

## SA3 Web Services Workshop Setup

Note: The instructions here have been modified from the those distributed at the workshop itself. In the workshop we used API keys that were set to expire shortly after the workshop. As those keys have now expired, you will need to generate your own key to access the GitHub API if you want to do those exercises.

### Get the Workshop Materials

1. Download SA3.zip from <https://github.com/Dyalog-Training/2024-SA3/releases/latest>
2. Unzip SA3.zip from one of the flash-drives to your computer.
3. Remember what folder you unzipped it to!

!!! Whenever you see an example referring to /SA3/, you'll use **your folder name** instead.

### If you want to do the exercises that use the GitHub API:

1. If you don't already have a GitHub account, create one. <https://github.com>
  - a. You'll need to remember your account user ID because you'll need it for some of the exercises.
2. Once you have signed into GitHub, go to <https://github.com/settings/personal-access-tokens/new>
  - a. Give your token a name
  - b. Set an expiration date. If you're planning on using this token just for these exercises, it is probably wise to set a short expiration date.
  - c. Specify the repositories for this token by selecting "All repositories".
  - d. Under Repository permissions, specify "Read and write" access level for the following entries:
    - i. Administration
    - ii. Commit statuses
    - iii. Contents
  - e. Click on "Generate token" at the bottom of the page
  - f. Make sure you copy the token since once you leave this screen the token will never be displayed again. You can insert the token in the GitHubAPIKey function that can be found in Materials folder in the unzipped contents.

### If you want to do the exercises that use the Google Maps API:

1. If you don't already have a Google account, create one. <https://google.com>
2. You'll need to create a project, enable the Google Maps API, and get an API key. Information for doing this can be found at: <https://developers.google.com/maps/get-started/>
3. You can insert the token in the GoogleAPIKey function that can be found in Materials folder in the unzipped contents.

## HttpCommand Using a REST API Exercise

### Setup

Load the material for this workshop (replacing /SA3/ with the folder you unzipped to).

```
]link.import # /SA3/Materials
```

You'll need to update GitHubAPIToken as described in the Workshop Setup page.

### Generically Configure the Request

Because we'll be issuing several requests to the GitHub API, we can set up a request object that we can reuse by changing its settings. This will save us from having to re-specify a number of settings that will be common to all the requests we send.

```
h←HttpCommand.New ''
h.BaseURL←'https://api.github.com'
'X-GitHub-API-Version' h.SetHeader '2022-11-28'
h.(AuthType Auth)←'bearer' GitHubAPIKey
h.ContentType←'application/json'
h.TranslateData←1 # translate JSON responses into APL
```

### Test Your Request

```
h.(Command URL)←'get' 'user/repos'
resp←h.Run
resp.IsOK # should return a 1
```

### Create a GitHub Repository

Now that we have a request generically configured, we can specify the particular settings to create a repository.

```
h.Command←'post'
h.URL←'user/repos'
p←[]NS ''
p.(name description)←'your-repo-name' 'some description'
h.Params←p
h.Show
r←h.Run
```

Documentation for the GitHub "Create a repository for the authenticated user" can be found [here](#).

## Update Your GitHub Repository

We're going to change the visibility of your repository from public to private.

```
h.Command←'patch'
h.URL←'repos/YOUR-GITHUB-USERID/your-repo-name'
p←⎕NS ''
p.(description visibility)←'new description' 'private'
h.Params←p
h.Show
r←h.Run
```

Documentation for the GitHub "Update a repository" can be found [here](#).

## List the Current Repositories for your GitHub User ID

```
h.Command←'get'
h.URL←'users/YOUR-GITHUB-USERID/repos' ⌘ lists only public repositories
h.Params←'' ⌘ remove previously set parameters
h.Show
r←h.Run
; r.Data.name
```

Documentation for the GitHub "List repositories for a user" can be found [here](#).

```
h.URL←'user/repos' ⌘ lists all repositories for the authenticated user
q←h.Run
r.Data.name ~ q.Data.name ⌘ private repositories
```

Documentation for the GitHub "List repositories for the authenticated user" can be found [here](#).

