



ConsumerView API

Developer Guide



Contents

1	Getting started	5
1.1	What is ConsumerView?	5
1.2	What is the ConsumerView API?	5
1.3	Purpose of this document	7
1.4	Document audience and pre-requisites	7
1.5	Getting a ConsumerView API account	8
1.5.1	Neartime domain names	8
1.5.2	Talking with your Experian Account Manager	8
1.5.3	Setting up your account	9
2	Logging in and changing the password	10
2.1	Logging in	10
2.1.1	About the log in request	10
2.1.2	Log in response	11
2.2	Changing the password	11
2.2.1	About the change password request	11
2.2.2	About the change password response	13
2.2.3	Processing the change password response	13
3	Sending and processing single asset lookups	14
3.1	Types of lookup request	14
3.1.1	Single record lookup request	14
3.1.2	Batch record lookup request	14
3.2	Input data required in a lookup request	14
3.2.1	Types of data	14
3.2.2	Authorisation data	15
3.2.3	Search data	15
3.2.4	Flags data (optional)	17
3.3	Using a single record lookup request	19
3.3.1	About the request	19
3.3.2	About the response	21
3.4	Using a batch record lookup request	22
3.4.1	About the request	22
3.4.2	About the response	24
3.5	Processing the response data	26
3.5.1	Matched responses	26
3.5.2	About the Match flag	27

3.5.3	Postcode-dominant variables and infilling	27
3.5.4	Non-matched and error responses	27
4	Sending and processing multi-asset lookups	28
4.1	Introduction	28
4.2	Input data required in a multi-asset lookup request	28
4.3	Using a single record, multi-asset lookup request	28
4.3.1	About the request	28
4.3.2	About the response	31
4.4	Processing the response data	34
4.4.1	Matched responses	34
4.4.2	Non-matched and error responses	34
5	HTTP responses	36
5.1	Introduction	36
5.2	200 (OK) responses	37
5.2.1	200 – Fully-matched lookup (batch record)	37
5.2.2	200 – Fully-matched lookup (single record, multi-asset)	37
5.2.3	200 – Fully-non-matched lookup (batch record)	39
5.2.4	200 – Fully-non-matched lookup (single record, multi-asset)	39
5.2.5	200 – Matched lookup (single record, single asset)	40
5.2.6	200 – Non-matched lookup (single record, single asset)	40
5.2.7	200 – Non-matched lookup (0-byte)	41
5.2.8	200 – Partially-matched lookup (batch record)	41
5.2.9	200 – Partially-matched lookup (single record, multi-asset)	42
5.2.10	200 – Successful change password	43
5.2.11	200 – Successful log in	43
5.3	400 – (Bad Request) responses	43
5.3.1	400 – Bad parameters, invalid	43
5.3.2	400 – Invalid input	43
5.4	401 – (Unauthorized) responses	44
5.4.1	401 – Asset Not Found.	44
5.4.2	401 – Invalid token	44
5.4.3	401 – Unauthorized Asset.	45
5.4.4	401 – Unauthorized Client	45
5.4.5	401 – Unauthorized Client, null token	46
5.4.6	401 – Unsuccessful, bad token or new password invalid	46
5.5	404 – (Not Found) response	47
5.6	405 – (Method Not Allowed) response	47
5.7	417 – (Expectation Failed) responses	48

5.7.1	417 – Batch limit exceeded	48
5.7.2	417 – Incorrect JSON	48
5.7.3	417 – Missing parameters or invalid	48
5.8	429 – (Too Many Requests) response	49
5.9	500 – (Internal Server Error) responses	49
5.9.1	500 – Internal Server Error	49
5.9.2	500 – Problem with authentication	50
5.9.3	500 – Problem with login	50
5.10	503 – (Service Unavailable) responses	51
5.10.1	503 – Internal refresh in progress	51
5.10.2	503 – Server error	51
A	Dummy data	52

1 Getting started

1.1 What is ConsumerView?

Experian's ConsumerView is a database of:

- A wide range of demographic, socio-economic, and behavioural variables for UK individuals at person-, household- and postcode-levels. The data provides a single, definitive, and consistent view of the UK adult population and helps clients to target, segment, and enrich their view of UK consumers.
- A diverse range of different modelled person-level propensities. Propensities use a score, a percentile, or a Yes/No flag to identify the likelihood of an individual to display a particular characteristic. Propensities cover areas as diverse as employment type, channel marketing preference, smart phone ownership and pension fund valuation.

Clients use ConsumerView to tailor their marketing communications and improve customer experience. The data is actionable and directly linked to a large database of consented compliant direct marketing data (email and direct mail).

Most of the person-, household- and postcode-level variables in the Experian ConsumerView database are updated monthly, while others are updated quarterly or annually. Propensities are updated annually.

1.2 What is the ConsumerView API?

The ConsumerView API is a secure Web-based software interface which allows users to enrich their customer data with a wide range of demographic, socio-demographic, life stage and lifestyle variables. The user can send a request for a single consumer or more than one consumer (i.e. batch record request). The API is hosted on Experian's Neartime server which offers access to several datasets including ConsumerView. The speed of the response for a single consumer request is typically less than 200 ms i.e. near real-time.

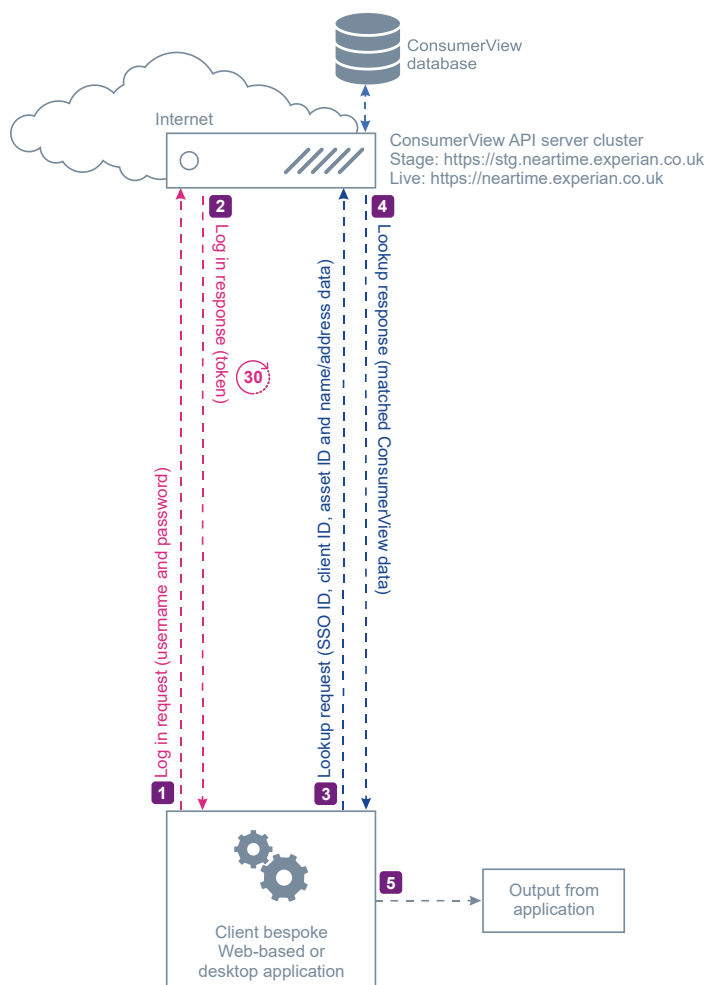
The API is primarily intended for clients who want to create their own bespoke Web-based or desktop applications and make use of the ConsumerView data. The API forms an integral component of these applications, allowing users to obtain near real-time ConsumerView data about selected UK consumers, e.g. the client's customer base.

These bespoke applications can extract, validate, and use the ConsumerView data for the client's own purposes. In its simplest form, the data could be used to create a consumer report to populate charts and graphs and determine the selection of representative images or other visual media. More sophisticated uses of the data could be to provide analysis of the returned data to give a decisioning capability, such as targeting mailshots, bespoke advertising, sales scripts, website personalisation and lead prioritisation; or to provide profiles of the client's customer base.

