REST isn't what you think it is

Peter Judge
Software Architect
pjudge@progress.com
whoami

Peter Judge

pjudge@progress.com

Software Architect*

@ Progress (Bedford, USA) since 2003

HTTP in and out: HTTP Client, DataObjectHandler for dataservices

Integration-y stuff

OE Best Practices / OERA / AutoEdge / CCS

4GL since 1996

* Aka programmer who knows PowerPoint
Basics
Hypertext Transfer Protocol (HTTP) is a method for encoding and transporting information between a client (such as a web browser) and a web server. HTTP is the primary protocol for transmission of information across the Internet.

HTTP follows a request-response paradigm in which the client makes a request and the server issues a response that includes not only the requested content, but also relevant status information about the request.

Using HTTP and HTML, clients can request different kinds of content (such as text, images, video, and application data) from web and application servers that host the content.

HTTP resources such as web servers are identified across the Internet using unique identifiers known as Uniform Resource Locators (URLs).

https://www.nginx.com/resources/glossary/http/
Hypertext Transfer Protocol (HTTP) is a method for encoding and transporting information between a client (such as a web browser) and a web server. HTTP is the primary protocol for transmission of information across the Internet.

HTTP follows a request-response paradigm in which the client makes a request and the server issues a response that includes not only the requested content, but also relevant status information about the request.

Using HTTP and HTML, clients can request different kinds of content (such as text, images, video, and application data) from web and application servers that host the content.

HTTP resources such as web servers are identified across the Internet using unique identifiers known as Uniform Resource Locators (URLs).

https://www.nginx.com/resources/glossary/http/
What is REST?

REST = REpresentational State Transfer

REST is an architectural style for network based software that requires stateless, cacheable, client-server communication via a uniform interface between components.

Understood as HTTP with strong* constraints

- **Named resources**: resources are named using a URL
- **Uniform interface**: all resources accessed via generic interface (GET/PUT/POST/DELETE)
- **Interconnected resource representations**: the representations of the resources are interconnected using URLs
- **Stateless**: each request must contain all the information necessary to understand the request; cannot take advantage of any stored context on the server

* https://martinfowler.com/articles/richardsonMaturityModel.html
What's in the box?

- **Classic AppServer**
  - Level 1: Resources
  - Level 2: HTTP Verbs
  - Level 3: Hypermedia Controls
  - all

- **Classic WebSpeed**
  - Level 1: Resources
  - Level 2: HTTP Verbs
  - Level 3: Hypermedia Controls
  - some

- **PASOE / REST**
  - Level 1: Resources
  - Level 2: HTTP Verbs
  - Level 3: Hypermedia Controls
  - none

- **PASOE / WEB**
  - Level 1: Resources
  - Level 2: HTTP Verbs
  - Level 3: Hypermedia Controls
  - all
Our app: Conference planning
A Speaker submits a Talk which is scheduled to a Timeslot in a Room and is given a Rating by an Attendee.
Conference – business logic

<table>
<thead>
<tr>
<th>Talk Logic</th>
<th>Speaker Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_talks.p</td>
<td>find_speaker.p</td>
</tr>
<tr>
<td>schedule_talk.p</td>
<td>get_name.p</td>
</tr>
<tr>
<td>new_talk.p</td>
<td>new_speaker.p</td>
</tr>
<tr>
<td>update_talk.p</td>
<td>update_speaker.p</td>
</tr>
<tr>
<td>list_speaker_talks.p</td>
<td>list_speaker_talks.p</td>
</tr>
<tr>
<td>remove_speaker.p</td>
<td>remove_speaker.p</td>
</tr>
</tbody>
</table>
Level 0: Swamp of POX

plain old xml
Embedded Remote Procedure Call

- Easy on the outside
  - One URL
  - One method/verb

- Less fun inside
  - Actual call method in body somewhere
  - Schema of arguments may be in body somewhere (or linked)

```
POST /api/web HTTP/1.1
Content-Type: application/json
[assorted headers]

{  "method": {  
    "name": "GetTalkById",
    "param": [
      {"talkId": "ABL-042"}
    ]
  } }
```
POST /api/web HTTP/1.1
[assorted headers]

{ "method": {
   "name": "GetTalkById",
   "param": [
      {"talkId": "ABL-042"}
   ]
}}

HTTP/1.1 200/OK
Content-Type: application/json
[assorted headers]

