Linked Data Platform

Platform versie 2 (via HMAC)

Beheren, publiceren, bevragen

	Versiebeheer:
2018-01-18	SDIJ heeft de eerste grove slag geslagen in het document dat door Semmtech is aangeleverd. Belangrijkste wijzigingen: - CROW huisstijl toegevoegd (headers) - De onderdelen mbt publiceren verwijderd (aangezien dit document betrekking heeft op het bevragen van het LDP) - Context kennisbank (en alle verwijzingen daarnaar) verwijderd, omdat dit document bedoeld is voor de context CPC (met name add-ons ed)

Inhoud

1 Beheren van LDP (via HMAC) 3
1.1 Introductie 3
1.2 Ondertekenen van Request 3
1.2.1 Werkwijze 3
1.2.2 Berekenen HMAC Signature 4
1.3 Versturen Request 5
1.4 Postman Pre-Request Script 6
1.4.1 Content in Binary 9
1.5 Overzicht van Services 9
2 Versie 2 | Publiceren nieuwe content 10
2.1 Autorisatie 10
2.2 Voorbeeld Data 10
CROW Schema v1 10
CROW Schema v2 11
2.3 Uploaden Model 11
2.3.1 Service URL 11
2.3.2 Aanroepen Service 12
2.3.3 Wijzigen van Namespace 13
2.4 Overige Operaties 13
2.4.1 /version/{VersionID} 13
2.4.2 /version/latest 14
2.4.3 /list 14
2.4.4 /delta/{FromVersionID}/{ToVersionID} 14
3 Bevragen gepubliceerde content 15
3.1 Introductie 15
3.2 SPARQL Services 15
3.2.1 GET 15
3.2.2 POST 16
3.2.3 Output Formats 16
3.2.4 CROW ProContract (CPC) 16
3.2.5 CROW Kennisbank (CKB) 17

1 Beheren van LDP (via HMAC)

1.1 Introductie

Deze onderdeel beschrijft de implementatie van het vernieuwde autorisatiemodel binnen het publicatie platform. Er is een demo omgeving beschikbaar gesteld waarin een aantal van de services en principes van het autorisatiemodel al kunnen worden bekeken/getest. Deze bevindt zich op de demo omgeving van Semmtech op http://acceptance2.laces.tech/crow-publication-v2. Uiteraard is er nog wel een username en password benodigd om toegang te krijgen tot de web pagina's van het platform, deze kunnen worden opgevraagd bij Semmtech.

1.2 Ondertekenen van Request

In plaats van het oversturen van een gebruikersnaam en wachtwoord, zullen alle request naar de admin services moeten worden voorzien van een HMAC signature. **H**ash based **M**essage **A**uthenti**C**ation (HMAC) is een message authentication algoritme waarbij een cryptografische hashing functie in combinatie met een private key wordt toegepast.

1.2.1 Werkwijze

Voor het berekenen van de signature kunnen de volgende stappen worden doorlopen.

HMAC authenticatie wordt gedaan volgens de volgende stappen:

1. Client en server kennen beide een geheime private key en een publieke client ID
2. Client genereert een request met daarin de volgende elementen:
    - een datum/tijd header (ofwel **CurrentDate**)
    - een unieke **NONCE** waarde, welke maar eenmalig kan worden gebruikt en reply attacks voorkomt
    - een **HMAC Signature** welke berekend is met behulp van de geheime private key
3. Client stuurt de request naar de server
4. Server leest de HMAC details en gebruikt de client ID om de private key behorende bij dit ID op te zoeken
5. Server gebruikt de private key om nogmaals de HMAC signature te berekenen (op dezelfde wijze zoals de client dit ook heeft gedaan)
6. Server checkt of de zelf berekende signature overeenkomt met de signature die de client heeft meegestuurd
7. Om reply attacks te voorkomen moet de datum/tijd waarde binnen een voor ingestelde bandbreedte liggen t.o.v. het moment waarop de request door de server is ontvangen, ook de NONCE mag slechts éénmaal worden gebruikt en zal worden opgeslagen op de server.

1.2.2 Berekenen HMAC Signature

Voor het berekenen van de HMAC signature worden een aantal velden gebruikt:

- **Method**: HTTP Method, zoals GET, PUT, POST of DELETE
- **CurrentDate**: Huidige datum/tijd (welke ook in de header wordt meegestuurd) in het formaat "2016-10-10T17:50:00Z"
- **URL**: De URL waarnaar de request wordt verzonden, bijvoorbeeld "http://demo2.semmweb.com/crow-publication-v2/contexts/cpc-admin/projects". Indien er querystring parameters worden meegestuurd, zullen ook deze moeten worden toegevoegd aan de URL (LET OP dat deze worden geURLEncode, aangezien de server ze ook zo zal ontvangen, dus een spatie zal moeten worden omgezet naar "%20" voordat de HMAC wordt berekend).
- **NONCE**: Unieke code welke slechts eenmaal kan worden gebruikt (gelijk aan de NONCE in de header van de request).
- Indien het om een PUT of POST gaat en de body van de request bevat data (length > 0) dan:
    - **Content-Type**: Het type van de content die wordt meegestuurd naar de server (gelijk aan de Content-Type header waarde). Indien een encoding wordt aangegeven, deze waarde niet opnemen in deze berekening (dus als het Content-Type gelijk is aan "text/turtle; charset=UTF-8" dan moet alleen "text/turtle" wordt gebruikt in de berekening van de HMAC).
    - **MD5**: Een MD5 hash van de meegestuurde content in de body (deze MD5 in waarde MOET in HEX en lower case worden aangeboden)

De bovenstaande waarden worden geconcateneerd met behulp van komma's, wat resulteert in de volgende HMAC Value string:

```
HMAC Value = {Method},{CurrentDate},{URL},{NONCE}[,{Content-Type},{MD5}]
HMAC Signature = Base64 ( HMAC-SHA256( {HMAC Value}, {Private Key} ) )
```

Samen met de private key zal de bovenstaande string volgens een HMAC-SHA256 algoritme worden gehashed (zie https://en.wikipedia.org/wiki/Hash-based_message_authentication_code).

