SOCIAL SURVEY API

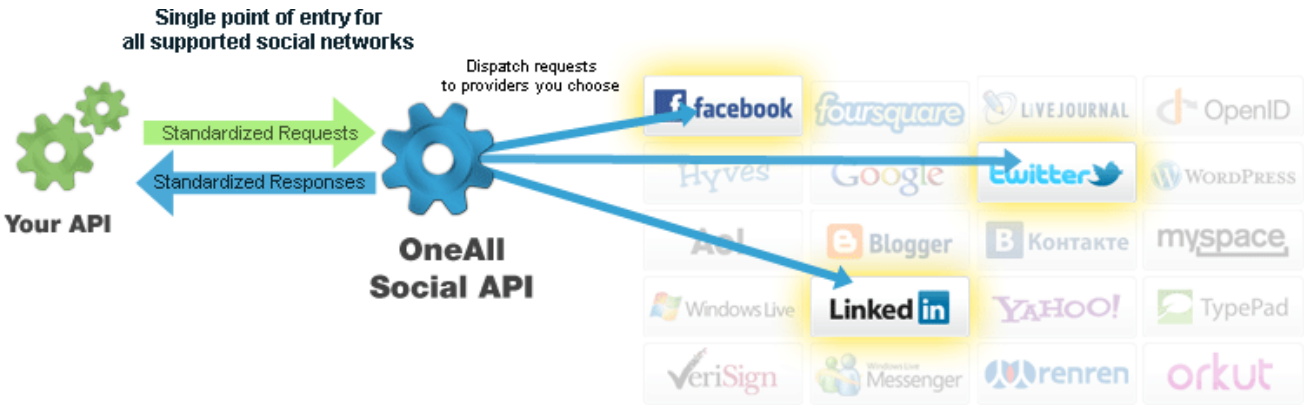
Tony Mears

"SocialSurvey API provides access to import existing review data from the Social Survey system. This data can be used for business reporting and/or integration of the reviews content into your company websites."

TABLE OF CONTENTS

Version Number	Change Description	Effective Date
3.0	Combine API Documents to one Manual	28-Feb-2019

API OVERVIEW



BENEFITS OF AN API

Application programming interfaces, or API's, are the way software talks to other software—so by definition, they're a technology that connects systems. Many enterprise leaders think of them primarily in these terms. But API's are much more. They're interfaces that enable developers to repeatedly leverage data, functions, and applications to build new products and services. They're how a business expresses itself via software, and they enable that business to rapidly expand into new contexts or adapt to meet changing user needs and preferences.

WHY USE AN API INSTEAD OF MOVING FILES AROUND

Clean Data: API's provide data in clean, programmatically accessible formats. No need to convert the data from one file format to another, no need to scrub out bad or polluted entries and no need to load it into your local system.
Timely Data: API's can provide data that is refreshed much more often than you can achieve with pulling, cleaning, and loading files.

Correct Data: API's always pull data from a single, authoritative source, so you know you're getting the "true" information. You often must rely on others to perform functions that you may not be able or permitted to do by yourself, such as signing up for a service. Similarly, virtually all software must request other software to do some things for it.

To accomplish this, the asking program uses a set of standardized requests, called application programming interfaces (API), that have been defined for the program being called upon. Almost every application depends on the API's of the underlying operating system to perform such basic functions as accessing the file system. A program's API defines the proper way for a developer to request services from that program.

Developers can make requests by including calls in the code of their applications. The syntax is described in the documentation of the application being called. By providing a means for requesting program services, an API is said to grant access to or open an application.

WHY YOU SHOULD USE AN API

Computers make a lot of things easier, especially tasks that involve collecting and sorting through tons of data. Let's say you wanted to know how many times a client submitted reviews for your agents to your company. You could feasibly go into your company's records, scan the "from" data input, and print each review individually for your audit.

On the other hand, if all reviews were uploaded to a central database, you could write a simple program that accesses that database and finds all the instances of the agents' name. This would take much less time and be much more accurate.

HOW DO API'S WORK?

Imagine a waiter in a restaurant. You, the customer, are sitting at the table with a menu of choices to order from, and the kitchen is the provider who will fulfill your order.

You need a link to communicate your order to the kitchen and then to deliver your food back to your table. It can't be the chef because she's cooking in the kitchen. You need something to connect the customer who's ordering food and the chef who prepares it. That's where the waiter — or the API — enters the picture.

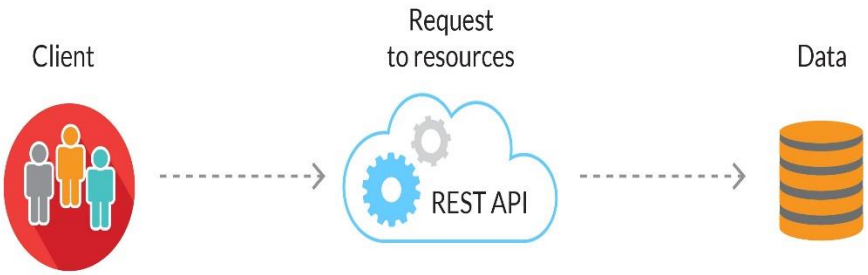
The waiter takes your order, delivers it to the kitchen, telling the kitchen what to do. It then delivers the response, in this case, the food, back to you. Moreover, if it's designed correctly, hopefully, your order won't crash!

WHY IS API SO POPULAR

The major social media networks all have API's. While you are prohibited from using them to duplicate the company's core service and sell that as your software, you can build on them to improve the experience.

Twitter's API allows you to access certain points of a public profile. As a basic use of the API, you could write a program where you can search for someone's username and it'll return the profile page. Instead of walking up to the Twitter office every time you have a request, the API gives access to the program to return the profile page.

If you use a third-party social media management platform like Facebook, then you've experienced the use and limitations of API's. They allow you to post, comment and like posts on behalf of your account. The advantage of software built on Social Survey API's is that you're able to view multiple accounts in one place.

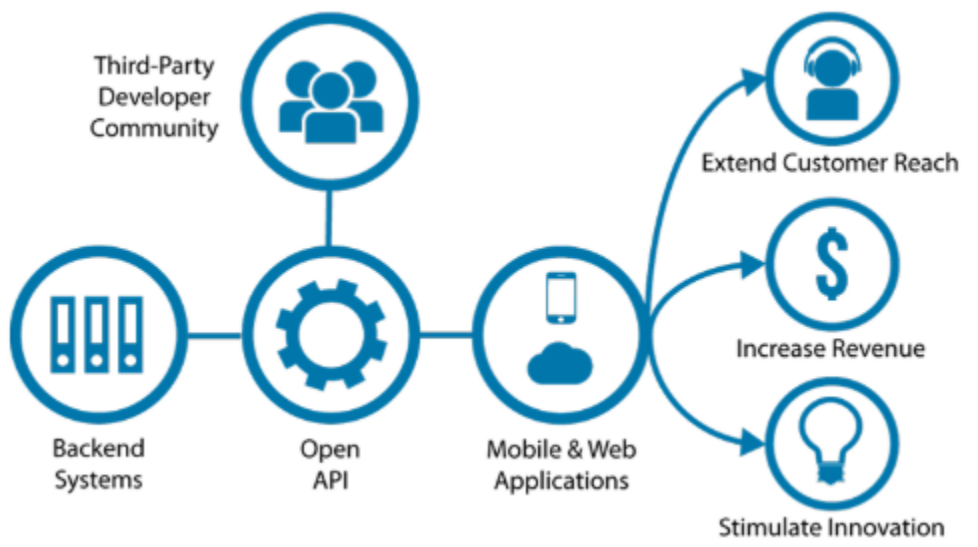


WHY USE SOCIAL SURVEY API'S

SocialSurvey supports the principle belief that the survey/review data belongs to you and you may manage it in accordance to your company's operating policies and business rules. You may request for the survey/review data via API or a file export from SocialSurvey. The API can provide you data for a certain date range or a specified number of latest reviews

SocialSurvey is known for managing your online reputation, but that's not what makes us special. What makes SocialSurvey special is the use of data in motion and engaging a network of people via our platform. We want your customers to be a part of this network, where their true voice is amplified and can spark discovery and connect you to new people.

