have been sent a Dyalog serial number; this serial number is individual to you and corresponds to the type of licence that you are entitled to use.

On Microsoft Windows, you can enter your Dyalog serial number during or after the installation process; on all other platforms the Dyalog serial number should be entered after installation.

Instructions on how to enter your Dyalog serial number are included in the following platform-specific documents:

- Dyalog for macOS Installation and Configuration Guide
- Dyalog for Microsoft Window Installation and Configuration Guide
- Dyalog for UNIX Installation and Configuration Guide
- Dyalog for Raspberry Pi User Guide

If you already have a registered or commercial version of Dyalog then you will already have a Dyalog serial number – if cannot remember it, then please contact sales@dyalog.com.

NOTE:

Using or entering a serial number other than the one issued to you is not permitted.
Transferring the serial number to anyone else is not permitted.
For the full licence terms and conditions, see:
https://www.dyalog.com/uploads/documents/terms_and_conditions.pdf

Key Features

There have been a number of performance improvements in 17.1, but the main area of improvement is in its stability. This has been achieved using extensive automated testing of random expressions to identify edge conditions requiring correction.

Language Enhancements

- ⎕JSON supports a new **Null** variant that specifies the APL representation of a JSON null value. It may be either ⊂'null' (the default) or ⎕NULL.
- The **Recurse** variant option for ⎕NINFO may be 2. This means the same as the value 1, but if any unreadable directories are encountered they are skipped (whereas if **Recurse** is 1, ⎕NINFO stops and generates an error).
- ⎕NINFO and other system functions and commands that handle files have been enhanced to deal with non-standard file names. This enhancement applies only to non-Windows platforms. See *Unusual File Names* on page 6.
- ⎕NINFO now reports whether a directory is readable or writable (exception: on Microsoft Windows, a left argument of 12 returns ¯1 for a directory).
- A 4th column has been added to Syntax Colour Tokens i-beam function. This is intended for the benefit of non-Windows users using the tty interface and indicates the video/foreground/background colour index.

GUI Enhancements

- The previously experimental HTMLRenderer object is now fully supported in Version 17.1. See *HTMLRenderer* on page 18.
- Setting the TitleHeight and/or TitleWidth properties of the Grid object now select the default height and/or width.

Other Enhancements

- The **RunAsService** parameter now accepts the value 2. This setting reduces the resources used by a Dyalog service by disabling the graphical user-interface features. ⎕WC object will fail with a LIMIT ERROR unless the object is Timer, which is the only one that remains enabled.
- The APL command line accepts two new options +s (force the display of the Session window) and +q (force a quit on error).

Miscellaneous Changes

- The *Log Tab* of the *Configuration Dialog* has been removed and its contents moved to the *Session Tab*.
- The default value of **APL_MAX_THREADS** has been changed from the number of virtual processors to 1. This means that parallel execution is no longer enabled by default. See *Programming Reference Guide: Parallel Execution*.
- In previous versions, if you attempted to write files to `c:\windows`, the Windows VirtualStore would be invoked, resulting in the file apparently being written to `c:\windows` where in actual fact it was written to `%localappdata%\virtualstore`. The Version 17.1`dyalog.exe.manifest` has been altered so that you will now get an error when attempting to write to `c:\windows`.

Unusual File Names

On non-Windows platforms, file names are exposed by the Operating System using UTF-8 encoding which Dyalog translates internally to characters.

In the Unicode Edition, if the UTF-8 encoding is invalid, Dyalog replaces each offending byte with a unique Unicode symbol (in the *Low Surrogate Area* of the Unicode charts) that is mapped back to the original byte by the other system functions (including `⎕NTIE` and `⎕NDELETE`) that take native file names as arguments. The display of a file name containing these mapped bytes may appear strange.

In the Classic Edition, offending bytes are replaced by the `?` symbol, which means that the names reported do not accurately identify the files.

Bug Fixes

A number of bug fixes implemented in Version 17.1 may change the way that existing code operates and are therefore documented in this section.

- 16013 `⎕NEXISTS` now returns 0 for an empty right argument; in previous versions it generated a `DOMAIN ERROR`.
- 16195 Interval Index no longer gives `DOMAIN ERROR` for consecutive duplicate values in the left argument. Consecutive duplicate values create an empty interval, and no items of the right argument will lie in an empty interval.

Announcements

Tools and Interfaces

In addition to the software provided as part of the Dyalog installation package, there are a growing number of tools and interfaces available for download from GitHub. For details, see https://www.dyalog.com/tools/tools-and-code-libraries.htm.

Withdrawal of Support for Version 15.0

The supported Versions of Dyalog APL are now Version 17.1, 17.0 and 16.0. Version 15.0 and earlier versions are no longer supported.

Planned Operating System Requirements for the next version

Dyalog Ltd expects that the next version of Dyalog will require the following minimum platform requirements:

Operating System	Version
Microsoft Windows	Windows 7/Server 2008 R2
AIX	POWER 8 (if you have a POWER 7 requirement, please contact sales@dyalog.com).
Linux	Debian 8
OS X	OS X Yosemite 10.10.x
Raspberry Pi	Raspbian Jessie

Further updates to this information will appear on the Forums as and when available.