UPDATE 25 januari 2018:

Om te voorkomen dat niet ASCII characters onderdeel zijn van de signature voordat deze wordt verzonden, dient het resultaat van het algoritme middels Base64 te worden ge-encode (https://en.wikipedia.org/wiki/Base64).

Als voorbeeld kan het volgende scenario worden gebruikt om een implementatie van bovenstaande te testen. Neem als HMAC Value de waarde "value" en als Private Key "secret", met deze waarden en volgen van de stappen, moet hier een Base64 encoded signature gelijk aan "UOA+vmW+mLuL8RuiyJLVTAeayisNOwFidpxtdXolQ08=" uitkomen. Wanneer een implementatie echter een waarde geeft van "NTBlMDNlYmU2NWJlOThiYjhiZjExYmEyYzg5MmQ1NGMwNzlhY2EyYjBkM2IwMTYyNzY5YzZkNzU3YTI1NDM0Zg==" of iets anders, kan het zijn dat het resultaat, voordat het naar de Base64 encoding gaat, al eerder geconverteerd wordt naar een HEX of ASCII formaat.

1.3 Versturen Request

Dit is een nieuwere manier om autorisatie informatie aan te leveren aan de server. Voorkeur van deze versie is dat alle informatie in één enkele header wordt doorgestuurd, en dat de header een gestandaardiseerde naam heeft.

Elke request zal minimaal de volgende header moeten bevatten:

- Authorization
- Content-Type (indien PUT of POST)

De waarde die wordt toegekend aan deze header moet de volgende opmaak hebben:

- De string moet starten met het woordje "HMAC"
- Daarna worden de volgende key-value paren toegevoegd, elk gescheiden middels een komma.
    - clientId
    - nonce
    - currentDate
    - signature

```
Authorization = HMAC clientId="{ClientID}", nonce="{NONCE}",
currentDate="{CurrentDate}", signature="{HMAC Signature}"
```

Voorbeeld GET

Let op dat deze keys hoofdletter gevoelig zijn! Zo zal bijvoorbeeld een *GET* request naar de service *http://acceptance2.laces.tech/crow-publication-v2/contexts/cpc-admin/projects* met een NONCE waarde gelijk aan *c555ffc421afc6ca12f4086c2c26442* op *10 oktober 2016 16:06:13*, voor de client met Client ID *admin* (en private key "password") resulteren in het onderstaande voorbeeld (signature is een voorbeeld):

```
GET /crow-publication-v2/contexts/cpc-admin/projects HTTP/1.1
Host: acceptance2.laces.tech
Authorization: HMAC clientId="admin", nonce="c555ffc421afc6ca12f4086c2c26442",
currentDate="2016-10-10T16:06:13Z",
signature="JnH89CHaYK6Gan5RCthBy4GZw7f47qGLrthnJ9cDTOw="
```

Voorbeeld POST

Let op dat deze keys hoofdletter gevoelig zijn! Een *POST* request naar de service *http://acceptance2.laces.tech/crow-publication-v2/contexts/cpc-admin/contexts* met een NONCE *057451f7-ba9d-48cf-bbd2-8420ed24d348* op *10 november 2016 10:50:04*, voor de client met Client ID *admin* (en private key "*password*") resulteert in het onderstaande voorbeeld.

De JSON body welke wordt megestuurd ziet er als volgt uit: *{ "id": "12345", "name": "Project X", "path": "/contexts/cpc", "enabled": true, "attributes": [], "permissions": [] }* en heeft een MD5 hash gelijk aan *6ce76fb5894d50731d914e98ba37a8dd.*

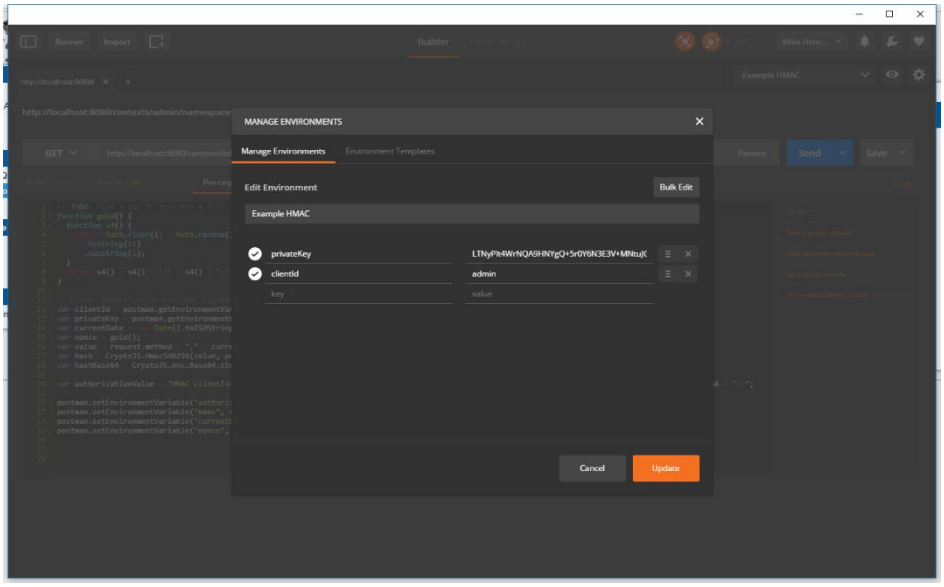
```
POST /crow-publication-v2/contexts/cpc-admin/contexts HTTP/1.1
Host: acceptance2.laces.tech
Authorization: HMAC clientId="admin", nonce="057451f7-ba9d-48cf-bbd2-8420ed24d348",
currentDate="2016-11-10T10:50:04Z",
signature="weQ6qrSuYEjaWf5wUTS1FOXDWwDS88VsJx2KEr2CVgo="
Content-Type: application/json

{ "id": "12345", "name": "Project X", "path": "/contexts/cpc", "enabled": true,
"attributes": [], "permissions": [] }
```