To help new customers discover and connect with you, we've built a Review API, using OAuth2 for authentication, through which you can seamlessly publish your reviews to your personal testimonial page.



To make it easy we have created full documentation on ways to consume our review API. There is no fuss to integrate, no messy builds to get up and running. You could use our API to customize how your reviews are displayed when you publish review data on your company site. You can also publish spotlight reviews that highlight the voice of your happiest customers. We're incredibly excited to help you unlock your creativity and bring even more customer testimonials and visibility to your business.

YOU MAY USE SOCIALSURVEY API TO:

POST transaction details to initiate survey sending

GET survey (and review) data collected for your account

DATA USAGE

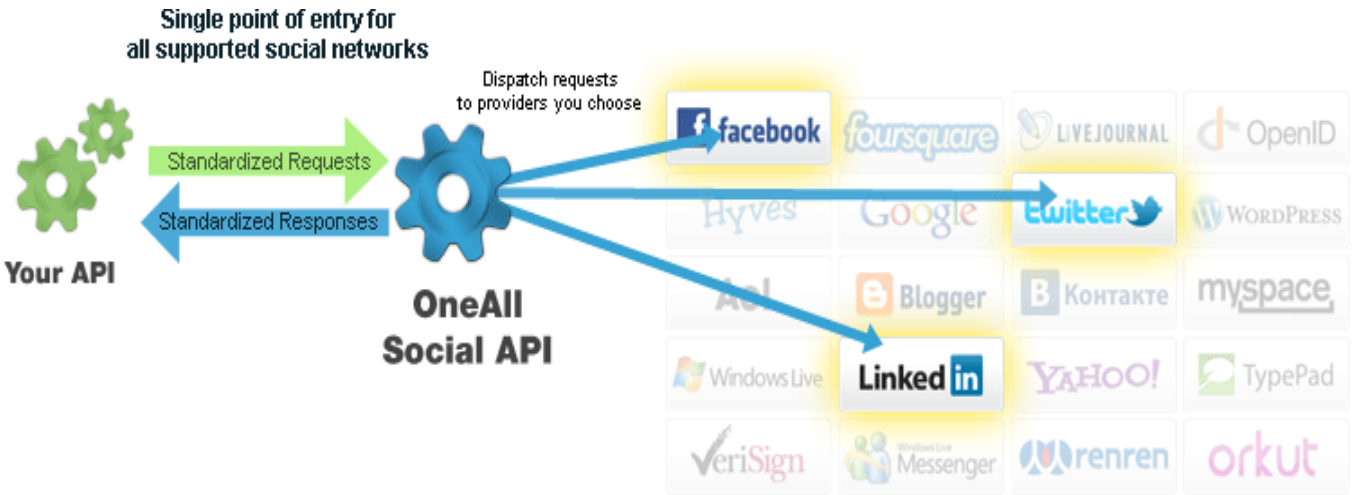
Social Survey's surveys API allows you to create and maintain a replica database of your survey/review data that you may use to integrate with custom CRM's for analytics, Intranet portals for reporting, or to host reviews on your own public facing website on the Internet.

Caution: Data provided by the surveys API may not be used for real time dynamically generated content and is provided solely for the purpose of generating and updating your locally hosted database.

Note: If users from your organization connected their SocialSurvey profiles with Zillow, it is possible that you may receive some Zillow Reviews data through the SocialSurvey API. Please note that Zillow reviews are subjected to the terms of use of Zillow's data policy. Please be aware that the current policy from Zillow regarding their data usage permits you to dynamically display their reviews content but it does not permit you to store information locally. Click here to view details of the https://www.zillow.com/howto/API/APITerms.htm for their data usage.

Social Survey

API DEVELOPER'S GUIDE



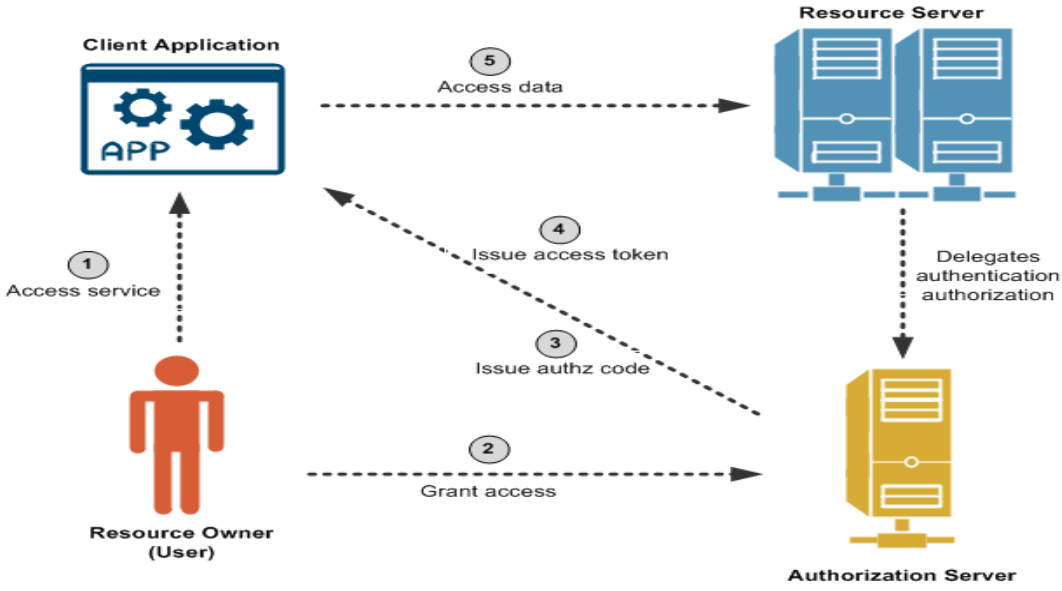
GETTING STARTED WITH SOCIALSURVEY API'S

Your SocialSurvey representative should have already provided you with an Access Token for your account. You may request a new or re-issued Access Token by email to support@socialsurvey.com.

OAUTH SECURITY PROTOCOL

OAuth is a standard for delegating authorization, it is standardized as API security best practice in the industry. The API consumer needs to provide a secret (=password) as part of the API request and the API provider authenticates and authorizes the request. In API scenarios, typically a third identity Is involved, the Identity of the end-user. This allows scenarios where the third part app of an API consumer gets access to the personal data of the end-user via API. It Is a very common pattern of practice In API design.

For example, the end-user wants his reviews from Social Survey to appear on Twitter, LinkedIn or Facebook, to allow this functionality, social media needs to have access to the end-user's accounts to read or share the reviews. OAuth Is a standard for delegating access, with OAuth the end-users get a Token from Social Survey to delegate his access rights to social media. The token represents the access rights for a subset of data, for a short time frame. Social Survey uses the access token to call the social media API's. Social Survey issues the token, so you delegate authority to exchange data.



WHAT IS HTTP

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.

HTTP works as a request-response protocol between a client and server.

A web browser may be the client, and an application on a computer that hosts a web site may be the server.

Example: A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

HTTP METHODS

HTTP methods have two important properties: HTTP methods can be safe and idempotent. Knowing the implications of these properties is especially useful for the API clients.

