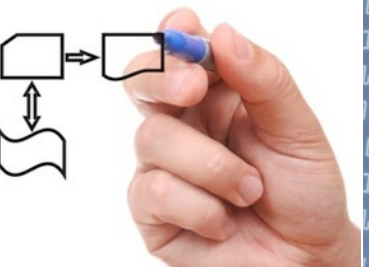
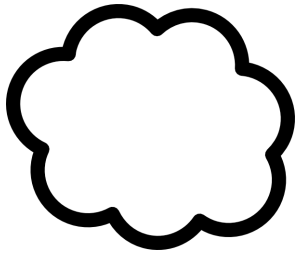
San Jose State University

CMPE 281
Cloud Technologies

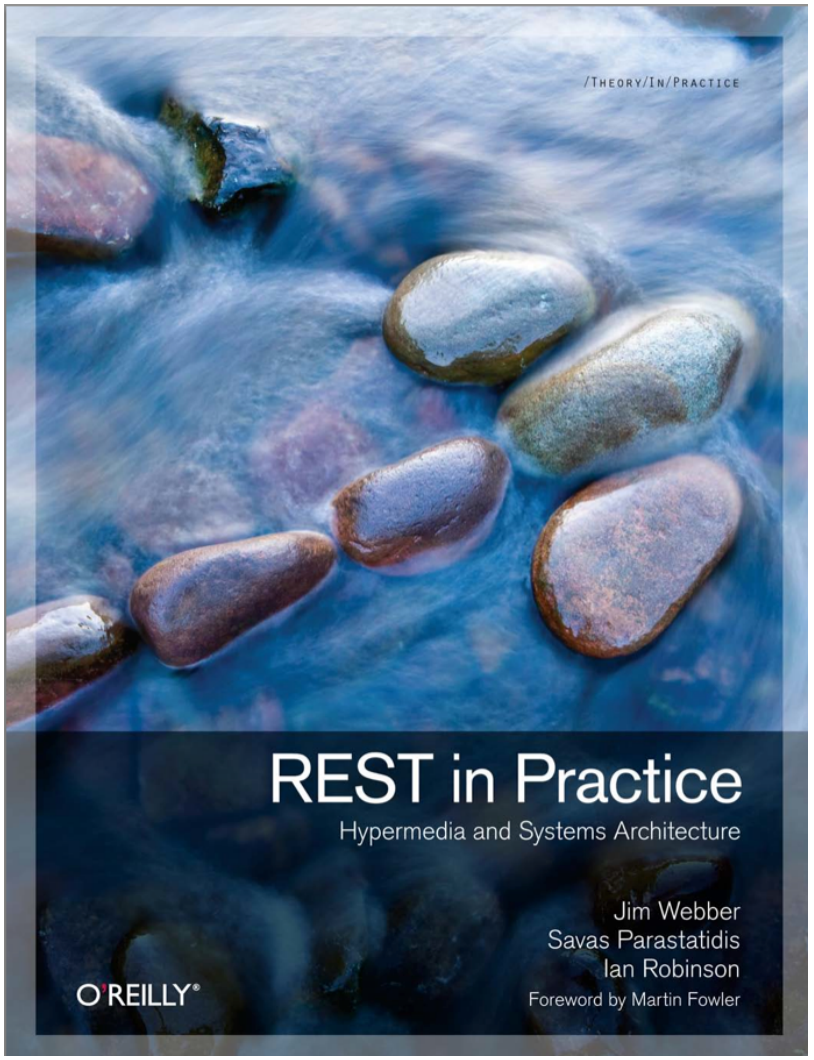
RESTbucks
"CRUD" REST API





CRUD Web Services

(*Excerpts from the "REST in Practice" book on the REST Architecture Style*)



Modeling Orders As Resources

In Restbucks, orders are core business entities, and as such, their life cycles are of real interest to us from a CRUD perspective. For the ordering parts of the Restbucks business process, we want to create, read, update, and delete order resources like so:

- Orders are created when a customer makes a purchase.
- Orders are frequently read, particularly when their preparation status is inquired.
- Under certain conditions, it may be possible for orders to be updated (e.g., in cases where customers change their minds or add specialties to their drinks).
- Finally, if an order is still pending, a customer may be allowed to cancel it (or delete it).

