The Open Graph protocol

Introduction

The Open Graph protocol enables any web page to become a rich object in a social graph. For instance, this is used on Facebook to allow any web page to have the same functionality as any other object on Facebook.

While many different technologies and schemas exist and could be combined together, there isn't a single technology which provides enough information to richly represent any web page within the social graph. The Open Graph protocol builds on these existing technologies and gives developers one thing to implement. Developer simplicity is a key goal of the Open Graph protocol which has informed many of the technical design decisions.

Basic Metadata

To turn your web pages into graph objects, you need to add basic metadata to your page. We've based the initial version of the protocol on RDFa which means that you'll place additional `<meta>` tags in the `<head>` of your web page. The four required properties for every page are:

- `og:title` - The title of your object as it should appear within the graph, e.g., "The Rock".
- `og:type` - The type of your object, e.g., "video.movie". Depending on the type you specify, other properties may also be required.
- `og:image` - An image URL which should represent your object within the graph.
- `og:url` - The canonical URL of your object that will be used as its permanent ID in the graph, e.g., "https://www.imdb.com/title/tt0117500/".

As an example, the following is the Open Graph protocol markup for The Rock on IMDB:

```
<html prefix="og: https://ogp.me/ns#">
<head>
<title>The Rock (1996)</title>
<meta property="og:title" content="The Rock" />
<meta property="og:type" content="video.movie" />
<meta property="og:url" content="https://www.imdb.com/title/tt0117500/" />
<meta property="og:image" content="https://ia.media-imdb.com/images/rock.jpg" />
...
</head>
...
</html>
```

Optional Metadata

The following properties are optional for any object and are generally recommended:

- `og:audio` - A URL to an audio file to accompany this object.
- `og:description` - A one to two sentence description of your object.
- `og:determiner` - The word that appears before this object's title in a sentence. An enum of (a, an, the, "", auto). If `auto` is chosen, the consumer of your data should chose between "a" or "an". Default is "" (blank).
- `og:locale` - The locale these tags are marked up in. Of the format `language_TERRITORY`. Default is `en_US`.
- `og:locale:alternate` - An array of other locales this page is available in.
- `og:site_name` - If your object is part of a larger web site, the name which should be displayed for the overall site. e.g., "IMDb".
- `og:video` - A URL to a video file that complements this object.

For example (line-break solely for display purposes):

```
<meta property="og:audio" content="https://example.com/bond/theme.mp3" />
<meta property="og:description"
  content="Sean Connery found fame and fortune as the
           suave, sophisticated British agent, James Bond." />
<meta property="og:determiner" content="the" />
<meta property="og:locale" content="en_GB" />
<meta property="og:locale:alternate" content="fr_FR" />
<meta property="og:locale:alternate" content="es_ES" />
<meta property="og:site_name" content="IMDb" />
<meta property="og:video" content="https://example.com/bond/trailer.swf" />
```

The RDF schema (in Turtle) can be found at ogp.me/ns.

Structured Properties

Some properties can have extra metadata attached to them. These are specified in the same way as other metadata with `property` and `content`, but the `property` will have extra `:`.

The `og:image` property has some optional structured properties:

- `og:image:url` - Identical to `og:image`.
- `og:image:secure_url` - An alternate url to use if the webpage requires HTTPS.
- `og:image:type` - A MIME type for this image.
- `og:image:width` - The number of pixels wide.
- `og:image:height` - The number of pixels high.
- `og:image:alt` - A description of what is in the image (not a caption). If the page specifies an og:image it should specify `og:image:alt`.