{"talks": [
   
   
   
   
   
   
   
   "id": "ABL-042",
   "name": "Finding deeper meaning in FOR EACH",
   "speaker": "peter.judge",
   "abstract": "Spicy jalapeno bacon andouille tenderloin",
   "talk_status": 1,
   "content_url": null,
   "content_type": null
}]}
POST /api/web HTTP/1.1
[assorted headers]
{
  "method": {
    "name": "GetTalkById",
    "param": [
      {"talkId": "ABL-099"}
    ]
  }
}

HTTP/1.1 200/OK
Content-Type: application/json
[assorted headers]
{
  "talkError": {
    "errorNum": "TE-001",
    "errorString": "Talk not found"
  }
}
Level 1: Resources
What's a resource?

- Business data (records)
- Documents or images
- Some work performed

- A URL is the address for that resource
- The JSON / XML / HTML / etc returned is a representation of that resource

Get all talks
Add speaker photo
Send acceptance email

/api/web/talks/abl-042

```
{  "id": "ABL-042",
   "name": "Finding deeper meaning in FOR EACH",
   "speaker": "peter.judge",
   "abstract": "Spicy jalapeno bacon andouille tenderloin",
   "talk_status": 1,
   "content_url": null,
   "content_type": null
}
```
Level 1 - success

POST /api/web/talks/abl-042 HTTP/1.1
[assorted headers]

{ "method": {
    "name": "GetTalkById",
    "param": [
        {"talkId": "ABL-042"}
    ]
}}

HTTP/1.1 200/OK
Content-Type: application/json
[assorted headers]

{"talks": [
    {
        "id": "ABL-042",
        "name": "Finding deeper meaning in FOR EACH",
        "speaker": "peter.judge",
        "abstract": "Spicy jalapeno bacon andouille",
        "talk_status": 1,
        "content_url": null,
        "content_type": null
    }
]}
Designing resource names

- Use URLs to name and identify resources: URI's are case-sensitive

- Collections-of vs. individual resource names
  - /speakers /talks
  - /speaker /talk

- What about nested resources?
  - /speakers/psc-123/talks/
  - /speakers/psc-123/talks/abl-042
  - /talks/abl-042

- Define the role of query strings
  - Should NOT use for identifiers

- Choose a style and stick with it
  - Singular or plural; caps or no
  - 301/Moved Permanently for the other

```json

{ "talks":
  [
    { "id": "ABL-042", ... },
    { "id": "PAS-027", ... },
    { "id": "SQL-327", ... }
  ]

// ALTERNATIVE

[
  { "id": "ABL-042", ... },
  { "id": "PAS-027", ... },
  { "id": "SQL-327", ... }
]

{ "id": "ABL-042",
  "name": "Finding deeper meaning in FOR EACH",
  "speaker": "peter.judge"
}
```
Defining resources in PASOE

https://localhost:8810/api/web/talks/abl-042

instance/webapp/transport/resources

[<abl-app>.<web-app>.WEB]

handlerName=<ooabl.type.name> : <relative-uri>

defaultHandler=<ooabl.type.name>

<relative-uri>

1* [ "/" [token|text] ]

Relative to /web

- MUST have leading /
- Either Text Customers
- Or Tokens {CustomerName} or {pathparam: regex}

<ooabl.type.name>

- OOABL implementation of Progress.Web.IWebHandler
  - OpenEdge.Web.WebHandler

Matched in (numeric) order then by best begins match then default handler
Which handler is used?

```
[conference.api.WEB]
defaultHandler = OpenEdge.Web.DefaultHandler
handler1 = API.TalksHandler : /speakers/{spkr}/talks/{tlk}
handler2 = API.SpeakersHandler : /speakers/{spkr}
handler3 = API.SpeakersHandler : /speakers
handler4 = API.TalksHandler : /talks/{tlk}
handler5 = API.TalksHandler : /talks
```

Client

```
http://localhost/api/web/speakers/psc-123/talks/abl-042
http://localhost/api/web/speakers/psc-123/talks
http://localhost/api/web/talks/abl-042
http://localhost/api/web/timeslots?speaker=psc-123
```
Determining the requested resource

```java
class API.TalksHandler implements Progress.Web.IWebHandler:
    method public integer HandleRequest():
        def var req as OpenEdge.Web.IWebRequest no-undo.

        req = new WebRequest().
        message
        req:URI:GetQueryValue('id') // ""
        req:PathParameterNames // "tlk,..."
        req:UriTemplate // "/talks/{tlk}" 
        req:PathInfo // "/talks/abl-042"
        req:TransportPath // "/web"
        req:GetPathParameter('tlk') // "abl-042"
```
Spring Security uses a concept called intercept-url to provide authorization on a resource