A description of all the ConsumerView variables and propensities that are currently available on the API are described in the ConsumerView API Variables and Propensities Reference Guide. However, the actual variables and propensities to which the client has access are subscription-dependent and based on the type of data they are interested in.

Clients who want to use the API, must first apply to obtain a Neartime Service account. Clients need to choose which ConsumerView variables and propensities that they want to subscribe and access, and how they want to use the API. The variables and propensities selected by the client are bundled into one or more assets, depending on the client's requirements, and given a unique ID known as an asset ID.

Refer to the process flow diagram below.



- 1
- 2 For a successful log in, the API sends an authenticated token which you can then use for lookup requests. Each token has a lifetime of up to 30 minutes, after which you must send a subsequent log in request.
- 3 Once logged in, you can query the API by supplying your SSO ID (Single Sign On ID) and client ID; the name and address details of a single consumer or more than one consumer you want to find; and the asset ID which identifies the licensed variables and propensities you want returned in a matching response.

- 4 The API returns the values of the variables and propensities defined in the asset that match the consumer(s) identified in the lookup request.
- 5 The values of the matching ConsumerView data are extracted from the response and used within your bespoke application to create the required output, e.g. a report, bespoke mailshot, advertising, or sales script.

You can also send a lookup request for more than one asset at a time. This is known as a multi-asset request and has the advantage of the application receiving all of the data in one request, rather than having to send individual single record requests for each asset. The assets can be associated with any dataset or API to which you subscribe, with the exception of the Catalist API. Multi-asset requests use a different endpoint to single and batch lookup requests. Multi-asset lookups are only supported for single record requests and not batch record requests.

1.3 Purpose of this document

This document provides technical information about:

- how to get an API account;
- the log in request and response, including examples;
- the single record lookup request and the response, including examples;
- the batch record lookup request and the response; including examples;
- the various HTTP responses that may be received from the API. For error responses this includes possible causes and their resolution.

Note: For information about all ConsumerView variables and propensities supported by the API, refer to the ConsumerView API Variables and Propensities Reference Guide. This includes a description of each variable and propensity and its possible values, as well as the name given to the variable or propensity in the response data, and a typical example response.

1.4 Document audience and pre-requisites

This document is intended to be used by developers wanting to use the API in their own bespoke Web-based or desktop applications.

There are many ways to create the HTTP requests needed to log in and query the API and to consume the responses. This can be from simple browser development tools and add-ins, tools and libraries such as cURL and JQuery, to development environments for programming languages such as C# and Java which provide libraries that include creating instances of an HTTP client or Web request.

It follows, therefore, that any API guide cannot be too prescriptive and can only give the 'bare bones' of information about the API. For this reason, the request examples used in this guide are in the Windows version of cURL. However, sufficient information is given about the construction of each HTTP request to allow you to build them using any preferred language or tool.

It is expected that you have a good understanding of:

- HTTP terminology and the general construct of POST method requests, including HTTP responses.
- The JSON (JavaScript Object Notation) format and syntax and the notion of key and value pairs.
- The cURL (Client URL) library, its installation, and its command line syntax. The examples in this guide uses the Windows version of cURL, but of course there are other versions including those for Linux and Python (urlLib2).

1.5 Getting a ConsumerView API account

1.5.1 Neartime domain names

You can ask to be registered to two Neartime Service domains:

- `stg.neartime.experian.co.uk` – this is the domain name for the Stage environment. You can use the Stage environment for testing purposes.
- `neartime.experian.co.uk` – this is domain name for the Live environment. When you have finished testing your application, you can switch to the URL for the Live environment.

1.5.2 Talking with your Experian Account Manager

If you want to use the API you must first talk with your Experian Account Manager.

The conversation will initially involve finding out:

- which API variables and propensities you want to use;
- the type of lookup request you want to use (single record, batch record or both).

You will also be asked about the bundling of the variables and propensities you intend to use into one or more assets. Each asset you have is given a unique ID.

Asset IDs let you make an API request for the variables and propensities in that asset. For example, you could put all variables and propensities that you have licensed or paid to use into a single asset. Alternatively, you could put the variables into two or more assets grouped by level (postcode-, household-, and person-) or by type (e.g. demographic, financial and segmentations etc). You could use a similar notion for any propensities you have licensed or paid to use or include them in assets that will contain variables.

Each variable or propensity can be assigned to one or more assets, i.e. the same variable or propensity could be in several assets depending upon your requirements.

Of course, if you have more than one asset you would need to send more than one lookup request (one per asset) to get all your data for a consumer or use a multi-asset request.

The number of assets you want created, and what they contain is largely decided by how you want to use the data. If required, you can also change the number of assets and what they contain at a later date.

1.5.3 Setting up your account

Registering the Neartime Service

Your Experian Account Manager will apply to the Experian Helpdesk and ask them to set up your account on the Experian Plexus Administration Site and give you access to the Neartime Service which facilitates the ConsumerView API.

The Plexus Administration Site is used for other services and applications such as Location Analyst, iCoder Online, Mosaic Audience and the Segmentation Portal. If you already use one of these services or sites, you will already have a username and password. In which case, the Helpdesk will only need to add the Neartime Service to your account and arrange for the access to the ConsumerView API (with your required variables, propensities, and assets) to be set up.

You can manage your password through any of the Plexus services or applications to which you have access, i.e. the same username and password is used for all services and sites to which you subscribe. You can also change your password using the API itself.

Any questions you have about your account should be sent to the Experian Helpdesk (EMSUKHelpdesk@experian.com).

Soon after the creation or update of your account you will receive an automated registration e-mail giving your username and password (if a new account has been created).

Getting access to the ConsumerView dataset

Giving you access to the Neartime Service is just part of the onboarding process. You will also need to be given access to the ConsumerView dataset on each of the two domains described in Subsection 1.5.1.

You will be notified by e-mail when this is complete and be given a 5-digit client ID, and depending on your requirements and contract, one or more asset IDs. You can then access the API.

Client IDs are used by the Neartime Service to match your username to a list of authorised usernames associated with that ID. For instance, you may have different users within your company that can access the API. The client ID is authorised to send a lookup request for an asset, but the requestor (username) is also logged. If different parts of your business need access to different variables, propensities, and assets to yourself, then please let us know, because it is likely that this will require a different client ID.

2 Logging in and changing the password

2.1 Logging in

2.1.1 About the log in request

To be able to query the ConsumerView database, you must first log in to the API to authenticate the session and receive a token. You must send your credentials, in JSON format, in the body of the request.

Currently, you are only allowed one token at any one time which is valid for 30 minutes. If you make any subsequent log in attempts, this invalidates the token of any active session you may have started and causes a new token to be created.

Note: You are recommended to put the code for the log in request into its own thread. Failure to do so would mean that any subsequent log in request being sent before the 30 minute expiry of a token already in use, would cause any search request which uses that token to fail and return a "401 – Invalid token" response.

HTTP summary details

The table below summarises the HTTP details of the log in request:

HTTP Element	Value
Request method:	POST
Standard header:	Content-Type:application/json
URL or endpoint:	https://[domain_name]/overture/login
Request body:	<pre>{ "userid": "USERID_VALUE", "password": "PASSWORD_VALUE" }</pre>

Note: See Subsection 1.5.1 for details of the domain names you can use for the [domain_name] placeholder.

Standard header

For the log in request, you must set the value of the standard Content-Type header to a MIME type of "application/json", i.e. Content-Type:application/json.

Request body key and value pairs

The key and value pairs in the log in request body (in JSON format) given above are defined as follows:

- "userid": "USERID_VALUE" – set the value to be the same as the username given in the registration e-mail e.g.
"userid": "client.username@client.com"
- "password": "PASSWORD_VALUE" – set the value to be the same as your current Plexus password, e.g. "password": "client_pAssw0rd". The password is case sensitive.

