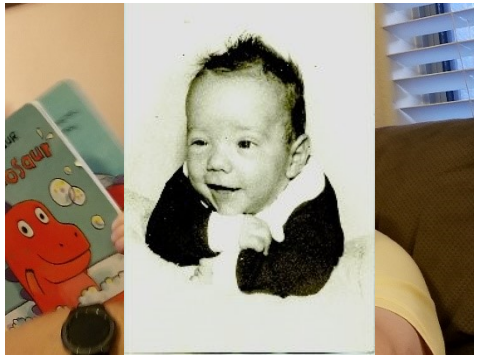
DYALOG
Eastbourne 2026

Web Services Overview

Brian Becker



About me...



- Learned APL in high school in 1975 (age 14)
- Have worked for 3 APL implementors – IPSA, STSC, Dyalog and a plethora of other companies
- Owner of Blue Dolphin Solutions, LLC since 1983
- Working with Dyalog since 2010, full-time since 2011
- APL Tools Architect with a focus on web-enablement

Agenda

- Three tools at a glance
  - WebSocketClient
  - HttpCommand
  - Jarvis
- Choosing the right tool combination
- Resources

The toolkit

Three tools cover every way APL talks to the web



HttpCommand

APL calls the web

Send a request, get a response: REST APIs, JSON services, file downloads and uploads.



Jarvis

The web calls APL

Publish APL functions as JSON or REST endpoints, serve static pages, push live updates.



WebSocketClient

APL talks both ways

Keep one connection open and send messages in both directions as they happen.

WebSocketClient

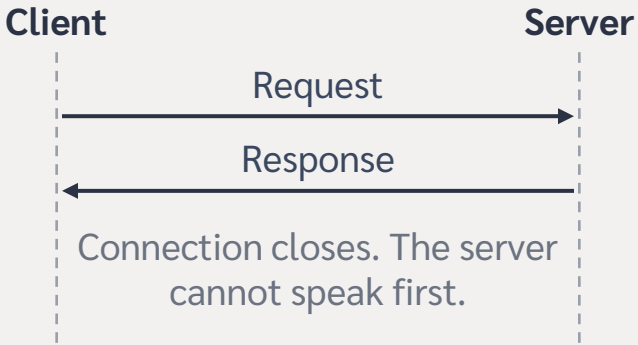
A live, two-way connection from APL to a server

- **Client side of the connection:** APL opens and holds a WebSocket
- **Two-way messaging:** either side can send at any time
- **No polling:** messages from the server arrive without APL asking for them
- **Talks to any WebSocket server:** including Jarvis
- **Version 1.0:** Released September 2026

HTTP vs WebSocket

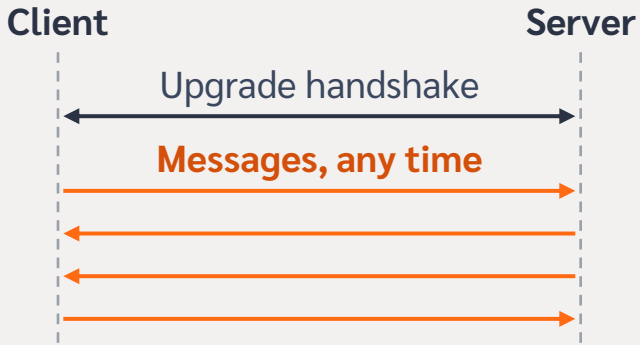
HTTP

Ask, answer, hang up



WEBSOCKET

Connect once, talk both ways



WebSocket message formats

Your application chooses the format. Both ends agree on it.

Text or binary

Plain text, JSON, XML or a custom binary format.

JSON examples

```
{"type":"join","user":"Drake"}
{"type":"msg","text":"Hello world!"}
```

Message Fragments

A long message can be split across several frames. Each frame carries a final flag, set only on the last one, so the receiver knows when the message is complete.

