Article
Benjamin De Boe · Jul 4, 2016 8m read

Introduction to the iKnow REST APIs

After a five-part series on sample iKnow applications (parts 1, 2, 3, 4, 5), let's turn to a new feature coming up in 2017.1: the **iKnow REST APIs**, allowing you to develop rich web and mobile applications. Where iKnow's core COS APIs already had 1:1 projections in SQL and SOAP, we're now making them available through a RESTful service as well, in which we're trying to offer more functionality and richer results with fewer buttons and less method calls. This article will take you through the API in detail, explaining the basic principles we used when defining them and exploring the most important ones to get started.

In this article, we'll assume you're at least somewhat familiar with REST, JSON and iKnow, for which you'll also find lots of relevant content on the Developer Community (tags REST, JSON and iKnow).

Guiding Principles

Rich is the default

Where the classic iKnow APIs in COS offered low-level functions to retrieve specific information, we wanted the REST APIs to be rich by default and answer an entire use case rather than a single question. For example, if you wanted a highlighted summary of a text in COS, you needed to first request the summary, and then the highlighted version of every sentence returned by the first query. Thanks to the flexible hierarchies we can serve back with JSON, much of this was easy to implement with REST APIs serving JSON. For example, the endpoint for getting source details will give you metadata and text in one go, optionally summarizing and/or highlighting it.

Also, in order to improve the development experience, we're including all those rich results in the returned JSON by default. In other words, you have to opt-out explicitly if you only want a particular piece of the information returned. The idea is that it's easier to add those flags once you're productizing your application than it is to look up and test all available flags while you are developing it.

Simplicity & consistency

Keeping things simple should be a goal for everyone at all times, and it is no different here. While there are often a decent amount of flags and parameters to tune the result you're getting back and thus you'd require a POST request, we'll also service a GET request as if it were a POST one with all default parameters. This means you can quickly check most queries from your browser, even without fancy REST plugins (although we liked postman a lot ourselves). In most cases where there is a string parameter to be supplied through the URL (GET request), you can also supply it as a property of the POST request object, overwriting anything that might be in the URL.

We also put great effort in consistency. Parameters as well as resulting JSON property names are 100% consistent across endpoints, the endpoint URLs are (we believe ;-)) nicely grouped and the overall service behaves the same way as the DeepSee REST API and others already in the kit when it comes to the basic path hierarchy per namespace. This means the default service endpoint URL starts with "http://localhost:57772/api/iKnow/v1/*namespace*/...", but you can create your own dedicated ones by subclassing %iKnow.REST.v1 to lock it to a particular namespace.

Debugging

If the resulting JSON doesn't seem to match what you expected, you can simply add a debug flag to your POST request and you'll get appropriate debug information. This includes a copy of the "final" request the query used,

after filling in missing parameter values with the defaults.

Getting at Sources

Simple queries

An obvious start is to check the available domains for your namespace, using the following URL and trivial response.

GET http://localhost:57772/api/iKnow/v1/samples/domains


```
{
  "domains": [
    {
      "id": 8,
      "name": "Aviation Events demo",
      "version": 5,
      "definitionClass": "Aviation.ReportDomain",
      "sourceCount": 1200
    },
    {
      "id": 12,
      "name": "Japanese Tests",
      "version": 5,
      "definitionClass": "Test.DomainDefJ",
      "sourceCount": 0
    }
  ]
}
```


Most other queries will target a specific domain, for example this one to fetch sources from a domain (using domain ID 8):

GET http://localhost:57772/api/iKnow/v1/samples/domain/8/sources


```
{
  "sources": [
    {
      "id": 1200,
      "extId": ":SQL:2010:20101209X94036",
      "metadata": {
        "DateIndexed": "06/30/2016 10:31:25",
        "EventDate": "12/03/2010 00:00:00",
        "Year": 2010,
        "Country": "United States",
        "State": "Oklahoma",
        "LightConditions": "Night",
        "HighestInjury": "Serious"
      },
      "snippet": "He stated that prior to the reduction in power, the fuel selector v
alves were selected to the main tanks, the fuel boost pumps were off, the mixture was
 set at rich, and all four fuel tanks indicated \"adequate fuel.\" ... The team that
recovered the wreckage reported that the right main fuel tank was full of fuel, and t
he left main fuel tank was burnt with signatures consistent with a fuel fed fire."
    },
    { ... }
  ]
```

```
}
```

This query will by default return the metadata and a snippet of each source. It will only output the first 200 rows, as the default for page and page size are 1 and 200, respectively. If you want to override these defaults, you could submit the following using a POST request to the same URL:

```
{
  "includeMetadata" : 0,      /* to disable retrieving metadata for each row */
  "includeSnippets" : 5,      /* to retrieve 5-sentence snippets for each row */
  "page" : 1, "pageSize": 10, /* to retrieve the first 10 records */
  "debug" : 1                 /* to display debug info in the result */
}
```

That's all there is to it to get started. Two additional request parameters that, like page and pageSize, are supported by many endpoints, are for filtering and highlighting. These are a bit more complex and deserve additional attention.