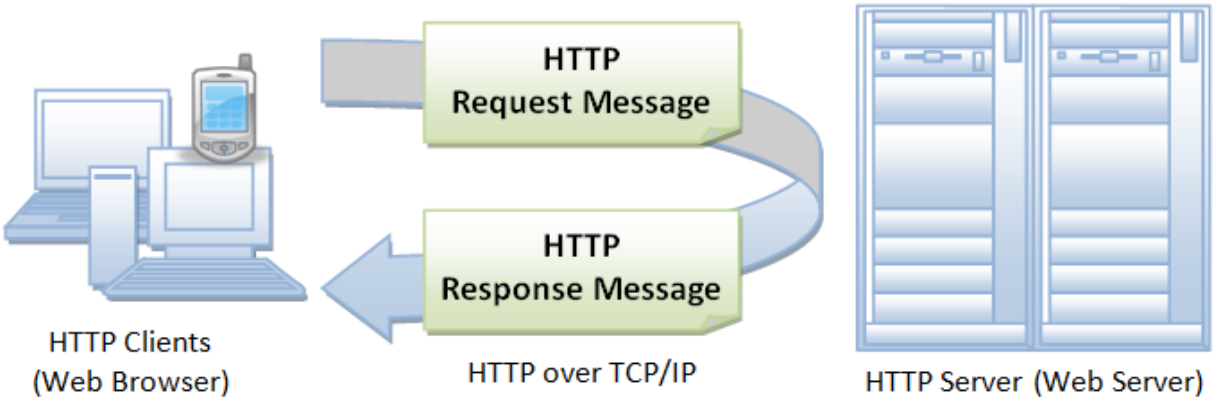
Why is it important to distinguish safe and unsafe HTTP methods? Safe methods can be used by spiders and web caches to pre-fetch without altering the state of the resource. Why distinguish between idempotent and non-idempotent? Clients might send a request, wait for an answer, but do not receive a response. If the method is idempotent, the client can simply re-send the request. Non-idempotent (ex: bank transfers), the client would need to check if the previous request was executed and roll it back, before retrying. Without this check, two transactions may be initiated.

The meaning of HTTP Methods

- **GET**
- **POST**
- **PUT**
- **HEAD**
- **DELETE**
- **PATCH**
- **OPTIONS**
- **TRACE**

Properties of HTTP Methods

- **<u>Idempotent methods</u> can be repeated without altering the end results. Executing the method multiple times has the same effect as executing the method only once. It is like a "setter" method in programming.**
- **<u>Safe methods</u> do not have side effects, do not change the state of the resource and are read-only. NOTE, that the value of the resource can still change, backend changes, or by other methods. POST and PUT are unsafe.**



THE GET METHOD

GET is used to request data from a specified resource. The GET operation Is Idempotent and the Information retrieved by GET can be cached, and the request can be resubmitted. GET request *do not* contain a payload, only the GET response contains a payload. If the resource exists. a *200 OK* status code Is returned. If the resource does not exist, a *404 NOT Found* status code Is returned.

GET Is one of the most common HTTP methods.

THE POST METHOD

POST is used to send data to a server to create/update a resource. The POST method creates the new resource anonymously as part of a collection resource. The creation Is anonymous because the Implementation of the POST method for the specific resource determines the URL of the new resource. If the client wants to determine the URL of a new resource the resource should Instead be created with the PUT command. The POST method Is used unsafe operations and for long running batch processes in combination with controller resources

POST is one of the most common HTTP methods.

THE PUT METHOD

PUT is used to send data to a server to create/update or modify a resource at a specific URL. The request typically contains a representation of the resource to be created or updated/modified. The resource to be created or the changes to be made Is expressed as a full representation In the HTTP body of the request. If the resource with the provide URI already exists, the PUT Is Interpreted as a modification. If the resource ***does not exist,*** It Is interpreted as a creation. The PUT method Is unsafe, but idempotent, so It can be resubmitted on a *4xx* or *5xx.*

> **The difference between POST and PUT is that PUT requests are idempotent. That is, calling the same PUT request multiple times will always produce the same result. In contrast, calling a POST request repeatedly have side effects of creating the same resource multiple times.**

THE HEAD METHOD

HEAD is almost identical to GET, but without the response body.

In other words, if GET /users returns a list of users, then HEAD /users will make the same request but will not return the list of users.

HEAD requests are useful for checking what a GET request will return before making a GET request - like before downloading a large file or response body.

THE DELETE METHOD

The DELETE method deletes the specified URL. The request does not contain any HTTP body. The response may contain a body, for example with the representation of the deleted resource. This method is idempotent, i.e., if called again in the future it needs it needs to provide the same response.

If the response contains a body, status code *200 OK* is returned. If the representation is not included. ***"does not contain"*** in the body of the HTTP response, a status code *204 No Content"* is returned.

THE OPTIONS METHOD

The OPTIONS method describes the communication options for the target resource. I can be used for retrieving a list of the HTTP methods, which are allowed on a given resource, Is Idempotent and needs to deliver to deliver the same response when resubmitted. The response with *ALLOW* header tells the client that the HTTP methods, GET, PUT and DELETE can be applied on the specified resource. OPTIONS request is idempotent and needs to deliver the same response when resubmitted (ex. After a *4xx or 5xx)* OPTIONS is read only and uncritical.

THE TRACE METHOD

TRACE is used for testing HTTP connections. When calling an API with the TRACE method, the API echoes back the HTTP headers and the body of the request. The HTTP body of the response includes the entire request including the request headers and the request body.

THE PATCH METHOD

The PATCH method is not part of the original HTTP standards; however, it is described separately in industry standards publications. The PATCH method is used to partially update an existing resource. This means the client does not send the complete resource representation to the server, but only certain fields describing the difference between the old and the new versions of the resource. To describe the difference between two JSON objects, use the following standards: JSON PATCH and JSON Merge Patch. Since PUT can also be used to update what is the difference between PUT and PATCH? PUT is used to do a "complete" update, PATCH is used for "partial" updates, only changing fields etc. PATCH cannot create a resource.

NON-STANDARD HTTP METHODS

Non-standard HTTP methods must be avoided. HTTP proxies and caches need to treat these methods as unsafe and non-idempotent. Non-standard HTTP methods have been used in the WebDav Project. Examples are MOVE, LOCK, UN-LOCK, PROPATCH, PROFIND.

> **Recommendation: Instead of inventing and using non-standard HTTP methods, use the POST method**

OVERVIEW OF STATUS CODES

The numerical status codes consist of 3 digits, for example *201*. The first digit groups the status codes into five groups:

- **Informational *(1xx)*** status codes: non-critical information
- **Success (*2xx)*** status codes: the request was processed successfully
- **Redirection (*3xx)*** status codes: the client needs to do another request based on the header parameters received in the response.
- **Client Error(*4xx)*** status codes: an error occurred on the client side
- **Server Error (*5xx)*** stats codes: an error occurred on the server side

More consideration must be given to the status codes indicating redirection and error. They are relatively easy to handle.

> ***Note: A complete list of error codes is in the appendix***

COMPARE GET VS. POST

The following table compares the two HTTP methods: GET and POST.

Function	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL. Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

RESOURCE PATHS

SocialSurvey APIs are supported on Social Survey's Test and Production platforms. The resource path for each platform are as follows:

Environment	Resource path
Testing	http://api.socialsurvey.info/v2/
Production	http://api.socialsurvey.me/v2/

All API resources are prefixed with the Resource Paths listed above.

SERVICE ENDPOINTS

Method	HTTP Request	Description
surveys	GET / surveys	Get existing survey data
incompletesurveys	GET /incomplete surveys	Gets existing survey data which has a status of incomplete
surveys	PUT/surveys	Create a new transaction for processing into a survey

USING THE API KEY TO ACCESS SOCIALSURVEY RESOURCES

Authorization is ALWAYS REQUIRED

"API KEY" tokens are different for Testing vs Production environments

There are two headers to be added in each API:

Name	Required	Location	Description
Content Type	Yes	Header	application/JSON
authorization	Yes	Header	Basic "API_Key"

CURL AUTHORIZATION

“curl” is a command line tool to test API. it is like “poster” addon available in “Google Chrome browser”. Or “Postman” app. Where you can test the API with URL/Request/Header parameters. In “curl” you need to pass the parameters in command line parameters as Linux commands. It is lightweight and easy to use. But the commands need to be prepared and pre-populated with all the parameters and you need to install the software manually.

