Using the Digg REST API
Programmatically access Digg's best features

Brian M. Carey

December 23, 2009

Digg is a social news Web site where users can submit news stories and links and also rank their popularity. Like most entries in the social networking genre, this Web site also provides an API that allows developers to programatically access the site's features. This article will show you how to use that API.

What is Digg?

Digg (http://www.digg.com) is a Web site that enables its users to discover, share, and recommend content from anywhere on the Internet. Usually this content is an article from a reputable news source or a blog entry. Users can then vote up (called digging) or vote down (called burying) someone else's submission. The stories that are the most dugg appear on the Web site's front page.

Frequently used acronyms

- API: Application programming interface
- HTTP: Hypertext Transfer Protocol
- REST: Representational State Transfer
- URL: Uniform Resource Locator
- XML: Extensible Markup Language

Digg operates as an editor-free article aggregation site. By editor-free, I mean that the top-level content of Digg is solely determined by popular vote among its users. Think of it as social media democracy. One notable exception to this rule occurred in 2007 when Digg pulled an article submission which contained the encryption key for certain HD DVD and Blu-ray discs.

Like many entries in the social media scene, Digg enables its users to discuss submitted articles. Users can comment on a submission, then other users can comment on the comment. And guess what? The whole comment thread can then be dugg just like an article anywhere else on the Internet! Welcome to the twenty-first century.

Many of the aforementioned reputable Web sites and blogs recognize the importance of Digg by including its icon on Web pages where articles and stories are posted. Usually, the Digg icon will
be labeled "Digg This!". It will frequently appear next to other icons representing similar sites (such as Reddit, StumbleUpon, and so on). When the icon is clicked, the user is submitting the article to Digg. Then, as mentioned previously, other users can digg or bury it.

Recognizing the importance of social media, Digg recently implemented a new feature known as Facebook Connect. As the name implies, Facebook Connect enables users of Digg and Facebook to link their accounts. This enables a Facebook Connect user to share dugg articles on their Facebook page. Facebook users can also create a Digg account without the usual registration process using Facebook Connect.

What is REST?

REST is an acronym for Representational State Transfer. The full explanation of everything entailed in a proper REST definition is outside of the scope of this article; however, it is available elsewhere on IBM developerWorks (see Resources). For the subject covered here, it is sufficient to state that REST enables developers to access information and resources using a simple HTTP invocation.

As an example, imagine that FishinHole.com operates a Web site that markets fishing tackle to its customers. Users who access the site can see a variety of lures, reels, rods, and so forth. They do this the old-fashioned way: by clicking links. In this way, FishinHole.com makes its services available to human beings.

But FishinHole.com also makes its services available to other Web applications by exposing its catalog of fishing tackle with REST. So, instead of clicking around, the Web application obtains information about lures, reels, rods, and so forth with a simple HTTP invocation. For example, http://www.fishinhole.com/catalog/rest/lures?format=xml returns a list of all lures offered by the company in XML format. As another example, http://www.fishinhole.com/catalog/rest/item?id=343221 returns information about item #343221 in the default format.

Think of REST this way: To obtain domain-specific data, you simply point a URL to a specific location. For the purposes of this article, that's really all it is. You can also think of it as a simplified Web service, but if you say that too loudly around the wrong people, you might find yourself in the middle of a debate.

Note: I should point out that FishinHole.com doesn't actually exist. So, if you paste any of those URLs into your browser, you might wonder why you got an error. I provide these examples simply to show the format of a typical REST invocation.

What is the Digg API?

The Digg API is provided by Digg as a service to enable Web developers to programmatically access Digg information. Think of it as software extracting key data from Digg so that the data can be displayed, in any format, on a completely separate Web page.

The Digg API accepts REST requests. See the previous section for more information about that. The response format is up to the individual developer, but is limited to XML, JavaScript Serialized
Object Notation (JSON), JavaScript, and serialized PHP. All responses are in 8-bit UCS/Unicode Transformation Format (UTF-8).

Once the response is received, it is up to the developer to parse and use that response as needed for a separate Web application.

You can use the Digg API for commercial purposes, as long as you follow the license agreements.

**Using the Digg REST API**

Like most APIs within the social media realm, Digg is easy to use. If you have even limited experience with the aforementioned technologies, you should have no trouble implementing a simple Digg API invocation.

