API DEVELOPMENT < HTTPS://WWW.KENNETHLANGE.COM/CATEGORY/API-DEVELOPMENT/>

Avoid Data Corruption in Your REST API with ETags

By **Kenneth Lange < https://www.kennethlange.com/author/kenneth-lange/>**

January 30, 2016 < https://www.kennethlange.com/avoid-data-corruption-in-your-rest-api-with-etags/>

There are few things worse than a really nasty data corruption issue.

Especially if it has occurred silently over a long period of time, so when it's discovered it's too late to rollback to a backup before the defect was introduced. It's even worse if it has also occurred randomly, so there is no pattern to base your fix upon.

Yet an awful lot of **REST APIs** ignore **concurrency control**, so if they are used by multiple clients who modify the same data at the same time then it can lead to **lost updates** and **stalled deletes**, which slowly ruins the data in the database.

This is totally unacceptable in most enterprise applications where data integrity is something you just don't fool around with…

So how can you avoid this messed up situation?

You use the concurrency control designed into the **HTTP protocol** as a simple, yet effective way to protect the integrity of your data.

Meet the ETag Header

If you want to use the concurrency control in the **HTTP Protocol**, you need to use the optional **Entity Tag** (ETag) header in the **HTTP request**.

The **ETag** is kind of like a version stamp for a resource and it's returned as part of the **HTTP response**.

For example, if you send a request to get a specific customer:

```
GET /customers/987123 HTTP/1.1
```

Then the **ETag** (if used) is included in the header in the response:

```
HTTP/1.1 200 OK
Date: Sat, 30 Jan 2016 09:38:34 GMT
ETag: "1234"
Content-Type: application/json


{
  "id":987123,
  "firstname":"Han",
  "lastname":"Solo",
  "job":"Smugler"
}
```

Each time the resource is updated on the server, the **ETag** header will be changed to reflect the content of the new version of the resource.

So to avoid lost updates, you simply take the value of the **ETag** header and put into the **If-Match** header on the **PUT** request:

```
PUT /customers/987123 HTTP/1.1

If-Match: "1234"
Content-Type: application/json

{
  "id":987123,
  "firstname":"Han",
  "lastname":"Solo",
  "job":"General"
}
```
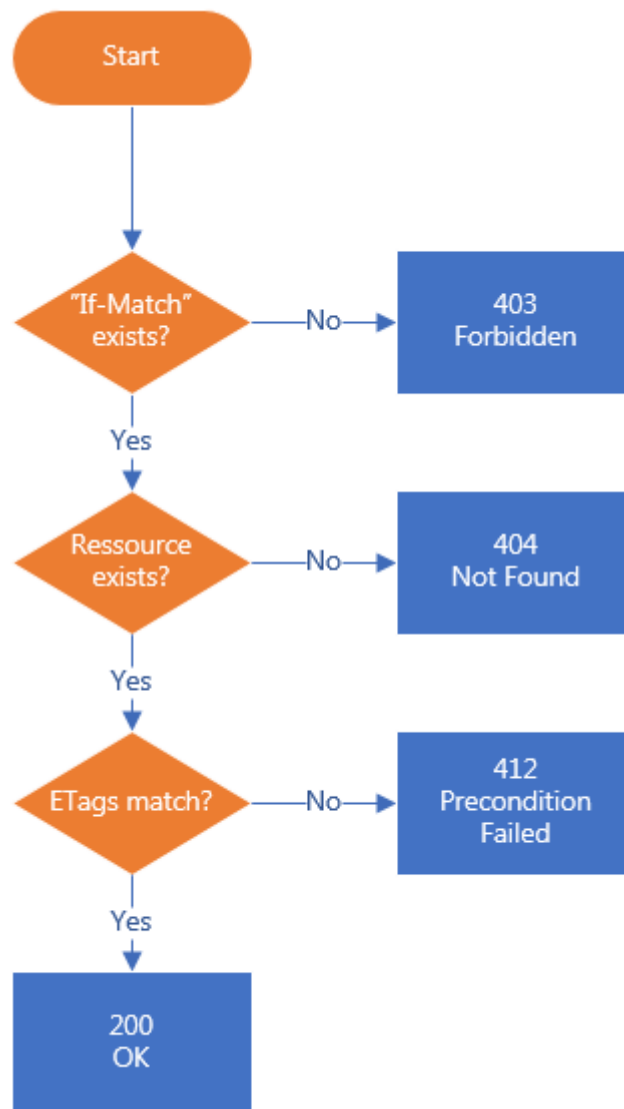
The **PUT** request above says that you want to update the customer resource on the server, but only if the **ETag** matches **1234** to make sure that the customer hasn't been updated since you sent the **GET** request. In this way, your request won't incidentally overwrite other users' updates.

Over at the Server

When the server gets the **PUT** request, it will execute the logic below:

First, the server checks that the **If-Match** header is included in the request. If not, it will tell the client that you cannot update this resource without the **If-Match** header.

Second, the server checks if the resource actually exists as it must exists before it can be updated.

Third, it checks if the **ETag** supplied in the **If-Match** header is the same as the latest **ETag** on the resource. If not, it tells the client that the precondition has failed.

Finally, if the request passes the three validations, then the server updates its resource.

If the server uses the same approach on **DELETE** request, you can also avoid stalled deletes.

Note: The logic above assumes that the server doesn't allow you to create new resources with a **PUT** request. If this is allowed then the first step should be to check if the resource exists, and if not then branch out and create it and skip the other steps.

Implementation Hints

There are several ways to implement **ETags** on the server.

One way is to make a hash of the resource, and put in the **ETag** header. But you need to make sure that the hash includes all updatable fields in the response, be sure that there cannot be hash collisions, and find a hashing algorithm that doesn't impact performance too much.

Another really simple way to implement **ETags** is to add a read-only etag column to the underlying database table and add a trigger that increases the value each time the row is updated. Of course, the server needs to be aware that the database changes this value behind the screen, so the server always uses the latest version.

Why not timestamps instead of ETags?

An easier way to implement concurrency control in **HTTP** is to use the **Last-Modified** and **If-Unmodified-Since** headers instead of **ETag** and **If-Match**. This difference is simply that these two headers use timestamps instead of **ETags**.

So, if it's easier why not use it?

The problem is that the timestamps use seconds as their finest precision, so if you have fast, high-frequency updates then there is a risk that two updates occur within the same second and you lose one of them.

So in enterprise software where data integrity is an absolute requirement, **Etags** are the safe choice.

———————————————————— // ————————————————————

← **7 Tips for Designing a Better REST API**   **< https://www.kennethlange.com/7-tips-for-designing-a-better-rest-api/>**

→ **Boost Your REST API with HTTP Caching**   **< https://www.kennethlange.com/boost-your-rest-api-with-http-caching/>**

———————————————————— // ————————————————————

© 2020 Kenneth Lange < https://www.kennethlange.com/>   **Up ↑**