Example log in request

The following is a simple Windows cURL command line example which is used to demonstrate the content of a log in request:

```
curl -X POST -H "Content-Type:application/json"
https://[domain_name]/overture/login -d
{"\"userid\": \"client.username@client.com\", \"password\": \"
client_pAssw0rd\"}"
```

2.1.2 Log in response

Successful log in

If log in request is successful, the API sends a 200 (OK) response. The body of the response is a single JSON key and string value pair. The value of the token key is a hexadecimal string. An example response is shown below:

```
{"token": "713dbcaa-21f5-4124-a576-176805b7b326"}
```

Error response

If the log in request is unsuccessful, the API sends a different response to a 200 (OK).

The server may send a 429 (Too Many Requests) response to limit the loading on the network. If this occurs you will need to wait a specified amount of time dependent on the endpoint, before retrying. The time remaining (in milliseconds) is included in the response message.

See Section 5 for information about the other types of response which you may receive for a log in request, what they mean, and how to resolve the issue.

Processing the log in response

In a real-world Web-based or desktop application, if the log in request is successful, the token is likely to be programmatically extracted from the JSON response and then passed through, or made accessible, to a lookup request (see Section 3) or to a change password request (see Subsection 2.2 below).

It is also expected that you would write application code to handle any response which is not successful.

2.2 Changing the password

2.2.1 About the change password request

As well as changing your password via the user interface of any of the Plexus applications to which you may subscribe (e.g. iCoder Online), you can also change it using the API. You can use this method at any time while you are logged in and have a valid token.

The new password:

- must not be the same as any of the previous 12 passwords you have used;
- must be at least 8 characters in length;
- must contain a mix of uppercase and lowercase letters;
- must contain at least one numeric digit (0 to 9);
- must contain at least one non-alphanumeric character, such as #, *, ! or _;
- must not contain spaces;
- must not contain repeated characters, e.g. bbb or 111;
- must not include your forename or surname.

Note: It is important to recognise that any new password that you set will also be used by any other Plexus applications, sites and APIs to which you have access, e.g. iCoder Online, Location Analyst, and the ConsumerView API.

If the same credentials are used by other areas of your business for any other applications, then any change of password should be communicated to these areas.

HTTP summary details

The table below summarises the details of the change password request:

HTTP Element	Value
Request method:	POST
Standard header:	Content-Type:application/json
URL or endpoint:	https:// [domain_name] /overture/changepassword
Request body:	<pre>{ "token": "TOKEN_VALUE", "password": "CURRENT_PASSWORD_VALUE", "newpassword": "NEW_PASSWORD_VALUE" }</pre>

Note: See Subsection 1.5.1 for details of the domain names you can use for the [domain_name] placeholder.

Standard header

You must set the value of a standard Content-Type header to a MIME type of "application/json", i.e. Content-Type:application/json.

Request body key and value pairs

The key and value pairs in the request body given above are defined as follows:

- "token": "TOKEN_VALUE" – set the value to be the same as the token you received in the log in response, e.g. "token": "713dbcaa-21f5-4124-a576-176805b7b326".
- "password": "CURRENT_PASSWORD_VALUE" – set the value to be the same as your current Plexus password, e.g. "password": "client_pAssw0rd". The password is case sensitive.

- "newpassword":"NEW_PASSWORD_VALUE" – set the value to the new password you want to use, observing the password requirements given above, e.g. "newpassword":"client_pAssw0rd_New2day".

Example change password request

The following is a simple Windows cURL command line example which is used to demonstrate the content of a change password request:

```
curl -X POST -H "Content-Type:application/json"
https://[domain_name]/overture/changepassword -d
"{\"token\":\"3049f8bd-1934-4efb-8e2a-13311d4663e0\",
\"password\":\"client_pAssw0rd\", \"newpassword\":\"
client_pAssw0rd_New2day\"}"
```

2.2.2 About the change password response

Successful change password

If the change password request is successful, the API sends a 200 (OK) response. The body of the response contains a key and Boolean value pair as shown below:

```
{
  "success": true
}
```

The new password should now be used for any future log-in requests.

Error response

If a problem occurred processing the change password request the server will give a different HTTP response to 200 (OK).

The server may send a 429 (Too Many Requests) response to limit the loading on the network. If this occurs you will need to wait a specified amount of time dependent on the endpoint, before retrying. The time remaining (in milliseconds) is included in the response message.

See Section 5 for information on the other types of response which may be received for a request, what they mean, and how to resolve the issue.

2.2.3 Processing the change password response

If the change password request is successful, any new log in requests must use the new password.

It is also expected that you would write application code to handle any response which is not successful.

3 Sending and processing single asset lookups

3.1 Types of lookup request

3.1.1 Single record lookup request

If you want to perform a lookup of the ConsumerView database for a single consumer you can use the POST method, where you supply the lookup data in a JSON request body. You can also include optional flags in the request which you can use to optimise the matching process.

Of course, if you want to perform a lookup for more than one consumer, then you can either send single record lookup requests one after the other or use the batch record lookup request described below.

3.1.2 Batch record lookup request

As the name suggests, a batch record lookup request lets you perform a lookup of the ConsumerView database for more than one consumer at a time.

Note: The details of a maximum 5,000 consumers can be added to a batch record lookup request.

A batch record lookup request uses the POST method, where you supply the lookup data for each consumer within an array of the JSON request body. You can also include optional flags for each consumer within the lookup request which you can use to optimise the matching process.

3.2 Input data required in a lookup request

3.2.1 Types of data

The input data which you send in the lookup request can be divided into 3 discrete types:

- **Authorisation data** – every lookup request must contain your authorisation credentials. These give you the permission to use the API and the variables and propensities in the requested asset. See Subsection 3.2.2 for further information.
- **Search data** – every lookup request must include parameters or keys which identify the required postcode, household or person you want to use to search the ConsumerView database. For a batch record lookup request, an array is formed, with each element holding the key and value pairs defining the search data for a consumer. See Subsection 3.2.3 for further information.
- **Flags** – you can include an optional `flags` string array. You can use these flags to switch off a specific pre-processing cleaning task performed by the API. You can include the flags to optimise the matching process in certain scenarios. See Subsection 3.2.4 for further information.

3.2.2 Authorisation data

To be able to perform a lookup, you must include the following authorisation data in each lookup request:

- your username which was sent to you in the registration e-mail. N.B. When used in lookups, the username is known as a Single Sign On ID (or ssold). This is because the session was authenticated by the log in request and is valid for the lifetime of the token (30 minutes).
- the token which you received in response to the log in request. If the token is older than 30 minutes, the lookup request will fail, and you will have to send another log in request to get a new token.
- your client ID which was included in the registration e-mail. A client ID is a 5-digit number, e.g. 12345.
- the ID of the required asset you want to use and is included in your registration e-mail. An asset ID is a 6-character alphanumeric string e.g. 88883A.

3.2.3 Search data

Search data refers to the name and address keys that you want to use to search the ConsumerView database for a matching person, household or postcode. For a batch record lookup request, you must create a JSON array, with each element of the array containing the search data for a specific consumer.

Key required for a match at postcode-level only

The table below identifies the key which you must send to perform a search at postcode-level only.

Search Level	Key Name	Description
Postcode	postcode	The postcode for the household, e.g. XX5 3NG.

Keys required for a match at household-, and postcode-levels

The table below identifies the combination of keys which you must send to perform a search at household- and postcode-levels (but not person-level).

Search Level	Key Name	Description
Household	addressline	The first address line for a household comprising a building number and a street or road name, e.g. 62 GRAPEFRUITS LANE.
	postcode	The postcode for the household, e.g. XX5 3NG.
Household	subbuildingno	A sub building number e.g. 1 (if required). This may be used when the buildingname or buildingno is split into individual units such as flats or apartments.
	buildingno	The number to identify the building or house, e.g. 2, 4, 8 or 62.
	street	The name of the postal road or street, e.g. GRAPEFRUITS LANE.
	town	The name of the postal town or city, e.g. YODELVILLE.
	postcode	The postcode for the household, e.g. XX5 3NG.

