



DYNA Fall 2025
A **DYALOG** USER MEETING

JAWS

Jarvis And WebSockets

Brian Becker
APL Tools Architect
Dyalog, LTD

Why Jarvis?

- Jarvis is a Web Service framework that helps you:
 - Expand your audience
 - Make it easier for you to make your APL application or data accessible
 - APL not required for the client
 - Lessen the need to learn underlying technologies (HTTP, etc)
 - Yet, provide low-level access if needed
 - Assume little - maximize flexibility for you

Jarvis

- ❖ JSON and REST Servis
- ❖ Operates in one of two modes
- ❖ JSON mode makes your APL functions accessible
- ❖ REST mode makes your data accessible

Jarvis

- ❖ JSON mode
 - ❖ Endpoints are APL functions
 - ❖ <https://someurl.com/compute> → corresponds to APL function "compute"
- ❖ REST mode
 - ❖ Endpoints are resources (real or virtual) to be retrieved and/or manipulated
 - ❖ <https://someotherurl.com/customers/123> → refers to customer with ID 123

A Web Service in 5 minutes...

Jarvis in Use

- JSON mode

- Most users use JSON mode

- <https://tryapl.org>

- REST mode

- Some REST mode users

- <https://dcms.dyalog.com>

- <https://dyalogprod.gos.dyalog.com/video-library/>

JSON Mode

- Endpoints are APL functions

$\text{avg} \leftarrow (+\neq) \div \neq$

$\text{avg} \leftarrow \{ (+\neq \omega) \div \neq \omega \}$

$\nabla \text{ r} \leftarrow \text{avg } \text{w}$

$[1] \text{ r} \leftarrow (+\neq \text{w}) \div \neq \text{w}$

∇

JSON Mode

- Endpoints are APL functions
- Client issues an HTTP POST request with a JSON payload
`'[3,1,4,1,5,9,2,6]'`
- Jarvis parses the request and converts the payload to an APL array
`3 1 4 1 5 9 2 6`
- Jarvis calls your function, passing the APL array as the right argument
- Your function returns an APL array
- Jarvis converts the APL array to JSON
- Jarvis formats and sends a proper HTTP response with the JSON result

JSON Objects $\equiv$ APL Namespaces

JSON: `{"name":"Jean", "bday":[1,23]}`

APLAN: `(name:'Jean' ⋄ bday:1 23)`

```
zodiac←{
  cusps←120 219 321 420 521 621 723 823 923 1023 1122 1222
  signs←13p'Capricorn' 'Aquarius' 'Pisces' 'Aries' 'Taurus'...
  w.sign←(1+cusps⊥100⊥w.bday)⊃signs
  w }
```

APLAN.Output on
'Was OFF'

```
zodiac (name:'Jean' ⋄ bday:1 23)
(bday:1 23 ⋄ name:'Jean' ⋄ sign:'Aquarius')
```

REST

Create, Read, Update, Delete

- Uses standard HTTP methods for CRUD

REST

- ✧ Uses standard HTTP methods for CRUD
 - ✧ GET – retrieve
 - ✧ POST – create
 - ✧ PUT – update/replace
 - ✧ DELETE – delete
 - ✧ PATCH – partial update

REST

- Uses standard HTTP methods for CRUD
 - GET /customers
 - GET /customers/123
 - DELETE /customers/123
 - POST /customers/345
{ "name":"fred", "dob":"19621031" }
 - PATCH /customers/345
{ "name":"bob" }
- GET – retrieve
- POST – create
- PUT – update/replace
- DELETE – delete
- PATCH – partial update

REST

- You write an APL function for each HTTP method you want your service to support
 - For a "read-only" service, you would write only a "Get" function
- Jarvis passes all the client request information to your function

REST

- ✧ Designing a consistent REST schema requires thought and effort
- ✧ Truly RESTful APIs have additional qualities like cacheability and statelessness
 - ✧ It's left up to you to decide how, to what extent, or even whether, to implement.
- ✧ Many "REST" web services are not truly RESTful

