Hypercat 3.00 Specification

Version	Authors	Change
2.0-rc1-2015-06-25	Toby Jaffey, 1248 John Davies, BT Pilgrim Beart, 1248	
2.01-rc1-2015-10-27	Pilgrim Beart, 1248	1. Added required hasDescription rel to example search catalogue 2. Changed multi-search from a POST to a GET
3.00-rc1-2016-02-23	Pilgrim Beart, 1248	1. Changed “HyperCat” to “Hypercat” throughout 2. Changed to new logo 3. Changed top-level catalogue JSON field name “item-metadata” to “catalogue-metadata” to more accurately reflects its function [as per BSI PAS212 public consultation process] 4. Changed per-item JSON field name “i-object-metadata” to “item-metadata” to more accurately reflect its function [ditto] 5. Renamed “Substring match search” to “Prefix match search” to more accurately reflect its function [ditto]. a. The body text changes b. The constant `"urn:X-hypercat:search:substring"` is therefore replaced with the constant `"urn:X-hypercat:search:prefix"` c. Query parameters `substring-rel/val/href` become `prefix-rel/val/href`

Hypercat is an open, lightweight JSON-based hypermedia catalogue format for exposing collections of URIs. Each Hypercat catalogue may expose any number of URIs, each with

any number of RDF-like triple statements about it. Hypercat is simple to work with and allows developers to publish linked-data descriptions of resources.

Hypercat is designed for exposing information about IoT assets over the web. It allows a server to provide a set of resources to a client, each with a set of semantic annotations. Implementers are free to choose or invent any set of annotations to suit their needs. Where implementers choose similar or overlapping semantics, the possibilities for interoperability are increased.

This specification is draft and subject to change.

Table of Contents

Specification items marked (M) are Mandatory.
Specification items marked (O) are Optional.

Hypercat 3.0 URN Change Notice

Hypercat File Format Specification (M)

Hypercat Server API Specification (O)

Hypercat Subscription (O)

Hypercat Resource Subscription (O)

Hypercat Signing (O)

Hypercat Security Access Hints (O)

Hypercat Security Credential Acquisition (O)

Hypercat Geographic Bounding Box Search (O)

Hypercat Lexicographic Range Search (O)

Hypercat Robots Exclusion Search (O)

Hypercat Multi-Search (O)

Hypercat Prefix Match Search (O)

Hypercat Linked Data rel (O)

Hypercat License rel (O)

Terms for requirement levels (MUST, MAY, SHOULD, etc.) are to be interpreted as in http://www.ietf.org/rfc/rfc2119.txt

Hypercat 3.0 URN Change Notice

Hypercat 3.0 deprecates the Hypercat 1.1 use of `tsbiot` in favour of `hypercat`.

Hypercat 1.1 Naming

Hypercat was developed as a solution to the IoT interoperability and data sharing needs of the Technology Strategy Board IoT Demonstrator Programme. As the 1.0 specification was formalised before the adoption of the name Hypercat it uses the term “tsbiot” in URN namespaces and the document MIME type.

In Hypercat 3.0 all occurrences of `tsbiot` are replaced by `hypercat`.

This includes all URNs, e.g. `urn:X-tsbiot:rels:supportsSearch` becomes `urn:X-hypercat:rels:supportsSearch`.

This includes the document MIME type, which now becomes `application/vnd.hypercat.catalogue+json`

Hypercat need not apply for a formal URN namespace at this time (see RFC3406).

Impact

New Hypercat clients and servers MAY choose to support the old-style naming in addition to new. However, existing clients will require updating to support Hypercat 3.0 documents.

Hypercat File Format Specification (M)

This section defines the format of Hypercat documents.

Hypercat Catalogues

A Hypercat catalogue is a file representing an unordered collection of resources on the web. Each item in a catalogue refers to a single resource by its URI, which may itself be a further catalogue.

All resource URIs in a catalogue MUST be unique within the catalogue, so they can be referred to individually. The same resource MUST NOT appear more than once in the same catalogue object's items (even if the entries have different metadata).

A catalogue MAY provide metadata for itself and MAY provide metadata for each catalogue item.

Catalogue Object

A catalogue object is a JSON object.
A catalogue object MUST contain all of the following properties:

Property Name	Meaning	Property Value
`items`	List of items	JSON array of zero[1] or more item objects
`catalogue-metadata`	An array of metadata objects describing the catalogue object	JSON array of metadata objects

Item Object

An item object (from the `items` array) is a JSON object, which MUST contain all of the following properties:

Property Name	Meaning	Property Value

[1] *An empty catalogue will not expose any data, hence does not allow any useful interoperability*

`href`	Identifier for the resource item	URI as a JSON string
`item-metadata`	An array of metadata objects describing the resource item	JSON array of metadata objects

The `catalogue-metadata` and `item-metadata` arrays MUST contain at least one metadata object for each of the mandatory metadata object relationships.

The `catalogue-metadata` and `item-metadata` arrays MAY contain multiple metadata objects with the same `rel` (and `val`) properties (they are bags/multisets of features).

Metadata Object

A metadata object is a JSON object which describes a single relationship between the parent object (either the catalogue or catalogue item) and some other entity or concept denoted by a URI.