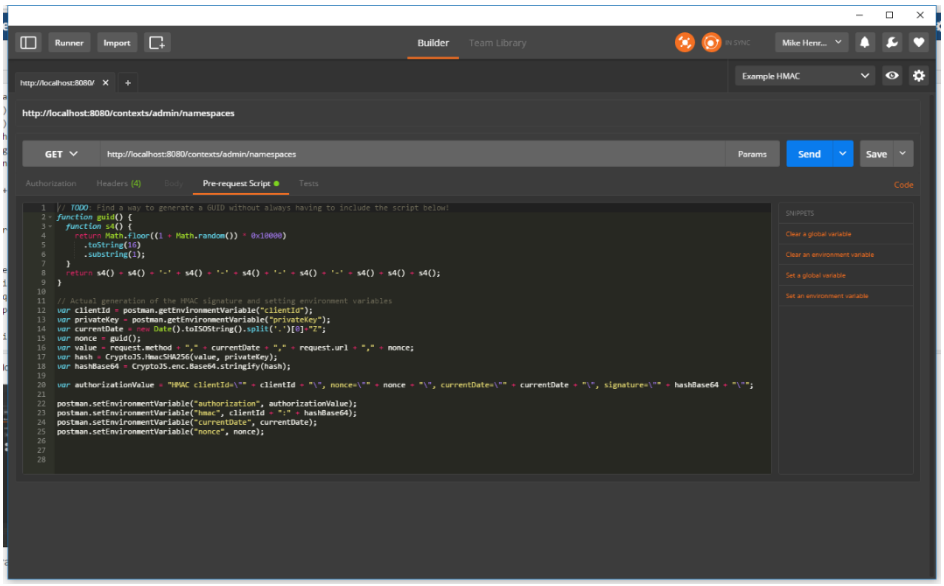
1.4 Postman Pre-Request Script

De REST client Postman (zie https://www.getpostman.com/) heeft de mogelijkheid om een JavaScript uit te voeren net voordat een request wordt verzonden naar de server. Hieronder wordt kort toegelicht hoe Postman kan worden geconfigureerd, zodat de HMAC authorisatie direct kan worden toegepast, zonder tussenkomst van een HMAC Generator als hierboven. Onderstaande script is geupdate en gestest in versie 5.5.0.

Om het script te gebruiken, moet je allereerst een environment instellen (tandwiel icoontje rechts van de dropdown met No Environment), in deze environment voeg je twee key-value paren toe (inclusief geldige waarden): één voor de clientId, en één voor de privateKey. Zie screenshot hieronder.



Nadat de Environment is aangemaakt, selecteer je deze environment in de dropdown. Vul vervolgens de URL in van de request welke je wilt uitvoeren (bijvoorbeeld http://acceptance2.laces.tech/crow-publication-v2/contexts/cpc-admin/namespaces). Voeg vervolgens in het tabblad Headers, een header toe met key "Authorization" en als waarde "{{authorization}}". Waarden tussen dubbele accolades, zullen tijdens het versturen van een request worden vervangen met gelijknamige waarden die in een environment zijn ingesteld.



Open tenslotte het tabblad Pre-request Script en plak het onderstaande JavaScript erin:

UPDATE 15 februari 2017: In de nieuwste versie van Postman (4.9.3+) kan geen gebruik meer gemaakt worden van het automatisch checken van de mode waarin content aan de body wordt meegegeven (of in ieder geval niet via request.dataMode). Indien content wordt meegegeven als binary, moet de regel met dataMode = "binary"; erin staan. Zo niet mag deze regel worden uitgecomment (door een dubbele slash '//' ervoor te zetten).

UPDATE 6 maart 2017: Er is een debugging feature toegevoegd in de HMAC mechanisme ter ondersteuning aan het achterhalen van incorrecte input van HMAC waarden. Hiertoe kan het onderstaande script worden gebruikt (bijgewerkt per 6 maart) en zal een extra "HMAC-Information" header in de request moeten worden meegegeven, welke als waarde "{{hmacInformation}}". Indien deze header wordt meegestuurd in de request en er treedt een "401 - Unauthorized - HMAC signatures do not match, request will be discarded" op, dan zal de response ook een header "HMAC-Error" bevatten waarin wordt aangegeven welke waarden afwijken.

UPDATE 11 september 2017: In de laatste versie (geconstateerd in versie 5.2.0) moet waar de "Content-Type" uit de headers wordt gehaald, de headers bevraagd worden met "content-type" (dus allemaal lower-case letters, i.p.v. "Content-Type"). Wanneer bij het uitvoeren van de request, de volgende foutmelding wordt gegeven: "There was an error in evaluating the Pre-request Script: TypeError: Cannot read property 'split' of undefined" dan moet de lower-case variant worden gebruikt.

