General REST usage

As explained in the introduction, the REST API is based on a very limited list of general principles:

- each resource (uri) interacts with a part of the system (Content, URL aliases, user groups...),
- for each resource, one or more verbs are available, each having a different effect (delete a Content, get a URL Alias, list user groups...),
- media-type request headers indicate what type of data (Content / ContentInfo), and data format (JSON or XML), are expected as a response, and what can be requested.

Anatomy of a REST call

What we can learn from a GET request

This verb is used to query the API for information. It is one of the two operations web browsers implement, and the most commonly used.

Request

The only requirement for this verb is usually the resource URI, and the accept header. On top of that, cache request headers can be added, like `If-None-Match`, but those aren't fully implemented yet in eZ Publish 5.0.

Load ContentInfo request

```
GET /content/objects/23 HTTP/1.1
Accept: application/vnd.ez.api.ContentInfo+xml
```

Response headers

The API will reply with:

- an **HTTP response code**,
- a few **headers**,
- the XML representation of the ContentInfo for content with ID 23 in XML format, as specified in the Accept header.

Load ContentInfo response

```
HTTP/1.1 200 OK
Accept-Patch: application/vnd.ez.api.ContentUpdate+xml;charset=utf8
Content-Type: application/vnd.ez.api.ContentInfo+xml
Content-Length: xxx
```

The length of our content, provided by the Content-Length header, isn't *that* useful.

HTTP Code

The API responded here with a standard 200 OK HTTP response code, which is the expected response code for a "normal" GET request. Some GET requests, like getting a content's current version, may reply with a 301 Moved permanently, or 307 Temporary redirect code.

Errors will be indicated with HTTP error codes, like 404 for Not Found, or 500 for Internal server error. The REST specifications provide the list of every HTTP response code you can expect from implemented resources.

Content-Type header

As long as a response contains an actual HTTP body, the Content-Type header will be used to specify which Content-Type is contained in the response. In that case, a ContentInfo (`Content-Type: application/vnd.ez.api.ContentInfo`) in XML (`Content-Type: applicatio`

n/vnd.ez.api.ContentInfo**+xml**)

Accept-Patch header

This is a very interesting one.

It tells us we can modify the received content by patching (**`Accept-Patch`**: `application/vnd.ez.api.ContentUpdate+xml;charset=utf8`) it with a ContentUpdateStruct (`Accept-Patch: application/vnd.ez.api.`**`ContentUpdate`**`+xml;charset=utf8`) in XML (`Accept-Patch: application/vnd.ez.api.ContentUpdate`**`+xml`**`;charset=utf8`) format. Of course, JSON would also work, with the proper format.

This last part means that if we send a PATCH /content/objects/23 request with a ContentUpdateStruct XML payload, we will update this Content.

> REST will use the **`Accept-Patch`** header to indicate you how to **modify** the returned **data**.

Other headers: Location

Depending on the resource, request & response headers will vary. For instance, creating content, or getting a content's current version will both send a `Location` header to provide you with the requested resource's ID.

Those particular headers generally match a specific list of HTTP response codes. Location is sent by `201 Created`, `301 Moved permanently`, `307 Temporary redirect responses`. This list isn't finished, but you can expect those HTTP responses to provide you with a Location header.

Other headers: Destination

This request header is the request counterpart of the Location response header. It is used in a COPY / MOVE operation, like copying a Content, on a resource to indicate where the resource should be moved to, using the ID of the destination.