All metadata objects MUST include all of the following properties:

Property Name	**Meaning**	**Property Value**
`rel`	A relationship between the parent object and a target noun, expressed as a predicate (verb)	URI of a relationship as a JSON string
`val`	The entity (noun) to which the `rel` property applies	JSON string (optionally URI of concept or entity). Where URLs are used, they MAY be relative. Relative links MUST be interpreted as in RFC 1808. MAY be empty.

For example, the following metadata MUST be interpreted as saying that the parent object is of colour blue:

```
{"rel":"urn:X-hypercat:rels:isColour","val":"blue"}
```

Mandatory Metadata Relationships

All arrays of metadata must include all of the following relationships:

rel value	**Meaning**	**val value**
urn:X-hypercat:rels:hasDescription:en	Resource has a human readable description in English	Description as a JSON string

In addition, all top-level catalogue-metadata objects (describing the catalogue object) MUST include all of the following relationships:

rel value	**Meaning**	**val value**
urn:X-hypercat:rels:isContentType	Data provided by resource is of given type	application/vnd.hypercat.catalogue+json

Optional Metadata Relationships

A metadata object MAY include any or all of the following relationships where applicable:

rel value	**Meaning**	**val value**
urn:X-hypercat:rels:isContentType	Data provided by resource SHOULD be of given MIME type	RFC2046 MIME type as JSON string (eg. "text/csv")
urn:X-hypercat:rels:hasHomepage	A reference to a human readable web page concerning the resource	URL as a JSON string
urn:X-hypercat:rels:containsContentType	This catalogue contains resources of given content type. Only meaningful for metadata objects contained by or pointing to catalogue objects	RFC2046 MIME type as JSON string (eg. "text/csv")
urn:X-hypercat:rels:supportsSearch	This catalogue's server supports a search mechanism. Only meaningful for metadata objects contained by or pointing to catalogue objects	URI of a defined search mechanism as a JSON string

MIME Type

A catalogue is represented by a JSON document of MIME type `application/vnd.hypercat.catalogue+json` containing a single catalogue JSON object.

Extensibility

New forms of metadata MAY be added by defining new metadata relationships. A client encountering an unknown metadata relationship SHOULD ignore it. Servers and clients are free to support different sets of metadata relationships.

For example, indication of a new search method supported by a catalogue server MAY be added, discovered and identified by defining a new value for the relation `urn:X-hypercat:rels:supportsSearch`

A new language or style for human readable descriptions MAY be added, discovered and identified by defining a new `urn:X-hypercat:rels:hasDescription` variant (e.g. German `urn:X-hypercat:rels:hasDescription:de`).

A complete replacement catalogue object format MAY be implemented by declaring a new MIME type for `urn:X-hypercat:rels:isContentType`. Old style catalogues may point to new style and vice versa without version ambiguity.

In addition to the properties and object structures specified in this section, a catalogue may contain any number of other properties and objects as implementers see fit.

It is RECOMMENDED that ALL metadata extensions be confined to defining new valid `rel`/`val` data pairs.

Hypercat Server API Specification (O)

This section defines an optional HTTP/HTTPS API which may be provided to allow clients to interact with Hypercat catalogues.

Every Hypercat server MUST provide a publicly readable `/cat` endpoint serving a Hypercat document. All other API operations are (O), optional.

Catalogue Operations

Operations to create, read, update and delete individual items within a catalogue are defined. Operations to create, update and delete entire catalogues are NOT defined and are considered out of scope.

Read Catalogue

To read an entire catalogue, a client MAY `GET` the catalogue URL.
On success, the server MUST return a JSON catalogue object.
On success, the server MUST return an HTTP 200 status code.
Reading part of a catalogue is accomplished using one of the defined search mechanisms.

Create / Insert Catalogue Item

To add a new item to a catalogue, a client MAY `POST` an item object (JSON) to a catalogue URL.
The server MAY place the new item object in any catalogue it chooses. The ability for the server to choose the catalogue allows a server to organise "uploaded" resource catalogues in any pattern it wishes.
On success, the server MUST return an HTTP location header with the URL of the catalogue to which the item was added.
On success, the server MUST return an HTTP 201 status code.
Creating an entire, new catalogue is not defined and is out of scope.

Read Catalogue Item

To read an individual item from a catalogue, a client MAY use the “simple search” mechanism, or any other defined search mechanism supported by the catalogue server.

Update Catalogue Item

To replace an existing item in a catalogue, a client MAY `PUT` or `POST` an item object to a catalogue URL. To specify which existing item is to be replaced, the client MUST provide a

query parameter of `href` with a value corresponding to a resource URI held in the catalogue (appropriately escaped for URI use).

On success, the server MUST respond with a status code of 200. On success, the server MAY respond with the item object.
Updating an entire catalogue in a single operation is not defined and is out of scope.

Note that a `POST` operation can create or update an item, whereas a `PUT` will only update.

Delete Catalogue Item

To delete an item from a catalogue, a client MAY `DELETE` from a catalogue URL. To identify the item to be deleted, the client MUST use a query parameter of `href` with a value corresponding to a resource URI held in the catalogue (appropriately escaped for URI use).

On success, the server MUST respond with a status code of 200.

Deleting an entire catalogue, as opposed to individual items within a catalogue, is not defined and is out of scope.