WebSocketClient

Enables APL to hold an open, two-way connection

STEP 1
Write a hook
Your function runs on each incoming message

STEP 2
Create
Make an instance and set the server URL

STEP 3
Connect
Open the socket and start listening

STEP 4
Talk, then close
Send messages any time; close when done

```
ws←WebSocketClient.New (URL:'wss://someurl.org' ⋄ OnWsReceive:'Receive')
ws.Connect
ws.Send 'Hello world'
```

Also handles **secure connections, custom headers and authentication, proxies, redirection**

HttpCommand

Call any web service from APL

- **Ships with Dyalog:** `]load`, `]get`, or `]import` it into any workspace
- **Every HTTP method:** GET, POST, PUT, PATCH, DELETE, …
- **JSON both ways:** `TranslateData←1` turns arrays into JSON and back
- **HTTPS built in:** Conga handles the secure connection

```
      ]load HttpCommand
      url←'api.github.com/repos/dyalog/jarvis'
      r←HttpCommand.GetJSON 'get' url
      r.Data.name
Jarvis
```

HttpCommand Uses

- Interact with web services
- Web scraping
- Upload files
- Download files
- Interact with LLMs as web services through their APIs

Some HttpCommand features

Data translation

`TranslateData←1` handles JSON, XML and form data

Credentials

Header substitution keeps secrets out of your code

Save to file

`OutFile` writes a download straight to disk

HTTPS

Secure connections, with client certificates when needed

Authentication

Basic and bearer token auth built in

Event streams

`EnableSSE←1` reads text/event-stream

Server Sent Events (SSE)

HTTP connection stays open and the server sends events as they happen

On the wire

Content-Type: text/event-stream

```
event: quote
data: {"px":42.1}

data: {"px":42.3}
```

Plain text. A blank line ends each event.

How it works

Ordinary HTTP, held open

- One request asks for `text/event-stream`
- The server holds the response open and writes events
- Each event: `data:` lines, optional `event:` and `id:`
- Browsers reconnect and resume from the last `id`
- One way only: server to client

SSE in HttpCommand and Jarvis

HttpCommand reads an event stream; Jarvis writes one

HttpCommand

APL receives a stream

- `EnableSSE←1` reads events as they arrive
- Works with any text/event-stream service
- LLM APIs using streams reply token by token

Jarvis

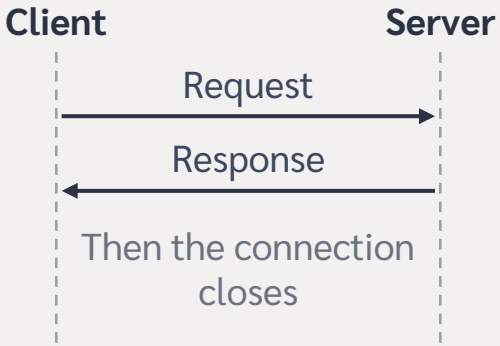
APL sends a stream

- `SendSSE` pushes events to selected listeners
- Browsers reconnect on their own
- Feeds live dashboards, tickers and progress bars