Example curl Authorization Syntax

curl –header “Content-Type: application/json” --header “Authorization: Basic “API_KEY””-X GET “https://api.socialsurvey.me/v2/surveys”

cURL Code Example

```
curl -d "userid=1&filecomment=This is an image file" --data-binary
@"/home/user1/Desktop/test.jpg" localhost/uploader.php
```

PUT SURVEYS REQUEST BODY

Submit Transactions to Survey

Submit Transactions data to Social Survey application for surveying clients

Resource URL

https://api.socialsurvey.me/v2/surveys

*ensure https is used due to secure connection

Method

PUT

PUT-Surveys Response Code

{

"transactionInfo": {

"transactionRef": string,

"transactionDate": Date/Time "yyyy-mm-dd hh:mm:ss",

"transactionCity": string,

"transactionState": string,

"transactionType: string,

"customer1FirstName": string,

"customer1LastName": string,

"customer1Email": string,

```
        “customer2FirstName”: string,
        “customer2LastName”: string,
        “customer2Email”: string,
        “buyerAgentFirstName”: string,
        “buyerAgentLastName”: string
        “buyerAgentEmail”: string,
        “sellerAgentFirstName”: string
        “sellerAgentLastName”: string,
        “sellerAgentEmail: string
},
        “serviceProviderInfo”: {
        “serviceProviderName”: string,
        “serviceProviderEmail”: string
}
}
```

PUT SURVEYS REQUEST BODY TABLE

Property Name	Value	Description
transactionInfo	nested object	The transaction data to be processed into a survey
transactionInfo.transactionRef	String (Mandatory)	The ID from your system that enables association to its original transaction (e.g. LoanNumber, InvoiceNumber, etc.)
transactionInfo.transactionDate	Date/Time (Mandatory) yyyy-mm-dd hh:mm:ss	The date and time the transaction occurred; typically associated with the closed or funding date of a transaction
transactionInfo.transactionCity	String (Optional)	The city where the transaction occurred
transactionInfo.transactionState	String (Optional)	The state where the transaction occurred
transactionInfo.transactionType	String (Optional)	The type of transaction provided (e.g. Purchase, Refinance, Rental, etc.)
transactionInfo.customer1FirstName	String (Mandatory)	The first name of the primary customer to be surveyed
transactionInfo.customer1LastName	String (Mandatory)	The last name of the primary customer to be surveyed
transactionInfo.customer1Email	String (Mandatory)	The email address of the primary customer to be surveyed
transactionInfo.customer2FirstName	String (Optional/mandatory)	The first name of the secondary customer (co-borrower) to be surveyed. NOTE: This field is mandatory if there is an email in “transactionInfo.customer2Email”
transactionInfo.customer2LastName	String (Optional)	The last name of the secondary customer (co-borrower) to be surveyed

transactionInfo.customer2Email	String (Optional)	The email address of the secondary customer (co-borrower) to be surveyed
transactionInfo.buyerAgentFirstName	String (Optional/mandatory)	The first name of the Buyer's Agent to be Surveyed.
		NOTE: This field is mandatory if there is an email in "transactionInfo. buyerAgentEmail"
transactionInfo.buyerAgentLastName	String (Optional)	The last name of the Buyer's Agent to be Surveyed
transactionInfo.buyerAgentEmail	String (Optional)	The email of the Buyer's Agent to be Surveyed
transactionInfo.sellerAgentFirstName	String (Optional/mandatory)	The first name of the Seller's Agent to be Surveyed NOTE: This field is mandatory if there is an email in "transactionInfo. sellerAgentEmail"
transactionInfo.sellerAgentLastName	String (Optional)	The last name of the Seller's Agent to be Surveyed
transactionInfo.sellerAgentEmail	String (Optional)	The email of the Buyer's Agent to be Surveyed
serviceProviderInfo	nested object	The object that contains the servicer information
serviceProviderInfo.serviceProviderName	String (Mandaotry)	The name of the person who provided service for the transaction/person
serviceProviderInfo.serviceProviderEmail	String (Mandatory)	The email address of the person who provided service for the transaction/person

PUT-Surveys Response Code

```
{
    “transactionInfo”: {
    “transactionRef”: string,
    “transactionDate”: Date/Time “yyyy-mm-dd hh:mm:ss”,
    “transactionCity”: string,
    “transactionState”: string,
    “transactionType: string,
        “customer1FirstName”: string,
        “customer1LastName”: string,
        “customer1Email”: string,
        “customer2FirstName”: string,
        “customer2LastName”: string,
        “customer2Email”: string,
            “buyerAgentFirstName”: string,
            “buyerAgentLastName”: string
            “buyerAgentEmail”: string,
            “sellerAgentFirstName”: string
            “sellerAgentLastName”: string,
            “sellerAgentEmail: string
            },
```

```
                    "serviceProviderInfo": {
                    "serviceProviderName": string,
                    "serviceProviderEmail": string
}
}
```

GET SURVEY REQUESTS TABLE

Property Name	Value	Description
msg	nested object	The server message, which includes the message and the server
msg.message	string	The server response message
msg.code	integer	The server response code
data	nested object	The object, which contains the surveyId object
data.surveyId	nested object	The object which contains the key value pairs of "customer1Email":"surveyId" and "customer2Email":"surveyId"
data.surveyId.customer1Email	long	The surveyId associated with customer1Email
data.surveyId.customer2Email	long	The surveyId associated with customer2Email

GET SURVEY REQUESTS FROM SOCIALSURVEY

GET requests include all required data in the URL. Forms in HTML can use either method by specifying method="POST" or method="GET" (default) in the <form> element.

Resource URL

https://API.socialsurvey.me/v2/surveys

HTTP Method

GET

GET SURVEYS REQUEST BODY

Do not supply a request body with this method
Provide parameters in URL (see parameters below)

GET-Surveys Response Code


```
{
      “msg”: {
            “message”: string,
            “code”: integer
},
      “data”: {
            “surveys”: [
{
            “surveyId”: long,
            Buyersroad, Inc. 8
            “reviewId”: string,
            “transactionInfo”: {
            “transactionRef”: string,
            “surveySentDateTime”: Date/Time (UTC Format),
            “transactionDateTime”: Date/Time (UTC Format),
            “transactionCity”: string,
            “transactionState”: string,
            “transactionType”: string,
            “customerFirstName”: string,
            “customerLastName”: string,
            “customerEmail”: string,
```

```
            “buyerAgentFirstName”: string,
            “buyerAgentLastName”: string
            “buyerAgentEmail”: string,
            “sellerAgentFirstName”: string
            “sellerAgentLastName”: string,
            “sellerAgentEmail: string
            },
            “serviceProviderInfo”: {
            “serviceProviderName”: string,
            “serviceProviderEmail”: string
},
“review”: {
      “source”: string,
            “reviewCompletedDateTime”: Date/Time (UTC Format),
            “reviewUpdatedDateTime”: Date/Time (UTC Format),
                  “rating”: string,
                  “summary”: string,
                  “description”: string,
                  “agreedToShare”: boolean,
                              “verifiedCustomer”: boolean,
                              “retakeSurvey”: boolean,
                              “surveyResponses”: [
```

```
{
    “question”: string,
        “type” string,
            “answer”: string
}
],
    “reportedAbusive”: boolean
},
        “reviewStatus”: string
}
]
}
}
```