Authentication

All HTTP(S) requests may be authenticated with a key, which is a URI.
All keys must be valid URIs.
Keys may be presented in two ways.

Basic-auth (RFC2617) may be used, with the key as username and no password provided. Such a header is constructed as follows:

```
"Authorization: " + base64(KEY + ':' + '')
```

Alternatively, a key may be passed in an x-api-key header, constructed as follows:

```
"x-api-key: " + KEY
```

How keys are generated, distributed and mapped to permissions in a hub are out-of-scope and implementation specific.

HTTP Status Codes

All HTTP requests MUST return a valid response or an appropriate status code.

Unless otherwise specified, the server MUST return one of the following status codes to indicate the cause of any failure:

Code	Meaning
204	"No Response"
400	"Bad request" (e.g. malformed input)
409	"Conflict" (e.g. insert existing href)
401	"Unauthorised"
404	"Not found"
501	"Not implemented"

Catalogue Search Extension

Catalogues may be searched (filtered) to find items matching a specified set of metadata.

If a catalogue provides a search capability it MAY advertise it to a client through having a `"urn:X-hypercat:rels:supportsSearch"` metadata relation.

Simple Search Mechanism

Simple search is a Hypercat extension allowing the search or filtering of a Hypercat catalogue according to specified parameters.

A Hypercat server MAY advertise that it supports the simple search mechanism by providing the metadata relation `"urn:X-hypercat:rels:supportsSearch"`/`"urn:X-hypercat:search:simple"` in a catalogue object's `catalogue-metadata`.

A simple search is performed by providing a query string (http://tools.ietf.org/html/rfc1738) to a catalogue. If multiple search parameters are supplied, the server MUST return the intersection of items where the all search parameters match in a single item, combining the parameters with boolean AND.

A “simple search” searches only a single catalogue resource. It does not include other linked or nested catalogues.

All query parameters MUST be URL encoded. All query parameters are optional.

Parameter	Meaning	Allowed Value
`rel`	Any metadata relation	URI as a JSON string
`val`	Any metadata value	URI as a JSON string
`href`	A resource URI	URI as a JSON string

Examples

Given the following catalogue:

```
{
    "catalogue-metadata":[
        {
             "rel":"urn:X-hypercat:rels:isContentType",
             "val":"application/vnd.hypercat.catalogue+json"
        },
        {
            "rel":"urn:X-hypercat:rels:hasDescription:en", "val":""
        },
        {
            "rel":"urn:X-hypercat:rels:supportsSearch",
            "val":"urn:X-hypercat:search:simple"
        }
    ],
    "items":[
        {
            "href":"http://A",
            "item-metadata":[
                {
                     "rel":"urn:X-hypercat:rels:hasDescription:en",
                     "val":"example item A"
                },
                {
                    "rel":"urn:X-hypercat:rels:1",
                    "val":"1"
```

```
                },
                {
                    "rel":"urn:X-hypercat:rels:2",
                    "val":"2"
                },
                {
                    "rel":"urn:X-hypercat:rels:3",
                    "val":""
                }
            ]
        }
    ]
}
```

The following query strings would all return a catalogue containing the one item above:

```
?rel=urn:X-hypercat:rels:1
?rel=urn:X-hypercat:rels:2
?rel=urn:X-hypercat:rels:3
?val=1
?val=2
?val=
?rel=urn:X-hypercat:rels:1&val=1
?rel=urn:X-hypercat:rels:3&val=
```

The following query strings would all return a catalogue with no items:

```
?rel=urn:X-hypercat:rels:4
?val=3
?rel=urn:X-hypercat:rels:1&val=2
?rel=urn:X-hypercat:rels:1&val=
```

Hypercat Subscription (O)

This section describes a simple subscription system for Hypercat catalogues using Server Sent Events. The core Hypercat specification provides an API for polling catalogues, but does not provide any means for subscribing and listening for changes. This section adds an optional mechanism for clients to subscribe to changes to a Hypercat where the server supports it.

Server Sent Events

The MIME media type `text/event-stream` describes a streaming text format for passing notifications to a listening client. Each message is delivered discretely over a streamed channel and contains both a name and body.
When used with HTTP(S), `text/event-stream` is supported in modern browsers using the HTML5 EventSource API. Provided CORS requirements are met, client-side javascript applications can stream from servers and react to events.
Server-side, supporting `text/event-stream` is as simple as writing event text to a held open HTTP(S) connection.

Model

A client subscribed to a stream of events from a Hypercat server will receive a stream of events. Every event contains an event name and a body. By first fetching a catalogue with HTTP(S), then accumulating these events, a client may keep a synchronised local copy of the catalogue being watched.

Event Endpoints

Authentication may be required to access the event endpoint, clients should assume that the same authentication systems can be used as were to fetch the catalogue over HTTP(S).

A catalogue which can be subscribed to MUST be annotated with the following metadata, this informs clients that the subscription method is available.

rel	**val**
`urn:X-hypercat:rels:eventsource`	URL of event endpoint

Event Format

All eventsource events have a name and a body.
For events concerning a specific catalogue item within a catalogue, the event name is the catalogue identifier concatenated with the URL encoded href of the catalogue item.