Planned Hardware Requirements for next version

The same as Dyalog Version 17.1.

System Requirements

Microsoft Windows

Dyalog APL Version 17.1 is supported on versions of Microsoft Windows from Windows 7 or Windows Server 2008 R2, up to and including Windows 10 and Windows Server 2016.

Microsoft .NET Interface

Dyalog APL Version 17.1 .NET Interface requires Version 4.0 or greater of the Microsoft .NET Framework. It does *not* operate with earlier versions of .NET.

For full Data Binding support (including support for the `INotifyCollectionChanged` interface[1]) Version 17.1 requires .NET Version 4.5. The Syncfusion libraries supplied with Version 17.1 require .NET 4.6.

The examples provided in the sub-directory `Samples/asp.net` require that IIS is installed. If IIS and ASP.NET are not present, the `asp.net` sub-directory will not be installed during the Dyalog installation.

AIX

For AIX, Version 17.1 requires AIX 7.2 or higher, and a POWER7 chip or higher.

Raspberry Pi

On the Raspberry Pi, Dyalog 32-bit Unicode supports Raspbian Jessie or later.

Non-Pi Linux

For non-Pi Linux, Version 17.1 only exists as 64-bit interpreters - there are no 32-bit versions. It is built on Debian 8; it should run on all recent distributions. See https://forums.dyalog.com/viewtopic.php?f=20&t=1504 for a list of tested platforms. Note that the HTMLRenderer has additional pre-reqs which are detailed in the Dyalog forums at https://forums.dyalog.com/viewtopic.php?f=20&t=1504.

macOS/Mac OS X

Version 17.1 requires Mac OS X Yosemite or El Capitan or macOS Sierra or later. The target Mac must have been introduced in 2010 or later.

[1]This interface is used by Dyalog to notify a data consumer when the contents of a variable, that is data bound as a list of items, changes.

Interoperability

Introduction

Workspaces and component files are stored on disk in a binary format (illegible to text editors). This format differs between machine architectures and among versions of Dyalog. For example, a file component written by a PC may well have an internal format that is different from one written by a UNIX machine. Similarly, a workspace saved from Dyalog Version 17.0 will differ internally from one saved by a previous version of Dyalog APL.

It is convenient for versions of Dyalog APL running on different platforms to be able to *interoperate* by sharing workspaces and component files. From Version 11.0, component files and workspaces can generally be shared between Dyalog interpreters running on different platforms. However, this is not always possible and the following sections describe limitations in interoperability:

Code and ⎕ORs

Code that is saved in workspaces, or embedded within ⎕ORs stored in component files, can only be read by the Dyalog version which saved them and later versions of the interpreter. In the case of workspaces, a load (or copy) into an older version would fail with the message:

```
this WS requires a later version of the interpreter.
```

Every time a ⎕OR object is read by a version later than that which created it, time may be spent in converting the internal representation into the latest form. Dyalog recommends that ⎕ORs should not be used as a mechanism for sharing code or objects between different versions of APL.

"Ordinary" Arrays

With the exception of the Unicode restrictions described in the following paragraphs, Dyalog APL provides interoperability for arrays that only contain (nested) character and numeric data. Such arrays can be stored in component files - or transmitted using `TCPSocket` objects and Conga connections, and shared between all versions and across all platforms.

Full cross-platform interoperability of component files is only available for large-span component files.

Null Items (⎕NULL) and Compressed Components

⎕NULLs and components from compressed component files that were created in Version 17.1 can be brought into Versions 14.1, 15.0, 16.0 and 17.0 provided that the interpreters have been patched to revision 35953 or higher. Attempts to bring ⎕NULL or compressed component into earlier versions of Dyalog APL or lower revisions of the aforementioned versions will fail with:

```
DOMAIN ERROR: Array is from a later version of APL.
```

Object Representations (⎕OR)

An attempt to ⎕FREAD a component containing a ⎕OR that was created by a later version of Dyalog APL will generate `DOMAIN ERROR: Array is from a later version of APL`. This also applies to APL objects passed via Conga or TCPSockets, or objects that have been serialised using `220⌶`.

32 vs. 64-bit Component Files

It is no longer possible to *create* or write to small-span (32-bit) files; however it is still currently possible to *read* from small span files. Setting the second item of the right argument of ⎕FCREATE to anything other than 64 will generate a `DOMAIN ERROR`.

Note that *small-span* (32-bit-addressing) component files cannot contain Unicode data. Unicode editions of Dyalog APL can only write character data which would be readable by a Classic edition (consisting of elements of ⎕AV).

External Variables

External variables are subject to the same restrictions as small-span component files regarding Unicode data. External variables are unlikely to be developed further; Dyalog recommends that applications which use them should switch to using mapped files or traditional component files. Please contact Dyalog if you need further advice on this topic.

32 vs. 64-bit Interpreters

There is complete interoperability between 32- and 64-bit interpreters, except that 32-bit interpreters are unable to work with arrays or workspaces greater than 2GB in size.

Note however that under Windows a 32-bit version of Dyalog APL may only access 32-bit DLLs, and a 64-bit version of Dyalg APL may only access 64-bit DLLs. This is a Windows restriction.

