Swiftriver – Content Duplication Service

Overview

As the name suggests, the Swiftriver Content Duplication Service (SiCDS) is responsible for ensuring – as much as is possible – that the Swiftriver Core and UI don't get cluttered up with duplicate content. The classic – well modern really – example is multiple re-tweets from the same original tweet!

Domain Terminology

In the Swiftriver domain, content is king! For this reason, a content item is the main aggregate data class that is used throughout the entire Swiftriver stack. A content item is a direct representation of a piece of information received by Swiftriver (it could be an email, a tweet, a news article etc.).

The originator of a content item is known as the source, this could be a twitter Id, and email address or a mobile phone number (in the case of SMS content).

The way in which the content was received by Swiftriver is known as the channel. The channel is not the same as the source so, for example; if 1000 content items were received from an SMS short-code then each of those content items would (probably) have their own source – the originating number – but they have all arrived on the same channel (in this case identified as the SMS short code)

So, in Swiftriver terms, this gives us:

Content from a **Source** arrives over a **Channel**.

SiCDS and Swiftriver

The architecture of Swiftriver has been designed with SOA (Service Orientated Architecture) in mind and the core of Swiftriver is simply responsible for routing incoming content to a predefined set of services, each of which is responsible for adding some kind of value to a set of content items.

In the case of the SiCDS; we are not so much looking for added value as we are for an indication of if this content has been seen before or not.

SiCDS Interface

The SOA design of Swiftriver means that the services used by the Swiftriver Core can be written in any language and either hosted locally – alongside the install of the Core – or as cloud based service.

With this in mind, a simple JSON based interface describes the interaction between the Swiftriver Core and the SiCDS.

Data to the SiCDS

Data is **POSTED** to the SiCDS in JSON format. The JSON will arrive and the form data collection of the POST variable with the name 'data'. An example follows:

```
POST["data"] =
  [
    {"key":"ds8a7d8df7s9df7sd8f7s9d8f",
     "content":[
       {"id":"df7sdfsd0f8sdsdd8fsdfdsfs980",
        "difs":[
          {"type":"email_address","value":"one@someaddress.com"},
          {"type":"email subject","value":"the subject of email one"},
          {"type":"email abstract","value":"The first X chars from email one"}
       ]},
       {"id":"duiwqoue87d8s7dwu89d8duedoe8",
        "difs":[
          {"type":"email_address","value":"two@someaddress.com"},
          {"type":"email_subject","value":"the subject of email two"},
          {"type":"email abstract",value":"The first X chars from email two"}
       ]}
    }
  ]
```

Or ...

```
POST["data"] =
  [
    {"key":"ds8a7d8df7s9df7sd8f7s9d8f",
     "content":[
       {"id":"ysudy87d7a87sd6wdy78ed68ead",
        "difs":[
          {"type":"unique tweet id","value":"the 1st twitter id of the source"},
          {"type":"tweet_text", "value":"this would be the content of the tweet"}
       ]},
       {"id":"usidcyas7d6sa76fd5f6sd5f7s7",
        "difs":[
          {"type":"unique tweet id","value":"the 2nd twitter id of the source"},
          {"type":"tweet_text", "value":"this would be the content of the tweet"}
       ]}
    }
  ]
```

So, I hope it's clear what the SiCSD will expect to receive but let's have a look at the key parts of the JSON in detail:

"key"

The Key is a unique identifier that is allocated to each Swiftriver instance, this is used to ensure that all requests are from a correctly identified Swiftriver instance. When the SiCDS is deployed, it will be configured to only answer requests from one (or multiple) keys.

"content"

Once the SiCSD has confirmed that the key provided in the JSON relates to a Swiftriver instance that it has been programmed to communicate with, it should begin to inspect the array of content items. Each content item is made up of an ID and a collection of Duplication Identification Fields (difs).

"id"

Each piece of content sent to the SiCSD will arrive with a unique Id. It's important for the SiCDS to extract this ID as it'll be used in the return message to the Swiftriver Core.

"difs"

Content "Duplication Identification Fields" or difs are a concept invented to help tackle the issue of duplicate content in Swiftriver. In essence, each content item is 'tagged' with a set of difs that are judged to be an accurate unique identifier for this content item. In essence, if all the listed difs for a content item match the difs for already recorded content item then you have a duplicate.

Data from the SiCDS

The return type of the **POST** to the SiCDS should be set with a header value of "Content-type: application/json". The calling service will be expecting a string of JSON similar to the following example.

```
[
  {"key":"ds8a7d8df7s9df7sd8f7s9d8f",
   "results":[
     {"id":"ysudy87d7a87sd6wdy78ed68ead","result":"unique"},
     {"id":"usidcyas7d6sa76fd5f6sd5f7s7","result":"duplicate"},
     {"id":"6gdfg9fg0dfg6f7gd8g6f7dgd5g","result":"duplicate"},
     {"id":"d9s08fd98fg87fd8gd9g7fd87gd","result":"unique"},
  ]}
]
```

As you can see, the return JSON should contain each and every content id passed to the SiCDS along with a 'result' chosen from either of the following string values: 'unique' or 'duplicate'.

One possible solution – in pseudocode

The implementation of the SiCDS is entirely in the hands of the developer, however for guidance – and if you are stuck for ideas, you can consider the following example of *how* the SiCDS could be implemented.

```
GET the json from the post variable

IF the id is a valid id
  CONTINUE
ELSE
  RETURN and error message in json

FOREACH content item in the json
  FOREACH dif in the content items difs collection
    CHECK if an entry exists in the data store for that dif type and value
  IF all the difs exists already
    MARK this content item as a duplicate
  ELSE
    MARK as unique

RETURN the collection of ids with their results
```