UPDATE 13 september 2017: HMAC-Information kan nu ook worden toegevoegd aan de headers, door gebruik te maken van de environment variable {{hmacInformation}}

```
function guid() {
  return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
    var r = Math.random() * 16 | 0,
      v = c == 'x' ? r : (r & 0x3 | 0x8);
    return v.toString(16);
  });
}

// Actual generation of the HMAC signature and setting environment variables
// The next two variables can be either retrieved from a EnvironmentVariable OR
hardcoded directly below
// In any case the private key will not be transmitted only used to calculated a
valid HMAC hash!
var clientId = postman.getEnvironmentVariable("clientId");
var privateKey = postman.getEnvironmentVariable("privateKey");
var applicationUrl = postman.getEnvironmentVariable("applicationUrl");
var currentDate = new Date().toISOString().split('.')[0] + "Z";
var nonce =  guid();
var method = request.method;
var url = request.url.replace("{{applicationUrl}}", applicationUrl);
var value = method + "," + currentDate + "," + url + "," + nonce;
var dataMode = request.dataMode;
var hmacInformation = "method=\"" + method + "\", url=\"" + url + "\",
contentType=\"" + contentType + "\"";

// NOTE: Manual MD5 is used in case of binary datamode, this MD5 hash
// should match the provided file (use md5sum or WinMD5Free to calculate)
var manualMD5 = "<paste-md5-hash-of-file-here>";

// NOTE!! If you use version 4.9.3+ and are adding the body as binary, add the
following line; otherwise comment out the next line!
// dataMode = "binary";
if (method == "POST" || method == "PUT") {
    var contentType = request.headers["content-type"];
    if (contentType !== undefined && contentType.length > 0) {
        contentType = contentType.split(';')[0];
        if (dataMode === "binary") {
            value += "," + contentType + "," + manualMD5;
        }
        else {
            var content = request.data;
            if (content !== undefined && content.length > 0) {
                var md5 = CryptoJS.MD5(content);
                value += "," + contentType + "," + md5;
                hmacInformation += ", contentType=\"" + contentType + "\", md5=\""
+ md5 + "\"";
            }
        }

    }
}
var hash = CryptoJS.HmacSHA256(value, privateKey);
var hashBase64 = CryptoJS.enc.Base64.stringify(hash);
var authorization = "HMAC clientId=\"" + clientId + "\", nonce=\"" + nonce + "\",
currentDate=\"" + currentDate + "\", signature=\"" + hashBase64 + "\"";
postman.setEnvironmentVariable("authorization", authorization);
postman.setEnvironmentVariable("hmacInformation", hmacInformation);
```

Wanneer je nu op de Send knop druk, zal allereerst het bovenstaande script worden uitgevoerd en de environment variable {{authorization}} worden berekend. Vervolgens zal de waarde worden toegevoegd aan de header met key "Authorization".

WAARSHUWING: Tijdens het gebruik van Postman als REST client, is het opgevallen dat, wanneer een body middels een extern bestand wordt aangeleverd, dit niet altijd vlekkeloos verloopt. Zo zijn we tegen problemen aangelopen met requests waarin de body alsnog leeg blijft hoewel in de interface duidelijk staat dat een bepaald Turtle bestand moet worden gebruikt. Het beste is om bij het opnieuw openen van Postman of na wijzigen van een bestand, éénmalig in Postman een selectie van een bestand te annuleren (hierdoor zal de body leeg gemaakt worden) en daarna nogmaals het betreffend Turtle bestand toe te voegen.

1.4.1 Content in Binary

Wanneer je de content niet wilt plakken in de body van de request, maar gebruik wilt maken van Postman's optie om een bestand toe te voegen (optie binary), dan zul je buiten Postman eerst de MD5 hash moeten berekenen voor het bestand, middels MD5 hash tools, zoals md5sum of WinMD5Free, en deze moeten plakken op de plek waar de tekst `<paste-md5-hash-of-file-here>` staat. Het script zal in binary mode zorgen dat de externe MD5 wordt gebruikt in het berekenen van de HMAC signature.

1.5 Overzicht van Services

Wederom zal het platform verschillende services ontsluiten, waarbij de configuratie van het publicatie platform kan worden aangepast. Deze services zijn beschreven in Swagger definities en hierdoor eenvoudig te gebruiken door andere ontwikkelaars. De volgende Swagger definities zijn beschikbaar:

- Admin API: http://acceptance2.laces.tech/crow-publication-v2/crow/ldp-admin/swagger.json
- ProContract API: http://acceptance2.laces.tech/crow-publication-v2/crow/context-services/procontract/swagger.json
- Kennisbank API: http://acceptance2.laces.tech/crow-publication-v2/crow/context-services/kennisbank/swagger.json

2 Versie 2 | Publiceren nieuwe content

2.1 Autorisatie

De autorisatie van aanroepen van onderstaande services zal verlopen op basis van een **HMAC Signature**, zoals eerder beschreven.

2.2 Voorbeeld Data

In onderstaande voorbeelden zullen we meerdere versies van een extreem simpel schema model importeren, de naam van dit model zal zijn "CROW Schema" en de namespace URI van dit specifieke model hebben we gelijkgesteld aan <http://linkeddata.crow.nl/publication-v2.2/ns/crow/2016/schema/>. De reden dat we deze namespace gebruiken is, omdat het berekenen van een delta of diff tussen twee versies, alleen zal worden uitgevoerd voor de namespace die gelijk is aan het adres waarop het model wordt aangeboden. In de onderstaande Turtle voorbeelden zal deze namespace ook zijn aangeduid met de prefix schema:, als ook als de base URI.