Search Level	Key Name	Description
Household	subbuildingno	A sub building number e.g. 1 (if required). This may be used when the <code>buildingname</code> or <code>buildingno</code> is split into individual units such as flats or apartments.
	buildingname	A building or house name e.g. MANOR LODGE or VICTORIA HOUSE.
	street	The name of the postal road or street, e.g. GRAPEFRUITS LANE.
	town	The name of the postal town or city, e.g. YODELVILLE.
	postcode	The postcode for the household, e.g. XX5 3NG.
Household	subbuilding	The sub building name e.g. CARETAKERS FLAT (if required). This may be used when the <code>buildingname</code> or <code>buildingno</code> is split into individual units such as flats or apartments.
	buildingname	A building or house name e.g. MANOR LODGE or VICTORIA HOUSE.
	street	The name of the postal road or street, e.g. GRAPEFRUITS LANE.
	town	The name of the postal town or city, e.g. YODELVILLE.
	postcode	The postcode for the household, e.g. XX5 3NG.
Household	subbuilding	The sub building name e.g. Caretakers Flat (if required). This may be used when the <code>buildingname</code> or <code>buildingno</code> is split into individual units such as flats or apartments.
	buildingno	The number to identify the building or house, e.g. 2, 4, 8 or 62.
	street	The name of the postal road or street, e.g. GRAPEFRUITS LANE.
	town	The name of the postal town or city, e.g. YODELVILLE.
	postcode	The postcode for the household, e.g. XX5 3NG.
Household	email	The email address for a whole household, e.g. dunroaming@somedomain.com.
Household	key_family	The unique 18-digit match key for a family within the household, e.g. 99999999999999992.
Household	key_household	The unique 18-digit match key for a household, e.g. 99999999999999991.

With the exception of the two match keys (`key_family` and `key_household`), whenever possible, the API validates and compares the address you have supplied to the Royal Mail's Postcode Address File (PAF) records. This requires you to use the data values which are mapped to the most appropriate element of the different PAF structures.

Keys required for a match at person-, household-, and postcode-levels

The table below identifies the combination of keys which you are required to send to perform a search at person-, household-, and postcode-levels. With the exception of the `mobile`, `email`, `key_individual` and `key_person` options, you must also send the keys for a valid household (as identified in the table above).

Search Level	Key Name	Description
Person	email	The consented email address for the person, e.g. John.Doe@somedomain.com.
Person	mobile	The consented mobile number for the person, e.g. 07837123456.
Person	initial	The single initial letter of the person's forename, e.g. J.
	surname	The surname of the person, e.g. DOE.
	[addresskeys]	The household address for the person given by the combination of address keys given in the table above, e.g. addressline and postcode.
Person	forename	The forename of the person, e.g. JOHN.
	surname	The surname of the person, e.g. DOE.
	[addresskeys]	The household address for the person given by the combination of address keys given in the table above, e.g. addressline and postcode.
Person	key_person	The unique 10-digit match key for a single individual living at the address, e.g. 9909909992.
Person	key_individual	The unique 18-digit match key for an individual living at the address, e.g. 999999999999999993.
		If two or more individuals in the same household and same family share the same initial, then they will have the same individual key. To get a tighter match, the key_person key should be used instead.

3.2.4 Flags data (optional)

About the pre-processing cleaning tasks

On receipt of a lookup request, the API normally performs 3 pre-processing cleaning tasks on the search data before attempting to find a match. The name and address keys which are valid for cleaning are: `initial`, `forename`, `surname`, `addressline`, `buildingno`, `subbuildingno`, `buildingname`, `subbuilding`, `street`, `town` and `postcode`.

The 3 pre-processing cleaning tasks are:

- Capitalisation – the API capitalises all alphabetic characters in the values of all name and address keys.
- Stripping of characters – the API removes all non-alphanumeric and hyphen characters (-) from the values of all received name and address keys. However, hyphen characters (-) are not removed from the values of the `buildingno` and `subbuildingno` keys because these are most likely to contain a hyphen for number ranges.
- Thoroughfare “hydration” – the API replaces any abbreviation of the ending of a thoroughfare, e.g. RD or ST with the full word, i.e. ROAD or STREET for the `addressline` and `street` keys.

The table below gives the full list of thoroughfare abbreviations and the full words which are produced following hydration by the API.

Abbreviation	Full Word
AV	AVENUE
AVE	AVENUE
CL	CLOSE
CNT	CRESCENT
CRT	COURT
CT	COURT
DR	DRIVE
GDNS	GARDENS
GR	GROVE
GRV	GROVE
LN	LANE
PK	PARK
PL	PLACE
RD	ROAD
SQ	SQUARE
ST	STREET
TER	TERRACE
WK	WALK

The reasons for switching off pre-processing cleaning tasks

For some lookup requests, you might want to switch off a particular pre-processing cleaning task for the following reasons:

- When you have already done capitalisation within your application or on your side. This is the only reason why you would want to switch off capitalisation.
- When characters such as the hyphen (-) in the name and address (e.g. a hyphenated `forename` or `surname`) are needed to provide a match.
- When the address you have is non-PAF valid and because the abbreviation of the street ending such as DR and ST (instead of DRIVE and STREET) will result in a match. This is the only reason you would want to switch off thoroughfare hydration.

The format of the flags array

If you want to switch off one or more of the pre-processing cleaning tasks you must add an optional flags string array into the JSON request body. The general format of this key is:

```
"flags": [{"KEY_NAME": ["FLAG1_VALUE",
"FLAG2_VALUE", "FLAG3_VALUE"]}]
```

The value of "KEY_NAME" identifies the name of one of the name and address keys (e.g. "addressline" or "street") or "all" for all keys.

Note: Currently, if you include flags array in the request you must set "KEY_NAME" to "all" because individual key names are not supported.

The 3 elements inside the array represent the values of the flags ("FLAG1_VALUE", "FLAG2_VALUE" and "FLAG3_VALUE") which you can use to switch off one or more of the 3 pre-processing cleaning tasks. The value can either be an empty string ("") if you want that task to still be run, or a string representing the task you want to switch off for the request, i.e. "nostrip", "nocaps" or "nohydrate".

An example of the `flags` array which switches off all 3 pre-processing tasks and for all keys is shown below:

```
"flags": [{"all": ["nocaps", "nostrip", "nohydrate"]}]
```

Note: When the option of specifying individual name and address keys is supported (instead of just "all"), it will be possible for the flags array to contain more than one element, with each one specifying the flags for the specified key. For example:

```
"flags": [{"addressline": ["", "", "nohydrate"],
"surname": ["", "nostrip", ""]}]
```

The `flags` array is applied on a per record basis. For a batch record lookup request, a `flags` array can be included within each element of the `batch` key array.

3.3 Using a single record lookup request

3.3.1 About the request

You can use a POST request to perform a lookup of the ConsumerView database for a single consumer by sending the authorisation keys and one or more keys which are used in the search. You must send this data, in JSON format, in the body of the request.

You can also include an optional `flags` string array in the request body. You can use the flags to switch off individual pre-processing cleaning tasks that are run at the API before the search is performed. Refer to section 3.2.4 for further information.

HTTP summary details

The table below summarises the details of the request:

HTTP Element	Value
Request method:	POST
Standard header:	Content-Type:application/json
URL or endpoint:	https://[domain_name]/overture/lookup
Request body (with the optional "flags" array):	<pre>{ "ssoId": "SSOID_VALUE", "token": "TOKEN_VALUE", "clientId": "CLIENTID_VALUE", "assetId": "ASSETID_VALUE", "KEY_1_NAME": "KEY_1_VALUE", ... "KEY_1+n_NAME": "KEY_1+n_VALUE", "flags": [{"KEY_NAME": ["FLAG1_VALUE", "FLAG2_VALUE", "FLAG3_VALUE"]}]} }</pre>

Note: See Subsection 1.5.1 for details of the domain names you can use for the `[domain_name]` placeholder.

Standard header

You must set the value of a standard `Content-Type` header to a MIME type of `"application/json"`, i.e. `Content-Type:application/json`.