Within the ordering service, these actions (which collectively constitute a protocol) move orders through specific life-cycle phases, as shown in Figure 4-1.

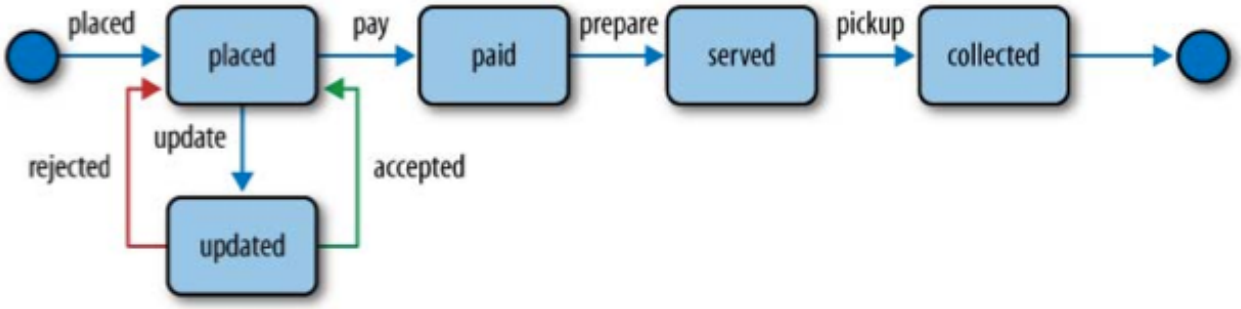


Figure 4-1. *Possible states for an order*

Each operation on an order can be mapped onto one of the HTTP verbs. For example, we use POST for creating a new order, GET for retrieving its details, PUT for updating it, and DELETE for, well, deleting it. When mixed with appropriate status codes and some commonsense patterns, HTTP can provide a good platform for CRUD domains, resulting in really simple architectures, as shown in Figure 4-2.

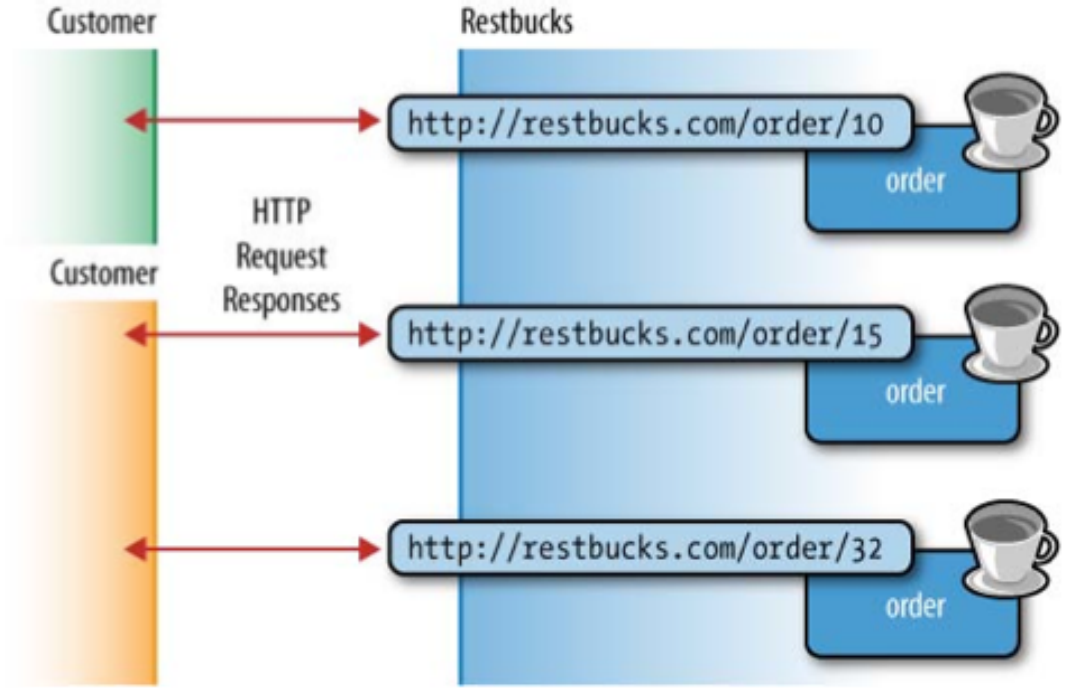


Figure 4-2. *CRUD ordering service high-level architecture*

While Figure 4-2 exemplifies a very simple architectural style, it actually marks a significant rite of passage toward embracing the Web's architecture. In particular, it highlights the use of URIs to identify and address orders at Restbucks, and in turn it supports HTTP-based interactions between the customers and their orders.