CROW Schema v1

Dit is de eerste versie van het schema, met daarin een definitie voor de ontologie als ook twee klassen en een property.

```
@base           <http://linkeddata.crow.nl/publication-v2.2/ns/crow/2016/schema/> .
@prefix schema: <http://linkeddata.crow.nl/publication-v2.2/ns/crow/2016/schema/> .
@prefix owl:    <http://www.w3.org/2002/07/owl#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

schema: a owl:Ontology ;
        rdfs:label "CROW Schema" ;
        rdfs:comment "This is the first version of the CROW Schema model." .

schema:Requirement a owl:Class ;
        rdfs:label "Requirement" ;
        rdfs:subClassOf owl:Thing .

schema:PhysicalObject a owl:Class ;
        rdfs:label "Physical Object" ;
        rdfs:subClassOf owl:Thing .

schema:hasPart a owl:ObjectProperty ;
        rdfs:label "has part" ;
        rdfs:domain schema:PhysicalObject ;
        rdfs:range schema:PhysicalObject .
```

Op het bovenstaande model worden vervolgens een aantal wijzigingen doorgevoerd.

- Er wordt een super klasse aan schema:Requirement toegevoegd schema:Specification toegevoegd
- De beschrijving van de ontologie wordt gewijzigd
- Er er wordt een niewe property schema:hasSpecification toegevoegd

Belangrijk om te zien is dat de base en namespace URI niet wijzigen aangezien het voor de modelleur geen ander model is.

```
@base            <http://linkeddata.crow.nl/publication-v2.2/ns/crow/2016/schema/> .
@prefix schema:  <http://linkeddata.crow.nl/publication-v2.2/ns/crow/2016/schema/> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

schema: a owl:Ontology ;
        rdfs:label "CROW Schema" ;
        rdfs:comment "This is the second version of the CROW Schema model." .

schema:Specification a owl:Class ;
        rdfs:label "Specification" ;
        rdfs:subClassOf owl:Thing .

schema:Requirement a owl:Class ;
        rdfs:label "Requirement" ;
        rdfs:subClassOf schema:Specification .

schema:hasSpecification a owl:ObjectProperty ;
        rdfs:label "has specification" ;
        rdfs:range schema:Specification .

schema:PhysicalObject a owl:Class ;
        rdfs:label "Physical Object" ;
        rdfs:subClassOf owl:Thing .

schema:hasPart a owl:ObjectProperty ;
        rdfs:label "has part" ;
        rdfs:domain schema:PhysicalObject ;
        rdfs:range schema:PhysicalObject .
```

De bovenstaande modellen zullen hieronder worden gebruikt.

2.3 Uploaden Model

2.3.1 Service URL

De delta import service is beschikbaar op het Publicatie Platform, onder het pad:

```
Service Path = {ApplicationUrl}/ns/{NamespacePart}/import
```

Hierin is de ApplicationUrl gelijk aan de locatie waar het Platform beschikbaar is gesteld, bijvoorbeeld http://linkeddata.crow.nl/publication-v2.2/. De NamespacePart bevat één of meerdere segmenten welke overeenkomen met de namespace binnen het aan te bieden model. Voor ons schema voorbeeld hierboven zal de NamespacePart dus gelijk zijn aan "crow/2016/schema".

LET OP: Gebruik van de onderstaande segmenten in de NamespacePart is niet toegestaan.

- /import
- /version
- /latest
- /list
- /delta
- /revert

Deze keywords worden reeds gebruikt in operaties en wanneer ze toegevoegd worden aan de NamespacePart tot ongewenste resultaten leiden.

Het volledige adres van de web service voor het importeren van het schema zal dus zijn:

```
Service URL = http://linkeddata.crow.nl/publication-v2.2/ns/crow/2016/schema/import
```

Elke volgende versie zal naar hetzelfde adres moeten worden gestuurd, om daarmee af te dwingen dat het verschil tussen de laatst geüploade versie en de nieuwe versie kan worden berekend.

Het gedeelte voor het laatste "import" keyword, in bovenstaande service URL, noemen we de **base URI** van een model.

2.3.2 Aanroepen Service

Let op dat de access key, welke gebruikt wordt in de aanroep van de import delta service, rechten moet hebben op het pad "/.*".

Naast het meesturen van het volledige model in de body van een request, zal de volgende informatie moeten worden toegevoegd.

- Authorization header, zie Versie 2 | Beheren van LDP (via HMAC)
- Content-Type header, op dit moment accepteert de service alleen nog maar "text/turtle" voor modellen uitgedrukt in Turtle, of "application/rdf+xml" voor het RDF in XML formaat.

Optioneel is het mogelijk om direct een naam op te geven voor de nieuwe namespace als ook om de namespace direct te activeren. De naam kan worden meegegeven door de querystring parameter name te gebruiken. Indien geen naam wordt meegegeven zal de uiteindelijk versie URL gebruikt worden als naam. Om de namespace direct actief te maken kan de parameter enabled worden gebruik welke in dit geval de waarde true moet meekrijgen.Hieronder staat een volledige aanroep voor het uploaden van versie 1.

```
POST /publication-v2.1/ns/crow/2016/schema/import?name=CROW%20Schema%20v1 HTTP/1.1
Host: linkeddata.crow.nl
Authorization: HMAC clientId="admin", nonce="06e89e4b-9d1e-4d83-bdec-e4ff073a1d11",
currentDate="2016-11-17T15:56:58Z",
signature="RFdFI0//79XRt+w+vAJ8EEFIlsEy9hlr8MaJjZQ8JSY="
Content-Type: text/turtle
```

In de response zal de server aangeven waar het model ter beschikking wordt gesteld, bovenstaande call retourneerde als response: "Import of content successful, and the graph is

accessible at http://linkeddata.crow.nl/publication-v2.2/ns/crow/2016/schema/version/799279295866777600". Ook bevat de response header Location de URL van deze versie van het CROW Schema.