Unicode vs. Classic Editions

Two editions are available on some platforms. Unicode editions work with the entire Unicode character set. Classic editions (which are only available to commercial and enterprise users for legacy applications) are limited to the 256 characters defined in the atomic vector, `⎕AV`.

Component files have a Unicode property. When this is enabled, all characters will be written as Unicode data to the file. The Unicode property is always off for small-span (32-bit addressing) files, as these cannot contain Unicode data. For large-span (64-bit addressing) component files, the Unicode property is set *on* by Unicode Editions and *off* by Classic Editions, by default. The Unicode property can subsequently be toggled on and off using `⎕FPROPS`.

When a Unicode edition writes to a component file that cannot contain Unicode data, character data is mapped using `⎕AVU`; it can therefore be read without problems by Classic editions.

A `TRANSLATION ERROR` will occur if a Unicode edition writes to a non-Unicode component file (that is either a 32-bit file, or a 64-bit file when the Unicode property is currently off) if the data being written contains characters that are not in `⎕AVU`.

Likewise, a Classic edition will issue a `TRANSLATION ERROR` if it attempts to read a component containing Unicode data that is not in `⎕AVU` from a component file.

A `TRANSLATION ERROR` will also be issued when a Classic edition attempts to `)LOAD` or `)COPY` a workspace containing Unicode data that cannot be mapped to `⎕AV` using the `⎕AVU` in the recipient workspace.

`TCPSocket` objects have an `APL` property that corresponds to the Unicode property of a file, if this is set to `Classic` (the default) the data in the socket will be restricted to `⎕AV`, if Unicode it will contain Unicode character data. As a result, `TRANSLATION ERROR`s can occur on transmission or reception in the same way as when updating or reading a file component.

The symbols ⊆, ⍳, ⍤, ⌸, ⌸ and ⌺ used for the Nest (Interval Index) and Where (Partition) functions, the Rank, Variant, Key and Stencil operators respectively are available only in the Unicode edition. In the Classic edition, these symbols are replaced by `⎕U2286`, `⎕U2378`, `⎕U2364`, `⎕U2360`, `⎕U2338` and `⎕U233A` respectively. In both Unicode and Classic editions Variant may be represented by `⎕OPT`.

Very large array components

An attempt to read a component greater than 2GB in 32-bit interpreters will result in a `WS FULL`.

TCPSockets and Conga

TCPSockets and Conga can be used to communicate between differing versions of Dyalog APL and are subject to similar limitations to those described above for component files.

Auxiliary Processors

A Dyalog APL process is restricted to starting an AP of exactly the same architecture from the same operating system. In other words, the AP must share the same word-width and byte-ordering as its interpreter process.

Session Files

Session (.dse) files can only be used on the platform on which they were created and saved. Under Microsoft Windows, Session files may only be used by the architecture (32-bit-or 64-bit) of the Version of Dyalog that saved them.

Chapter 2:

Object Reference Changes

AsChild Property

Applies To: HTMLRenderer

Description

The AsChild property is a Boolean (default 0) indicating how the HTMLRenderer object is displayed. AsChild must be set when the object is created and may not subsequently be changed.

If AsChild is 0 (the default) the HTMLRenderer is displayed in a separate top-level window. If the HTMLRenderer is created (with AsChild 0) as a child of another object it still appears as a separate window and its Size and Posn relate to the screen rather than to its parent object. However, it is a member of its parent object's hierarchy and will disappear when its parent is closed.

If AsChild is 1, the HTMLRenderer must be created as a child of a valid parent type other than Root (which is not supported) and is displayed in a sub-window within its parent.

This property only applies to Microsoft Windows. On other platforms it is ignored.

CEFVersion Property

Applies To: HTMLRenderer

Description

CEFVersion is a read-only property that reports the version of the Chromium Embedded Framework (CEF)[1] that is being used. This information is primarily used for debugging and support.

It is a 10-element vector containing the following:

Index	Description
[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 Hashes
[9]	GIT Hashes
[10]	GIT Hashes

[1]https://en.wikipedia.org/wiki/Chromium_Embedded_Framework

DoPopup Event 846

Applies To: HTMLRenderer

Description

This event is triggered when the HTMLRenderer client attempts to open a new window. This is often done using an HTML <a> tag with the target attribute set to open a URL in a new window. Note that this does not apply to JavaScript Popup Boxes.

Example:

```
<a href="www.dyalog.com" target="_blank">Dyalog Website</a>
```

By default the HTMLRenderer ignores a request for a new window, but if the DoPoup event, is enabled, it provides the information needed to process the request in the workspace.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 4-element vector as follows:

`[1]`	Object	ref or character vector
`[2]`	Event	`'DoPopup'` or 846
`[3]`	URL	the requested url
`[4]`	Attributes	requested window attributes (see below)

Attributes is a 7-element nested vector that specifies the requested attributes for the new window. The HTMLRenderer currently provides no mechanism to use this information.

To respond to the request for a new window, the callback function should open the requested URL as appropriate, for example, in a newly created HTMLRenderer object.

HTML Property

Applies To: HTMLRenderer

Description

The HTML property is a character vector that specifies the content to be rendered by the HTMLRenderer object. Dyalog does not perform any pre-processing of the text nor does it verify that it is properly formed HTML using single-byte character data, including any necessary escaping and encoding.