## Delete Your GitHub Repository

```
h.Command←'delete'
h.URL←'repos/plusdottimes/your-repo-name'
h.Params←''
h.Show
r←h.Run
```

Documentation for the GitHub "Delete a repository" can be found [here](#).

## Bonus Exercise

Construct a request to find out how many contributors there have been to the Dyalog Jarvis repository.

See [here](#).

## Jarvis Exercise 1 – A web service in 5 minutes or so

Jarvis is a framework that makes it easy for an APLer to deploy applications as web services. How easy? Try this...

```
)clear
```

Define an endpoint

```
sum←+/
```

Load Jarvis from the Workshop materials

```
]link.import # /SA3/Materials
```

Create a new Jarvis instance

```
j←Jarvis.New ''
```

And start it up

```
j.Start
```

Let's use HttpCommand to test our service

```
(HttpCommand.GetJSON 'post' 'localhost:8080/sum' (⍳10)).Data
```

Now let's open a browser to test our service

```
]open http://localhost:8080
```

In your browser,

- Notice that the endpoint is "sum".
- Now type [42,23,19,61] in the "JSON Payload" input area, then press "Send".

In the APL session, type:

```
rotate←ϕ
```

And then refresh your browser.

- Now select the "rotate" endpoint in the dropdown menu.
- Press "Send"

Now stop the service.

```
j.Stop
```

## Jarvis Exercise 2 – Debugging a Web Service

Create an endpoint that has an intentional error.

```
    ▽ r←req oops arg
[1]   ...
    ▽
```

Set **j.(Debug ErrorInfoLevel)←0**

Start the service **j.Start**

Open the Jarvis web interface

```
]open http://localhost:8080
```

Select the oops endpoint

Enter 1 in the JSON Payload area.

Click "Send" and look at the response.

Stop the service **j.Stop**

Set **j.ErrorInfoLevel←1**

Start the service again **j.Start**

Click "Send" again and notice the difference in the response

Now try **j.Stop ♦ j.ErrorInfoLevel←2 ♦ j.Start**

Click "Send" again and notice the difference in the response

Set **j.Debug←1**

Enter 1 in the JSON payload area and click "Send"

Switch to your APL session to see the suspension in **oops**

While you're here, take some time to inspect req

- How many **Headers** does req have?
- What does **req.GetHeader 'user-agent'** return?

→0 to exit the endpoint.

Set **j.Debug←0**

Experiment with different ways to error trap an endpoint

Try these endpoints with a JSON payload of "a"

```
sum1←{+/ω,0}  A will get caught by Jarvis' safety net
```

```
sum2←{11::{'payload not numeric' α.Fail 400 ♦ +/ω,0} A better error info
```

```
sum3←{(r←⎕NS'').(msg data)←'' ⍉
```

```
11::r←r.msg←'payload not numeric'
```

```
r←r.data←+/ω,0}  A information returned in the payload
```

### Jarvis Exercise 3 – Authentication in Jarvis

Write a simple function to perform authentication.

```
j.Stop  
  
▽ r←Login req  
[1]  A req is the request object  
[2]  A req.UserID is the userid, req.Password is the password  
[3]  A should return 0 if authentication succeeds or was not necessary  
[4]  ...your code here...  
▽  
  
j.AuthenticateFn←'Login'  
  
j.Start
```

In the examples below, substitute **user** and **pwd** with proper credentials for your Login function.

Try:

```
]open localhost:8080  
HttpCommand.GetJSON 'post' 'localhost:8080/sum' (110)  
HttpCommand.GetJSON 'post' 'user:pwd@localhost:8080/sum' (110)  
h←HttpCommand.New 'post' 'localhost:8080/sum'  
h.ContentType←'application/json'  
h.Run  
h.Auth←'user' 'pwd'  
h.Run  
j.Stop  
j.AuthenticateFn←''
```