SURVEYS RESPONSE TABLE

Property Name	Value	Description
msg	nested object	The server message, which includes the message and the server
msg.message	string	The server response message
msg.code	integer	The server response code
data	nested object	The object which contains surveys
data.surveys[]	list	The list of surveyData
data.surveys.surveyId	long	The ID associated with that survey. NOTE: Not available for Zillow reviews
data.surveys.reviewId	string	The unique ID associated with the review NOTE: reviewId is only available after the customer completes the survey
data.surveys.transactionInfo	nested object	The object which contains the transaction information

data.surveys.transactionInfo.transactionRef	string	An ID from your system, given in the initial survey request, that enables association to its original transaction (eg. LoanNumber, Invoice Number, etc.)
data.surveys.transactionInfo.surveySentDateTime	Date/Time (UTC Format)	The timestamp when the survey was sent
data.surveys.transactionInfo.transactionDateTime	Date/Time (UTC Format)	The timestamp of when the transaction occurred. Typically associated with the funding date or closed date of a transaction
data.surveys.transactionInfo.transactionCity	string	The city where the transaction occurred
data.surveys.transactionInfo.transactionState	string	The state where the transaction occurred
data.surveys.transactionInfo.transactionType	string	The type of transaction (e.g. Purchase, Refinance, Rental, etc.)
data.surveys.transactionInfo.customerFirstName	string	The customer's first name
data.surveys.transactionInfo.customerLastName	string	The customer's last name
data.surveys.transactionInfo.customerEmail	string	The customer's email address

data.surveys.serviceProviderInfo	nested object	The object which contains servicer information
data.surveys.serviceProviderinfo.serviceProviderName	string	The name of the person who provided the service for the transaction / person being surveyed
data.surveys.serviceProviderInfo.serviceProviderEmail	string	The email address of the person who provided the service for the transaction / person being surveyed
data.surveys.review	nested object	The object that contains the review data
data.surveys.review.source	string	The source of the review • agent • admin • encompass • dotloop • lonewolf • ftp • Zillow • 3rd party
data.surveys.review.reviewComopletedDateTime	Date/Time (UTC Format)	The timestamp of when the survey was completed
data.surveys.review.reviewUpdatedDateTime	Date/Time (UTC Format)	The timestamp of when the survey was updated (e.g. survey re-take)

data.surveys.review.rating	string	The overall rating for the survey response
data.surveys.review.summary	string	The summary of the transaction (e.g. “Completed transaction on July 2017 in San Francisco, California”)
data.surveys.review.description	string	The content of the review
data.surveys.review.agreedToShare	boolean	Whether the customer agreed to sharing on social media
data.surveys.review.verifiedCustomer	boolean	Whether the review has been verified against the originating transaction
data.surveys.review.retakeSurvey	boolean	Whether the review has been flagged as a re-take
data.surveys.review.surveyResponses	list	The survey responses
data.surveys.review.surveyResponses.question	string	The survey question
data.surveys.review.surveyResponses.type	string	The type of the response • Numeric • Text • Experience

data.surveys.review.surveyResponses.answer	string	The response provided by the customer
data.surveys.review.reportedAbusive	boolean	Whether the response has been marked as abusive
Data.surveys.reviewStatus	string	The state of the review (e.g. completed, incomplete)

GET – INCOMPLETE SURVEYS BODY

GET requests include all required data in the URL. Forms in HTML can use either method by specifying method="POST" or method="GET" (default) in the <form> element.

Resource URL

https://API.socialsurvey.me/v2/surveys

HTTP Method

GET

GET INCOMPLETE SURVEYS BODY

Do not supply a request body with this method
Provide parameters in URL (see parameters below)

GET –Incomplete Surveys Response Code


```
{
        “msg”: {
                “message”: string,
                “code”: integer
        }
        “surveys”: [
        {
                “surveyId”: long,
                “reviewId”: string,
                “transactionInfo”: {
                                        “transactionRef”: string,
                                        “surveySentDateTime”: Date/Time (UTC Format),
                                        “transactionDateTime”: Date/Time (UTC Format),
                                        “transactionCity”: string,
                                        “transactionState”: string,
                                        “transactionType”: string,
                                        “customerFirstName”: string,
                                        “customerLastName”: string,
                                        “customerEmail”: string,
        },
                “serviceProviderInfo”: {
                “serviceProviderName”: string,
                “serviceProviderEmail”: string
}
}
]
}
```

INCOMPLETE SURVEYS RESPONSE TABLE

Property Name	Value	Description
msg	nested object	The server message, which includes the message and the server
msg.message	string	The server response message
msg.code	integer	The server response code
data	nested object	The object which contains surveys
data.surveys[]	list	The list of survey data
data.surveys.surveyId	long	The ID associated with that survey.
		NOTE: Not available for Zillow reviews
data.surveys.reviewId	string	The unique ID associated with the review NOTE: reviewId is only available after the customer completes the survey
data.surveys.transactionInfo	nested object	The object which contains the transaction information

data.surveys.transactionInfo.transactionRef	string	An ID from your system, given in the initial survey request, that enables association to its original transaction (eg. LoanNumber, Invoice Number, etc.)
data.surveys.transactionInfo.surveySentDateTime	Date/Time (UTC Format)	The timestamp when the survey was sent
data.surveys.transactionInfo.transactionDateTime	Date/Time (UTC Format)	The timestamp of when the transaction occurred. Typically associated with the funding date or closed date of a transaction
data.surveys.transactionInfo.transactionCity	string	The city where the transaction occured
data.surveys.transactionInfo.transactionState	string	The state where the transaction occurred
data.surveys.transactionInfo.transactionType	string	The type of transaction (e.g. Purchase, Refinance, Rental, etc.)
data.surveys.transactionInfo.customerFirstName	string	The customer's first name
data.surveys.transactionInfo.customerLastName	string	The customer's last name
data.surveys.transactionInfo.customerEmail	string	The customer's email address
data.surveys.serviceProviderInfo	nested object	The object which contains servicer information

data.surveys.serviceProviderinfo.serviceProviderName	string	The name of the person who provided the service for the transaction / person being surveyed
data.surveys.serviceProviderInfo.serviceProviderEmail	string	The email address of the person who provided the service for the transaction / person being surveyed

PARAMETERS

Parameters are options you can pass with the endpoint (such as specifying the response format, or the amount returned) to influence the response. There are four types of parameters: header parameters, path parameters, query string parameters, and request body parameters.

The headers and parameters contain a wealth of information that can help you track down issues when you encounter them. HTTP Headers are an important part of the API request and response as they represent the meta-data associated with the API request and response.

Parameter Types

APIs have four types of parameters:

- **Header parameters**: Parameters included in the request header, usually related to authorization.
- **Path parameters:** Parameters within the path of the endpoint, before the query string (?). These are usually set off within curly braces.
- **Query string parameters**: Parameters in the query string of the endpoint, after the (?).
- **Request body parameters**: Parameters included in the request body. Usually submitted as JSON

Header Parameters

Header parameters are included in the request header. Usually, the header just includes authorization parameters that are common across all endpoints; as a result, the header parameters aren’t usually documented with each endpoint.

However, if your endpoint requires unique parameters to be passed in the header, you would document them in the parameter’s documentation within each endpoint.

Path Parameters

Path parameters are part of the endpoint itself and are not optional. For example, in the following endpoint, {user} and {surveyId} are required path parameters:

Path parameters are usually set off with curly braces, but some API doc styles precede the value with a colon or use a different syntax. When you document path parameters, indicate the default values, the allowed values, and other details.

With Path parameters, the order does matter. If the parameter is part of the actual endpoint (not added after the query string), you usually describe this value in the description of the endpoint itself.

Query Parameters

Query string parameters appear after a question mark (?) in the endpoint. The question mark followed by the parameters and their values is referred to as the "query string." In the query string, each parameter is listed one right after the other with an ampersand (&) separating them. The order of the query string parameters does not matter.

Request Body Parameters

Frequently, with POST requests (where you're creating something), you submit a JSON object in the request body. This is known as a request body parameter, and the format is usually JSON. This JSON object may be a lengthy list of key-value pairs with multiple levels of nesting.