See also: *URL* on page 30.

Note:

Typically, you will need to UTF-8 encode any text outside the Unicode range 0-127.

HTMLRenderer Object

Purpose:	The HTMLRenderer Object is a cross-platform mechanism for producing Graphical User Interfaces (GUI), based on HyperText Markup Language (HTML).
Parents	ActiveXControl, Form, Group, PropertyPage, Root, SubForm
Children	Timer
Properties	Type, HTML, Posn, Size, URL, Coord, Border, Visible, Event, Sizeable, Moveable, SysMenu, MaxButton, MinButton, IconObj, Data, Attach, Translate, KeepOnClose, AsChild, InterceptedURLs, CEFVersion, Caption, MethodList, ChildList, EventList, PropList
Methods	Detach, PrintToPDF, WebSocketSend, ShowDevTools, Wait
Events	Close, Create, HTTPRequest, WebSocketUpgrade, WebSocketReceive, WebSocketClose, WebSocketError, DoPopup, SelectCertificate

Description

The HTMLRenderer object renders HTML in a window on the screen. It may appear as a top-level window, similar to a Form, or be displayed within another GUI object according to the value of the Boolean AsChild property which must be specified when the HTMLRenderer is created. Several HTMLRenderer objects may co-exist in the Dyalog application.

The HTMLRenderer is implemented using the Chromium Embedded Framework (CEF)[1]. Note that if the **ENABLE_CEF parameter** is set to 0 (its default value is 1) the CEF is disabled and an attempt to create an HTMLRenderer object will fail with an error message.

The HTMLRenderer object can be considered as two components; a client implemented using CEF and an internal server which implements an interface from the APL workspace to the client. The client may communicate with both the internal server and external servers on the web. Thus it can combine and display information from external and internal feeds.

[1]https://en.wikipedia.org/wiki/Chromium_Embedded_Framework

Internal and external communications are distinguished primarily by the InterceptedURLs property which specifies which requests from the HTMLRenderer client component are to he handled by the internal server component and which are to be serviced by the internet. The default value of InterceptedURLs has been chosen so that in most cases it can be ignored.

The HTMLRenderer object supports both the HTTP protocol and the WebSocket (WS) protocol.

Using the HTTP protocol, the client requests a resource, such as a style-sheet, an image, or a complete web page, which the server then delivers. All communication is initiated by the client and involves the creating, use, and closing of a TCP/IP socket.

Using WS protocol, the client asks the server for a permanent communications channel (this is done by *upgrading* the TCP/IP socket to a WebSocket) which may subsequently be used for messages initiated by either the client or the server.

The internal server component of the HTMLRenderer is implemented by functions in the workspace.

HTTP protocol communications are handled by callback functions on the HTTPRequest event.

WS protocol communications are handled:

- by the WebSocketUpgrade event, which reports the initial connection and the WebSocket ID,
- by a callback on the WebSocketReceive event
- and by calling the WebSocketSend method.

The HTMLRenderer may be initialised by setting its HTML property to a character vector representing a base HTML document. This will typically contain references to other documents, 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. Requests with URLs that match a triggering pattern in the InterceptedURLs property will generate an HTTPRequest event on the HTMLRenderer, which can be directed to a callback function. The callback can either service the request or leave it to the CEF to handle it.

Requests with URLs that do not match a pattern in InterceptedURLs, or that match a pattern with a 0 in the second column, will be handled by the CEF.

Requests handled by the CEF push the request out to the network to be serviced by an external web server and require that the system has an active internet connection.

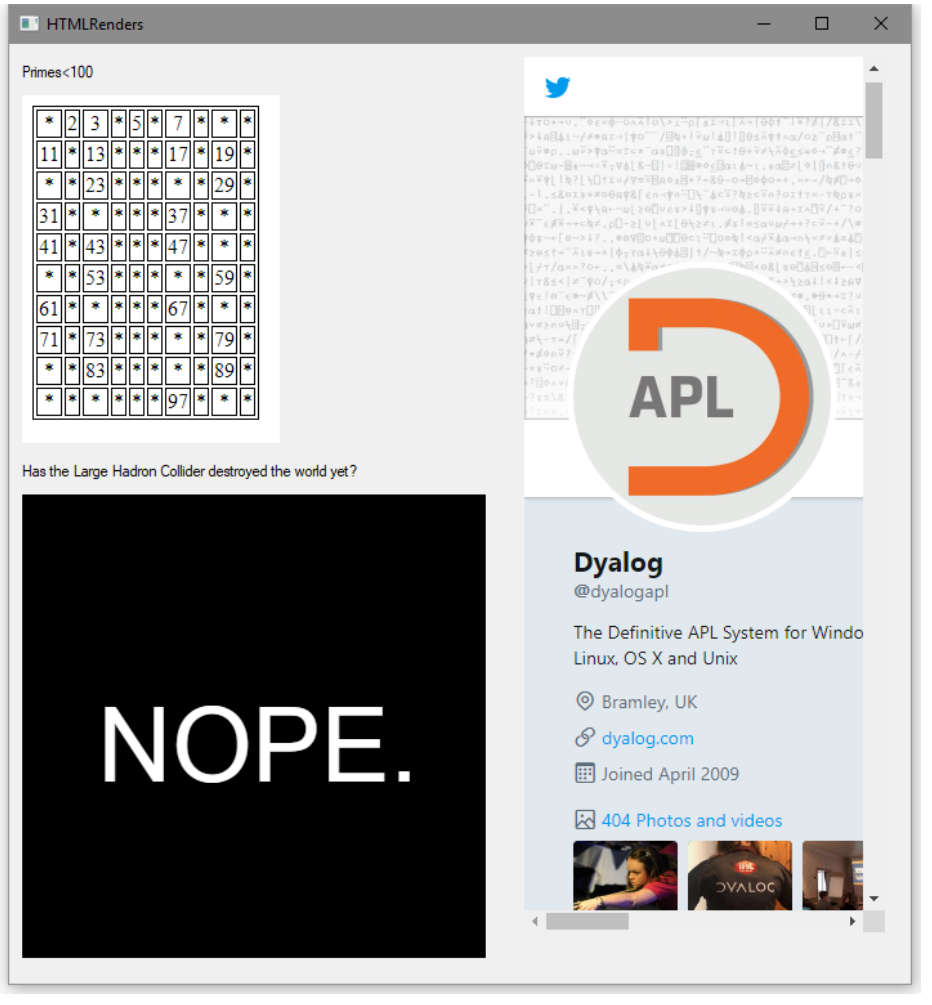
An alternative is to initialise the HTMLRenderer by setting its URL property. This is typically used to display external content , rather than content delivered from the workspace.