Since CRUD services embrace both HTTP and URIs, they are considered to be at level two in Richardson's maturity model. Figure 4-3 shows how CRUD services embrace URIs to identify resources such as coffee orders and HTTP to govern the interactions with those resources.

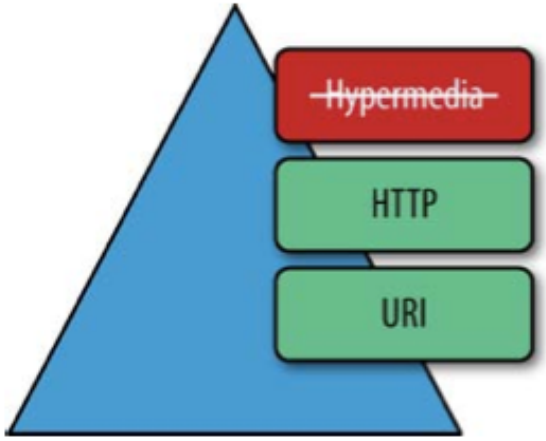


Figure 4-3. *CRUD services reach level two on Richardson's maturity model*

Level two is a significant milestone in our understanding. Many successful distributed systems have been built using level two services. For example, Amazon's S3 product is a classic level two service that has enabled the delivery of many successful systems built to consume its functionality over the Web. And like the consumers of Amazon S3, we'd like to build systems around level two services too!

Building CRUD Services

When you're building a service, it helps to think in terms of the behaviors that the service will implement. In turn, this leads us to think in terms of the contract that the service will expose to its consumers. Unlike other distributed system approaches, the contract that CRUD services such as Restbucks exposes to customers is straightforward, as it involves only a single concrete URI, a single URI template, and four HTTP verbs. In fact, it's so compact that we can provide an overview in just a few lines, as shown in Table 4-1.

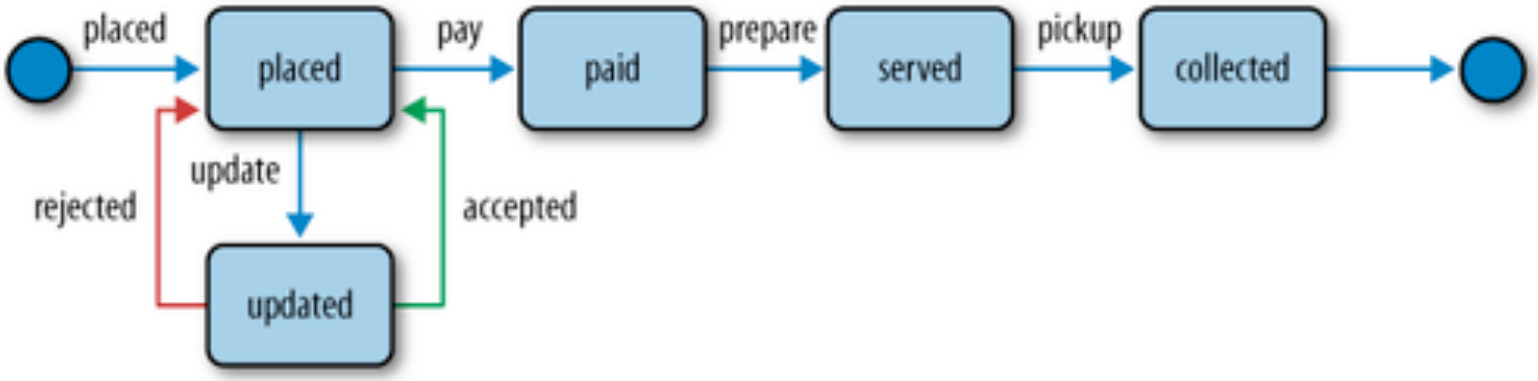
Table 4-1. *The ordering service contract overview*