For example, the endpoint may be something simple, such as /surveyId / (startSurveyID) But in the body of the request, you might include a JSON object with many key-value pairs. Documenting a JSON object is easy if the object is simple, with just a few key-value pairs at the same level. But if you have a JSON object with multiple objects inside objects, numerous levels of nesting, and lengthy conditional data, it can be tricky. And if the JSON object spans more than 100 lines, or 1,000, you'll need to think carefully about how you present the information.

Regardless of the parameter type, define the following with each parameter:

- Data type
- Max and min value

.

Parameter Data Types

These data types are the most common with APIs:

- **string**: An alphanumeric sequence of letters and/or numbers
- **integer**: A whole number — can be positive or negative
- **Boolean:** True or false value
- **object**: Key-value pairs in JSON format
- **array:** A list of values

Max and min values for Parameters

In addition to specifying the data type, the parameters should indicate the maximum, minimum, and allowed values. For example, if the weather API allows only longitude and latitude coordinates of specific countries, these limits should be described in the parameter's documentation.

Omitting information about max/min values or other unallowed values is a common pitfall in docs. Developers often don't realize all the "creative" ways users might use the APIs.

QUERY PARAMETERS

Query parameters are the most common type of parameters. They appear at the end of the request URL after a question mark (?), with different name=value pairs separated by ampersands (&). Query parameters can be required and optional. Query parameters can be primitive values, arrays and objects. API provides several ways to serialize objects and arrays in the query string.

GET – SURVEY PARAMETERS

GET requests include all required data in the URL. Forms in HTML can use either method by specifying method="POST" or method="GET" (default) in the <form> element.

Resource URL

https://API.socialsurvey.me/v2/surveys

HTTP Method

GET

GET – SURVEY PARAMETERS RESPONSE TABLE

Parameter Name	Value	Description
Optional Path Parameters		
surveyId	long	The surveyId of the survey of interest

Optional Query Parameters		
count	string	The number of returned surveys requested Default: 1000
start	string	The starting index Default:0
state	string	A survey response • “Unpleasant” • “OK” • “Great”
status (Depreciated) • Use the incompletesurveys endpoint	string	The status of the survey • “complete” • “incomplete”
startSurveyID	long	The starting surveyId to retrieve survey responses
startReviewDateTime	string	The starting DateTime, of when a survey was completed, to retrieve survey responses
startTransactionDateTime	string	The starting DateTime, of when a transaction was completed, to retrieve survey responses
user	string	The serviceProviderEmail of the transaction
IncludeManagedTeam	string	The filter that includes or excludes members that are managed by the specific user given in the User parameter

GET – INCOMPLETE PARAMETERS

GET requests include all required data in the URL. Forms in HTML can use either method by specifying method="POST" or method="GET" (default) in the <form> element.

Resource URL

https://API.socialsurvey.me/v2/surveys

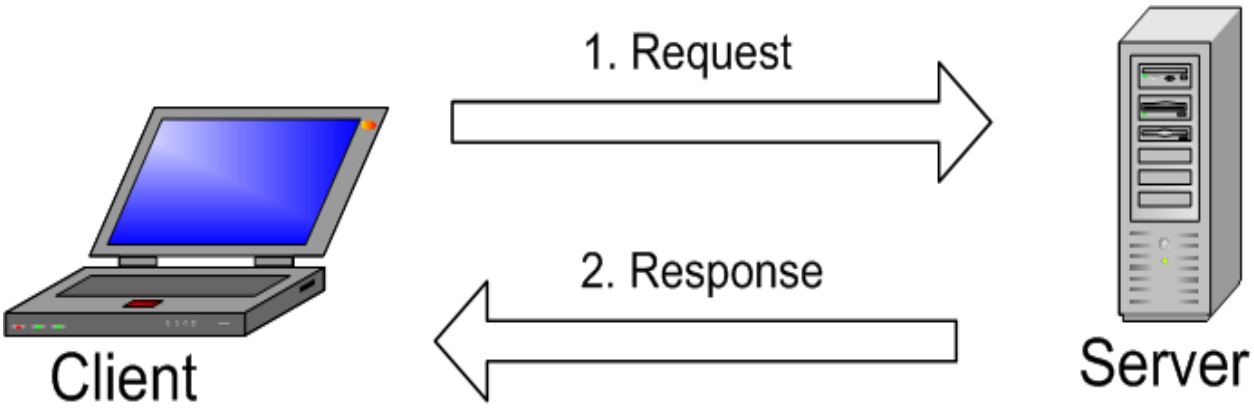
HTTP Method

GET

GET Incomplete Parameters Response Table

Parameter Name	Value	Description
Optional Query Parameters		
count	string	The number of returned surveys requested Default: 1000
start	string	The starting index Default:0
startSurveyID	string	The starting surveyId to retrieve survey responses
startTransactionDateTime	string	The starting DateTime, of when a transaction was completed, to retrieve survey responses
user	string	The serviceProviderEmail of the transaction
IncludeManagedTeam	string	The filter that includes or excludes members that are managed by the specific user given in the User parameter

USEFUL SERVER RESPONSES



GET - Surveys Successful

```
{
        “msg”: {
                “message”: “Request Successfully processed”,
                “code”: 200
        },
“data”: {
        “surveys”: [<SURVEY_DATA>]
}
}
```

GET Surveys Invalid Filter


```
{
        “msg”: {
                “message”: “Unsupported filter parameter: <QUERY
                PARAMETER>”,
                “code”: 400
}
}
```


GET Surveys/surveyId InvalidsurveyID


```
        “msg”: {
                “message”: “No record found for id”,
                “code”: 404
}
}
```


GET Invalid Parameter Passed


```
{
        “msg”: {
                “message”: “Passed parameter <PASSED PARAMETER> is
                invalid”,
                “code”: 400
}
}
```


PUT Surveys Server Error


```
{
        “msg”: {
“message”: “Could not process request due to server error”,
“code”: 500
}
}
```

PUT Service Provider Mismatch

```
{
        “msg”: {
                “message”: “Can not process the record. No service provider found
                with email address :
                “serviceProviderEmail”,
                “code”: 406
}
}
```

PUT Customer1Email is Null or Empty

```
{
        “msg”: {
                “message”: “Invalid input passed. Customer1Email can’t be null or
                empty”,
                “code”: 400
}
}
```

PUT Customer has Already Been Surveyed

```
{
        “msg”: {
                “message”: “Cannot process the record. A survey request for
                customer “customer1FirstName”
                has already been received”,
                “code”: 406
}
```

PUT Transaction is One Year or Older


```
{
        “msg”: {
                “message”: “Can not process the record. Request for customer
                “customer1FirstName” is older
                than 365 days.”,
                “code”: 406
}
}
```


PUT Invalid Date Format


```
{
        “msg”: {
                “message”: “Transaction Date with invalid format”,
                “code”: 400
}
}
```


PUT Customer Email Provided but No First name


```
{
        “msg”: {
                “message”: “Invalid input passed. customer2FirstName can't be null
                or empty”,
                “code”: 400
}
}
```

PUT Successful With 1 Customer


```
{
        “msg”: {
                “message”: “Survey successfully created”,
                “code”: 201
},
        “data”: {
                “surveyId”: {
                “customer1Email”: “surveyId”
}
}
}
```


PUT Successful With Customer


```
{
        “msg”: {
                “message”: “Survey successfully created”,
                “code”: 201
},
        “data”: {
                “surveyId”: {
                “customer1Email”: “surveyId”,
                “customer2Email”: “surveyId”
}
}
}
```

APPENDIX A

ERROR STATUS CODE DEFINITIONS PER - WORLD WIDE WEB CONSORTIUM

WEB STANDARDS ORGANIZATION

https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Each Status-Code is described below, including a description of which method(s) it can follow and any metainformation required in the response.