Request body key and value pairs

The key and value pairs in the request body given above are defined as follows:

- `"ssoId": "SSOID_VALUE"` – set the value to be the same as the username given in the registration e-mail and used at log in e.g. `"ssoId": "client.username@client.com"`.
- `"token": "TOKEN_VALUE"` – set the value to be the same as the token you received in the log in response, e.g. `"token": "713dbcaa-21f5-4124-a576-176805b7b326"`.
- `"clientId": "CLIENTID_VALUE"` – set the value to be the same as the client ID which you received in the registration email, e.g. `"clientId": "12345"`.
- `"assetId": "ASSETID_VALUE"` – set the value to be the same as one of the asset IDs which you received in the registration email, e.g. `"assetId": "88883A"`. The one you use is dependent on the variables and propensities you want returned in the response.
- `"KEY_1_NAME": "KEY_1_VALUE", ... "KEY_1+n_NAME": "KEY_1+n_VALUE"` – this represents the name and value pairs of one or more keys used in the lookup of the ConsumerView database. The number, names and values of these keys are dependent on the search level and what you must specify to obtain a match at this level. For example,
`"forename": "JOHN", "surname": "DOE", "addressline": "10 SWINEGATE", "postcode": "XX319RR"`.
Refer to section 3.2.3 for a full list of input parameters and their description.
- `"flags": [{"KEY_NAME": ["FLAG1_VALUE", "FLAG2_VALUE", "FLAG3_VALUE"]}]` – this represents the key name and value of 3 flags for an optional flags array which you can use to stop one or more of the pre-processing cleaning tasks from running for this request. Refer to section 3.2.4 for more information.

Example request

The following is a simple Windows cURL command line example which is used to demonstrate the content of a single record lookup request for a person-level search of the ConsumerView database:

```
curl -X POST -H "Content-Type:application/json"
https://[domain_name]/overture/lookup -d
{"ssoId\":\"client.username@client.com\",
 \"token\":\"3049f8bd-1934-4efb-8e2a-13311d4663e0\",
 \"clientId\":\"10002\", \"assetId\":\"88889A\",
 \"initial\":\"G\", \"surname\":\"DIXON\", \"addressline\":\"86
 WALDORF STREET\", \"postcode\":\"XX319PR\"}
```

3.3.2 About the response

Matched lookup

If there has been a successful match with the search data you have supplied, the API sends a 200 (OK) response. The body of the response contains key and string value pairs for each variable and propensity that has been defined in the asset you have used. The response also contains a *Match* flag. An example of this is shown below:

```
{
  "h_age_fine":"07",
  "p_financial_advice_band":"1",
  "p_gender":"0",
  "p_length_of_residency":"11",
  "p_0212_perc_newspaper_main_sun_v1b":"77",
  "Match":"P"
}
```

Note: The response data does not include any variables or propensities which are higher than the level given by the *Match* flag and which cannot be infilled. For example, the person-level variable *p_mosaic_uk_6_shopper_type* will not be returned if the *Match* flag is *PC* or *H*, because it cannot be infilled (i.e. it has no postcode-dominant value).

Variables which are in the response data and which are higher than the level given by the *Match* flag, will have been infilled by a postcode-dominant value.

Non-matched lookup

If the lookup request results in a non-match, a 200 (OK) response is given but with empty curly braces, i.e. {}.

Error response

If a problem occurred processing the request the server will give a different HTTP response to a 200 (OK).

The server may send a 429 (Too Many Requests) response to limit the loading on the network. If this occurs you will need to wait a specified amount of time dependent on the endpoint, before retrying. The time remaining (in milliseconds) is included in the response message.

See Section 5 for information on the other types of response which may be received for a request, what they mean, and how to resolve the issue.

3.4 Using a batch record lookup request

3.4.1 About the request

You can use a POST request to perform a lookup of the ConsumerView database for more than one consumer. You must send the authorisation keys as you would with a single record lookup request. The difference is that the search data for each consumer is encapsulated into the elements of a `batch` key array: with one element of the array holding the lookup data for a single consumer.

Note: The details of a maximum 5,000 consumers can be added to a batch consumer lookup request.

You can also include an optional `flags` string array for each consumer in the `batch` key array. You can use the flags to switch off individual pre-processing cleaning tasks that are run at the API before the search is performed. Refer to section 3.2.4 for further information.

HTTP summary details

The table below summarises the details of the request:

HTTP Element	Value
Request method:	POST
Standard header:	Content-Type:application/json
URL or endpoint:	https://[domain_name]/overture/batch
Request body (with the optional “flags” array):	<pre>{ "ssoId":"SSOID_VALUE", "token":"TOKEN_VALUE", "clientId":"CLIENTID_VALUE", "assetId":"ASSETID_VALUE", "batch":[{"KEY_1_NAME":"KEY_1_VALUE", ... "KEY_1+n_NAME":"KEY_1+n_VALUE", "flags":[{"KEY_NAME":["FLAG1_VALUE", "FLAG2_VALUE","FLAG3_VALUE"]}]}], ... {"KEY_1_NAME":"KEY_1_VALUE", ... "KEY_1+n_NAME":"KEY_1+n_VALUE", "flags":[{"KEY_NAME":["FLAG1_VALUE", "FLAG2_VALUE","FLAG3_VALUE"]}]}] }</pre>

Note: See Subsection 1.5.1 for details of the domain names you can use for the `[domain_name]` placeholder.

Standard header

You must set the value of a standard `Content-Type` header to a MIME type of `"application/json"`, i.e. `Content-Type:application/json`.

Request body key and value pairs

The key and value pairs in the request body given above are defined as follows:

- `"ssoId": "SSOID_VALUE"` – set the value to be the same as the username given in the registration e-mail and used at log in e.g. `"ssoId": "client.username@client.com"`.
- `"token": "TOKEN_VALUE"` – set the value to be the same as the token you received in the log in response, e.g. `"token": "713dbcaa-21f5-4124-a576-176805b7b326"`.
- `"clientId": "CLIENTID_VALUE"` – set the value to be the same as the client ID which you received in the registration email, e.g. `"clientId": "12345"`.
- `"assetId": "ASSETID_VALUE"` – set the value to be the same as one of the asset IDs which you received in the registration email, e.g. `"assetId": "88883A"`. The one you use is dependent on the variables and propensities you want returned in the response.
- `"batch": [{"Consumer1_KVPs"}, ..., {"Consumer1+n_KVPs"}]` – this represents the key and value pairs (KVPs) for each consumer in the `batch` array. The number, names and values of these keys are dependent on the search level for each consumer and what you must specify to obtain a match at this level.

For example, for a two element (2 consumer array) the `batch` key array may look like:

```
batch: [{"forename": "JOHN", "surname": "DOE", "addressline": "24 SPITTLEGATE", "postcode": "XX319RR"}, {"forename": "JUSTIN", "surname": "DAVIS", "addressline": "23 HAYCOCK STREET", "postcode": "XX138EH"}]
```

Refer to section 3.2.3 for a full list of input parameters and their description.

Each element in the `batch` array can also include an optional `flags` array in the general form of `"flags": [{"KEY_NAME": ["FLAG1_VALUE", "FLAG2_VALUE", "FLAG3_VALUE"]}]`. This represents the key name and value of 3 flags which you can use to stop one or more of the pre-processing cleaning tasks from running for that consumer in the request. Refer to section 3.2.4 for more information.

Example request

The following is a simple Windows cURL command line example which is used to demonstrate the content of a batch record lookup request for a person-level search of the ConsumerView database. For the sake of brevity, the batch array only includes the details of two consumers and does not include any `flags` arrays:

```
curl -X POST -H "Content-Type:application/json"
https://[domain_name]/overture/batch -d
{"ssoId":"client.username@client.com",
"token":"3049f8bd-1934-4efb-8e2a-13311d4663e0",
"clientId":"10002","assetId":"88889A",
"batch":[{"initial":"J","surname":"JOHNSON",
"addressline":"24 INNER STREET","postcode":
"YY249QZ"},{"initial":"K","surname":"HOWELLS",
"addressline":"87 LONGACRE","postcode":"QQ661ZZ"}]}
```

3.4.2 About the response

Fully-matched lookup

If the lookup request results in a match for all consumers in the request, the API sends a 200 (OK) response. The body of the response contains a JSON array.