**A simple example**

Consider the following simple example:

http://services.digg.com/1.0/endpoint?method=story.getPopular

You can simply plug that into the URL bar of your favorite browser and press Enter. After a few seconds, you should see XML output on your screen that looks like Listing 1. (Note: In Listing 1, the value of the src attribute on the thumbnail element splits to multiple lines due to formatting constraints. It should appear as a single string of characters: src="http://digg.com/movies/The_real_incredibly_mundane_reason_Darth_Vader_wears_a_mask/t.jpg".

**Listing 1. A simple example: Output**

```
  <href>http://digg.com/movies/The_real_incredibly_mundane_reason_Darth_Vader_wears_a_mask</href>
  <promote_date>1259150128</promote_date>
  <status>popular</status>
  <media>news</media>
  <description>Darth Vader is one of the coolest -- and coolest-looking -- characters in science fiction, yet much of his appearance was the result of nothing more than practicality. At least that's how Ralph McQuarrie, the legendary conceptual designer of George Lucas' original Star Wars films, remembers the evolution of the Sith's darkest Dark Lord</description>
  <title>The real, incredibly mundane reason Darth Vader wears a mask</title>
  <user name="mklopez" registered="1121877948" profileviews="128127">
    <icon>http://digg.com/users/mklopez/l.png</icon>
  </user>
  <topic name="Movies" short_name="movies"/>
  <container name="Entertainment" short_name="entertainment"/>
  <thumbnail originalwidth="550" originalheight="324" contentType="image/jpeg" src="http://digg.com/movies/The_real_incredibly_mundane_reason_Darth_Vader_wears_a_mask/t.jpg" width="80" height="80"/>
  <shorturl short_url="http://digg.com/d31Ay85" view_count="210"/>
</story>
```

Your response will contain a lot more story elements than what you see in Listing 1. That listing is abbreviated for space purposes.

First of all, note that the response is in XML format. This is the default unless you specify a return format.
Most of the attributes and child elements are self-explanatory. However, I'll cover them briefly here.

The `story` element itself represents an article, somewhere on the Web, which has been dugg. All of the relevant information about this article is expressed either as an attribute or child element.

The `link` attribute, as the name implies, is the actual URL of the article itself. This represents a permalink to the article. In other words, that URL should work forever as a pointer to the article being expressed by the story.

The `submit_date` attribute represents the date that the article was submitted to Digg. This is expressed in UNIX® time, or the number of seconds that have elapsed since January 1, 1970.

The `diggs` attribute represents the number of times that the article has been dugg. This is a net number, meaning that it actually represents the number of overall diggs minus the number of overall buries.

The `id` attribute is the Digg identification number of the article.

The `comments` attribute represents the number of comments (on the Digg site itself) associated with that article.

The `href` attribute represents the permalink to the article reference on Digg. This is distinct from `link` in that `link` points to the article on the site where it is actually hosted, whereas `href` points to the Digg representation of that article.

The `promote_date` attribute is the date (again, in UNIX time) that the article was promoted in Digg. In other words, this is the time that the article last moved up in ranking.

The `status` attribute represents the current status of the article. In this case, the value is `popular` because this article, as of this writing, is one of the most popular articles on Digg. Articles with a status of `popular` appear on the Digg home page.

The `media` attribute represents the type of article. In this case, the article is a news article.

The `description` element provides a brief synopsis of the article. This is so that users of Digg can look at a summary of the article and decide whether or not they would like to view the entire article.

The `title` element provides the title of the article.

The `user` element provides information about the user who submitted the article to Digg. Note that the attributes provide key information about the user, such as the date that the user registered with Digg, how many times the user's profile has been viewed, and the URL of the user's icon.

The `topic` element provides information about which topic is associated with this article. In this case, the topic is `Movies`. Note that, although it is not represented that way in XML format, topic is actually a child of the container category.
The `container` element, as just mentioned, is the logical parent of the `topic` element. In this case, 
the container is `Entertainment`.

The `thumbnail` element provides key information about the thumbnail image which displays next to 
the Digg entry. If you go to http://www.digg.com right now, you will almost certainly see more than 
a few Digg entries with thumbnail images next to them. This element provides information about 
those images, not the least of which is the URL for the image itself.

The `shorturl` element provides a shortened URL for this article. This is convenient if you want to 
submit the URL to Twitter, which supports a very limited number of characters per tweet.

**A simple change**

Let's say, for example, that you liked that story so much you want to see some other stories, 
but only those in the Entertainment container. Fortunately, Digg provides an API for exactly that 
purpose.

Consider the following URL:

http://services.digg.com/1.0/endpoint?method=story.getPopular&container=Entertainment

This will retrieve a list of stories in that category (`Entertainment`, in this case) that have recently 
received enough Diggs to designate the story as "Popular." If you plug that URL into your browser 
and press Enter, you will see a list of Digg articles that come from the Entertainment container.

