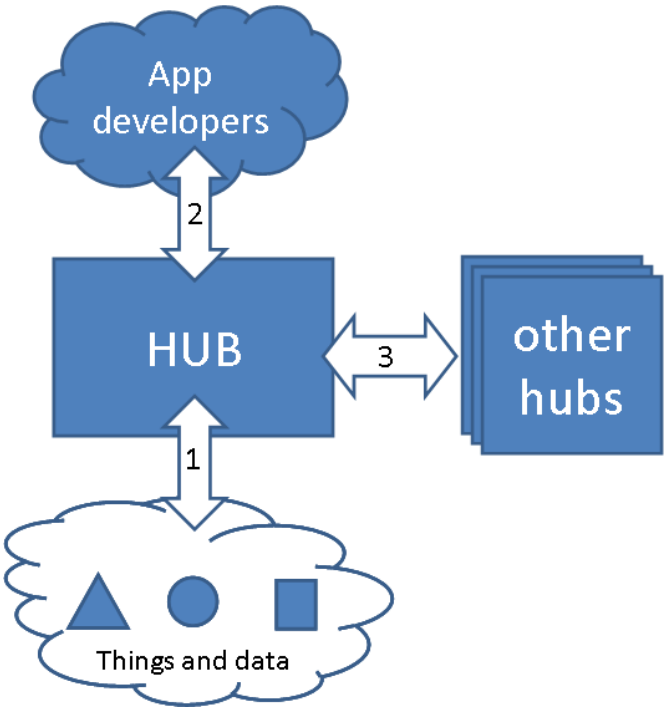
*This document is the "specification" part of a wider "Interoperability Action Plan" document.*

## 3.0 High Level Architecture

Full interoperability for all of the "things" within the Internet of Things is a massive undertaking, beyond the scope of this demonstrator. For most developers, what is more important is access to the data their individual implementations which produce and an understanding of what that data represents. The goal of this project is to provide and demonstrate that interoperability at this level is possible.

A simplified view of the IoT is given below. Each consortium member will be implementing one or more hubs, which communicate with their underlying devices to feed data, and possibly expose actuators and control points to the hub (1). Each hub can then interact with applications which can discover these resources and access them (2). Hubs may also communicate with other hubs (3), although this will be through the same interface. The agreed focus of this interoperability demonstrator is Interface 2:




The following section defines the specification for Interface 2, which will be provided in at least one hub implemented by each consortium. Each consortium will also develop at least one application to access data from another hub.

Because of the limited timescale for this project, the details of security and commercial access are considered out of scope, but may be developed independently by consortium members.

# 4.0 Core Interoperability Specification

## 4.1 Identity

The identity of Apps and Hubs will be URI's.

## 4.2 Catalogues

Applications access data on another hub by means of a catalogue which shall be implemented on every hub which allows one or more applications to access it. A catalogue is a specific type of resource representing an unordered collection of resource items. Each item in a catalogue refers to a single resource by its URI, which may itself be a further catalogue.

All compliant hub servers MUST provide a catalogue with zero or more items at `{BASE URL}/cat`. A compliant hub server MUST return a response to a valid catalogue request, even if the catalogue has zero items available for that request.

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

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.

## 4.3 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 |
| "item-metadata" | An array of metadata objects describing the catalogue object | JSON array of metadata objects |

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

### 4.3.1 Item Object

An "items" object is a JSON object, which MUST contain all of the following properties:

| Property Name | Meaning | Property Value |
|---|---|---|
| "href" | Identifier for the resource item | URI as a JSON string |
| "i-object-metadata" | An array of metadata objects describing the resource item | JSON array of metadata objects |

The `metadata` array MUST contain a metadata object for each of the mandatory metadata object relationships.

The `metadata` array MAY contain multiple metadata objects with the same `rel` (and `val`) properties (`metadata` is a bag/multiset of features).

### 4.3.2 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 appliesJSON 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. | |

The metadata object `{"rel":"urn:X-tsbiot:rels:isColour", "val":"blue"}` MUST be interpreted as saying the parent object is of colour blue.

### 4.3.3 Mandatory Metadata Relationships

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

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

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

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

### 4.3.4 Optional Metadata Relationships

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

| **`rel` value** | **Meaning** | **`val` value** |
|---|---|---|
| `"urn:X-tsbiot:rels:isContentType"` | Data provided by resource SHOULD be of given type | RFC2046 MIME type as JSON string (eg. `"text/csv"`) |

| | | |
|---|---|---|
| "urn:X-tsbiot:rels:hasHomepage" | A reference to a human readable web page concerning the resource | URL as a JSON string |
| "urn:X-tsbiot: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-tsbiot:rels:supportsSearch" | This catalogue 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 (see section 4.4 ) |

## 4.4 Catalogue Search

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-tsbiot:rels:supportsSearch" metadata relation.

In future, other search mechanisms may be defined. At present, only the simple search mechanism is specified.

### 4.4.1 Simple Search Mechanism

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

```
{
    "item-metadata":[
        {
            "rel":"urn:X-tsbiot:rels:isContentType",
            "val":"application/vnd.tsbiot.catalogue+json"
```

```
        },
        {
            "rel":"urn:X-tsbiot:rels:hasDescription:en",
            "val":""
        },
        {
            "rel":"urn:X-tsbiot:rels:supportsSearch",
            "val":"urn:X-tsbiot:search:simple"
        }
    ],
    "items":[
        {
            "href":"http://A",
            "i-object-metadata":[
                {
                    "rel":"urn:X-tsbiot:rels:1",
                    "val":"1"
                },
                {
                    "rel":"urn:X-tsbiot:rels:2",
                    "val":"2"
                },
                {
                    "rel":"urn:X-tsbiot:rels:3",
                    "val":""
                }
            ]
        }
    ]
}
```

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

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

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

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

## 4.5 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.

### 4.5.1 Read Catalogue

To read an entire catalogue, a client MAY `GET` the catalogue URL.

On success, the server MUST return a JSON catalogue object, as defined in this document.

On success, the server MUST return an HTTP 200 status code.

Reading part of a catalogue is accomplished using Search (see section 4.4 ).

### 4.5.2 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.

### 4.5.3 Read Catalogue Item

To read an individual item from a catalogue, a client MAY use the simple search mechanism defined in section 4.4 .

### 4.5.4 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 use a query parameter of `href` with a value corresponding to a resource URI held in the catalogue.

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.

### 4.5.5 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.

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.

### 4.5.6 Security

All HTTP(S) requests may be authenticated with a key, which is a URI.

Keys may be presented in two ways (HTTP headers):

x-api-key: KEY
Authorization: base64(KEY:"")

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

## 4.6 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” |

## 4.7 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.

A new search method MAY be added, discovered and identified by defining a new value for the relation `urn:X-tsbiot:rels:supportsSearch`

A new language or style for human readable descriptions MAY be added, discovered and identified by defining a new `urn:X-tsbiot:rels:hasDescription` variant.

A complete replacement catalogue object format MAY be implemented by declaring a new MIME type. 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 document, 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.

### 4.8 Best Practice

To help improve interoperability of data within hubs, members will work to define Best Practice for metadata and resource formats, as well as any items identified as out of scope.

## 7.0 IP

All of the technical content within this document, primarily covering the catalogue and item specifications, is deemed to be open. No member of the demonstrator will assert IP against the use by any party of any such content described within this document.

## 8.0 Dissemination

In the spirit of interoperability, Consortia will work with the TSB to disseminate this specification and provide access to open hubs to encourage further demonstration of the interoperability that is a key factor of this project. It is anticipated that a high level dissemination plan covering the overall interoperability will be produced to cover this.