HTTP vs WebSocket vs SSE

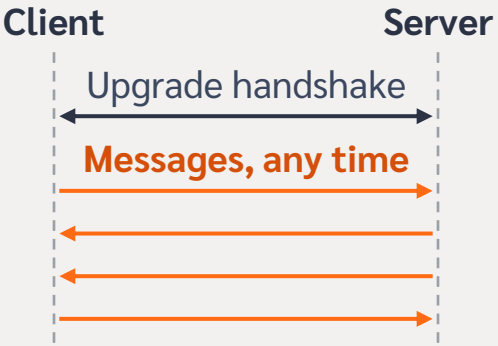
HTTP

Ask, answer, hang up



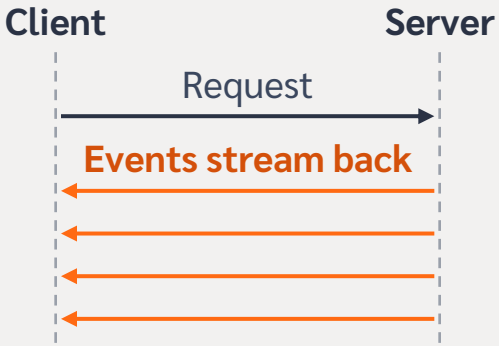
WEBSOCKET

Talk both ways



SSE

Ask once, then listen



HttpCommand, SSE and LLMs

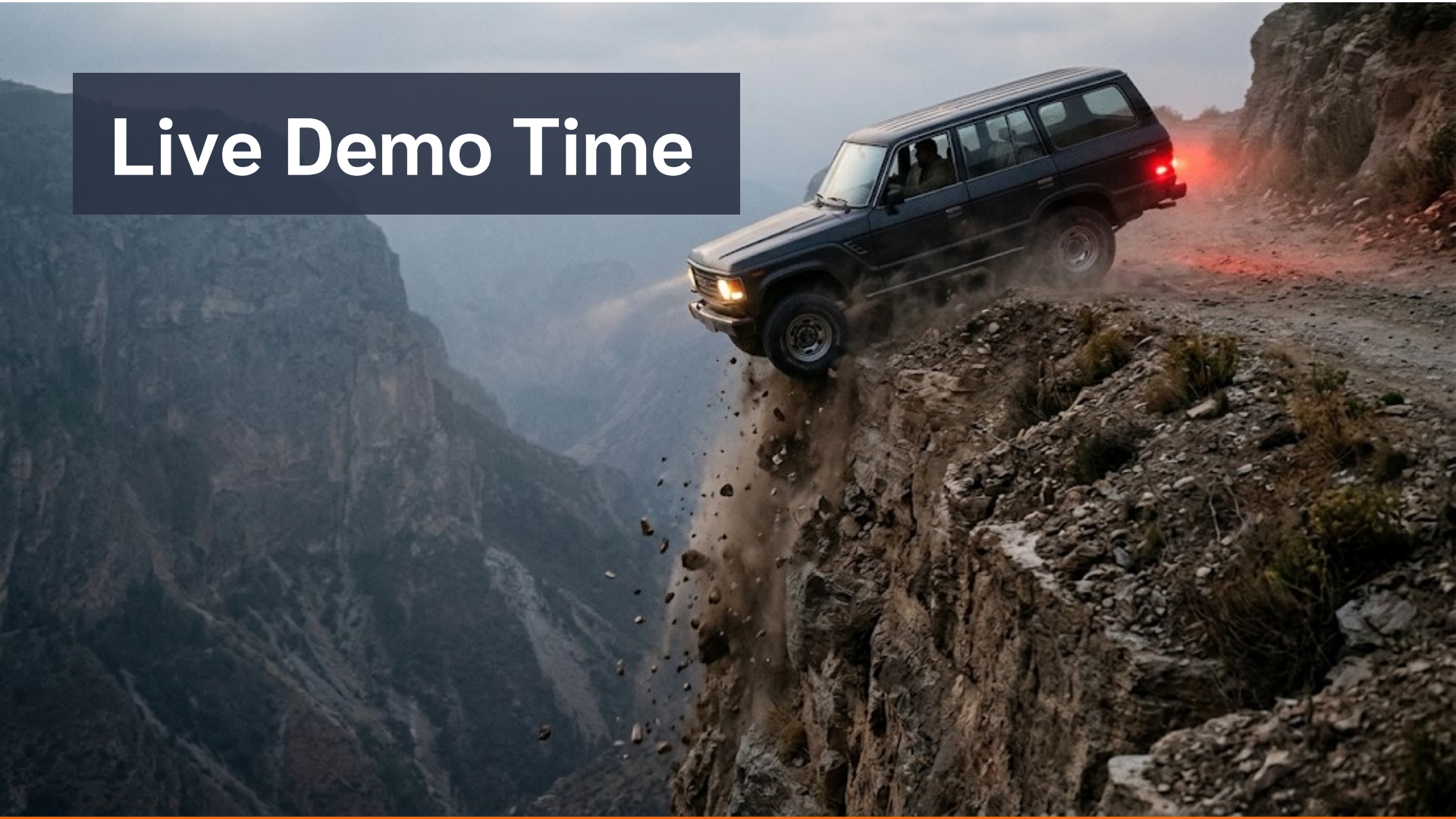
```
h←HttpCommand.New (EnableSSE:1)
```

Why stream?
A model writes its reply one token at a time. Streaming shows the first words very quickly, not after the whole answer is finished.