`<item href>` is the URL href field from a catalogue item

Action	**Event Name**	**Event Body**
Notification of item change	<item href>	Hypercat item object
Notification of item deletion	<item href>	Empty string

Example

Client fetches a catalogue, sees the available subscription method and begins streaming events from the provided endpoint

```
GET /cat/events
```

Client recieves an item update event for item http://example.org/item

```
id: 14:23:51
event: http://example.org/item
data:
{"href":"http://example.org/item","item-metadata":[{"rel":"urn:X-
hypercat:rels:hasDescription:en","val":"thing"},{"rel":"...","val
":"..."},{"rel":"...","val":"..."}]}
```

Client recieves an item deletion event for item http://example.org/item

```
id: 14:23:51
event: http://example.org/item
data:
```

Search

This section covers the case of streaming every change to an item in a Hypercat. However, some users may wish to only receive events pertaining to particular items, or where `rel` and `val` criteria are met. This filtering might be provided by passing URL parameters describing criteria in the streamed GET request.
The details of the search mechanism are out of scope for this section.

Retrofitting

The simplest Hypercat server is a traditional HTTP server. Implementing this section requires a more capable server, able to hold open long-lived connections and update clients with events. The mechanisms in this section can be implemented as a proxy for an existing Hypercat server, polling from the existing server and pushing events to clients. This enables adoption of subscription and streaming without impacting existing servers.

Alternative Technologies

MQTT

MQTT is a popular lightweight pubsub protocol where clients connect to a central broker. The MQTT protocol is designed for both publishing and subscribing and has a compact on-the-wire representation to allow use by lightweight devices.
MQTT runs over TCP and as such every client and server needs an MQTT implementation. Security is provided by using SSL/TLS for the TCP socket and exchanging credentials with the broker.

Although MQTT clients exist for most languages and programming environments, there are relatively few brokers. Every catalogue server wishing to implement an event interface based on MQTT would either need to implement an MQTT broker or connect to one.
If a separate broker is used, its authentication system must be linked to the catalogue server.
In contrast, server sent events are implemented as line oriented ASCII over an HTTP session.

EventSource Info

(Taken from http://www.w3.org/TR/2012/WD-eventsource-20120426/)

To enable servers to push data to Web pages over HTTP or using dedicated server-push protocols, this specification introduces the EventSource interface.

Using this API consists of creating an EventSource object and registering an event listener.

```
var source = new EventSource('updates.cgi');
source.onmessage = function (event) {
  alert(event.data);
};
```

On the server-side, the script ("updates.cgi" in this case) sends messages in the following form, with the text/event-stream MIME type:

```
data: This is the first message.

data: This is the second message, it
data: has two lines.

data: This is the third message.
```

Authors can separate events by using different event types. Here is a stream that has two event types, "add" and "remove":

```
event: add
data: 73857293

event: remove
data: 2153

event: add

data: 113411
```

The script to handle such a stream would look like this (where addHandler and removeHandler are functions that take one argument, the event):

```
var source = new EventSource('updates.cgi');
source.addEventListener('add', addHandler, false);
source.addEventListener('remove', removeHandler, false);
```

Hypercat Resource Subscription (O)

This section describes the issue of subscription to resources discovered in Hypercat catalogues.

Resource Subscription

Hypercat provides the ability to link to resources by URL/URI. Often, those resources may contain data required by applications which can change in real-time. There exist multiple IoT use-cases where client applications require real-time data feeds from devices, hubs or other services.

Catalogues vs. Resources

Hypercat catalogues are JSON documents, typically served over HTTP(S). A Hypercat catalogue may list any number of resources by URI of any type, accessed with any protocol. Due to the wide range of resource types which may be linked, it is difficult to imagine a single, universal, mechanism for subscribing to resource data.

One possibility, considered in the past, is to place all data directly into Hypercat catalogues. For example, having a temperature sensor generate live data into a Hypercat document as item metadata. While this approach does enable existing Hypercat subscription methods to be used, it falls short. Hypercat has a simple data model of rel, val pairs of metadata, where vals are always strings. While this is ideal for ensuring interoperability of high-level metadata and aiding discovery of resources, it is a poor fit for real-world IoT asset and device data. For this reason, structured, live or complex IoT data is often held in resources, referenced in Hypercat documents by URL/URI.

Resource Types

Today, there already exist many media-types protocol combinations suitable for streaming data which can be linked to by a Hypercat, each providing a different specific mechanism of subscribing to the resource. Where they are used, conventions should be established for linking to the resource and, optionally, mapping the received data to Hypercat best practice ontologies.

MQTT

MQTT is a lightweight pubsub protocol where clients connect to a central broker.
The MQTT protocol is designed for both publishing and subscribing and has a compact on-the-wire representation to allow use by lightweight devices. MQTT runs over TCP and provides security by using SSL/TLS for the TCP socket, with clients exchanging credentials

with the broker.

MQTT URLs may be embedded in Hypercat documents. Currently, no formal URL scheme is available. Research should be undertaken to establish best practices for linking to streamed data with MQTT.

Hypercat rels may be declared to inform clients of the kind of data to expect from an MQTT stream. For example, subscribing to the provided topic may provide payloads in a particular MIME-type. MQTT does not, as standard, provide the client with any hints as to the data type it will receive.