- Role-based access controls in PASOE

```java
### <instance>/webapps/<webapp>/WEB-INF/oeablSecurity.csv
### Intercept-url definitions for the WEB transport URIs
"/web/talks/**","*","hasAnyRole('ROLE_ADMIN,ROLE_ATTENDEE')"
"/web/speakers/psc-123/**","*","hasAnyRole('ROLE_SPEAKER')"
"/web/**","*","denyAll()"
```
Level 2: HTTP Verbs (methods)
## HTTP methods

HTTP methods, also called verbs, are used to specify which action to apply to the resource specified by the Uniform Resource Identifier (URI). What exactly this resource represents, or the exact action performed on the resource, depends on the application server.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Safe?</th>
<th>Idempotent?</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Request a representation of the resource</td>
<td>yes</td>
<td>yes</td>
<td>HTTP 1.0</td>
</tr>
<tr>
<td>HEAD</td>
<td>Request only the headers for the resource</td>
<td>yes</td>
<td>yes</td>
<td>HTTP 1.0</td>
</tr>
<tr>
<td>POST</td>
<td>Process the request body with the resource</td>
<td>no</td>
<td>no</td>
<td>HTTP 1.0</td>
</tr>
<tr>
<td>PUT</td>
<td>Create or update a new resource with the contents of the request body</td>
<td>no</td>
<td>yes</td>
<td>HTTP 1.1</td>
</tr>
<tr>
<td>DELETE</td>
<td>Remove the specified resource</td>
<td>no</td>
<td>yes</td>
<td>HTTP 1.1</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Return the HTTP methods the specified resource supports</td>
<td>yes</td>
<td>yes</td>
<td>HTTP 1.1</td>
</tr>
<tr>
<td>TRACE</td>
<td>Echo the received request</td>
<td>yes</td>
<td>yes</td>
<td>HTTP 1.1</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Convert the connection to a transparent tcp/ip tunnel, usually to allow</td>
<td>—</td>
<td>—</td>
<td>HTTP 1.1</td>
</tr>
<tr>
<td></td>
<td>SSL/TLS through an unencrypted HTTP proxy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PATCH</td>
<td>Apply partial modifications to the resource</td>
<td>no</td>
<td>yes</td>
<td>RFC-5789</td>
</tr>
</tbody>
</table>

### Safe Methods

Certain methods are specified to be safe, which means that executing them will not modify the resource or have other side effects on the overall state of the server. Unsafe methods may cause side effects, such as modifying a resource, sending an email or initiating the processing of a credit card.

### Idempotent Methods

Some methods are idempotent, meaning that executing identical requests multiple times will have the same effect as executing only one request. For example, DELETE requests are idempotent because once a resource is deleted it can’t be deleted again. Conversely, POST requests are not idempotent because a second POST request may send a second email or process the same credit card a second time.

By definition, safe methods are also idempotent.

[https://github.com/bigcompany/know-your-http](https://github.com/bigcompany/know-your-http)
# HTTP methods

HTTP methods, also called verbs, are used to specify which *action* to apply to the *resource* specified by the Uniform Resource Identifier (URI). What exactly this resource represents, or the exact action performed on the resource, depends on the application server.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Safe?</th>
<th>Idempotent?</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Request a representation of the resource</td>
<td>yes</td>
<td>yes</td>
<td>read</td>
</tr>
<tr>
<td>HEAD</td>
<td>Request only the headers for the resource</td>
<td>yes</td>
<td>no</td>
<td>create</td>
</tr>
<tr>
<td>POST</td>
<td>Process the request body with the resource</td>
<td>yes</td>
<td>no</td>
<td>update</td>
</tr>
<tr>
<td>PUT</td>
<td>Create or update a new resource with the contents of the request body</td>
<td>yes</td>
<td>no</td>
<td>delete</td>
</tr>
<tr>
<td>DELETE</td>
<td>Remove the specified resource</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Return the HTTP methods the specified resource supports</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>TRACE</td>
<td>Echo the received request</td>
<td>yes</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>CONNECT</td>
<td>Convert the connection to a transparent tcp/ip tunnel, usually to allow SSL/TLS through an unencrypted HTTP proxy</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>PATCH</td>
<td>Apply partial modifications to the resource</td>
<td>yes</td>
<td>yes</td>
<td>update</td>
</tr>
</tbody>
</table>