If neither HTML nor URL is set when the HTMLRenderer is created, it will generate an HTTPRequest event with a requested url of `http://dyalog_root`.

When the HTMLRenderer is displayed in its own window, the window caption is set by an assignment to its Caption property. The window caption may subsequently change when content is displayed (typically by the title tag in the html). The Caption property reports the current window caption.

Example

```
     ∇ Example;enc;Q;U;tw
[1]    'f'⎕WC'Form' 'HTMLRender'
[2]    f.(Coord Size)←'Pixel'(730 700)
[3]    'pco'⎕CY'dfns'
[4]    'f.l1'⎕WC'Label' 'Primes<100'(10 10)
[5]    'f.p'⎕WC'HTMLRenderer'('AsChild' 1)('Posn' 40 10)(270 200)
[6]    f.p.HTML←HTMLTable('*'@(0∘pco)10 10⍴⍳100)
[7]    Q←'Has the Large Hadron Collider destroyed the world yet?'
[8]    'f.l2'⎕WC'Label'Q(320 10)
[9]    'f.q'⎕WC'HTMLRenderer'('ASChild' 1)
[10]   f.q.(Posn Size)←(350 10)(360 360)
[11]
U←'http://hasthelargehadroncolliderdestroyedtheworldyet.com'
[12]   f.q.URL←U
[13]   tw←'<a class="twtimeline"'
[14]   tw,←'href="https://twitter.com/dyalogapl">'
[15]   tw,←'Tweets by dyalogapl</a>'
[16]   tw,←'<script async
src="http://platform.twitter.com/widgets.js"'
[17]   tw,←'charset="utf-8"></script>'
[18]   'f.t'⎕WC'HTMLRenderer'('AsChild' 1)
[19]   f.t.(Posn Size)←(10 400)(680 280)
[20]   f.t.HTML←tw
     ∇

     enc←{'<',⍺,'>',(∊⍵),'</',((~∨\' '=⍺)/⍺),'>'}

     HTMLTable←{'table border="1"'enc∊(⊂'tr')enc∘∊⍤1⊢(⊂'td')enc⍤1⊢⍵}
```


HTMLRenders
Primes<100
Has the Large Hadron Collider destroyed the world yet?
NOPE.
APL
Dyalog
@dyalogapl
The Definitive APL System for Windo
Linux, OS X and Unix
Bramley, UK
dyalog.com
Joined April 2009
404 Photos and videos

HTTPRequest Event 840

Applies To: HTMLRenderer

Description

An HTTPRequest event is raised whenever the HTMLRenderer requests a url from the workspace. See InterceptedURLs property. The request could be generated by a form submission, clicking on a hyperlink, an AJAX request or a link to a resource like a style sheet, image or JavaScript file. An HTTPRequest event is also raised when the HTMLRenderer is initialised and both HTMLand URL are empty.

The callback function must "fill in the blanks" in the event message and return the modified event message as its result.

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is an 11-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'HTTPRequest' or 840
[3]		'ProcessRequest' (unused)
[4]	Handle	Initially 0, must be set to 1.
[5]	Status	Integer HTTP status code (initially 0).
[6]	Message	Character vector containing the HTTP status message (initially empty).
[7]	MIME	Character vector containing the MIME type (initially empty). See below.
[8]	URL	Character vector containing the requested URL.
[9]	Headers	Character vector containing the HTTP Request headers (initially empty).
[10]	Body	Character vector containing the HTTP Request body (initially empty).
[11]	Method	Character vector containing the HTTP method e.g. 'GET' or 'POST'.

To process the request, the callback function should return the message with only the following items changed. Note that only elements `[4 5 6 10]` are always required, and it is important to set element `[4]` to 1.