## Jarvis Exercise 4 – Maintaining Server-side State with Sessions

```
▽ SessionInit req
[1]  A req is the request object
[2]  A req.Session is the new session namespace
[3]  A rc is optional, but should be 0 if the session initialized properly
[4]  req.Session.Data←0
▽
  j.Stop
  j.SessionInitFn←'SessionInit'
  j.SessionTimeout←.1 A session times out in 6 seconds
  j.SessionPollingTime←.1 A how often to check for timed out sessions?
  j.SessionUseCookie←1 A keep track of the session ID via HTTP cookie
  accum←{α.Session.Data→α.Session.Data,←'mm:ss'(1200⌊)1 ⌊DT<⌊TS}
  j.Start
]open localhost:8080
```

Select the accum endpoint

Enter anything in the JSON Payload

Click "Send"

Repeat this a few times

Wait 10 seconds

Click "Send"

## Jarvis/HttpCommand Exercise – Putting it all together

Working in teams of 2 or 3, build a web service that:

- Has at least one endpoint
- At least one endpoint should use HttpCommand to call a web service and does something with the response payload

The URLs referenced below can be displayed in your APL session using the URLs function (this saves you having to type them in).

```
)clear  
  
]link.import # /SA3/Materials
```

### Some Ideas:

A weather service that takes an address and returns some current weather or forecast information.

- Use Google Maps Geocoding API to get a latitude and longitude for a location  
<https://developers.google.com/maps/documentation/geocoding/requests-geocoding?hl=en>  
You need to use the GoogleAPIKey function to retrieve the API key for this service.
- Then uses the lat/long to call the open-meteo weather forecast API to retrieve some weather data and return it  
<https://open-meteo.com/en/docs>

A service that tells you about the 5 most recently updated GitHub public repositories for a user or organization

- <https://docs.github.com/en/rest/repos/repos?apiVersion=2022-11-28#list-organization-repositories>  
You'll need to use GitHubAPIKey

An HTML page based web service that takes an address and embeds a map (centered around the address) lower in the page.

- Use the Google Maps Static Map API to get the map image  
<https://developers.google.com/maps/documentation/maps-static/start?hl=en>  
You need to use the GoogleAPIKey function to retrieve the API key for this service.
- We've provided an HTML page implemented in function MapPage to get you started.



## Solutions

We have written some sample solutions which can be loaded by

```
]link.import # /SA3/Solutions
```

`GetCurrentWeather` returns the current weather conditions for a given location.

```
(GetCurrentWeather 'Dyalog, LTD').data
interval                900      seconds
precipitation            0        mm
relative_humidity_2m    82        %
temperature_2m          14.1     °C
time                    2024-10-12T13:15 iso8601
wind_direction_10m      245      °
wind_gusts_10m          38.2     km/h
wind_speed_10m          21.1     km/h
```

`Latest5` returns the information about the 5 most recently updated repositories for a user or organization or the `HttpCommand` response namespace is the request was not successful.

```
Latest5 'Dyalog'
AtfIn      GitHub  2024-10-07T10:52:22Z
PracticeProblems GitHub 2024-10-05T20:36:44Z
SSGMon     GitHub  2024-10-03T08:50:48Z
pcr2       GitHub  2024-10-02T14:08:18Z
ullu       GitHub  2024-08-30T07:08:12Z
```

To complete the map page exercise, you will need to create a `Jarvis` instance and set `HTMLInterface` to point to the `MapPage` function which returns the HTML content for the web page. When a request is submitted, the web page calls the `GetMap` endpoint to retrieve the map contents.

```
j←Jarvis.New ''
j.HTMLInterface←'' 'MapPage' A get HTML content from the MapPage function
j.Start
2024-10-12 @ 09.23.06.078 - Starting Jarvis 1.18.4
2024-10-12 @ 09.23.06.112 - Conga copied from C:\Program Files\Dyalog\Dyalog
APL-64 19.0 Unicode/ws/conga
2024-10-12 @ 09.23.06.115 - Local Conga v3.5 reference is #.Jarvis.[LIB]
2024-10-12 @ 09.23.06.126 - Jarvis starting in "JSON" mode on port 8080
2024-10-12 @ 09.23.06.131 - Serving code in #
2024-10-12 @ 09.23.06.135 - Click http://192.168.223.136:8080 to access web
interface
0 Server started
```

## Location

Address or Location Name

Dyalog, LTD

Map It!

## Map