CoAP

CoAP is a lightweight RESTful protocol designed to take web-like interaction down to the smallest devices. Operating over UDP, typically in IPv6 environments, CoAP extends the concept of client-server Internet interaction to the microcontroller level. Unlike HTTP, CoAP has an inbuilt mechanism for observing resources, allowing clients to request server pushed updates in real-time.

CoAP has a fully documented URI scheme which may be used in Hypercat documents. https://tools.ietf.org/html/rfc7252#section-6.1

Although a CoAP server can notify clients with a content-type header, it may be desirable to define extra best-practice rels as hints to clients about the data they can expect from a CoAP endpoint.

RSS/Atom

RSS and Atom are XML based media-types intended to aid in syndication of content on the web. Although not intended for streaming, RSS and Atom provide a mechanism for clients to poll and determine if a new resource is available. Placing a link to an RSS or Atom document in a Hypercat should indicate to clients that they can monitor the RSS/Atom file for links to updated resources.

XMPP

XMPP is an XML-based messaging protocol designed for a range of client-server messaging applications. Being very flexible, XMPP has been adapted to many purposes including streaming data about IoT resources.

http://xmpp.org/extensions/xep-0323.html

Where XMPP resources are linked to from Hypercat, there exists a formal definition of a URI scheme. However, as with other protocols, it may be advantageous to provide extra metadata rels telling the client what data type to expect at the end of the XMPP link.

https://www.ietf.org/rfc/rfc4622.txt

Hypercat Signing (O)

This section describes a secure signing system for Hypercat catalogues, using JSON Web Signature (JWS), RFC7515, with a custom serialisation for Hypercat.
Hypercat is a JSON format for representing catalogues of resources and metadata about them. JWS provides integrity protection for content and is serialised to JSON, making it an good fit for Hypercat.

Signing Hypercats

Digital signatures for Hypercat documents can be provided at several levels.
For individual items contained in a Hypercat (`items` array), a signature may be added to the `item-metadata` for the item. In this case, the digest should cover all of the item (`href` and `item-metadata`).

For signing an entire Hypercat document, the signature may be added to the `catalogue-metadata`, with the digest covering both `catalogue-metadata` and all of `item-metadata`.

Verifying Hypercats

To verify a Hypercat, the signature is generated and compared. When generating the signature, the rels related to signatures should not be included in the digest.

JSON Web Signature

JWS allows for the digital signing (MACing) of content using JSON data structures. A signed document comprises three parts, a JOSE header, a JWS payload and a JWS signature.

JOSE header

The JOSE header declares the algorithm used to sign the payload, in JSON.
Example:

```
{ “header”: “alg”: “HS256” }
```

A per-item/per-catalogue, Hypercat rel is used for this purpose, e.g.

```
{ rel: “urn:X-hypercat:rels:jws:alg”, val: “RS256” }
```

JWS payload

The JWS payload to be signed is an ordered list of of octets, typically a string.
When generating and comparing the signed digest hash of a document, ordering is critical. Unfortunately, though JSON does specify a string serialisation (`JSON.stringify`), it does not guarantee ordering of properties within objects.

In order to produce a stable string representation of a Hypercat item, a deterministic `JSON.stringify` will be needed by producers and verifiers, e.g. https://github.com/substack/json-stable-stringify
Alternately, given the simplicity of Hypercat object structures, this may be a well-defined ordering (**TBD**).

JWS signature

The JWS signed hash is a base64 encoded string representing the hash digest signed with the chosen algorithm.

A per item, Hypercat `rel` is used for this purpose, e.g.

```
{ rel: "urn:X-hypercat:rels:jws:signature", val:
"YOWPexyGHKu4T_1M_vtlEnNlqmFOclqp4Hy6hVHfFT4" }
```

Keys

In order to verify a signature, a public key is required. This should be specified in PEM format in a val.

```
{ rel: "urn:X-hypercat:rels:jws:key", val: "-----BEGIN RSA PUBLIC
KEY-----…" }
```

The problem of verifying the authenticity and provenance of a key is considered out of scope for this specification.

Examples

An entire catalogue signature.

```
{
    "catalogue-metadata":[
```

```
        {
            "rel":"urn:X-hypercat:rels:isContentType",
            "val":"application/vnd.hypercat.catalogue+json"
        },
        {
            "rel":"urn:X-hypercat:rels:hasDescription:en",
            "val":"Signing example"
        },
        {
            rel: "urn:X-hypercat:rels:jws:key",
            val: "-----BEGIN RSA PUBLIC KEY-----..."
        },
        {
            rel: "urn:X-hypercat:rels:jws:signature",
            val: "YOWPexyGHKu4T_1M_vtlEnNlqmFOclqp4Hy6hVHfFT4"
        }
        {
            rel: "urn:X-hypercat:rels:jws:alg",
            val: "RS256"
        }
    ],
    "items":[
        ...
    ]
}
```