`[4]`	Handle	1
`[5]`	Status	Success is indicated by 200.
`[6]`	Message	Success is indicated by `'OK'`.
`[7]`	MIME	Defaults to `'text/html'` and need be specified only if the response (Body) is not HTML.
`[9]`	Headers	Not normally required.
`[10]`	Body	Typically this will contain HTML.

MIME types include:

- text/html
- text/css
- text/plain
- text/csv
- application/javascript
- application/xml
- application/json

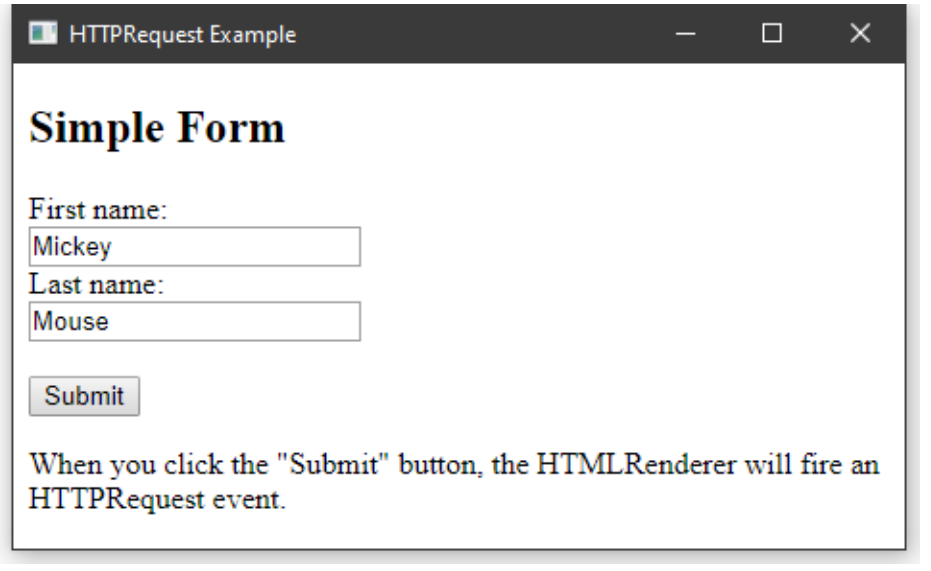
For a complete list of media/MIME types, see:

https://www.iana.org/assignments/media-types/media-types.xhtml:

Example

```
<!DOCTYPE html>
<html>
<head>
<Title>HTTPRequest Example</Title>
</head>
<body>

<h2>Simple Form</h2>

<form action="Hello">
  First name:<br>
  <input type="text" name="firstname" value="Mickey">
  <br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse">
  <br><br>
  <input type="submit" value="Submit">
</form>

<p>When you click the "Submit" button, the HTMLRenderer
 will fire an HTTPRequest event.</p>

</body>
</html>
```

```
     ∇ msg←SayHello msg;url;names;first;last;response
[1]    url←8⊃msg
[2]    'Requested URL is: ',url
[3]    names←(url∊'?&')⊂url
[4]    first last←{(⍵⍳'=')↓⍵}¨names
[5]    response←'<!DOCTYPE html><html><head>'
[6]    response,←'<Title>Hi Folks</Title>'
[7]    response,←'</head><body>'
[8]    response,←'<h1 align="center">Hello '
[9]    response,←first,' ',last,'</h1>'
[10]   response,←'</body></html>'
[11]   msg[4 5 6 10]←1 200 'OK'response
     ∇

     ∇ run
[1]    'hr'⎕WC'HTMLRenderer'
[2]    hr.Size←25 25
[3]    hr.HTML←html
[4]    hr.onHTTPRequest←'SayHello'
     ∇

          run
Requested URL is: http://dyalog_
root/Hello?firstname=Mickey&lastname=Mouse
```


HTTPRequest Example
Simple Form
First name:
Mickey
Last name:
Mouse
Submit
When you click the "Submit" button, the HTMLRenderer will fire an HTTPRequest event.


Hi Folks
Hello Mickey Mouse

InterceptedURLs Property

Applies To: HTMLRenderer

Description

The InterceptedURLs property is a 2-column matrix that specifies whether the HTMLRenderer will attempt to satisfy a request for a resource from the workspace or, via the CEF, from the internet. If directed to the workspace, the request will trigger an HTTPRequest event if the protocol is `http`, or a WebSocketUpgrade event if the protocol is `ws`.

The first column is a wild-carded character scalar or vector containing a pattern to match. The second column is a Boolean indicating whether or not the HTMLRenderer should trigger an event. InterceptedURLs may contain any number of rows.

If the requested url is a relative rather than an absolute url, it is prepended by the string `http://dyalog_root/`. So, for example, if the HTML property contained :

```
<link rel="stylesheet" href="style.css">
<script src="app.js"></script>
```

the HTMLRenderer will request `http://dyalog_root/style.css` and `http://dyalog_root/app.js` respectively.

When the value of InterceptedURLs is its default (`(0 2⍴'')` it is treated as if it were set to `((1 2⍴'*://dyalog_root/*' 1)`. So by default, requests for a relative url will fire an event in the workspace while absolute urls will be directed by the CEF to the internet.