Verb	URI or template	Use
`POST`	`/order`	Create a new order, and upon success, receive a `Location` header specifying the new order's URI.
`GET`	`/order/{orderId}`	Request the current state of the order specified by the URI.
`PUT`	`/order/{orderId}`	Update an order at the given URI with new information, providing the full representation.
`DELETE`	`/order/{orderId}`	Logically remove the order identified by the given URI.

The contract in Table 4-1 provides an understanding of the overall life cycle of an order. Using that contract, we can design a protocol to allow consumers to create, read, update, and delete orders. Better still, we can implement it in code and host it as a service.

NOTE

What constitutes a good format for your resource representations may vary depending on your problem domain. For Restbucks, we've chosen XML, though the Web is able to work with any reasonable format, such as JSON or YAML.



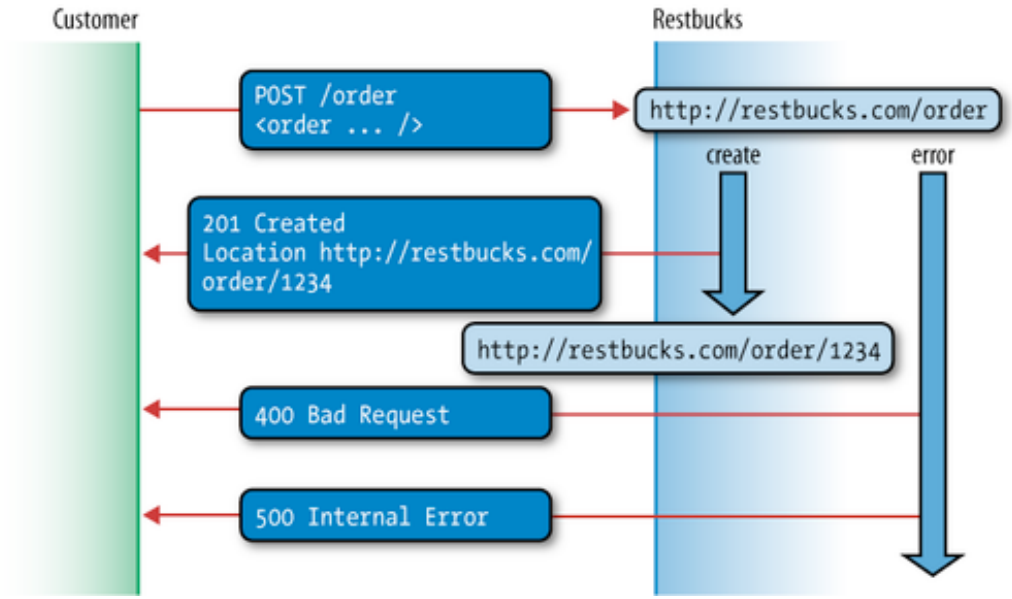
Each operation on an order can be mapped onto one of the HTTP verbs. For example, we use `POST` for creating a new order, `GET` for retrieving its details, `PUT` for updating it, and `DELETE` for, well, deleting it. When mixed with appropriate status codes and some commonsense patterns, HTTP can provide a good platform for CRUD domains, resulting in really simple architectures

Verb	URI or template	Use
`POST`	`/order`	Create a new order, and upon success, receive a `Location` header specifying the new order's URI.
`GET`	`/order/{orderId}`	Request the current state of the order specified by the URI.
`PUT`	`/order/{orderId}`	Update an order at the given URI with new information, providing the full representation.
`DELETE`	`/order/{orderId}`	Logically remove the order identified by the given URI.

