Hurl HTTP requests as tests

CONCEPTS

A .hurl file is a list of entries: a request, then the response you expect.

Request METHOD URL, headers, body
↓ HTTP
Response HTTP 200
↓ [Captures]
Variables {{name}}
↓ next entry
[Asserts] pass or fail, exit code

Entries run in order, so a capture in one is usable as {{name}} in the next.

RUN

`hurl <file.hurl>`
Run a file

`hurl <a.hurl> <b.hurl>`
Run several files in order

`hurl --test <files>`
Test mode: progress and summary

`hurl --glob "<pattern>"`
Run files matching a glob

`hurl --continue-on-error <file>`
Do not stop at the first failed assert

REQUEST

`GET <url>`
First line: method and URL

`<Name>: <value>`
Header lines straight after the URL

`[Query] <key>: <value>`
Query string params, unencoded

`[Form] <key>: <value>`
URL-encoded form fields

`[BasicAuth] <user>: <password>`
Basic authentication

`{ "key": "value" }`
JSON body, Content-Type set for you

RESPONSE & ASSERTS

`HTTP 200`
Expect this status

`[Asserts]`
Section of checks, one per line

`status < 300`
Status check

`header "<Name>" contains "<text>"`
Header check

`jsonpath "$.a.b" == "<value>"`
Check a JSON value

`jsonpath "$.items" count == <n>`
Check list length

`xpath "//h1" exists`
Check HTML or XML

CAPTURES & VARIABLES

`[Captures]`
Section that stores response values

`id: jsonpath "$.id"`
Capture a JSON value

`tok: header "X-Token"`
Capture a header

`{{name}}`
Use a variable in URL, header, body

`hurl --variable <name>=<value>`
Set a variable

HTTP OPTIONS

`-L, --location`
Follow redirects

`-H, --header <Name:value>`
Add a header to every request

OUTPUT & REPORTS

`-v, --verbose`
Request and response details on stderr

`-h, --help / -V, --version`
Help and version