Per-item signature

```
{
    "catalogue-metadata":[
        {
            "rel":"urn:X-hypercat:rels:isContentType",
            "val":"application/vnd.hypercat.catalogue+json"
        },
        {
            "rel":"urn:X-hypercat:rels:hasDescription:en",
            "val":"Signing example"
        }
    ],
    "items":[
        "href":"https://example.org/sensor1",
        "item-metadata": [
            {
                rel: "urn:X-hypercat:rels:jws:key",
                val: "-----BEGIN RSA PUBLIC KEY-----..."
            },
            {
                rel: "urn:X-hypercat:rels:jws:signature",
```

```
                val:
"YOWPexyGHKu4T_1M_vtlEnNlqmFOclqp4Hy6hVHfFT4"
            }
            {
                rel: "urn:X-hypercat:rels:jws:alg",
                val: "RS256"
            }
            {
                "rel":"urn:X-hypercat:rels:isContentType",
                "val":"application/senml+json"
            },
            {
                "rel":"urn:X-hypercat:rels:hasDescription:en",
                "val":"Sensor number one"
            }
        ]
    ]
}
```

Hypercat Security Access Hints (O)

Hypercat Openness

In order to build a successful ecosystem, systems supporting Hypercat should provide open data with traversable links whenever possible. However, many systems will wish to provide resources or catalogues only to authenticated clients. Sometimes, these will need to be entirely obscured from view. However, where they are discoverable (but not accessible) without authentication, clients require information about how to authenticate for access to the resource.

Metadata

A single new `rel` is described for the per-item `item-metadata` object to inform clients how to authenticate for access to the linked resource.

rel	**val**
`urn:X-hypercat:rels:accessHint`	URI

Where the `val` is a URL, it should point at a machine or human readable description of the authentication method required. Some catalogues may choose to use well-defined URNs to signal a key, or protocol required for authentication.

Where multiple `accessHint` declarations are present, the client should assume that the resource can be accessed using multiple authentication systems.

Cross domain trust

In the case where a Hypercat provides `accessHint` for a resource, it is possible for the hint to be incorrect for the resource. For this reason, clients MUST only trust `accessHint` declarations for resources held on the same domain as the linking Hypercat.

Example

```
{
 ...
  "items": [
```

```
    {
      "href": "http://example.org/A",
      "item-metadata": [
        {
          "rel": "urn:X-hypercat:rels:accessHint",
          "val": "urn:X-hypercat:rels:oauth2.0"
        }
      ]
    }
  ]
}
```

Hypercat Security Credential Acquisition (O)

This section describes a Hypercat rel for signalling to a client where a key may be sought for access to a resource or catalogue.

Hypercat Key Provisioning

The Hypercat 1.1 specification said little about the topic of acquiring keys for access to catalogues and resources. This was left as a purely out-of-band activity. While this was fine for small numbers of catalogues and clients, it presents a barrier to scale.

Many methods of acquiring access credentials are possible. These could involve a simple request, an exchange of further credentials or even a financial contract. Using the described `rel`, any of these are possible, but none are specified, being considered out of scope for this section.

Metadata

A single new `rel` is described for the per-item `item-metadata` object and catalogue level `catalogue-metadata` object to inform clients where to acquire credentials for the resource or catalogue respectively.

rel	**val**
`urn:X-hypercat:rels:acquireCredential`	URL

The `val` must always be a URL, resolving to a self-describing web page or resource helping the client acquire credentials. In its simplest form, the URL may be a human readable web page explaining what to do. More advanced systems may wish to use machine readable documents to enable automatic negotiation of credentials.

Where multiple `acquireCredential` declarations are present, the client should assume that credentials can be acquired in multiple ways.

Hypercat Geographic Bounding Box Search (O)

This section describes a Hypercat search extension allowing for the search and retrieval of catalogue items with longitude and latitude within a 2D bounding box. The mechanism relies on position data being encoded with specific, well-known, metadata `rel`s.

Geographic Search

The current `simpleSearch` mechanism supported by Hypercat allows matching of items whose `rel`s are a match for some given search criteria. A geographic search allows for filtering of items who fall within a geographic region, defined by a bounding box. While a simple bounding box will be insufficient for some applications, clients of the API are free to further refine search results by other geographic criteria.

In order to search items using the described method, longitude and latitude must be encoded with the following metadata `rel`s for each Hypercat item:

`rel`	**Meaning**	**Example**
`http://www.w3.org/2003/01/geo/wgs84_pos#lat`	WGS84 Latitude	`51.508775`
`http://www.w3.org/2003/01/geo/wgs84_pos#long`	WGS84 Longitude	`-0.116993`

Catalogue Metadata

To signal to a client that a served Hypercat supports geographic bounding box search, the catalogue metadata section must include the `rel` `val` pair:

```
{ "rel": "urn:X-hypercat:rels:supportsSearch", "val":
"urn:X-hypercat:search:geobound" }
```

On seeing these parameters, a client may then provide the query parameters below to execute a geographic bounding box search.

Query Parameters

All query parameters below are mandatory for a search request.

Parameter Name	Meaning
`geobound-minlat`	Inclusive lower bound of latitude of bounding box
`geobound-maxlat`	Inclusive upper bound of latitude of bounding box
`geobound-minlong`	Inclusive lower bound of longitude of bounding box
`geobound-maxlong`	Inclusive upper bound of longitude of bounding box

As with other search mechanisms, the server returns a complete valid Hypercat document containing only items matching the search criteria.