**Supported Features**

You can do a number of useful things with the Digg API. Having provided a couple of simple 
examples, now look in detail at some of its specific features.

**Get Popular Stories**

This Get Popular Stories endpoint is the one that you have already seen, and probably the one 
that most developers will use. As the name implies, this is the endpoint used to retrieve articles 
from Digg.

The method name for this endpoint is:

/story.getPopular

But of course you must specify the full URL as follows:

http://services.digg.com/1.0/endpoint?method=story.getPopular

As with the other endpoints, this one accepts specifications regarding exactly what should be 
returned. In this example, there is no specification, so Digg returns all of the popular articles. In the 
example in the previous section, the specification was for popular articles from the Entertainment 
container (`&container=Entertainment`).
If, for whatever reason, you do not provide any specifications, you will receive all articles from Digg, within limits. That URL would look like the one just above the previous paragraph. The result returned from that URL only includes the first 10 articles in the interest of saving bandwidth. You can increase the number of articles returned, up to 100, using the count parameter, as follows:

http://services.digg.com/1.0/endpoint?method=story.getPopular&count=30

This URL will return 30 articles as opposed to the default of 10.

Another useful option for listing stories is the option to list the upcoming stories:

http://services.digg.com/1.0/endpoint?method=story.getUpcoming

This returns the XML equivalent of what you would see if you manually went to the following URL:

http://digg.com/all/upcoming

You have already seen how to extract stories from a given container. But there are more options available to users who wish to use that feature. One option enables users to retrieve the top articles from a specific container. Sticking with the Entertainment category, that URL looks like this:

http://services.digg.com/1.0/endpoint?method=story.getTop&container=Entertainment

Note that the getTop method is used here as opposed to the getPopular method.

You can also view upcoming articles for a particular category by using the getUpcoming method as opposed to the getTop method in the URL above:

http://services.digg.com/1.0/endpoint?method=story.getUpcoming&container=Entertainment

### Get comments

The Get Comments endpoint returns a list of comments as specified in the REST invocation. Recall that Digg users can comment on submitted articles. This feature returns some of those comments, filtered based on the specifications in the request.

The method name for this endpoint is story.getComments.

To see all comments for a particular story, use this URL:

http://services.digg.com/1.0/endpoint?method=story.getComments&story_id=17390757

The story_id request parameter is required, and should be set to the story id of the article with the comments you want to examine. The output should look like Listing 2:
Using the Digg REST API

Listing 2. Digg comments: Output

```
<?xml version="1.0" encoding="utf-8" ?>
<events timestamp="1261234852" total="14" offset="0" count="10">
<comment date="1259165131" id="29563488" story="17390757"
up="10" down="1" replies="1" replyto="" user="Manchildcartoon"
level="0" root="29563488">That's not true...THAT'S IMPOSSIBLE!!!</comment>
<!-- The other 9 comments appear here -->
</events>
```

This returns 10 comments in XML format (remember, Digg only returns 10 unless you specifically ask for more). Each comment appears in a `comment` element as a child of the `events` element. I show one comment in the output above—the other 9 comments look just like this one. The parameters specify the date of the comment, the story identifier, the identifier of the event, and the user who entered the comment. Note that a Digg event can also be an actual digg as opposed to a simple comment; in that case, the event would appear within a `digg` element.

Get users

The Get Users feature, as can be implied from the name, gets Digg users as specified in the REST API.

The method name for this endpoint is:

```
/user.getAll
```

With that in mind, consider the following URL:

```
http://services.digg.com/1.0/endpoint?method=user.getAll&usernames=Manchildcartoon
```

It is important to point out that, for this feature, the `usernames` parameter is required. It should be set to a comma-delimited list of user names whose information you want to retrieve. This will return a result that looks like **Listing 3:**

Listing 3. Digg users: Output

```
<?xml version="1.0" encoding="UTF-8"?>
<users count="1" offset="0" timestamp="1261394886" total="1">
<user icon="/users/Manchildcartoon/l.png" name="Manchildcartoon" profileviews="565" registered="1228079424"/>
</users>
```

As you can see, the results include information about the specific Digg user, including the user’s icon (probably their picture), when the user registered, and the number of times the user’s profile has been viewed.

Conclusion

The Digg API employs a powerful set of features for users who want to programmatically access dugg articles, Digg events, and Digg users. Using these features, Web developers can provide instant access to popular Web content without redirecting their users to digg.com.