2.3.3 Wijzigen van Namespace

Nadat een model is geüpload via de import delta service, zal er een namespace entiteit worden toegevoegd aan de lijst van namespaces. Door een call te doen naar het adres http://linkeddata.crow.nl/publication-v2.2/contexts/cpc-admin/namespaces zien we dat de lijst van namespaces onze eerste versie van het CROW Schema bevat:


```
...
  {
    "attributes": [
      {
        "name": "urn:semmweb:namespaces:created",
        "value": "2016-11-17T16:53:28Z"
      }
    ],
    "id": "799279295866777600",
    "name": "CROW Schema v1",
    "enabled": true,
    "uri": "http://linkeddata.crow.nl/publication-
v2.2/ns/crow/2016/schema/version/799279295866777600"
  }
...
```


Via de LDP Admin webservice, en specifiek de PUT op de namespaces web service, kan de informatie van de namespace worden bewerkt (details over de precieze aanroep zijn te vinden in de Swagger documentatie van deze service op http://linkeddata.crow.nl/publication-v2.2/crow/ldp-admin/swagger.html).

2.4 Overige Operaties

Onderstaande operaties verwachten dat gebruikers zich via een HMAC autoriseren.

2.4.1 /version/{VersionID}

Om een specifieke versie van een model op te vragen, moet achter de base URI van het model het keyword /version/{VersionID} worden toegevoegd. Voor onze eerste versie van het schema was dit http://linkeddata.crow.nl/publication-v2.2/ns/crow/2016/schema/version/799279295866777600. Deze URL is terug te vinden in de property URI in de namespaces web services en wordt tevens geretourneerd door een import via de Location header. Achter het keyword /version is de unieke (numerieke) identificatie van de versie/namespace gegeven.

Er kan gebruik worden gemaakt van de Accept header in een request, om het betreffende model terug te krijgen in "text/turtle"- of "application/rdf+xml"-formaat.

2.4.2 /version/latest

Wanneer niet bekend is welke versie van een model de laatste is, kan gebruik worden gemaakt van de combinatie /version/latest, in plaats van de numeriek identificatie. Zo zal na het uploaden van de eerste versie, de URL http://linkeddata.crow.nl/publication-v2.2/ns/crow/2016/schema/version/latest hetzelfde resultaat geven als http://linkeddata.crow.nl/publication-v2.2/ns/crow/2016/schema/version/799279295866777600.

Er kan gebruik worden gemaakt van de Accept header in een request, om het betreffende model terug te krijgen in "text/turtle"- of "application/rdf+xml"-formaat.

2.4.3 /list

Een lijst van alle versies van een model kan worden opgevraagd door het keyword /list achter de base URI te plaatsen. Bijvoorbeeld http://linkeddata.crow.nl/publication-v2.2/ns/crow/2016/schema/list geeft het volgende JSON resultaat:

```
[
  {
    "id": 799283849643601920,
    "versioned_graph": "http://linkeddata.crow.nl/publication-
v2.2/ns/crow/2016/schema/799283849643601920",
    "graph": "http://linkeddata.crow.nl/publication-v2.2/ns/crow/2016/schema",
    "date": "2016-11-17T17:11:34Z",
    "creator": "http://linkeddata.crow.nl/publication-v2.2/user/anonymous"
  },
  {
    "id": 799279295866777600,
    "versioned_graph": "http://linkeddata.crow.nl/publication-
v2.2/ns/crow/2016/schema/799279295866777600",
    "graph": "http://linkeddata.crow.nl/publication-v2.2/ns/crow/2016/schema",
    "date": "2016-11-17T16:53:28Z",
    "creator": "http://linkeddata.crow.nl/publication-v2.2/user/anonymous"
  }
]
```

2.4.4 /delta/{FromVersionID}/{ToVersionID}

Om het verschil te zien tussen twee versies, kan /delta/{FromVersionID}/{ToVersionID} worden gebruikt. Het resultaat zal een TriG resultaat zijn (zie https://www.w3.org/TR/trig/) in de default graph informatie over de berekende delta en daarnaast een graph for "added" triples en "removed" triples. Zo geeft de http://linkeddata.crow.nl/publication-v2.2/ns/crow/2016/schema/delta/799283849643601920 als resultaat:

```
<http://semmtech.com/versions> {
        # Graph met informatie over de <urn:delta:source> en <urn:delta:target>
        # namespaces gebruikt in de berekening van de delta.
}
<urn:delta:added> {
        # Graph met toegevoegde triples.
}
<urn:delta:removed> {
        # Graph met verwijderde triples.
}
```

3 Bevragen gepubliceerde content

3.1 Introductie

Deze pagina beschrijft hoe de content gepubliceerd op het LDP platform kan worden benaderd en welke queries er kunnen worden uitgevoerd op deze content. De onderstaande informatie is bedoeld voor ontwikkelaars die de content uit het LDP platform willen benaderen, om deze vervolgens in hun eigen systeem te gebruiken. Er wordt vanuit gegaan dat deze ontwikkelaars bekend zijn met (RESTful) webservices en concepten die nodig zijn voor het aanroepen van dergelijke webservices.