### Safe Methods

Certain methods are specified to be *safe*, which means that executing them will not modify the resource or have other side effects on the overall state of the server. Unsafe methods may cause side effects, such as modifying a resource, sending an email or initiating the processing of a credit card.

### Idempotent Methods

Some methods are *idempotent*, meaning that executing identical requests multiple times will have the same effect as executing only one request. For example, DELETE requests are idempotent because once a resource is deleted it can’t be deleted again. Conversely, POST requests are not idempotent because a second POST request may send a second email or process the same credit card a second time. By definition, *safe* methods are also *idempotent*.

[https://github.com/bigcompany/know-your-http](https://github.com/bigcompany/know-your-http)
Level 2 - success

GET /api/web/talks HTTP/1.1
[assorted headers]

<empty body>

HTTP/1.1 200/OK
Content-Type: application/json
[assorted headers]

{"talks": [
{"id": "abl-042", "name": "Finding deeper meaning"},
{"id": "pas-027", "name": "Spin your /WEB"},
{"id": "db-999", "name": "Up? Down? Does it matter?"}
]}

Level 2 – success

GET /api/web/talks/abl-042 HTTP/1.1
[assorted headers]

<empty body>

HTTP/1.1 200/OK
Content-Type: application/json
[assorted headers]

```json
{
    "id": "ABL-042",
    "name": "Finding deeper meaning in FOR EACH",
    "speaker": "peter.judge",
    "abstract": "Spicy jalapeno bacon andouille",
    "talk_status": 1,
    "content_url": null,
    "content_type": null
}
```
Level 2 – options

OPTIONS /api/web/talks/abl-042 HTTP/1.1
[assorted headers]

HTTP/1.1 200/OK
Content-Type: application/json
Allow: GET, PUT, DELETE
[assorted headers]

Missing POST = no new talks at this URL
Level 2 – updates

POST /api/web/talks/abl-042 HTTP/1.1
[assorted headers]

{  "id": null,
   "name": "Exploits of a mom",
   "speaker": "bobby.tables",
   "abstract": "Leberkas capicola chicken corned beef",
   "content_url": "https://xkcd.com/327/",
   "content_type": "image/png"
}

HTTP/1.1 405/Method Not Allowed
[assorted headers]

{"talkError": {
   "errorNum": "TE-007",
   "errorString": "This resource already exists: request terminated"
}}
Level 2 – updates

PUT /api/web/talks/abl-042 HTTP/1.1
[assorted headers]

{ "id": "ABL-042",
  "name": "Finding deeper meaning in FOR EACH",
  "speaker": "peter.judge",
  "abstract": "Spicy jalapeno bacon andouille",
  "talk_status": 2,
  "content_url": "http://slides/pug/42.pdf",
  "content_type": "application/pdf" }

HTTP/1.1 200/OK
[assorted headers]
Level 2 – updates

POST /api/web/talks/ HTTP/1.1
[assorted headers]

```json
{
  "name": "Exploits of a mom",
  "speaker": "bobby.tables",
  "abstract": "Leberkas capicola chicken corned beef",
  "content_url": "https://xkcd.com/327/",
  "content_type": "image/png",
  "id": null
}
```

HTTP/1.1 201/Created
Location: /api/web/talks/sql-327
[assorted headers]

```json
{
  "id": "SQL-327",
  "name": "Exploits of a mom",
  "speaker": "bobby.tables",
  "abstract": "Leberkas capicola chicken corned beef",
  "content_url": "https://xkcd.com/327/",
  "content_type": "image/png",
  "talk_status": 1
}
```
Coding a web handler

class API.TalksHandler inherits OpenEdge.Web.WebHandler:
    method protected integer HandlePost(input pReq as IWebRequest):
        resp = new OpenEdge.Web.WebResponse().

        if pReq:PathInfo eq '/talks/' then
            do:
                table ttTalk:read-json('handle', web-context).
                run talks/bl/new_talk.p (input table ttTalk, output talkId).