POST

```
POST /order HTTP/1.1
Host: restbucks.com
Content-Type: application/xml
Content-Length: 239

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <name>latte</name>
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
  </items>
</order>
```

```
HTTP/1.1 201 Created
Content-Length: 267
Content-Type: application/xml
Date: Wed, 19 Nov 2008 21:45:03 GMT
Location: http://restbucks.com/order/1234

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <name>latte</name>
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
  </items>
  <status>pending</status>
</order>
```

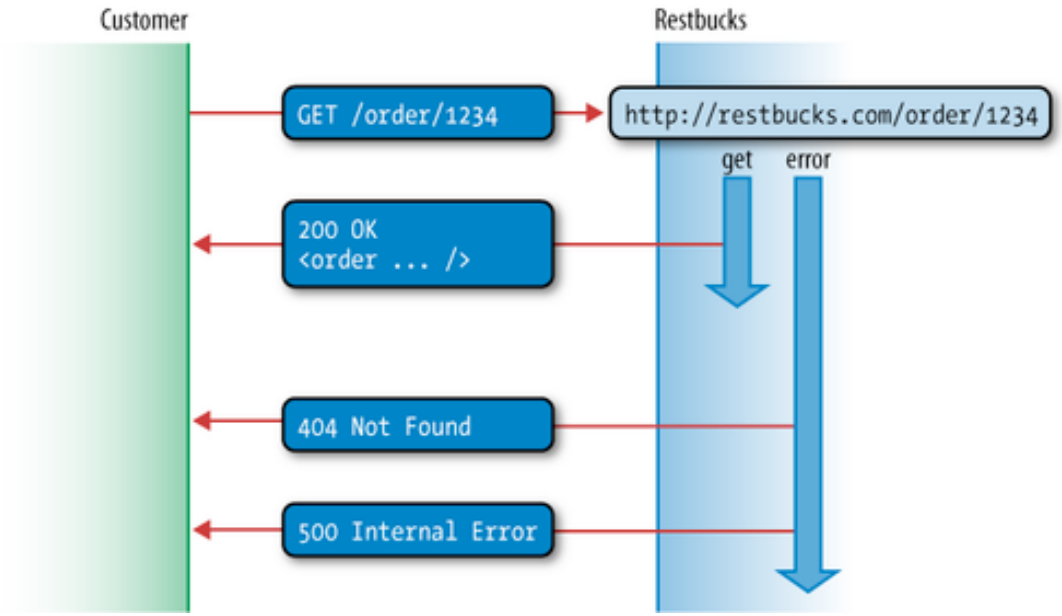



```
HTTP/1.1 400 Bad Request
Content-Length: 250
Content-Type: application/xml
Date: Wed, 19 Nov 2008 21:48:11 GMT

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <!-- Missing drink type -->
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
  </items>
</order>
```

GET

```
GET /order/1234 HTTP/1.1
Host: restbucks.com
```

```
HTTP/1.1 200 OK
Content-Length: 241
Content-Type: application/xml
Date: Wed, 19 Nov 2008 21:48:10 GMT

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <name>latte</name>
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
  </items>
</order>
```

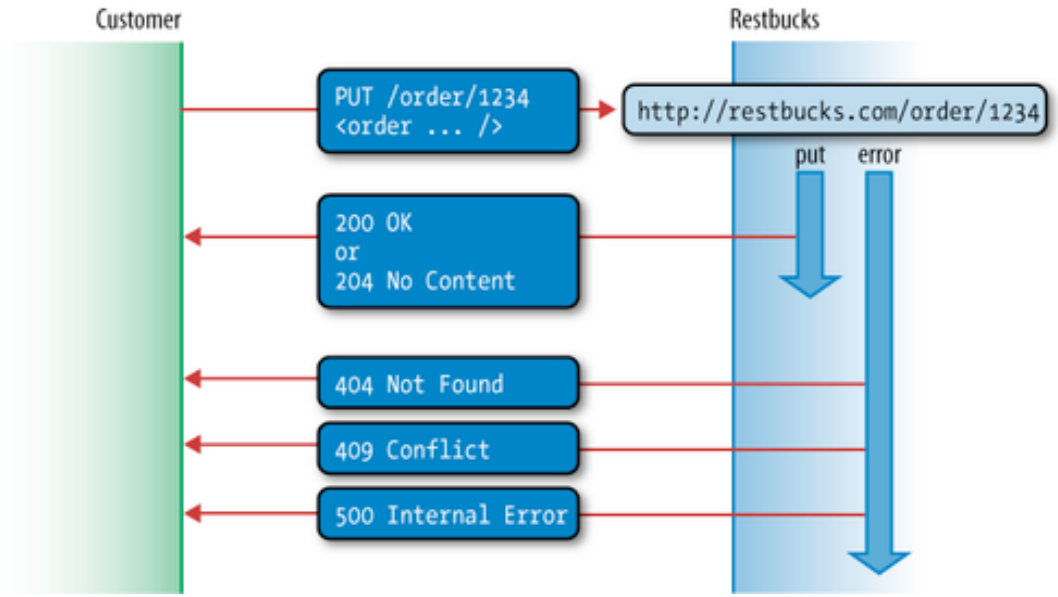



```
HTTP/1.1 404 Not Found
Date: Sat, 20 Dec 2008 19:01:33 GMT
```