Daarnaast zullen alle calls die gedaan worden altijd moeten worden voorzien van een geldige HMAC signature, voor details en procedure hoe deze op te stellen, zie Beheren van LDP via HMAC.

3.2 SPARQL Services

Er zijn in deze versie van het platform een tweetal services ontsloten voor het bevragen van het platform. Beide services zullen op basis van een set aan parameters bepalen welke namespaces zullen moeten worden opgenomen in de uit te voeren query. Beide service verwachten altijd een Authorization header (zie Beheren van LDP via HMAC) om de aanroep te kunnen authenticeren.

3.2.1 GET

Indien de SPARQL service wordt benaderd middels een GET operatie zal er een query parameter in de querystring moeten worden meegegeven met hierin de betreffende query. Omdat dit een querystring parameter is, zal de opgegeven query eerst moeten worden ge-URL-encode alvorens mee te sturen. Nemen we bijvoorbeeld onderstaande query:

```
select * { ?s ?p ?o } limit 10
```

Zal deze middels een URL encoding tool, zoals http://www.urlencoder.org/, moeten worden omgezet naar een veilige string die in een URL kan worden meegegeven. URL encoded variant van bovenstaande query is:

```
select%20%2A%20%7B%20%3Fs%20%3Fp%20%3Fo%20%7D%20limit%2010
```

LET OP! Officieel mag een asterisk karakter ongecodeerd voorkomen in een URL. Echter hebben we situaties gehad waarbij na ontvangst van een request de '*' alsnog gecodeerd is (buitenom de client die gebruikt is). De oorzaak hiervoor is nog niet bekend. Tot die tijd wordt aangeraden om, indien een '*' wordt gebruikt in de query, deze ook te coderen middels '%2A'.

We zullen in onderstaande services voorbeelden geven waarin we bovenstaande query zullen uitvoeren.

3.2.2 POST

Wanneer de POST operatie wordt gebruikt kan de query gewoon worden opgegeven als de body van de request. Hierbij zal wel de Content-Type header worden worden meegegeven met als waarde "application/sparql-query".

3.2.3 Output Formats

Wanneer er een Accept header (of output querystring parameter) wordt meegegeven met één van onderstaand waarde, zal het resultaat van de query in dit formaat worden geretourneerd. Indien de waarde onbekend is, of niet wordt meegegeven, zal standaard het resultaat in XML worden teruggegeven. Wanneer zowel een Accept als output worden meegegeven prevaleert de output over Accept.

Accept (MIME-type)	Parameter	Beschrijving
application/sparql-results+xml	xml	Geeft het resultaat terug in JSON formaat (standaard waarde)
application/sparql-results+json	json	Geeft het resultaat terug in JSON formaat (standaard waarde)
text/csv	csv	Geeft het resultaat terug in CSV (komma gescheiden) formaat (standaard waarde)
text/tab-separated-values	tabs	Geeft het resultaat terug in TSV (tabs gescheiden) formaat (standaard waarde)
text/html	html	Geeft het resultaat terug in HTML opmaak met daarin een tabel met binding resultaten (standaard waarde)

3.2.4 CROW ProContract (CPC)

De service voor het benaderen van ProContract data is beschikbaar op het onderstaande pad:

```
/contexts/cpc/select
```

De volledige URL van deze service op de release 2.2 op de acceptatie server zal dus zijn:

http://acceptatie.linkeddata.crow.nl/publication-v2.2/contexts/cpc/select

Naast de query is er nog een aantal (querystring) parameters welke optioneel kunnen worden meegegeven:

Parameter	Beschrijving
projectId	ID van het project
projectToolCode	Autorisatie code welke nodig is als extra validatie op de tool en project combinatie. De code is dus uniek voor de tool en project ID combinatie.
trace	Wanneer deze de waarde "namespaces" zal extra informatie worden teruggegeven in de Trace response headers over de gebruikte namespaces (URIs).
output	Zie "Output Formats"

Op basis van bovenstaande parameters zal elke request de volgende logica gebruiken om de lijst van (session) namespaces te bepalen. Over deze verzameling van namespaces zal vervolgens de de query worden uitgevoerd:

1. Project namespaces: indien project met ID bestaat, het project actief is (enabled) én de projectToolCode klopt voor combinatie project ID en tool, zullen alle namespaces worden gebruikt waartoe het project rechten heeft.
2. Tool namespaces: alle namespaces waartoe de tool rechten heeft. Tool wordt bepaald door te kijken naar de client ID en access key die gekoppeld is aan deze tool.
3. Open namespaces: alle namespaces die een attribuut met key "urn:semmweb:namespaces:openNamespace" kennen (ongeacht de waarde van dit attribuut).
4. Context Namespaces: alle namespaces welke gekoppeld zijn aan de context met naam "cpc".
5. Subscription namespaces: alle namespaces die gekoppeld zijn aan de subscription behorende bij de tool.

Een volledige aanroep van de SPARQL query hierboven op de ProContract service:

```
GET http://acceptatie.linkeddata.crow.nl/publication-
v2.2/contexts/cpc/select?query=select%20%2A%20%7B%20%3Fs%20%3Fp%20%3Fo%20%7D%20limi
t%2010&projectId=1234&projectToolCode=password
```

Uiteraard zal hier een geldige Authorization header aan moeten worden toegevoegd waarin een HMAC wordt berekend met de volledige URL.