                assign resp:StatusCode = integer(OpenEdge.Net.HTTP.StatusCodeEnum:Created) //201
                location = new URI(pReq:URI:Scheme, pReq:URI:Host, pReq:URI:Port)
                location:Path = pReq:WebAppPath // /api
                + pReq:TransportPath // /web
                + pReq:PathInfo // /talks/
                + talkId // sql-327

                resp:SetHeader('Location', location:ToString()).
            end.
        // Add cleanup; error handling for path validation and other issues
Sidebar: Updated security with HTTP Methods

### Intercept-url definitions for the WEB transport URIs

#### Anyone can see info about a talk, or all talks

```
"/web/talks/**","GET","allowAll()
```

#### Anyone can submit a talk

```
"/web/talks/","POST","allowAll()
```

#### Only admins can update talks

```
"/web/talks/**","","hasAnyRole('ROLE_ADMIN')"
```

#### Admins and Speakers can update speaker info

```
"/web/speakers/","GET","allowAll()
"/web/speakers/","PUT,POST,DELETE","hasAnyRole('ROLE_ADMIN')"
"/web/speakers/**","PUT","hasAnyRole('ROLE_ADMIN, ROLE_SPEAKER')"
```
Level 3: Hypermedia Controls

hypermedia (n)
The use of text, data, graphics, audio and video as elements of an extended hypertext system in which all elements are linked so that the user can move among them at will

https://en.wiktionary.org/wiki/hypermedia
Why hypermedia controls?

- It means you can start at the Very Beginning and the RESTful API will let you know where you can go
  - Who am I?
  - What's next, last, first in a set of records?
  - Do I have any related resources, and where are they?

- Linked resources are self-describing
  - Methods/verbs
    - OPTIONS
  - Content types
    - Accept / Content-Type

```json
{  "id": "abl-042",  "name": "Finding deeper meaning in FOR EACH",  "abstract": "Spicy jalapeno bacon andouille",  "speaker": "peter.judge",  "talk_status": 1,  "content_url": null,  "content_type": null,  "links": {    "self": {"href": "/talks/abl-042" },    "speaker": {"href": "/speakers/psc-123" },    "times": {"href": "/timeslot/?talk=abl-042" }  }}
```
GET /api/web HTTP/1.1
Accept: application/json, application/xml
[assorted headers]

HTTP/1.1 200/OK
Content-Type: application/vnd.conf+json
[assorted headers]

{
  "links": {
    "talks": {"href": "/api/web/talks" },
    "speakers": {"href": "/api/web/speakers" },
    "times": {"href": "/api/web/timeslots" },
    "attendees":{"href": "/api/web/attendees" }
  },
  "text": "Ribeye venison jerky, ham hock pancetta turducken jowl biltong porchetta strip steak. Venison flank ham hock meatloaf ball tip chuck, porchetta picanha drumstick pork loin tongue frankfurter bacon. Pancetta doner pork fatback Fatback pork belly burgdoggen landjaeger frankfurter." 
}
But in reality …

- "Pure REST" aka HATEOAS is poorly understood … and rarely fully ("properly") implemented
  - Standardized content types are underused
  - OPTIONS method underused

- Most modern RESTful APIs combine
  - Well-defined resources & methods (level 2)
  - Some level of hypermedia (linked resources)
  - (External) API documentation

  [swagger.io](http://swagger.io), [apidocjs.com](http://apidocjs.com), [json-schema.org](http://json-schema.org), Data Services Catalog
Level 3 - success

GET /api/web/talks?top=5 HTTP/1.1
Accept: application/json, application/xml
[assorted headers]

HTTP/1.1 200/OK
Content-Type: application/vnd.conf.talks+json
[assorted headers]

```json
{
  "links": [
    {
      "self": {
        "href": "/api/web/talks?skip=0&top=5"
      },
      "first": {
        "href": "/api/web/talks?skip=0&top=5"
      },
      "next": {
        "href": "/api/web/talks?skip=5&top=5"
      },
      "prev": {
        "href": "null"
      },
      "last": {
        "href": "/api/web/talks?skip=86"
      }
    },
    {
      "id": "abl-042",
      "name": "Finding deeper meaning"
    },
    {
      "id": "pas-027",
      "name": "Spin your /WEB"
    },
    {
      "id": "db-999",
      "name": "Up? Down? Does it matter?"
    }
  ]
}
```
GET /talks?recs=5 HTTP/1.1
Accept: application/json, application/xml
[assorted headers]