A full image example:

```
<meta property="og:image" content="https://example.com/ogp.jpg" />
<meta property="og:image:secure_url" content="https://secure.example.com/ogp.jpg" />
<meta property="og:image:type" content="image/jpeg" />
<meta property="og:image:width" content="400" />
<meta property="og:image:height" content="300" />
<meta property="og:image:alt" content="A shiny red apple with a bite taken out" />
```

The `og:video` tag has the identical tags as `og:image`. Here is an example:

```
<meta property="og:video" content="https://example.com/movie.swf" />
<meta property="og:video:secure_url" content="https://secure.example.com/movie.swf" />
<meta property="og:video:type" content="application/x-shockwave-flash" />
<meta property="og:video:width" content="400" />
<meta property="og:video:height" content="300" />
```

The `og:audio` tag only has the first 3 properties available (since size doesn't make sense for sound):

```
<meta property="og:audio" content="https://example.com/sound.mp3" />
<meta property="og:audio:secure_url" content="https://secure.example.com/sound.mp3" />
<meta property="og:audio:type" content="audio/mpeg" />
```

Arrays

If a tag can have multiple values, just put multiple versions of the same `<meta>` tag on your page. The first tag (from top to bottom) is given preference during conflicts.

```
<meta property="og:image" content="https://example.com/rock.jpg" />
<meta property="og:image" content="https://example.com/rock2.jpg" />
```

Put structured properties after you declare their root tag. Whenever another root element is parsed, that structured property is considered to be done and another one is started.

For example:

```
<meta property="og:image" content="https://example.com/rock.jpg" />
<meta property="og:image:width" content="300" />
<meta property="og:image:height" content="300" />
<meta property="og:image" content="https://example.com/rock2.jpg" />
<meta property="og:image" content="https://example.com/rock3.jpg" />
<meta property="og:image:height" content="1000" />
```

means there are 3 images on this page, the first image is `300x300`, the middle one has unspecified dimensions, and the last one is `1000`px tall.

Object Types

In order for your object to be represented within the graph, you need to specify its type. This is done using the `og:type` property:

```
<meta property="og:type" content="website" />
```

When the community agrees on the schema for a type, it is added to the list of global types. All other objects in the type system are CURIEs of the form

```
<head prefix="my_namespace: https://example.com/ns#">
<meta property="og:type" content="my_namespace:my_type" />
```

The global types are grouped into verticals. Each vertical has its own namespace. The `og:type` values for a namespace are always prefixed with the namespace and then a period. This is to reduce confusion with user-defined namespaced types which always have colons in them.

Music

- Namespace URI: `https://ogp.me/ns/music#`

`og:type` values:

`music.song`

- `music:duration` - integer >=1 - The song's length in seconds.
- `music:album` - music.album array - The album this song is from.
- `music:album:disc` - integer >=1 - Which disc of the album this song is on.
- `music:album:track` - integer >=1 - Which track this song is.
- `music:musician` - profile array - The musician that made this song.

`music.album`

- `music:song` - music.song - The song on this album.
- `music:song:disc` - integer >=1 - The same as `music:album:disc` but in reverse.
- `music:song:track` - integer >=1 - The same as `music:album:track` but in reverse.
- `music:musician` - profile - The musician that made this song.
- `music:release_date` - datetime - The date the album was released.

`music.playlist`

- `music:song` - Identical to the ones on music.album
- `music:song:disc`
- `music:song:track`
- `music:creator` - profile - The creator of this playlist.

`music.radio_station`

- `music:creator` - profile - The creator of this station.

Video

- Namespace URI: `https://ogp.me/ns/video#`

`og:type` values:

`video.movie`

- `video:actor` - profile array - Actors in the movie.
- `video:actor:role` - string - The role they played.
- `video:director` - profile array - Directors of the movie.
- `video:writer` - profile array - Writers of the movie.
- `video:duration` - integer >=1 - The movie's length in seconds.
- `video:release_date` - datetime - The date the movie was released.
- `video:tag` - string array - Tag words associated with this movie.

`video.episode`

- `video:actor` - Identical to video.movie
- `video:actor:role`
- `video:director`
- `video:writer`
- `video:duration`
- `video:release_date`
- `video:tag`
- `video:series` - video.tv_show - Which series this episode belongs to.

`video.tv_show`

A multi-episode TV show. The metadata is identical to video.movie.