Each element of the array represents the results for the corresponding consumer in the request, i.e. element 0 in the response array relates to the consumer defined in element 0 in the lookup request array.

Each element in the array contains key and string value pairs for each variable and propensity that has been defined in the asset you have used. Each element also contains a `Match` flag.

An example of this is shown below. For the sake of brevity, the example is limited to a few variables and only two elements (or two consumers):

```
[{
  "h_age_fine":"07",
  "p_financial_advice_band":"1",
  "p_gender":"0",
  "p_length_of_residency":"11",
  "p_0212_perc_newspaper_main_sun_v1b":"77",
  "Match":"P"
},
{
  "h_age_fine":"12",
  "p_financial_advice_band":"3",
  "p_gender":"0",
  "p_length_of_residency":"4",
  "p_0212_perc_newspaper_main_sun_v1b":"20",
  "Match":"P"
}]
```

Note: The response data does not include any variables or propensities which are higher than the level given by the `Match` flag and which cannot be infilled. For example, the person-level variable `p_mosaic_uk_6_shopper_type` will not be returned if the `Match` flag is `PC` or `H`, because it cannot be infilled (i.e. it has no postcode-dominant value).

Variables which are in the response data and which are higher than the level given by the `Match` flag, will have been infilled by a postcode-dominant value.

Partially-matched lookup

If the lookup request results in a non-match for one or more consumers (but not *all*) in the request, a 200 (OK) response is given but empty curly braces, i.e. `{ }` are returned for that consumer in the response array. For example, the response below shows that for the first consumer in the request (element 0 in the array) matched at person-level, while for the second consumer (element 1 in the array) no match was found:

```
[ {
  "h_age_fine": "07",
  "p_financial_advice_band": "1",
  "p_gender": "0",
  "p_length_of_residency": "11",
  "p_0212_perc_newspaper_main_sun_v1b": "77",
  "Match": "P"
},
{ } ]
```

Note: The response data does not include any variables or propensities which are higher than the level given by the `Match` flag and which cannot be infilled. For example, the person-level variable `p_mosaic_uk_6_shopper_type` will not be returned if the `Match` flag is `PC` or `H`, because it cannot be infilled (i.e. it has no postcode-dominant value).

Variables which are in the response data and which are higher than the level given by the `Match` flag, will have been infilled by a postcode-dominant value.

Fully-non-matched lookup

If a batch consumer lookup request is made where no consumers were matched, a 200 (OK) response is given, but with empty curly braces for each element in the array. For example, for a 2-element array: `[ { }, { } ]`

Error response

If a problem occurred processing the request the server will give a different HTTP response to a 200 (OK).

The server may send a 429 (Too Many Requests) response to limit the loading on the network. If this occurs you will need to wait a specified amount of time dependent on the endpoint, before retrying. The time remaining (in milliseconds) is included in the response message.

See Section 5 for information on the other types of response which may be received for a request, what they mean, and how to resolve the issue.

3.5 Processing the response data

3.5.1 Matched responses

If the lookup request results in a successful match, then it is likely that you will write code to consume the response data. In simple terms, this means your code will extract and validate the values for each variable and propensity in the response, and then perhaps conditionally use the variables and propensities in some way according to their value. For example, to populate a consumer report and/or to be used by downstream decisioning software.

Rules for a successful match

The primary rule for a successful match is that the search data (e.g. name and address) is accurate and spelled and formatted correctly so that it is the same as the ConsumerView database.

Assuming that the primary rule has been fulfilled, a successful match can only be given if the result of the search can be resolved to a single person, household, or postcode. For example:

- If two or more people share the same initial and surname and live at the same household address, then a search using the initial and surname and identifiable household details will result in a match at household-level and not person-level. If the people have different forenames, then a search using forename and surname and identifiable household details will result in a match at person-level.
- If two or more people share the same forename and surname (e.g. a father and son) and live at the same address, then a search using forename and surname and identifiable household details will result in a match at household-level and not person-level.
- If two or more houses exist with the same postcode, then a search which does not uniquely identify the house (e.g. a building number) but does provide other address details will result in a match at postcode level, provided there is only one postcode for the thoroughfare. If there are two or more postcodes for the thoroughfare (e.g. for different sides of the road) then the search will result in no matches, since neither the household nor the postcode can be resolved.

Variables and propensities included in the response data

The names of all variables and propensities include a "p_", "h_" or "pc_" prefix. To be included in the response data, those variables and propensities with a:

- "p_" prefix need a match which is resolved to a single person or are infilled by a postcode-dominant value (if one exists).
- "h_" prefix need a match which is resolved to a single person or household or are infilled by a postcode-dominant value (if one exists).

- "pc_" prefix need a match which is resolved to a single person, household or postcode.

The response data only contains variables and propensities where matching has been found or has been infilled by a postcode-dominant value. The data does not include any variables or propensities which are higher than the level given by the `Match` flag and which cannot be infilled.

Further information about variables and propensities

The ConsumerView API Variables and Propensities Reference Guide describes each variable and propensity that is available via the API. It identifies the name that is given in the JSON response; and the possible values.

3.5.2 About the Match flag

A successful match response for a consumer also includes a `Match` flag. For person-, household- and postcode-level and propensities the `Match` flag identifies the highest matching search which was found and has 3 possible values (from lowest to highest):

- `"Match": "PC"` – data matched at postcode-level.
- `"Match": "H"` – data matched at household-level.
- `"Match": "P"` – data matched at person-level.

3.5.3 Postcode-dominant variables and infilling

Most person- and household-level variables also have a postcode-dominant version. In the event of a non-match of the variable at that level, the value of the postcode-dominant version is used to infill the variable.

Where a variable is available at both person- and household level, postcode-dominant variables may also be available for both (for example, `p_affluence_v2` and `h_affluence_v2`). In this instance it is quite possible the postcode-dominant values are different for each (for example, `"p_affluence_v2": "07"` and `"h_affluence_v2": "08"`).

Consumer propensities are at person-level only and do not have a postcode-dominant version, so cannot be infilled. Therefore, in the event of not matching at person-level, any propensities to which you subscribe are not returned in the response.

Infilling is automatic and cannot be switched off or on via the API. However, you can write code to check the value of the `Match` flag to determine if infilling has been applied. For example, if the `Match` flag is set to postcode-dominant variable in the response to a search, then any person-level variables in that asset will have been infilled by a postcode-dominant value (if available). Depending on your requirements, you can then check the values of any infilled variables and decide whether you want to use that value or overwrite them with a different value, e.g. not available.

3.5.4 Non-matched and error responses

It is also expected that you would write application code to handle any response where no match is found (`{ }` given) for a consumer, or which indicates that there was a problem (see Section 5 for details).

4 Sending and processing multi-asset lookups

4.1 Introduction

You can send a lookup request for more than one asset at a time. This is known as a multi-asset request and has the advantage of your application receiving all of the data in one request, rather than having to send individual single record requests for each asset.

The asset IDs that you send in the multi-asset request can be for any dataset to which you have subscribed, except one for the Catalist API. If you have licensed two or more assets for a particular dataset (e.g. ConsumerView) then you can specify two or more in the multi-asset request. Similarly, a multi-asset request can include assets for disparate datasets, such as ConsumerView, Suppressions and WorldView.

All the relevant search fields that would be required for a single record lookup request for each dataset should be included in a multi-asset asset request. Some datasets share the same fields, e.g. the name and address fields. Whereas for other datasets, some search fields are unique to that dataset; for example, the `latitude`, `longitude` and `country` fields are available to perform lookups of the WorldView and EMEA Mosaic datasets only.

You are limited to sending a single record, multi-asset request, i.e. multi-asset batch record requests are not supported.

Multi-asset requests use the POST method, but use a different endpoint to their single record, single asset counterpart.

4.2 Input data required in a multi-asset lookup request

The types of data you provide in a multi-asset lookup request are exactly the same as what you would provide in a single record, single asset request. For example, for the ConsumerView API, refer to section 3.2.

Other Neartime APIs may support other keys or fields, not relevant or available to this API. Therefore, if you are making a request which includes an asset for one of these APIs you should include those keys or fields needed to provide a successful match and response.