Note that if code in the page creates a web socket intended for internal use, with anything other than `dyalog_root` as the url, the url must match a pattern in InterceptedURLs with 1 in the second column. The following example does not require a matching pattern in InterceptedURLs.

```
// Create a new WebSocket.
 window.socket = new WebSocket('ws://dyalog_root/');
```

Examples:

The following will trigger an 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
```

PrintToPDF Method 845

Applies To: HTMLRenderer

Description

This method writes the content displayed in an object to a specified file in Portable Document Format (pdf).

The argument to PrintToPDF is a simple character scalar or vector containing a file name. Note that the method does not add any extension to the file name that is supplied.

The method returns a Boolean result which indicated whether or not the operation succeeded. If the file name contains a directory path, the path must already exist. The user must have permission to write the file.

SelectCertificate Event 848

Applies To: HTMLRenderer

Description

This event is triggered when HTMLRenderer requests a resource from a server that requires a certificate.

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 7-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'SelectCertificate' or 848
[3]	Index	Integer (see below)
[4]	Addr	Host address
[5]	Port	Host port
[6]		'is proxy'
[7]	Certificates	See below

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

Name	Description
DER	Distinguished Encoding Rules. Character Vector.
Subject	Namespace (see below)
Issuer	Namespace (see below)
SerialNumber	Integer

The **Subject** and **Issuer** namespaces contain the following variables:

Name	Description
CommonName	Character vector
CountryName	Character vector
DisplayName	Character vector

The application should respond to this event by selecting a certficate from the list of available certificates reported by the 7th element of the event message. This is done by having a callback function that sets the 3rd element of the event message (Index) to the 0-origin index in Certificates and returns the event message as its result.

Example

```
      ∇ arg←cb arg
[1]    ⍝ SelectCertificate callback function
[2]     arg[3]←0 ⍝ Select the first certificate
      ∇
```

ShowDevTools Method 849

Applies To: HTMLRenderer

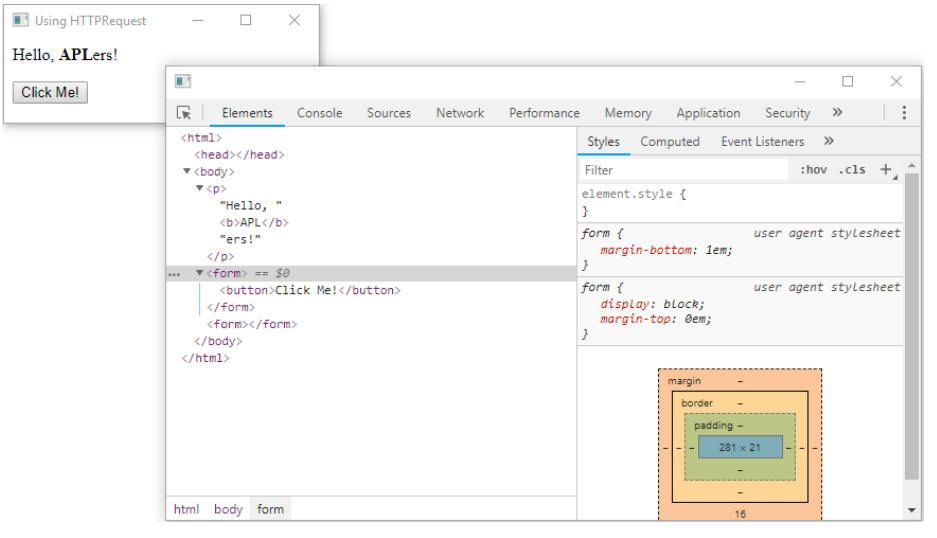
Description

This method displays or hides the CEF development tools window that displays information about the content being displayed by the HTMLRenderer.

It takes a Boolean argument 0(hide) or 1(display).

Using the example illustrated in the HttpRequest topic ...

```
      hr.ShowDevTools 1
```



URL Property

Applies To: HTMLRenderer

Description

The URL property is a character vector that specifies the url of a resource to be requested by the HTMLRenderer. Its default value is an empty character vector.

Specifying URL is an alternative way to setting the HTML property in order to display content in the HTMLRenderer.

When you set the URL property, the HTMLRenderer will request the corresponding resource (from either the internet or the workspace via an HTTPRequest event) and the display will change according to the response. The HTML property is ignored and remains unchanged.

When you set the HTML property, the content of the HTMLRenderer will change accordingly. The current value of the URL property is ignored and remains unchanged.

If you set BOTH URL and HTML in the same statement, the value of URL takes precedence and the assignment to HTML is ignored (it remains unchanged).

WebSocketClose Event 843

Applies To: HTMLRenderer

Description

This event is triggered when the HTMLRenderer client closes the WebSocket. It is for notification only.

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 4-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'WebSocketClose' or 843
[3]	ID	Character vector containing the ID of the WebSocket

When used as a method, the result is 0.

Example