Response body

Load ContentInfo response body Expand source

```
<?xml version="1.0" encoding="UTF-8"?>
<Content href="/content/objects/23" id="23"
  media-type="application/vnd.ez.api.Content+xml" remoteId="qwert123">
  <ContentType href="/content/types/10"
media-type="application/vnd.ez.api.ContentType+xml" />
  <Name>This is a title</Name>
  <Versions href="/content/objects/23/versions"
media-type="application/vnd.ez.api.VersionList+xml" />
  <CurrentVersion href="/content/objects/23/currentversion"
    media-type="application/vnd.ez.api.Version+xml"/>
  <Section href="/content/sections/4" media-type="application/vnd.ez.api.Section+xml"
/>
  <MainLocation href="/content/locations/1/4/65"
media-type="application/vnd.ez.api.Location+xml" />
  <Locations href="/content/objects/23/locations"
media-type="application/vnd.ez.api.LocationList+xml" />
  <Owner href="/user/users/14" media-type="application/vnd.ez.api.User+xml" />
  <lastModificationDate>2012-02-12T12:30:00</lastModificationDate>
  <publishedDate>2012-02-12T15:30:00</publishedDate>
  <mainLanguageCode>eng-US</mainLanguageCode>
  <alwaysAvailable>true</alwaysAvailable>
</Content>
```

The XML body is a serialized version of a ContentInfo struct. Most REST API calls will actually involve exchanging XML / JSON representations of

the public API. This consistency, which we took very seriously, was a hard requirement for us, as it makes documentation much better by requiring *less* of it.

As explained above, the API has told us that we could modify content object 23 by sending a `vendor/application/vnd.ez.ContentUpdate+xml`. This media type again matches a Value in the API, ContentUpdateStruct.

> The REST API data structs mostly match a PHP Public API value object

Value objects representation

Value objects like ContentInfo basically feature two types of fields: basic, local fields (modified, name...) and foreign field(s) references (sectionId, mainLocationId).

Local fields will be represented in XML / JSON with a primitive type (integer, string), while foreign key references will be represented as a link to another resource. This resource will be identified with its URI (`/content/objects/23/locations`), and the media-type that should be requested when calling that resource (`media-type="application/vnd.ez.api.LocationList+xml"`). Depending on how much data you need, you may choose to crawl those relations, or to ignore them.

> **XSD files**
> For each XML structure known to the REST API, you can find XSD files in the XSD folder of the specifications. Those will allow you to validate your XML, and learn about every option those XML structures feature.
> https://github.com/ezsystems/ezpublish-kernel/tree/master/doc/specifications/rest/xsd

Request parameters

So far, we have seen that responses will depend on:

- The URI,
- Request headers, like the Accept one

URI parameters are of course also used. They usually serve as filters / options for the requested resource. For instance, they can be used to customize a list's offset/limit, to filter a list, specify which fields you want from a content... For almost all resources, those parameters must be provided as GET ones. This request would return the 5 first relations for Version 2 of Content 59:

GET request with limit parameter

```
GET /content/objects/59/versions/2/relations&limit=5 HTTP/1.1
Accept: application/vnd.ez.api.RelationList+xml
```

> **Working with value objects IDs**
> Resources that accept a reference to another resource expect this reference to be given as a REST ID, not a Public API ID. As such, the URI to request users that are assigned the role with ID 1 would be `GET /api/ezp/v2/user/users?roleId=/api/ezp/v2/user/roles/1`.

Custom HTTP verbs

In addition to the usual GET, POST, PUT and DELETE HTTP verbs, the API supports a few custom ones: COPY, MOVE (http://tools.ietf.org/html/rfc2518), PATCH (http://tools.ietf.org/html/rfc5789) and PUBLISH. While it should generally not be a problem, some HTTP servers may fail to recognize those. If you face this situation, you can customize a standard verb (POST, PUT) with the `X-HTTP-Method-Override` header.

PATCH HTTP request

```
POST /content/objects/59 HTTP/1.1
X-HTTP-Method-Override: PATCH
```

Both methods are always mentioned, when applicable, in the specifications.

Specifying a siteaccess

One of the principles of REST is that the same resource (Content, Location, ContentType, ...) should be unique. The purpose is mostly to make it simple to cache your REST API using a reverse proxy like Varnish. If the same resource is available at multiple locations, cache purging becomes much more complex.

Due to this, we decided not to enable siteaccess matching with REST. In order to specify a siteaccess when talking to the REST API, a custom header, `X-Siteaccess`, needs to be provided. If it isn't, the default one will be used:

X-Siteaccess header example

```
GET / HTTP/1.1
Host: api.example.com
Accept: application/vnd.ez.api.Root+json
X-Siteaccess: ezdemo_site_admin
```