INFORMATIONAL 1XX

This class of status code indicates a provisional response, consisting only of the Status-Line and optional headers, and is terminated by an empty line. There are no required headers for this class of status code. Since HTTP/1.0 did not define any 1xx status codes, servers MUST NOT send a 1xx response to an HTTP/1.0 client except under experimental conditions.
A client MUST be prepared to accept one or more 1xx status responses prior to a regular response, even if the client does not expect a 100 (Continue) status message. Unexpected 1xx status responses MAY be ignored by a user agent.
Proxies MUST forward 1xx responses, unless the connection between the proxy and its client has been closed, or unless the proxy itself requested the generation of the 1xx response. (For example, if a
proxy adds a "Expect: 100-continue" field when it forwards a request, then it need not forward the corresponding 100 (Continue) response(s).)

100 Continue

The client SHOULD continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the server. The client SHOULD continue by sending the remainder of the request or, if the request has already been completed, ignore this response. The server MUST send a final response after the request has been completed. For detailed discussion of the use and handling of this status code.

101 Switching Protocols

The server understands and is willing to comply with the client's request, via the Upgrade message header field for a change in the application protocol being

used on this connection. The server will switch protocols to those defined by the response's Upgrade header field immediately after the empty line which terminates the 101 response.

The protocol SHOULD be switched only when it is advantageous to do so. For example, switching to a newer version of HTTP is advantageous over older versions, and switching to a real-time, synchronous protocol might be advantageous when delivering resources that use such features.

SUCCESSFUL 2XX

This class of status code indicates that the client's request was successfully received, understood, and accepted.

200 OK

The request has succeeded. The information returned with the response is dependent on the method used in the request, for example: **GET** an entity corresponding to the requested resource is sent in the response; **HEAD** the entity-header fields corresponding to the requested resource are sent in the response without any message-body; **POST** an entity describing or containing the result of the action; **TRACE** an entity containing the request message as received by the end server.

201 Created

The request has been fulfilled and resulted in a new resource being created. The newly created resource can be referenced by the URI(s) returned in the entity of the response, with the most specific URI for the resource given by a Location header field. The response SHOULD include an entity containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content-Type header field. The origin server MUST create the resource before returning the 201-status code. If the action cannot be carried out immediately, the server SHOULD respond with 202 (Accepted) response instead.

A 201 response MAY contain an ETag response header field indicating the current value of the entity tag for the requested variant just created, see

202 Accepted

The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place. There is no facility for re-sending a status code from an asynchronous operation such as this.

The 202 response is intentionally non-committal. Its purpose is to allow a server to accept a request for some other process (perhaps a batch-oriented process that is only run once per day) without requiring that the user agent's connection to the server persist until the process is completed. The entity returned with this response SHOULD include an indication of the request's current status and either a pointer to a status monitor or some estimate of when the user can expect the request to be fulfilled.

203 Non-Authoritative Information

The returned metainformation in the entity-header is not the definitive set as available from the origin server, but is gathered from a local or a third-party copy. The set presented MAY be a subset or superset of the original version. For example, including local annotation information about the resource might result in a superset of the metainformation known by the origin server. Use of this response code is not required and is only appropriate when the response would otherwise be 200 (OK).

204 No Content

The server has fulfilled the request but does not need to return an entity-body, and might want to return updated metainformation. The response MAY include new or updated metainformation in the form of entity-headers, which if present SHOULD be associated with the requested variant.

If the client is a user agent, it SHOULD NOT change its document view from that which caused the request to be sent. This response is primarily intended to allow input for actions to take place without causing a change to the user agent's active document view, although any new or updated metainformation SHOULD be applied to the document currently in the user agent's active view.

The 204 response MUST NOT include a message-body, and thus is always terminated by the first empty line after the header fields.

205 Reset Content

The server has fulfilled the request and the user agent SHOULD reset the document view which caused the request to be sent. This response is primarily intended to allow input for actions to take place via user input, followed by a clearing of the form in which the input is given so that the user can easily initiate another input action. The response MUST NOT include an entity.

206 Partial Content

The server has fulfilled the partial GET request for the resource. The request MUST have included a Range header field indicating the desired range, and MAY have included an If-Range header field to make the request conditional.

The response MUST include the following header fields:

- Either a Content-Range header field indicating
- the range included with this response, or a multipart/byteranges
- Content-Type including Content-Range fields for each part. If a
- Content-Length header field is present in the response, its
- value MUST match the actual number of OCTETs transmitted in the
- message-body.
- Date
- ETag and/or Content-Location, if the header would have been sent
- in a 200 response to the same request
- Expires, Cache-Control, and/or Vary, if the field-value might
- differ from that sent in any previous response for the same
- variant

If the 206 response is the result of an If-Range request that used a strong cache validator (see section 13.3.3), the response SHOULD NOT include other entity-headers. If the response is the result of an If-Range request that used a weak validator, the response MUST NOT include other entity-headers; this prevents inconsistencies between cached entity-bodies and updated headers. Otherwise, the response MUST include all of the entity-headers that would have been returned with a 200 (OK) response to the same request.

A cache MUST NOT combine a 206 response with other previously cached content if the ETag or Last-Modified headers do not match exactly, see

A cache that does not support the Range and Content-Range headers MUST NOT cache 206 (Partial) responses.

REDIRECTION 3XX

This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request. The action required MAY be carried out by the user agent without interaction with the user if and only if the method used in the second request is GET or HEAD. A client SHOULD detect infinite redirection loops, since such loops generate network traffic for each redirection.

300 Multiple Choices

The requested resource corresponds to any one of a set of representations, each with its own specific location, and agent- driven negotiation information (section 12) is being provided so that the user (or user agent) can select a preferred representation and redirect its request to that location.

Unless it was a HEAD request, the response SHOULD include an entity containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content- Type header field. Depending upon the format and the capabilities of

the user agent, selection of the most appropriate choice MAY be performed automatically. However, this specification does not define any standard for such automatic selection.

If the server has a preferred choice of representation, it SHOULD include the specific URI for that representation in the Location field; user agents MAY use the Location field value for automatic redirection. This response is cacheable unless indicated otherwise.

301 Moved Permanently

The requested resource has been assigned a new permanent URI and any future references to this resource SHOULD use one of the returned URIs. Clients with link editing capabilities ought to automatically re-link references to the Request-URI to one or more of the new references returned by the server, where possible. This response is cacheable unless indicated otherwise.

The new permanent URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).

If the 301 status code is received in response to a request other than GET or HEAD, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

302 Found

The requested resource resides temporarily under a different URI. Since the redirection might be altered on occasion, the client SHOULD continue to use the Request-URI for future requests. This response is only cacheable if indicated by a Cache-Control or Expires header field.

The temporary URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).

If the 302 status code is received in response to a request other than GET or HEAD, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

303 See Other

The response to the request can be found under a different URI and SHOULD be retrieved using a GET method on that resource. This method exists primarily to allow the output of a POST-activated script to redirect the user agent to a selected resource. The new URI is not a substitute reference for the originally requested resource. The 303 response MUST NOT be cached, but the response to the second (redirected) request might be cacheable.

The different URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).

304 Not Modified

If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server SHOULD respond with this status code. The 304 response MUST NOT contain a message-body, and thus is always terminated by the first empty line after the header fields.

If a clockless origin server obeys these rules, and proxies and clients add their own Date to any response received without one by section caches will operate correctly.

- ETag and/or Content-Location, if the header would have been sent
- in a 200 response to the same request
- Expires, Cache-Control, and/or Vary, if the field-value might
- differ from that sent in any previous response for the same
- variant

If the conditional GET used a strong cache validator (see section 13.3.3), the response SHOULD NOT include other entity-headers. Otherwise (i.e., the conditional GET used a weak validator), the response MUST NOT include other entity-headers; this prevents inconsistencies between cached entity-bodies and updated headers. If a 304 response indicates an entity not currently cached, then

the cache MUST disregard the response and repeat the request without the conditional.