REST Example - DCMS

```
url ← 'https://dcms.dyalog.com/videos'
```

```
⊢ r←HttpCommand.GetJSON (Command:'get' ⋄ URL:url ⋄ Timeout:60)
```

```
[rc: 0 | msg: | HTTP Status: 200 "OK" | #Data: 700]
```

```
↑(3↑r.Data).(presenter event youtube_id)
```

```
Adám Brudzewsky APL Quest Mj4wyLKrBho
```

```
Adám Brudzewsky APL Quest w-rzx2VNqbY
```

```
Adám Brudzewsky APL Quest pxo2BtoMxP4
```

```
url ← url, '/Mj4wyLKrBho'
```

```
⊢ r←HttpCommand.GetJSON 'get' url
```

```
[rc: 0 | msg: | HTTP Status: 200 "OK" | #Data: 1(namespace)]
```

```

▽ res←Get req;allowed;nl
[1]   :If GLOBAL.debug   A :If here because prefer not to compute  $\tau$ TS for every request
[2]   □←'> GET REQUEST at ', $\tau$ TS
[3]   □←req.Endpoint
[4]   last←req
[5]   :EndIf
[6]   :Trap GLOBAL.debug↓0
[7]   req.Endpoint←{ $\omega$ / $\sim$ 1( $\vdash\vee\phi$ )'/' $\neq\omega$ }req.Endpoint   A Remove multiple slashes
[8]   :If req.Endpoint( $\supset\epsilon\sim$ )'/videos'
[9]   res←read.videos.Handle req
[10]  :ElseIf 1=+/req.Endpoint $\equiv$ ''/person' '/organisation' '/event' '/event_type' ...
[11]  res←read.Table 1↓req.Endpoint
[12]  :ElseIf 1=+/req.Endpoint $\equiv$ ''/presenters' '/dtv_events'
[13]  res←CACHE $\oplus$ 1↓req.Endpoint
[14]  :ElseIf req.Endpoint $\equiv$  '/version'
[15]  res←Version
[16]  :Else
[17]  req.Fail 404
[18]  :EndIf
[19]  :Else
[20]  'Internal Server Error'req.Fail 500
[21]  :EndTrap

```

▽

Clients

- ◆ Anything that can "speak" HTTP
 - ◆ Microservices
 - ◆ APL applications
 - ◆ Web Front Ends (JavaScript, React, Vue, etc)
 - ◆ Python, C#, Java
 - ◆ Mobile Apps
 - ◆ Command line tools (curl, httpie)

UI-agnostic

- Jarvis has a simple built-in HTML interface to allow you to test JSON mode endpoints – that's it.
- Presentation is left up to the client
 - Service-based applications may not need any UI
- Let your UI people do what they do best

"Hook" Functions

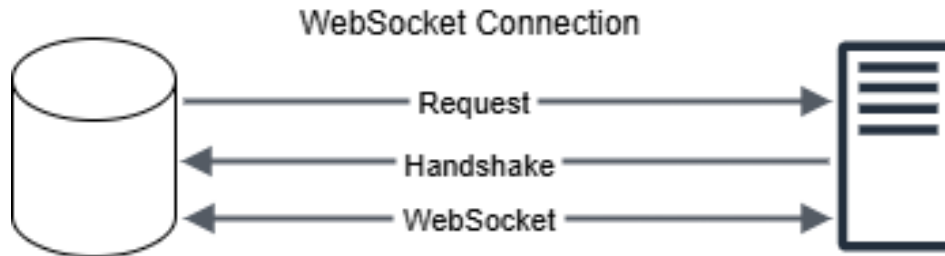
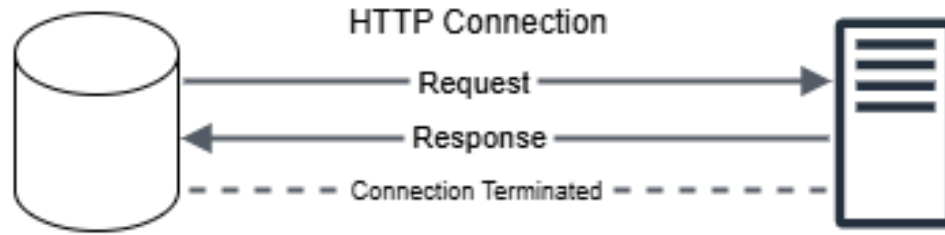
- Application Initialization
- Request Validation
- Authentication
- Session Initialization
- Request Post Processing
- Application Shutdown

Okay, but...

- ❖ JSON and REST modes work well for Request-Response
- ❖ What about more dynamic interactions?
 - ❖ Server needs to push data to the clients
 - ❖ Financial, Chat, Gaming, Monitoring
 - ❖ Long running requests
- ❖ That's where WebSockets come in...

WebSockets

- An HTTP connection can be "upgraded" to a WebSocket
 - The HTTP connection is established from the Client to the Server
 - The Client requests to upgrade the connection to a WebSocket
 - If the Server approves the upgrade, it sends an upgrade response
 - The connection is now bi-directional (full-duplex)



HTTP

- Request-Response (client initiates)
- Short-lived; closes after each request
- Higher message overhead (headers, etc)
- Higher latency (new connection per request)
- Use Cases
 - Web pages
 - Web service APIs

WebSocket

- Full-duplex (both sides can send anytime)
- Persistent; stays open until explicitly closed
- Lower overhead after initial handshake
- Lower latency (persistent connection)
- Use Cases
 - Real time data (trading, chat, gaming, monitoring)
 - Server needs to "push" data to client
 - Long running endpoints

WebSockets in Jarvis

- More "Hook" functions
 - Authentication
 - AutoUpgrade
 - AutoUpgradeReq
 - Receive
 - Close
 - Error

Demo Time

Questions?