PUT

```
PUT /order/1234 HTTP/1.1
Host: restbucks.com
Content-Type: application/xml
Content-Length: 246

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <milk>skim</milk>
      <name>cappuccino</name>
      <quantity>1</quantity>
      <size>large</size>
    </item>
  </items>
</order>
```

```
HTTP/1.1 200 OK
Content-Length: 275
Content-Type: application/xml
Date: Sun, 30 Nov 2008 21:47:34 GMT

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <milk>skim</milk>
      <name>cappuccino</name>
      <quantity>1</quantity>
      <size>large</size>
    </item>
  </items>
  <status>preparing</status>
</order>
```

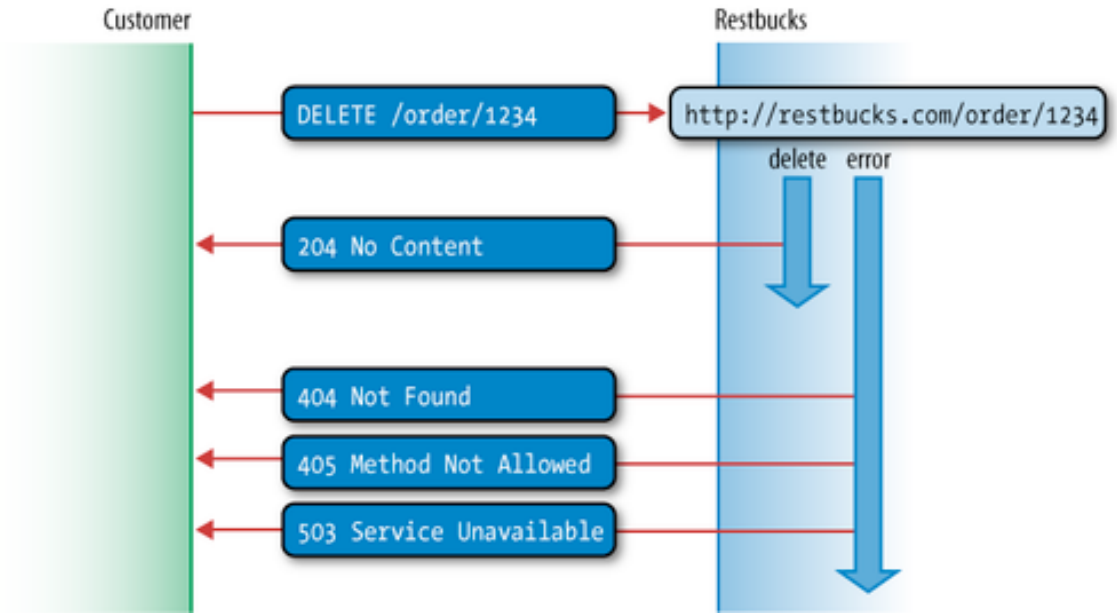



```
HTTP/1.1 204 No Content
Date: Sun, 30 Nov 2008 21:47:34 GMT
```

DELETE

```
DELETE /order/1234 HTTP/1.1
Host: restbucks.com
```

```
HTTP/1.1 204 No Content
Date: Tue, 16 Dec 2008 17:40:11 GMT
```




```
HTTP/1.1 404 Not Found
Content-Length: 0
Date: Tue, 16 Dec 2008 17:42:12 GMT
```