Filtering

Filtering means restricting the scope of your query to a certain subset of your domain, based on filter criteria. In COS, this was achieved through passing in instances of %iKnow.Filters.Filter subclasses. The following example filters by year:

```
{ ...,
  "filter" : { "field": "Year", "operator": ">", "value": 2010 }
}
```

And this one uses grouped filters to also include a few records identified by source ID:

```
{ ...,
  "filter" : { "operator": "OR", "filters": [
                    { "field": "Year", "operator": ">", "value": 2010 },
                    { "ids": [123, 124, 125] }
                ] }
}
```

Highlighting

Almost every piece of text that consists of sentences from the original record (sentence results, summaries, snippets, ...) can be highlighted by specifying a "highlight" property in your POSTed request object. The options correspond to what you may already be familiar with from the %iKnow.Queries.SentenceAPI:GetHighlighted() method.

For example, we can use this to highlight the summarized contents of a particular document, putting square brackets around all concepts, curly ones around all relationships and plain parentheses around any path-relevants:

POST http://localhost:57772/api/iKnow/v1/samples/domain/8/sources/123/details

```
{ "summarize": 10,  /* for retrieving a 10-sentence summary */
  "highlight": [
    { "style": "[]", "role": "concept" },
    { "style": "{}", "role": "relation" },
    { "style": "()", "role": "pathRelevant" }
```

```
  ]
?}
```

And the following will add HTML tags to bolden occurrences a specific set of entities, put all entities affected by negation in italics and underline the negation markers, add a grey background to all dictionary matches and put those of the dictionaries with IDs 12 and 67 in blue:

```
{ ...,
  "highlight": [
    { "style": "<b>", "entities": [ "airplane", "helicopter" ] },
    { "style": "<i>", "attribute": "negation" },
    { "style": "<u>", "attributeWords": "negation" },
    { "style": "<font style='background-color: grey;'>", "anyMatch": 1 },
    { "style": "<font style='color: blue;'>", "dictionaries": [ 12, 67 ], "matchType"
: "full" }
  ]
}
```

Getting at the Rest

Getting at Sources

Here are a couple more queries to get to a list of sources, all very similar to the general one we saw earlier:

GET http://localhost:57772/api/iKnow/v1/samples/domain/8/sources/by/entity/h...

or, if you prefer passing in the search string through a POST request object property (and avoid confusion with URL encoding issues):

POST http://localhost:57772/api/iKnow/v1/samples/domain/8/sources/by/entity[/x]

```
{ "entity": "helicopter" }
```

or, when using multiple entities:

```
{ "entity": [ "helicopter", "tail" ], "setOperation" : "intersect" }
```

Obviously there are similar methods to retrieve sources based on a CRC:

GET http://localhost:57772/api/iKnow/v1/samples/domain/8/sources/by/crc/line...

or

POST http://localhost:57772/api/iKnow/v1/samples/domain/8/sources/by/crc[/x]

```
{ "crc": {"master": "line", "relation": "of", "slave": "trees" }, ... }
```

Getting at Entities

The typical starting point for retrieving entities would be the equivalent of the GetTop() query in the EntityAPI:

GET http://localhost:57772/api/iKnow/v1/samples/domain/8/entities

Which returns a rather basic data structure:

```
{
  "entities": [
    {
      "id": 11149,
      "value": "airplane",
      "frequency": 6343,
      "spread": 983
    },
    {
      "id": 54528,
      "value": "pilot",
      "frequency": 6197,
      "spread": 1083
    }, ...
}
```

The results of this query can be fine-tuned by a number of optional parameters, on top of the page, pageSize and filter parameters we saw earlier. A fully-configured request could look like this:

```
{
  "sortBy": "dominance",
  "role": "concept",
  "page": 1, "pageSize": 20,
  "blacklists": [ 1, 12 ],
  "includeMetrics": ["frequency", "spread", "dominance"],
  "debug" : 1
}
```

A couple of other endpoints that will definitely look familiar to those who worked with the EntityAPI before:

GET http://localhost:57772/api/iKnow/v1/samples/domain/8/entities/***airplane***/**details**

GET http://localhost:57772/api/iKnow/v1/samples/domain/8/entities/***airplane***/**similar**

GET http://localhost:57772/api/iKnow/v1/samples/domain/8/entities/***airplane***/**related**

The latter will return "related" entities based on proximity by default, unless you set the sortBy parameter to "frequency" or "spread"

Getting further

This is just a first overview of the general outline of this REST API. There are many more methods, including those to deal with dictionaries, blacklists and other elements. More developer-oriented information is available in the class reference for %iKnow.REST.v1, with full reference documentation available in Swagger format. Read this article to learn more about how that reference documentation can be leveraged to automatically generate a GUI to explore and test the APIs.

#Best Practices #Caché #InterSystems Natural Language Processing (NLP, iKnow)

Source URL:https://community.intersystems.com/post/introduction-iknow-rest-apis