```
hr.WebSocketClose '223d0f781e95113'
```

WebSocketError Event 844

Applies To: HTMLRenderer

Description

This event is triggered an error occurs on the WebSocket. It is for notification only.

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 4-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'WebSocketError' or 844
[3]	ID	Character vector containing the ID of the WebSocket
[4]	URL	The remote url (character vector)

WebSocketReceive Event 842

Applies To: HTMLRenderer

Description

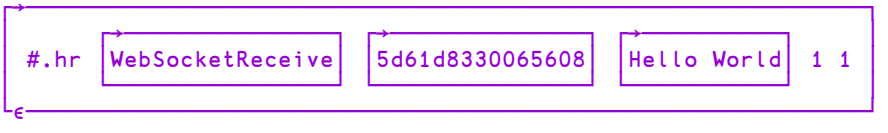
This event is triggered when data is received over a WebSocket. This event is reported for information only. The result (if any) of a callback function will be ignored.

Note that the WebSocket protocol provides for the possibility for the data to be sent in chunks, causing a succession of WebSocketReceive events. The FIN bit of the last chunk will be 1. The CEF does not currently implement "chunking", so FIN will always be 1.

The event message reported as the result of □DQ, or supplied as the right argument to your callback function, is a 6-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'WebSocketReceive' or 842
[3]	ID	Character vector containing the ID of the WebSocket
[4]	Data	Character or integer vector.
[5]	FIN	Boolean. 1 indicates that this is the last chunk; 0 that there is more to come.
[6]	Datatype	1 = character, 2 - numeric

Example



WebSocketSend Method 847

Applies To: HTMLRenderer

Description

This method is used to send data to a WebSocket. The argument to WebSocketSend is a 2 or 3-element vector as follows:

[1]	ID	Character vector containing the ID of the WebSocket
[2]	Data	Character or integer vector. Integers must be in the range 0-255.
[3]	FIN	Boolean. 1 indicates that this is the last chunk; 0 that there is more to come. This is not currently supported by the CEF and should be omitted.

The result is 0.

Example

```
      hr.WebSocketSend  '5d61d8330065608'  'Hello World'
0
```

WebSocketUpgrade Event 841

Applies To: HTMLRenderer

Description

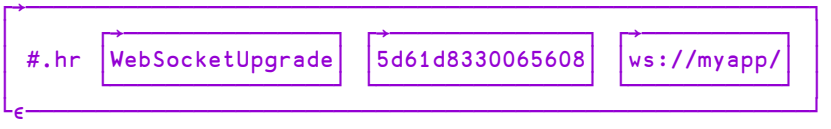
This event is reported when the client component of an HTMLRenderer object opens a WebSocket and the requested URL matches a pattern specified by the InterceptedURLs property. If there is no match, the connection request is processed as an external request by the Chromium Embedded Framework (CEF)[1].

The protocol for establishing the connection is handled internally then this event is reported when the connection has been made. The WebSocket ID is subsequently required to send a message by calling the WebSocketSend method or to close the connection using the WebSocketClose method. Note that several WebSocket connections may be made concurrently. Should the connection fail, a WebSocketError event will be reported instead.

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 4-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'WebSocketUpgrade' or 841
[3]	ID	Character vector containing the ID of the WebSocket
[4]	URL	The requested url of the WebSocket

Example



[1] https://en.wikipedia.org/wiki/Chromium_Embedded_Framework

Chapter 3:

Non-Windows Specific Features

Summary

This section summarises the changes specific to Dyalog APL Version 17.1 on non-Windows platforms. This list currently consists of:

- AIX
- Linux (including the Raspberry Pi)
- macOS/ Mac OS X

Hardware/Software Requirements

These details are subject to change, particularly for Linux. It is possible that using the HTMLRenderer will require more recent versions of distributions than are required for non-HTMLRenderer usage. More information can be found on the Dyalog Forums.

Our policy is, where possible, to support operating system versions which will still be in standard support within 2 or 3 months of the release date of a version of Dyalog APL.

RIDE and Dyalog APL 17.1

Dyalog Version 17.1 supports RIDE 4 only; earlier versions of RIDE are not supported. Dyalog recommends that RIDE 4.2 is used by. RIDE 4.2 can be used with Versions 17.0 and 16.0 too.

RIDE 4.2 is supported on Raspberry Pi models 2 and 3 only; models Zero and 1 are not supported (the underlying libraries which RIDE is built on are not available for the Pi Zero and 1). The *Dyalog RIDE Reference Guide* details how to configure the APL session to support the underscored alphabet; contact support@dyalog.com if you wish to be able to generate key-chords which result in the underscored alphabet being entered into APL.

Note that on Linux and Pi, if RIDE 4.2 is installed after Dyalog 16.0 an extra icon will be added to the window manager's start menu which will start Dyalog with a RIDE front end.

HTMLRenderer on Linux

The HTMLRenderer object is now supported in Dyalog Version 17.1 for Linux (not on the Raspberry Pi). See the *HTMLRenderer User Guide* for more infomation. See the Dyalog Forums for information about the pre-reqs needed for using the HTMLRenderer on Linux:

Location of configuration and log files

In Dyalog 17.1 the location of various configuration and log files has been changed so that they are all put in one directory. See the *UNIX Installation and Configuration Guide* for more information.

SQAPL on macOS

Dyalog 17.1 for macOS includes support for SQAPL. However, it is necessary to install iODBC and suitable drivers for your database before SQAPL can work. *The SQL Interface Guide* describes the steps that are typically necessary to get SQAPL connected to a MySQL database.

Index

U

V

W