Hypercat Lexicographic Range Search (O)

This section describes a Hypercat search extension allowing for the search and retrieval of catalogue items with metadata values between a given lexicographic range. The mechanism is designed to be general purpose, but is inspired by the need to return catalogue items within a given time or date range. An example time based `rel` is described for use with the search mechanism.

Lexicographic Search

The current `simpleSearch` mechanism supported by Hypercat allows matching of items whose `rel`s are an exact match for some given search criteria. A lexicographic range search allows searching for items which, when sorted lexicographically fall between a minimum and maximum.

The primary intended use for this search mechanism is to return items with `rel`s in a particular time range. Hypercat `val`s are always strings, so time based metadata is always represented as a string. When sorting and searching, therefore, it is critical that time be represented with a lexicographically orderable time string, such as ISO8601[2], with UTC and 'Z' suffix for the timezone.

Catalogue Metadata

To signal to a client that a served Hypercat supports lexicographic range search, the catalogue metadata section must include the `rel val` pair:

```
{ "rel": "urn:X-hypercat:rels:supportsSearch", "val":
"urn:X-hypercat:search:lexrange" }
```

On seeing these parameters, a client may then provide the query parameters below to execute a lexicographic range search.

Example Item Metadata

When lexicographic range search is used with dates and times, they must be represented using a format which can be meaningfully sorted as a string. For this purpose a new, example, metadata relation is described for use with catalogue items:

[2] http://en.wikipedia.org/wiki/ISO_8601#General_principles

rel	Meaning	Example val
`urn:X-hypercat:rels:lastUpdated`	ISO8601 UTC timedate string of last known update to resource	`2007-03-01T13:00:00Z`

Query Parameters

All query parameters below are mandatory for a search request.

Parameter Name	Meaning	Example
`lexrange-rel`	Specifies the `rel` to search on	`urn:X-hypercat:rels:lastUpdated`
`lexrange-min`	Lower bound of range to return (inclusive)	`2007-03-01T13:00:00Z`
`lexrange-max`	Upper bound of range to return (non-inclusive)	`2007-04-02T12:07:41Z`

As with other search mechanisms, the server returns a complete valid Hypercat document containing only items matching the search criteria.

Hypercat Robots Exclusion Search (O)

This section describes a Hypercat extension allowing a catalogue to reference a `robots.txt` file.

The Robots Exclusion standard

The robots exclusion standard (also known as `/robots.txt`) is a de-facto standard, and is not owned by any standards body.

It is used by websites to communicate with web crawlers and other web robots. The standard specifies the instruction format to be used to inform the robot about which areas of the website can and/or cannot be processed or scanned. It is specified here: http://www.w3.org/TR/html4/appendix/notes.html#h-B.4.1.1

Catalogue Metadata

To signal to a client that a served Hypercat has an associated `robots.txt` file the catalogue metadata section may include the `rel val` pair:

```
{ "rel": "urn:X-hypercat:rels:hasRobotstxt", "val":
"[BASE_URL]/robots.txt" }
```

If a robots exclusion file is provided it MUST be located at the BASE_URL of a catalogue and it MUST be named `robots.txt`.

Hypercat Multi-Search (O)

Multi-Search

Hypercat supports several different search extensions covering domains such as time, location and string matching. However, these search extensions only allow for simple interactions with a catalogue. Currently, the only way to, for example, filter a catalogue by “geographic bounding box” and “item creation date” is for a client to submit two independent queries then produce the intersection of the queries itself. This works, but is inefficient, requiring more data to be transmitted to the client than it needs. For large datasets, it is impractical.

Multi-search allows a client to combine single or multiple search mechanisms supported by a server to produce a Hypercat document containing only the items of interest.

Catalogue Metadata

To signal to a client that a served Hypercat supports multi-search, the catalogue metadata section must include the `rel val` pair:

```
{ "rel": "urn:X-hypercat:rels:supportsSearch", "val":
"urn:X-hypercat:search:multi" }
```

On seeing these parameters, a client may then execute a search by making a GET request with the URL query parameter "multi" with the value being a URL encoded, stringified multi-search JSON object.

For example, to request the result of the multi-search query {"query":"?rel=A"}, the following GET request would be made:

```
http://example.org/cat?multi=%7B%22query%22%3A%22%3Frel%3DA%22%7D
```

Where the multi-search object is too large to be passed as a URL query parameter, the POST mechanism may be used. All services supporting Hypercat multi-search MUST support the GET mechanism. Support for the POST mechanism is optional.

Multi-Search JSON Object

Multi-search accepts a single JSON object.
The simplest search contains a single `query` parameter, specifying the URL parameters to be executed.

Property Name	Property Meaning	Example
`query`	JSON string, holding URL query string as passed to underlying search mechanism	`"?rel=A"` (for use of simpleSearch extension)
`intersection`	JSON array of objects, containing `query`, `intersection` or `union`.	`"intersection": [ {"query": "?rel=A"}, {"query": "?rel=B"} ]`
`union`	JSON array of objects containing `query`, `intersection` or `union`.	`"union": [ {"query": "?rel=C"}, { "query": "?rel=D"} ]`

As shown below, searches may be nested to allow complex mixing of union and intersection.

As with other search mechanisms, the server returns a complete valid Hypercat document containing only items matching the search criteria.