Note: All Neartime APIs ignore any keys or fields that it does not recognise.

4.3 Using a single record, multi-asset lookup request

4.3.1 About the request

You can use a POST request to perform a lookup of multiple assets for a single record by sending the authorisation keys (including the required asset IDs) and one or more keys which are used in the search. You must send this data, in JSON format, in the body of the request.

For some datasets, e.g. ConsumerView, you can also include an optional `flags` string array in the request body. You can use the flags to switch off individual pre-processing cleaning tasks that are run at the API before the search is performed. Refer to section 3.2.4 for further information about the `flags` array for ConsumerView.

HTTP summary details

The table below summarises the details of the request:

HTTP Element	Value
Request method:	POST
Standard header:	Content-Type:application/json
URL or endpoint:	https://[domain_name]/overture/MultiAssetLookup
Request body (with the optional “flags” array):	<pre>{ "ssoId": "SSOID_VALUE", "token": "TOKEN_VALUE", "clientId": "CLIENTID_VALUE", "assetId": ["ASSETID1_VALUE", ... "ASSETID_1+n_VALUE"], "KEY_1_NAME": "KEY_1_VALUE", ... "KEY_1+n_NAME": "KEY_1+n_VALUE", "flags": [{"KEY_NAME": ["FLAG1_VALUE", "FLAG2_VALUE", "FLAG3_VALUE"]}]} }</pre>

Note: See Subsection 1.5.1 for details of the domain names you can use for the `[domain_name]` placeholder.

Standard header

You must set the value of a standard `Content-Type` header to a MIME type of “application/json”, i.e. `Content-Type:application/json`.

Request body key and value pairs

The key and value pairs in the request body given above are defined as follows:

- `"ssoId": "SSOID_VALUE"` – set the value to be the same as the username given in the registration e-mail and used at log in e.g. `"ssoId": "client.username@client.com"`.
- `"token": "TOKEN_VALUE"` – set the value to be the same as the token you received in the log in response, e.g. `"token": "713dbcaa-21f5-4124-a576-176805b7b326"`.
- `"clientId": "CLIENTID_VALUE"` – set the value to be the same as the client ID which you received in the registration email, e.g. `"clientId": "12345"`.
- `"assetId": "ASSETID_VALUE"` – this is a string array of asset IDs you want to use in your request, e.g. `"assetId": ["88883A", "88883B", "88883C"]`.

- `"KEY_1_NAME": "KEY_1_VALUE", ... "KEY_1+n_NAME": "KEY_1+n_VALUE"`
– this represents the name and value pairs of one or more keys used in the lookup. The number, names and values of these keys are dependent on the search level and the assets you want to use. For example:

```
"forename": "JOHN", "surname": "DOE", "addressline": "10
SWINEGATE", "postcode": "XX319RR", "latitude": "52.8619",
"longitude": "-0.629722".
```

For the ConsumerView API, refer to section 3.2.3 for a full list of input keys and their description. For other Neartime APIs, refer to the same section but within that API Developer Guide.

- `"flags": [{"KEY_NAME": ["FLAG1_VALUE", "FLAG2_VALUE", "FLAG3_VALUE"]}]` – this represents the key name and value of 3 flags for an optional flags array which you can use to stop one or more of the pre-processing cleaning tasks from running for this request. This is available for some datasets, e.g. ConsumerView, but not all and is ignored by these if included. Refer to section 3.2.4 for further information about the flags array for ConsumerView.

Example request

The following is a simple Windows cURL command line example which is used to demonstrate the content of a single record, multi-asset lookup request:

```
curl -X POST -H "Content-Type:application/json"
https://[domain_name]/overture/MultiAssetLookup -d
"{\"ssoId\": \"client.username@client.com\",
\"token\": \"3049f8bd-1934-4efb-8e2a-13311d4663e0\",
\"clientId\": \"10002\", \"assetId\": [\"10002A\", \"10002B\",
\"10002C\", \"10002D\", \"10002E\", \"10002G\"], \"forename\":
\"YAK\", \"surname\": \"PORTER\", \"addressline\": \"62
Grapefruits Lane\", \"postcode\": \"XX5 3NG\",
\"latitude\": \"50.00001\", \"longitude\": \"-0.99999\",
\"country\": \"GB\"}"
```

In this example, the request the asset IDs relate to the following datasets and APIs:

- 10002A – EMEA Mosaic API asset
- 10002B - WorldView API asset
- 10002C – Residential Property API asset
- 10002D – Microcells API asset
- 10002E – Suppressions API asset
- 10002G – ConsumerView API asset

4.3.2 About the response

Fully-matched lookup

If the lookup request results in a match for all assets in the request, the Neartime API sends a 200 (OK) response containing a JSON array of objects.

Each element or object in the array contains the results and asset ID for one of the assets specified in the request. Each object in the array has the following general format:

```
{
  "result": {
    "[VARIABLE_1_NAME]": "[VARIABLE_1_VALUE]",
    ...
    "[VARIABLE_1+n_NAME]": "[VARIABLE_1+n_VALUE]"
  },
  "assetId": "[ASSETID_VALUE]"
}
```

Each element in the `result` object contains key and string value pairs for each variable that has been defined in the asset you have used. For the ConsumerView API assets, the `result` object also contains a `Match` flag. An example of this is shown below.

```
[
  {
    "result": {
      "country_mosaic": "A01",
    },
    "assetId": "10002A"
  },
  {
    "result": {
      "disposable_income": "50",
      "country_code": "ZZ",
      "dominant_age_band": "15-30",
      "grid_code": "T99_000001_00001",
      "worldview_segment": "A"
    },
    "assetId": "10002B"
  },
  {
    "result": {
      "building_description": "DETACHED HOUSE",
      "property_roof_shape": "PITCHED",
      "property_wall_type": "CAVITY",
      "reception_room_count": "2",
      "property_built_in_period": "1500 TO 1550",
      "bedroom_count": "1",

```

```

        "property_wall_material": "GRANITE",
        "property_is_new_build": "FALSE",
        "dwelling_description": "HOUSEBOAT",
        "bathroom_count": "5+",
        "dwelling_type": "HOUSE",
        "building_type": "DETACHED"
    },
    "assetId": "10002C"
},
{
    "result": {
        "h_age_fine": "07",
        "h_shareholding_value": "0",
        "h_presence_of_young_person_at_address": "1",
        "p_discretionary_income_value": "35",
        "p_fsi_score": "-0.2574",
        "p_regional_normalised_person_income_band": "3",
    },
    "assetId": "10002D"
},
{
    "result": {
        "DeadFInd": "F",
        "DeadSInd": "F",
        "DeadIInd": "F"
    },
    "assetId": "10002E"
},
{
    "result": {
        "p_0071_perc_pension_standard_v1b": "18",
        "h_council_tax_band": "0",
        "h_income_value_v3": "22650",
        "h_fuelpoverty_v2_flag": "N",
        "h_households_with_children_v3": "0",
        "p_directorships_flag": "0",
        "pc_mosaic_uk_6_digital_group": "B",
        "h_mosaic_uk_6_type": "43",
        "Match": "P"
    },
    "assetId": "10002G"
}
]

```

Note: The same rule about which variables and propensities are in the response for a single record, single asset request is also applicable to single record, multi-asset requests.

Partially-matched lookup

If the lookup request results in a non-match for one or more assets (but not *all*) in the request, a 200 (OK) response is given but empty curly braces for the content of the `result` object for that asset in the response array, i.e.

```
{
  "result": {},
  "assetId": "10002E"
}
```

For example, the response below shows that for the first two assets in the request (element 0 and 1 in the array) no match was found, while for the third asset (element 2 in the array) a match was found:

```
[
  {
    "result": {},
    "assetId": "10002A"
  },
  {
    "result": {},
    "assetId": "10002B"
  },
  {
    "result": {
      "building_description": "DETACHED HOUSE",
      "property_roof_shape": "PITCHED",
      "property_wall_type": "CAVITY",
      "reception_room_count": "2",
      "property_built_in_period": "1500 TO 1550",
      "bedroom_count": "1",
      "property_wall_material": "GRANITE",
      "property_is_new_build": "FALSE",
      "dwelling_description": "HOUSEBOAT",
      "bathroom_count": "5+",
      "dwelling_type": "HOUSE",
      "building_type": "DETACHED"
    },
    "assetId": "10002C"
  }
]
```

Note: The same rule about which variables and propensities are in the response for a single record, single asset request is also applicable to single record, multi-asset requests.