How it works
Set `EnableSSE←1`. The server replies with Content-Type text/event-stream, keeps the connection open and sends each chunk as an event.

Why not WebSocket?
One request, one-way reply: plain HTTP is enough, and it passes through proxies and load balancers unchanged.

Live Demo Time



Jarvis

- Expose ordinary APL functions as HTTP endpoints with very little ceremony, so the rest of the world can call your APL code without knowing it's APL.

What is an endpoint?

A URL that a client can call, such as `http://foo.org/sum`. `sum` is the endpoint
Jarvis maps each endpoint to an APL function:
the request payload becomes its argument and its result becomes the response.

Jarvis

- Expose ordinary APL functions as HTTP endpoints with very little ceremony, so the rest of the world can call your APL code without knowing it's APL.
- You write normal APL and Jarvis handles the plumbing.
- Many new projects use Jarvis
- Used internally for TryAPL and DCMS

Jarvis Paradigms

A Jarvis service runs one paradigm: JSON or REST

JSON

Endpoints are APL functions

- Each function name is a URL
- Payloads are sent as JSON
- Jarvis translates from JSON to APL and back

```
sum←⊃+/
POST /sum  [[1,3,5],[2,4,6]]
[3,7,11]
```

REST

Endpoints are resources

- HTTP method sets the action
- One function per HTTP method
- Cumbersome to implement
  - Stark to the rescue!

```
GET /customers/10
r←GET request
{"id":10,"name":"Acme"}
```

A Sampling of Jarvis Features

User Hooks

Inject your code to customize behaviors

CORS

Serve browser apps on other domains

Static files

Serve HTML, CSS and JavaScript too

Sessions

Track state across a user's requests

Live push

WebSockets and SSE stream updates

Docker

Deploy with your own container based on the dyalog/jarvis image

WebSockets vs SSE

Choose SSE when only the server needs to talk

	WebSockets	Server-Sent Events
Direction	Both ways, any time	Server to client only
Connection	Upgrades HTTP to ws:// or wss://	Plain HTTP stream
Data	Text and binary	UTF-8 text only
Reconnection	Your code handles it	Browser reconnects on its own
Jarvis function	WsSend	SendSSE
Best for	Chat rooms, interactive games	Dashboards, stock tickers