Examples

Search catalogue using a single `simpleSearch` query:

```
{
    "query": "?rel=A"
}
```

Search catalogue, returning the intersection of two simpleSearch queries:

```
{
    "intersection": [
        { "query": "?rel=A" },
        { "query": "?rel=B" }
    ]
}
```

Search catalogue, returning the union of two `simpleSearch` queries:

```
{
    "union": [
        { "query": "?rel=C" },
        { "query": "?rel=D" }
    ]
}
```

Search catalogue, returning the intersection of one `simpleSearch` query with the union of two other queries:

```
{
    "intersection": [
        { "query": "?rel=A" },
        "union": [
            { "query": "?rel=C" },
            { "query": "?rel=D" }
    ]
}
```

Hypercat Prefix Match Search (O)

This section describes a Hypercat search extension allowing for the search and retrieval of catalogue items with metadata values or `href` link matching a specified prefix. The mechanism is designed to be general purpose working with any `rel`.

Prefix Match Search

The current `simpleSearch` mechanism supported by Hypercat allows matching of items where specified `rel`s have `val`s which are an exact match for some given search criteria. A prefix match search allows searching for items where the `val` specified in the query match the first N characters of the `val` in the `item` (where N is the number of characters in the query `val`).

Catalogue Metadata

To signal to a client that a served Hypercat supports prefix match search, the catalogue metadata section must include the `rel` `val` pair:

```
{ "rel": "urn:X-hypercat:rels:supportsSearch", "val":
"urn:X-hypercat:search:prefix" }
```

On seeing these parameters, a client may then provide the query parameters below to execute a prefix match search.

Example Item Metadata

As with simple search, any `rel` can be used.

Query Parameters

A prefix search is performed by providing a query string (http://tools.ietf.org/html/rfc1738) to a catalogue. If multiple search parameters are supplied, the server MUST return the intersection of matched items, combining the parameters with boolean AND.

A simple search searches only a single catalogue resource. It does not include other linked or nested catalogues.

All query parameters MUST be URL encoded. All query parameters are optional.

Parameter	Meaning	Allowed Value
`prefix-rel`	Any metadata relation	URI as a JSON string
`prefix-val`	Any metadata value	URI as a JSON string
`prefix-href`	A resource URI	URI as a JSON string

String matching examples

The following examples illustrate the operation of the string matcher.

Needle (in query)	Haystack (in catalogue)	Matches
`foo`	`foobarbaz`	Yes
`bar`	`foobarbaz`	No
`foobar`	`foobarbaz`	Yes
`foobarbaz`	`foobarbaz`	Yes
`xfoo`	`foobarbaz`	No

Hypercat Linked Data rel (O)

This section describes a new Hypercat rel for specifying that the item at hand is an instance of an RDF class..

Hypercat & Linked Data

A feature which would help to make best use of the data found in Hypercat catalogues or linked resources would be the ability to link Hypercat resources to external Linked Data data sources, thus providing connection to the Linked Data cloud and potentially facilitating semantic queries.

Metadata

A single new `rel` is described

rel	Meaning	val
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	The resource is an instance of an RDF class	URI

The val should point at an RDFS class as specified at http://www.w3.org/TR/rdf-schema/

Hypercat License (O)

This section describes a new Hypercat rel for marking up the license conditions under which a Hypercat catalogue or linked resource may be used.

Hypercat Licenses

In order to make use of the data found in Hypercat catalogues or linked resources, clients must be able to determine the license under which the data is released.
The chosen license may allow or disallow dissemination, sale or other forms of use by the client, and may define obligations on the catalogue publisher.

Metadata

In order to allow maximum flexibility, a single new `rel` is described which acts as a signal to clients that data is provided under a set of license conditions. While some possible `val`s will be described, we expect users to select from the wide range of licenses available across jurisdictions and data types.

rel	**val**
`urn:X-hypercat:rels:hasLicense`	URI

Where the `val` is a URL, it should point at a machine or human readable version of a license, preferably a permalink, e.g. https://creativecommons.org/licenses/by/4.0/ or https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html
The `hasLicense` metadata may be used in both `catalogue-metadata` and `item-metadata` sections to mark the license conditions of the containing Hypercat and of a linked resource.

Where multiple `hasLicense` declarations are present, the client should assume that the resource is available under multiple licenses.

Cross domain trust

In the case where a Hypercat provides `hasLicense` for a resource, it is possible for the license to be incorrect for the resource. For example, a rogue catalogue could mark that all `.gov.uk` datasets are for limited use, when they are not. For this reason, clients MUST only trust `hasLicense` declarations for resources held on the same domain as the linking Hypercat.

Example

```
{
  "catalogue-metadata": [
    {
      "rel": "urn:X-hypercat:rels:isContentType",
      "val": "application/vnd.hypercat.catalogue+json"
    },
    {
      "rel": "urn:X-hypercat:rels:hasDescription:en",
      "val": ""
    },
    {
      "rel": "urn:X-hypercat:rels:hasLicense",
      "val": "http://hypercat.io/catlicense4.3"
    }
  ],
  "items": [
    {
      "href": "http://example.org/A",
      "item-metadata": [
        {
          "rel": "urn:X-hypercat:rels:hasLicense",
          "val": "http://example.org/publicfreedata4.3"
        }
      ]
    }
  ]
}
```