<empty body>

{
  "talks": [
    {
      "links": {
        "first": {"href": "/api/web/talks?skip=0&top=5" },
        "next": {"href": "/api/web/talks?skip=5&top=5" },
        "self": {"href": "/api/web/talks?skip=0&top=5" },
        "prev": {"href": null },
        "last": {"href": "/api/web/talks?skip=86" }
      },
      "id": "abl-042",
      "name": "Finding deeper meaning in FOR EACH",
      "abstract": "Spicy jalapeno bacon andouille",
      "speaker": "peter.judge",
      "talk_status": 1,
      "content_url": null,
      "links": {
        "self": {"href": "/api/web/talks/abl-042" },
        "speaker": {"href": "/api/web/speakers/psc-123" },
        "times": {"href": "/api/web/timeslots?talk=abl-042" }
      }
    },
    {
      "id": "pas-027",
      "name": "Spin your /WEB"
    },
    {
      "id": "db-999",
      "name": "Up? Down? Does it matter?"
    }
  ]
}
Documentation to the rescue

- Message body schemas
- Content type(s)
- Query string
- Status/error codes
… REST APIs must be hypertext-driven …

- JSON is weakly-typed / schema-less
  - Custom types let us "strongly suggest" the schema
    application/schema+json

- Of course there's not just one
  - JSON-LD [https://www.w3.org/TR/ld+json](https://www.w3.org/TR/ld+json)
  - HAL [http://stateless.co/hal_specification.html](http://stateless.co/hal_specification.html)
  - SIREN [https://github.com/kevinswiber/siren](https://github.com/kevinswiber/siren)
  - JSON API [http://jsonapi.org/](http://jsonapi.org/)

```
{  
  "_links": {  
    "self": {  
      "href": "/orders"  
    },  
    "curies": [  
      {"name": "ea",  
        "href": "http://example.com/docs/rels/{rel}"},  
        "templated": true }],  
  "next": {  
      "href": "/orders?page=2" },  
  "ea:find": {  
      "href": "/orders{?id}"  
    },  
    "ea:admin": [  
        {"href": "/admins/2", "title": "Fred"},  
        {"href": "/admins/5", "title": "Kate"}  
    ],  
    "currentlyProcessing": 14,  
    "shippedToday": 20,  
  "_embedded": {  
    "ea:order": [{  
    "_links": {  
      "self": {  
        "href": "/orders/123"  
      },  
      "ea:basket": {  
        "href": "/baskets/98712"  
      },  
      "ea:customer": {  
        "href": "/customers/7809"  
      }  
    },  
    "total": 30.00,  
    "currency": "USD",  
    "status": "shipped"  
  ] } }  
```
Manipulating content types

ABL basics

2. SAX-READER, SAX-WRITER, READ- and WRITE-XML()
3. Low-level: STRING(), REPLACE(), COPY-LOB, PUT-BYTES()

But these have no concept of the semantics of the data; maybe with temp-tables & datasets, but not always

- Pluggable, extensible MessageWriters in OpenEdge.Net.*
  - Keyed on MIME type
  - Know how to turn (say) JSON into a temp-table and vice versa
Coding a web handler

// get current parameters
topRecs = integer(pReq:URI:GetQueryValue('top')).
skipRecs = integer(pReq:URI:GetQueryValue('skip')).

// get the requested talk data
run logic/talk/read_talks.p (input '', skipRecs, topRecs, output table ttTalk).

resp = new OpenEdge.Web.WebResponse().
resp:StatusCode = 200. // OK
resp:ContentType = 'application/vnd.conf.talks+json'.
resp:Entity = new JsonObject().
resp:Entity:Add('talks', new JsonArray()).

// Build link base from the requested resource
linkBase = req:WebAppPath // /api
     + req:TransportPath. // /web
Coding a web handler

// add links for the collection (set/list) of talks
talks: Add('links', new JsonObject()).

links: Add('first', linkBase + '/talks?skip=0&top=' + string(topRecs)).
links: Add('next', linkBase + '/talks?skip=' + string(skipRecs + topRecs) + '&top=' + string(topRecs)).
links: Add('prev', linkBase + '/talks?skip=' + string(skipRecs - topRecs) + '&top=' + string(topRecs)).
links: Add('last', linkBase + '/talks?skip=' + string(totRecs - topRecs)).