`video.other`

A video that doesn't belong in any other category. The metadata is identical to video.movie.

No Vertical

These are globally defined objects that just don't fit into a vertical but yet are broadly used and agreed upon.

`og:type` values:

`article` - Namespace URI: `https://ogp.me/ns/article#`

- `article:published_time` - datetime - When the article was first published.
- `article:modified_time` - datetime - When the article was last changed.
- `article:expiration_time` - datetime - When the article is out of date after.
- `article:author` - profile array - Writers of the article.
- `article:section` - string - A high-level section name. E.g. Technology
- `article:tag` - string array - Tag words associated with this article.

`book` - Namespace URI: `https://ogp.me/ns/book#`

- `book:author` - profile array - Who wrote this book.
- `book:isbn` - string - The ISBN
- `book:release_date` - datetime - The date the book was released.
- `book:tag` - string array - Tag words associated with this book.

`profile` - Namespace URI: `https://ogp.me/ns/profile#`

- `profile:first_name` - string - A name normally given to an individual by a parent or self-chosen.
- `profile:last_name` - string - A name inherited from a family or marriage and by which the individual is commonly known.
- `profile:username` - string - A short unique string to identify them.
- `profile:gender` - enum(male, female) - Their gender.

`website` - Namespace URI: `https://ogp.me/ns/website#`

No additional properties other than the basic ones. Any non-marked up webpage should be treated as `og:type` website.

Types

The following types are used when defining attributes in Open Graph protocol.

Type	Description	Literals
Boolean	A Boolean represents a true or false value	true, false, 1, 0
DateTime	A DateTime represents a temporal value composed of a date (year, month, day) and an optional time component (hours, minutes)	ISO 8601
Enum	A type consisting of bounded set of constant string values (enumeration members).	A string value that is a member of the enumeration
Float	A 64-bit signed floating point number	All literals that conform to the following formats: 1.234 -1.234 1.2e3 -1.2e3 7E-10
Integer	A 32-bit signed integer. In many languages integers over 32-bits become floats, so we limit Open Graph protocol for easy multi-language use.	All literals that conform to the following formats: 1234 -123
String	A sequence of Unicode characters	All literals composed of Unicode characters with no escape characters
URL	A sequence of Unicode characters that identify an Internet resource.	All valid URLs that utilize the https:// or https:// protocols

Discussion and support

You can discuss the Open Graph Protocol in the Facebook group or on the developer mailing list. It is currently being consumed by Facebook (see their documentation), Google (see their documentation), and mixi. It is being published by IMDb, Microsoft, NHL, Posterous, Rotten Tomatoes, TIME, Yelp, and many many others.

Implementations

The open source community has developed a number of parsers and publishing tools. Let the Facebook group know if you've built something awesome too!

- Facebook Object Debugger - Facebook's official parser and debugger
- Google Rich Snippets Testing Tool - Open Graph protocol support in specific verticals and Search Engines.
- PHP Validator and Markup Generator - OGP 2011 input validator and markup generator in PHP5 objects
- PHP Consumer - a small library for accessing of Open Graph Protocol data in PHP
- OpenGraphNode in PHP - a simple parser for PHP
- PyOpenGraph - a library written in Python for parsing Open Graph protocol information from web sites
- OpenGraph Ruby - Ruby Gem which parses web pages and extracts Open Graph protocol markup
- OpenGraph for Java - small Java class used to represent the Open Graph protocol
- RDF::RDFa::Parser - Perl RDFa parser which understands the Open Graph protocol
- WordPress plugin - Facebook's official WordPress plugin, which adds Open Graph metadata to WordPress powered sites.
- Alternate WordPress OGP plugin - A simple lightweight WordPress plugin which adds Open Graph metadata to WordPress powered sites.

The Open Graph protocol was originally created at Facebook and is inspired by Dublin Core, link-rel canonical, Microformats, and RDFa. The specification described on this page is available under the Open Web Foundation Agreement, Version 0.9. This website is Open Source.