If a cache uses a received 304 response to update a cache entry, the cache MUST update the entry to reflect any new field values given in the response.

305 Use Proxy

The requested resource MUST be accessed through the proxy given by the Location field. The Location field gives the URI of the proxy. The recipient is expected to repeat this single request via the proxy. 305 responses MUST only be generated by origin servers.

306 (Unused)

The 306 status code was used in a previous version of the specification, is no longer used, and the code is reserved.

307 Temporary Redirect

The requested resource resides temporarily under a different URI. Since the redirection MAY be altered on occasion, the client SHOULD continue to use the Request-URI for future requests. This response is only cacheable if indicated by a Cache-Control or Expires header field.

The temporary URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s) , since many pre-HTTP/1.1 user agents do not understand the 307 status. Therefore, the note SHOULD contain the information necessary for a user to repeat the original request on the new URI.

If the 307 status code is received in response to a request other than GET or HEAD, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

CLIENT ERROR 4XX

The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a HEAD request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. These status codes are applicable to any request method. User agents SHOULD display any included entity to the user.

If the client is sending data, a server implementation using TCP SHOULD be careful to ensure that the client acknowledges receipt of the packet(s) containing the response, before the server closes the input connection. If the client continues sending data to the server after the close, the server's TCP stack will send a reset packet to the client, which may erase the client's unacknowledged input buffers before they can be read and interpreted by the HTTP application.

400 Bad Request

The request could not be understood by the server due to malformed syntax. The client SHOULD NOT repeat the request without modifications.

401 Unauthorized

The request requires user authentication. The response MUST include a WWW-Authenticate header field containing a challenge applicable to the requested resource. The client MAY repeat the request with a suitable Authorization header field If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials. If the 401 response contains the same challenge as the prior response, and the user agent has already attempted authentication at least once, then the user SHOULD be presented the entity that was given in the response, since that entity might include relevant diagnostic information. HTTP access authentication is explained in "HTTP Authentication: Basic and Digest Access Authentication" [43].

402 Payment Required

This code is reserved for future use.

403 Forbidden

The server understood the request, but is refusing to fulfill it. Authorization will not help and the request SHOULD NOT be repeated. If the request method was not HEAD and the server wishes to make public why the request has not been fulfilled, it SHOULD describe the reason for the refusal in the entity. If the server does not wish to make this information available to the client, the status code 404 (Not Found) can be used instead.

404 Not Found

The server has not found anything matching the Request-URI. No indication is given of whether the condition is temporary or permanent. The 410 (Gone) status code SHOULD be used if the server knows, through some internally configurable mechanism, that an old resource is permanently unavailable and has no forwarding address. This status code is commonly used when the server does not wish to reveal exactly why the request has been refused, or when no other response is applicable.

405 Method Not Allowed

The method specified in the Request-Line is not allowed for the resource identified by the Request-URI. The response MUST include an Allow header containing a list of valid methods for the requested resource.

406 Not Acceptable

The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.

Unless it was a HEAD request, the response SHOULD include an entity containing a list of available entity characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most

appropriate choice MAY be performed automatically. However, this specification does not define any standard for such automatic selection.

If the response could be unacceptable, a user agent SHOULD temporarily stop receipt of more data and query the user for a decision on further actions.

407 Proxy Authentication Required

This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy. The proxy MUST return a Proxy-Authenticate header field containing a challenge applicable to the proxy for the requested resource. The client MAY repeat the request with a suitable Proxy-Authorization header field. HTTP access authentication is explained in "HTTP Authentication: Basic and Digest Access Authentication" [43].

408 Request Timeout

The client did not produce a request within the time that the server was prepared to wait. The client MAY repeat the request without modifications at any later time.

409 Conflict

The request could not be completed due to a conflict with the current state of the resource. This code is only allowed in situations where it is expected that the user might be able to resolve the conflict and resubmit the request. The response body SHOULD include enough

information for the user to recognize the source of the conflict. Ideally, the response entity would include enough information for the user or user agent to fix the problem; however, that might not be possible and is not required.

Conflicts are most likely to occur in response to a PUT request. For example, if versioning were being used and the entity being PUT included changes to a resource which conflict with those made by an earlier (third-party) request, the server might use the 409 response to indicate that it can't complete the request. In this case, the response entity would likely contain a list of the differences between the two versions in a format defined by the response Content-Type.

410 Gone

The requested resource is no longer available at the server and no forwarding address is known. This condition is expected to be considered permanent. Clients with link editing capabilities SHOULD delete references to the Request-URI after user approval. If the server does not know, or has no facility to determine, whether the condition is permanent, the status code 404 (Not Found) SHOULD be used instead. This response is cacheable unless indicated otherwise.

The 410 response is primarily intended to assist the task of web maintenance by notifying the recipient that the resource is intentionally unavailable and that the server owners desire that remote links to that resource be removed. Such an event is common for limited-time, promotional services and for resources belonging to individuals no longer working at the server's site. It is not necessary to mark all permanently unavailable resources as "gone" or to keep the mark for any length of time -- that is left to the discretion of the server owner.

411 Length Required

The server refuses to accept the request without a defined Content- Length. The client MAY repeat the request if it adds a valid Content-Length header field containing the length of the message-body in the request message.

412 Precondition Failed

The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server. This response code allows the client to place preconditions on the current resource metainformation (header field data) and thus prevent the requested method from being applied to a resource other than the one intended.

413 Request Entity Too Large

The server is refusing to process a request because the request entity is larger than the server is willing or able to process. The server MAY close the connection to prevent the client from continuing the request.

If the condition is temporary, the server SHOULD include a Retry- After header field to indicate that it is temporary and after what time the client MAY try again.

414 Request-URI Too Long

The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret. This rare condition is only likely to occur when a client has improperly converted a POST request to a GET request with long query information, when the client has descended into a URI "black hole" of redirection (e.g., a redirected URI prefix that points to a suffix of itself), or when the server is under attack by a client attempting to exploit security holes present in some servers using fixed-length buffers for reading or manipulating the Request-URI.

415 Unsupported Media Type

The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.

416 Requested Range Not Satisfiable

A server SHOULD return a response with this status code if a request included a Range request-header field, and none of the range-specifier values in this field overlap the current extent of the selected resource, and the request did not include an If-Range request-header field. (For byte-ranges, this means that the first- byte-pos of all of the byte-range-spec values were greater than the current length of the selected resource.)

When this status code is returned for a byte-range request, the response SHOULD include a Content-Range entity-header field specifying the current length of the selected resource. This response MUST NOT use the multipart/byteranges content- type.

417 Expectation Failed

The expectation given in an Expect request-header field (see section 14.20) could not be met by this server, or, if the server is a proxy, the server has unambiguous evidence that the request could not be met by the next-hop server.

SERVER ERROR 5XX

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request. Except when responding to a HEAD request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. User agents SHOULD display any included entity to the user. These response codes are applicable to any request method.

500 Internal Server Error

The server encountered an unexpected condition which prevented it from fulfilling the request.

501 Not Implemented

The server does not support the functionality required to fulfill the request. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any resource.

502 Bad Gateway

The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.

503 Service Unavailable

The server is currently unable to handle the request due to a temporary overloading or maintenance of the server. The implication is that this is a temporary condition which will be alleviated after some delay. If known, the length of the delay MAY be indicated in a Retry-After header. If no Retry-After is given, the client SHOULD handle the response as it would for a 500 response.

504 Gateway Timeout

The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server specified by the URI (e.g. HTTP, FTP, LDAP) or some other auxiliary server (e.g. DNS) it needed to access in attempting to complete the request.

505 HTTP Version Not Supported

The server does not support, or refuses to support, the HTTP protocol version that was used in the request message. The server is indicating that it is unable or unwilling to complete the request using the same major version as the client, as described in section 3.1, other than with this error message. The response SHOULD contain an entity describing why that version is not supported and what other protocols are supported by that server.