// add each Talk as data + links
for each ttTalk:
   talks: Add(new JsonObject()).
   // add Talk data
   record: Read(buffer ttTalk:handle).

   // add Talk links
   record: Add('links', new JsonObject()).

   links: Add('self', linkBase + '/talks/' + ttTalk.id).
   links: Add('speaker', linkBase + '/speakers/' + ttTalk.speaker).
   links: Add('times', linkBase + '/timeslots?talk=' + ttTalk.id).
end.
What else?
JSDO / Data Object Services

"Data Object Service with Catalog"

- Predefined resources
- Defined verbs
- Fixed content type
- No hyperlinks

- Primarily used by JavaScript DataObject
  - ProDataSet client proxy
  - Catalog provides metadata
  - Operations
  - Field based data types and validation
DataObjectHandler

- Generic RESTful WebHandler
  - Configure with `/pdo/` in `openedge.properties`
  - Added for JSDO use-cases …
    … but “invoke” operations means it can do almost anything
  - Add (customer-requested) flexibility to responses
    (headers, status codes etc)

- JSON mapping file for service configuration
  - Internal – not public – config for routing requests and mapping inputs & outputs

- Extend via published events or content type plugins
  - Does require Work
**DataObjectHandler: what can I map?**

```
"/talks/{tlk}": {
  "POST": {
    "contentType": "application/json",
    "statusCode": 201,
    "options": {
      "responseEnvelope": true
    },
    "entity": {
      "name": "logic/talk/new_talk.p",
      "function": "add_talk",
      "arg": [ {
        "ablName": "ttTalk",
        "ioMode": "INPUT",
        "ablType": "table",
        "msgElem": { "type": "body", "name": null }
      }, {
        "ablName": "pcChar",
        "ioMode": "output",
        "ablType": "character",
        "msgElem": { "type": "header", "name": "location" }
      }]}
  }
}
```

**URI mapping**
- `/talks`
- `/{service}/data/{resource}`
- `/{collection}/coll-id`

**Status codes**
- 202 / Accepted
- 418 / I'm a teapot

**Envelopes**
- `requestEnvelope`: "input"
- `errorEnvelope`: "oops"

**IO Modes**
- "input" "output" "input-output" "return"

**ABL data types (also extent variants)**
- "character", "longchar", "integer", "int64", "decimal",
- "logical", "rowid", "recid", "date", "datetime", "datetime-tz",
- "raw", "memptr", "dataset", "temp-table",
- "class <ooabl.type.name>"

**HTTP Message elements**
- Request-only: "path", "query", "httpMethod", "request",
- Response-only: "none", "statusCode", "statusReason"
- Both: "cookie", "header", "field", "body"
And after REST?

Single-client-request architectures?

🌟 GraphQL

GraphQL APIs are organized in terms of types and fields, not endpoints. Access the full capabilities of your data from a single endpoint.

http://graphql.org/

🌟 Falcorn

Falcorn allows you to model all of your backend data as a single JSON resource on the application server.

https://netflix.github.io/falcor/
"The Glory of REST" with OpenEdge

✓ Easy to implement REST in PASOE
  1. Configuration in openedge.properties to make sure the appropriate resources are called
  2. Web handlers to process requests and call business logic

Supporting players

- Helper code in OpenEdge.Net.* & OpenEdge.Web.*
- Data Object Handler
- Data Service Catalog ("JSDO catalog")

https://documentation.progress.com/output/oehttpclient/

✓ Using REST (resources, etc) will let your app run on standard internet infrastructure / hardware
Appendix: Linky-loo

- [https://softwareengineering.stackexchange.com/a/357882/6588](https://softwareengineering.stackexchange.com/a/357882/6588)
- [https://martinfowler.com/articles/richardsonMaturityModel.html](https://martinfowler.com/articles/richardsonMaturityModel.html)
- [https://www.iana.org/assignments/link-relations/link-relations.xhtml](https://www.iana.org/assignments/link-relations/link-relations.xhtml)
- [https://sookocheff.com/post/api/on-choosing-a-hypermedia-format/](https://sookocheff.com/post/api/on-choosing-a-hypermedia-format/)