Live Demo Time

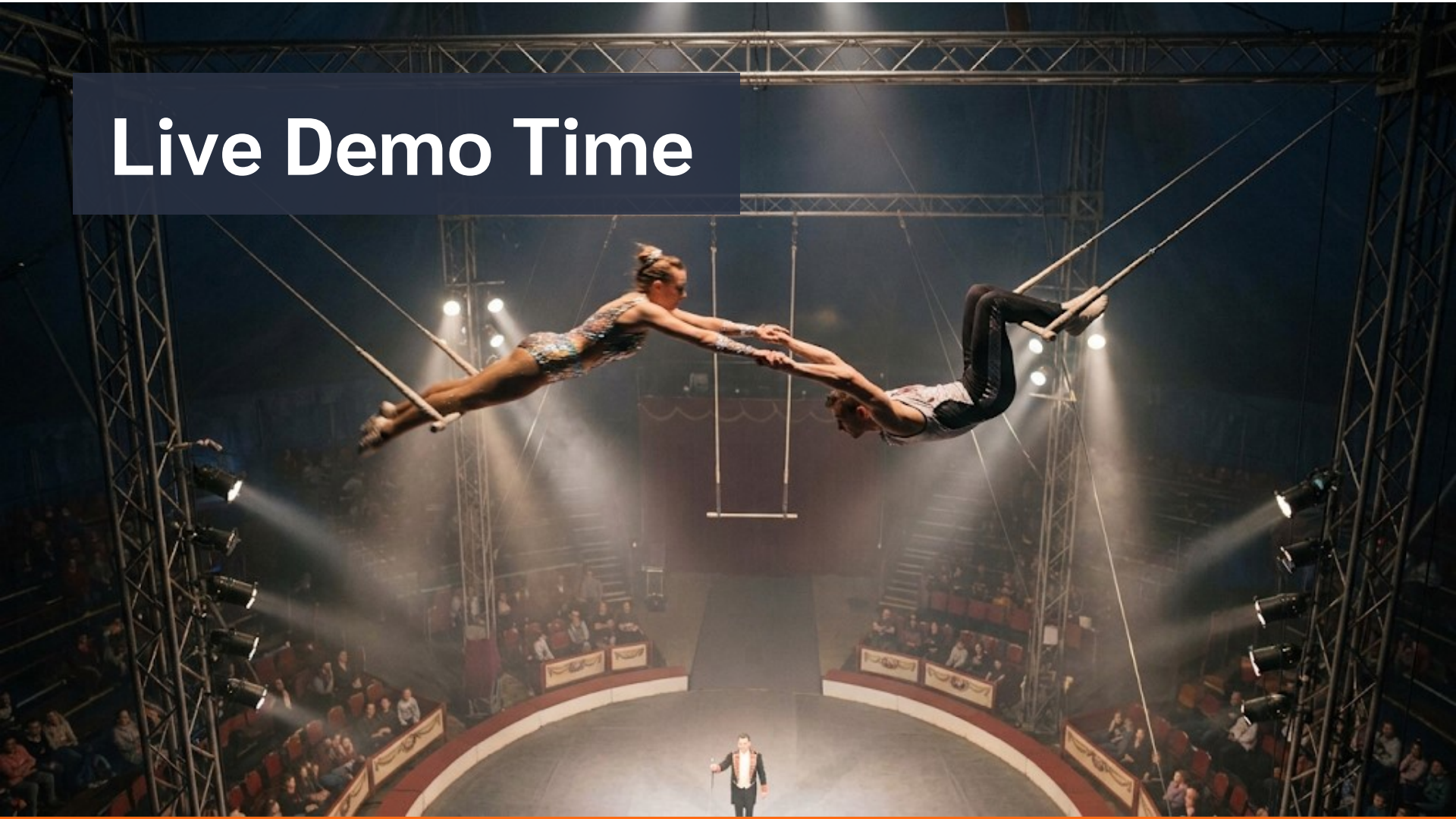
Jarvis and AI – an update

- At DYNA26 I gave a presentation where I asked Google's Gemini to build a Wordle®-like web service.
  - After several iterations (and a little nudging on my part) it came up with a very respectable solution.
- Recently I used Claude and VSCode to attempt the same thing...

Build a Wordle-like web application using Dyalog Jarvis as its backend

- Use the color scheme from Dyalog.com as much as possible
- Generate a sample "dictionary" of 100 words to guess from
- There should be an onscreen keyboard
- There should be a "New Game" button to start a new game
- Use Jarvis's SSE support to send encouraging messages every 30 seconds or so
- Congratulate a successful guess
- Offer a consoling message if the user doesn't guess the word
- The game should be responsive so that it works on laptops, phones, tables, etc.
- Wordle is trademarked name owned by the New York Times Company. Give the application an APL-ish name that won't conflict with trademark issues.
- In portrait orientation on a monitor, there's too much space between the heading, the game board, and the keyboard.

Live Demo Time



Choosing a tool

Pick the tool by who starts the conversation

Who starts	Your situation	Use
APL	APL needs data from a web API	**HttpCommand**
The client	Browsers or programs call your APL code	**Jarvis**
APL pushes	A web page needs live updates from APL	**Jarvis + SSE**
Either side	Browsers and APL message both ways	**Jarvis + WebSockets**
Remote server	APL listens to an outside WebSocket feed	**WebSocketClient**
Both, later	Long job: reply now, call back when done	**Jarvis + HttpCommand**

Resources

Documentation, source and hands-on exercises

	Documentation	Source
HttpCommand	dyalog.github.io/HttpCommand	github.com/Dyalog/HttpCommand
Jarvis	dyalog.github.io/Jarvis	github.com/Dyalog/Jarvis
WebSocketClient	dyalog.github.io/WebSocketClient	github.com/Dyalog/WebSocketClient

Workshop: Asynchronous Services

Brian Becker and Holden Hoover, 12 October. Build a web hook, a live SSE ticker and a WebSocket chat with all three tools.

github.com/dyalog-training/2026-MP2