3.2.5 CROW Kennisbank (CKB)

De service voor het benaderen van Kennisbank data is beschikbaar op het onderstaande pad:

```
/contexts/ckb/select
```

De volledige URL van deze service op de release 2.2 op de acceptatie server zal dus zijn:

http://acceptatie.linkeddata.crow.nl/publication-v2.2/contexts/ckb/select

Naast de query zijn er nog een aantal (querystring) parameters welke optioneel kunnen worden meegegeven:

Parameter	Beschrijving
trace	Wanneer deze de waarde "namespaces" heeft zal extra informatie worden teruggegeven in de Trace response headers over de gebruikte namespaces (URIs).
output	Zie "Output Formats"

Op basis van bovenstaande parameters zal elke request de volgende logica gebruiken om de lijst van (session) namespaces te bepalen. Over deze verzameling van namespaces zal vervolgens de de query worden uitgevoerd:

1. Tool namespaces: alle namespaces waartoe de tool rechten heeft. Tool wordt bepaald door te kijken naar de client ID en access key die gekoppeld is aan deze tool.
2. Open namespaces: alle namespaces die een attribuut met key "urn:semmweb:namespaces:openNamespace" kennen (ongeacht de waarde van dit attribuut).
3. Context Namespaces: alle namespaces welke gekoppeld zijn aan de context met naam "cpc".
4. Subscription namespaces: alle namespaces die gekoppeld zijn aan de subscription behorende bij de tool.

Een volledige aanroep van de SPARQL query hierboven op de Kennisbank inclusief de trace zal zijn:

```
GET http://acceptatie.linkeddata.crow.nl/publication-
v2.2/contexts/ckb/select?query=select%20%2A%20%7B%20%3Fs%20%3Fp%20%3Fo%20%7D%20limi
t%2010&trace=true
```

Uiteraard zal hier een geldige Authorization header aan moeten worden toegevoegd waarin een HMAC wordt berekend met de volledige URL.

3.3 Voorbeeld SPARQL queries

Hieronder volgen twee voorbeelden voor het bevragen van data.

1. Decompositie van objecten (bijv. fysieke objecten of activiteiten)
2. Lijst van specificaties

3.3.1 Decompositie van objecten

Voor objecten in een decompositie zijn we geïnteresseerd in de volgende informatie:

- Name
- Description
- Remark
- Parent-child relatie

De root elementen van een hiërarchische structuur presenteren.

```
# This query returns assembly roots, which are resources which are themselves not
being used as parts in the scheme:hasConceptAsPart relation.
# the part has itself.
# OUTPUT:
#   ?root = URI of the root resource
#   ?nameOfRoot = Name of the root resource
#   ?childCount = Number of children (parts) the root has
#   ?descriptionOfRoot = Description of the root resource
#   ?remarkOfRoot = Remark of the root resource
SELECT DISTINCT ?root ?nameOfRoot ?descriptionOfRoot ?remarkOfRoot (COUNT(?part) AS
?childCount)
WHERE { }
GROUP BY ?root ?nameOfRoot ?descriptionOfRoot ?remarkOfRoot
ORDER BY LCASE(?nameOfRoot)
```

De onderliggende elementen van een hiërarchische structuur presenteren.

```
# This query returns assembly wholes and their parts, together with the number of
children
# the part has itself.
# INPUT:
#   ?whole = Can be used to find only parts for the given whole (URI)
# OUTPUT:
#   ?part = URI of the assembly part
#   ?nameOfPart = Name of assembly part
#   ?partChildCount = Number of children (parts) the assembly part has
#   ?descriptionOfPart = Description of the part
#   ?remarkOfPart = Remark on the part
SELECT DISTINCT ?part ?nameOfPart ?descriptionOfPart ?remarkOfPart
(COUNT(?grandchild) AS ?partChildCount)
WHERE { }
GROUP BY ?part ?nameOfPart ?descriptionOfPart ?remarkOfPart
ORDER BY LCASE(?nameOfPart)
```

3.3.2 Lijst van specificaties

Voor specificaties zijn we geïnteresseerd in de volgende informatie:

- Name
- Definition
- Remark
- Code
- Status
- Source document (Name, Description)
- Verification method (Name, Description)
- Related Subject (for example physical objects, activities)

Specificatie definitie

```
# This query returns requirements.
# INPUT:
#   ?description = Description filter for requirements
#   ?title = Title filter for requirements
# OUTPUT:
#   ?requirement = URI of the requirement
#   ?label = Label of requirement
#   ?definition = Definition of requirement
#   ?code = Code of requirement
#   ?status = Status of requirement
#   ?remark = Remark of requirement
SELECT ?requirement ?label ?definition ?code ?status ?remark
WHERE { }
```

Koppeling van een specificatie aan een brondocument

```
# This query returns source documents attached to a given requirement
# INPUT:
#   ?requirement = URI of the requirement for which source documents should be
given
# OUTPUT:
#   ?document = URI of the document
#   ?name = Name or label of the document
#   ?description = Description or definition of the document
SELECT ?document ?name ?description
WHERE { }
```

Verificatiemethode van een specificatie

```
# This query returns verification methods attached to a given requirement
# INPUT:
#   ?requirement = URI of the requirement for which verification methods should be
given
# OUTPUT:
#   ?method = URI of the method
#   ?name = Name or label of the method
#   ?description = Description or definition of the method
SELECT ?method ?name ?description
WHERE { }
```

Koppeling van een specificatie aan een subject

```
# This query returns subjects for which requirements exist.
# INPUT:
#   ?subject = URI of the subject for which requirements should be given
# OUTPUT:
#   ?requirement = URI of the requirement
#   ?label = Label of requirement
#   ?definition = Definition of requirement
#   ?code = Code of requirement
#   ?status = Status of requirement
#   ?remark = Remark of requirement
SELECT ?requirement ?label ?definition ?code ?status ?remark
WHERE {  }
```