Fully-non-matched lookup

If the lookup request is made where no match was found for any of the assets specified in the request, a 200 (OK) response is given, but empty curly braces for the content of the `result` object for all assets in the response array.

For example, the response below shows that for all assets in the request (element 0, 1 2 in the array) no match was found:

```
[
  {
    "result": {},
    "assetId": "10002A"
  },
  {
    "result": {},
    "assetId": "10002B"
  },
  {
    "result": {},
    "assetId": "10002C"
  }
]
```

Error response

If a problem occurred processing the request the server will give a different HTTP response to a 200 (OK).

The server may send a 429 (Too Many Requests) response to limit the loading on the network. If this occurs you will need to wait a specified amount of time dependent on the endpoint, before retrying. The time remaining (in milliseconds) is included in the response message.

See Section 5 for information on the other types of response which may be received for a request, what they mean, and how to resolve the issue.

4.4 Processing the response data

4.4.1 Matched responses

If the lookup request results in a successful match, then it is likely that you will write code to consume the response data. In simple terms, this means your code will extract and validate the values for each variable in the response, and then perhaps conditionally use the variables in some way according to their value.

The rules for creating a successful matched response for a single asset, single record request are also applicable to single record, multi-asset requests.

4.4.2 Non-matched and error responses

It is also expected that you would write application code to handle any response where no match is found for an asset, or which indicates that there was a problem.

For a multi-asset request, any error responses such as the 400- and 500-series response are either related to the entire call, or one or more assets defined in the request. For example, if there is an issue with the authentication `token` supplied in the request then this obviously is related to the entire call, and the appropriate 401 response would be given. However, if there is an asset-specific error response, then a 200 (OK) response is given for the request, but the `result` object holds the actual specific response for that asset. For example:

```
{
  "result": {
    "code": 401,
    "response": "Asset Not Found"
  },
  "assetId": "10002C"
}
```

See Section 5 for details of all responses.

5 HTTP responses

5.1 Introduction

The following provides the details of the possible responses to the log in request, the change password request, and the lookup requests. In some cases, the text of the response is the standard HTML response given by the server, while in other cases a custom response has been created to be used specifically with the API.

The details given provide the meaning of the response, and if it is an error, the possible causes and the resolution.

The names in parentheses are the standard HTTP names for the responses.

For a multi-asset request, any error responses such as the 400- and 500-series response are either related to the entire call, or one or more assets defined in the request. For example, if there is an issue with the authentication `token` supplied in the request then this obviously is related to the entire call, and the appropriate 401 response would be given. However, if there is an asset-specific error response, then a 200 (OK) response is given for the request, but the `result` object holds the actual specific response for that asset. For example:

```
{
  "result": {
    "code": 401,
    "response": "Asset Not Found"
  },
  "assetId": "10002C"
}
```

The relevant subsections for each response identify whether it would be treated as an asset-specific response for a multi-asset request.

5.2 200 (OK) responses

5.2.1 200 – Fully-matched lookup (batch record)

If a batch record lookup request is made where all consumers in the request were matched, a 200 (OK) response is given.

The body of the response contains a JSON array. Each element in the array comprises key and string value pairs for each variable and propensity for the respective consumer in the request. Each element also includes a `Match` flag. For example, for a two element array:

```
[{
  "pc_mosaic_uk_7_type": "29",
  "Match": "PC"
},
{
  "pc_mosaic_uk_7_type": "09",
  "Match": "PC"
}]
```

5.2.2 200 – Fully-matched lookup (single record, multi-asset)

If a single record, multi-asset lookup request is made where all assets in the request were matched, a 200 (OK) response is given.

The body of the response contains a JSON array. Each element in the array comprises a `result` object and an `assetId` key for the respective asset in the request. The `result` object contains the string value pairs for each variable and propensity.

If the lookup request results in a match for all assets in the request, the Neartime API sends a 200 (OK) response containing a JSON array of objects.

Each element or object in the array contains the results and asset ID for one of the assets specified in the request. Each object in the array has the following general format:

```
{
  "result": {
    "[VARIABLE_1_NAME]": "[VARIABLE_1_VALUE]",
    ...
    "[VARIABLE_1+n_NAME]": "[VARIABLE_1+n_VALUE]"
  },
  "assetId": "[ASSETID_VALUE]"
}
```

Each element in the `result` object contains key and string value pairs for each variable that has been defined in the asset you have used. For the ConsumerView API assets, the `result` object also contains a `Match` flag. An example of this is shown below.

```
[
  {
    "result": {
      "country_mosaic": "A01",
    },
    "assetId": "10002A"
  },
  {
    "result": {
      "disposable_income": "50",
      "country_code": "ZZ",
      "dominant_age_band": "15-30",
      "grid_code": "T99_000001_00001",
      "worldview_segment": "A"
    },
    "assetId": "10002B"
  },
  {
    "result": {
      "building_description": "DETACHED HOUSE",
      "property_roof_shape": "PITCHED",
      "property_wall_type": "CAVITY",
      "reception_room_count": "2",
      "property_built_in_period": "1500 TO 1550",
      "bedroom_count": "1",
      "property_wall_material": "GRANITE",
      "property_is_new_build": "FALSE",
      "dwelling_description": "HOUSEBOAT",
      "bathroom_count": "5+",
      "dwelling_type": "HOUSE",
      "building_type": "DETACHED"
    },
    "assetId": "10002C"
  },
  {
    "result": {
      "h_age_fine": "07",
      "h_shareholding_value": "0",
      "h_presence_of_young_person_at_address": "1",
      "p_discretionary_income_value": "35",
      "p_fsi_score": "-0.2574",
      "p_regional_normalised_person_income_band": "3",
    },
    "assetId": "10002D"
  },
]
```

```

{
  "result": {
    "DeadFInd": "F",
    "DeadSInd": "F",
    "DeadIInd": "F"
  },
  "assetId": "10002E"
},
{
  "result": {
    "p_0071_perc_pension_standard_v1b": "18",
    "h_council_tax_band": "0",
    "h_income_value_v3": "22650",
    "h_fuelpoverty_v2_flag": "N",
    "h_households_with_children_v3": "0",
    "p_directorships_flag": "0",
    "pc_mosaic_uk_6_digital_group": "B",
    "h_mosaic_uk_6_type": "43",
    "Match": "P"
  },
  "assetId": "10002G"
}
]

```

Note: The same rule about which variables and propensities are in the response for a single record, single asset request is also applicable to single record, multi-asset requests.

5.2.3 200 – Fully-non-matched lookup (batch record)

If a batch record lookup request is made where no consumers were matched, a 200 (OK) response is given, but with empty curly braces for each element in the array. For example, for a 2-element array:

```
[ {}, {} ]
```

This is the normal response where a search of the ConsumerView database resulted in no match for any consumers in the request.

5.2.4 200 – Fully-non-matched lookup (single record, multi-asset)

If the lookup request is made where no match was found for any of the assets specified in the request, a 200 (OK) response is given, but empty curly braces for the content of the `result` object for all assets in the response array.

For example, the response below shows that for all assets in the request (element 0, 1 2 in the array) no match was found:

```
[
  {
    "result": {},
    "assetId": "10002A"
  },
  {
    "result": {},
    "assetId": "10002B"
  },
  {
    "result": {},
    "assetId": "10002C"
  }
]
```

5.2.5 200 – Matched lookup (single record, single asset)

This response is given to a single record, single asset lookup request where there has been a successful match with the search data supplied. The body of the response comprises key and string value pairs for each variable and propensity that has been defined in the asset. The response also includes a `Match` flag. For example:

```
{
  "pc_mosaic_uk_7_type": "29",
  "Match": "PC"
}
```

5.2.6 200 – Non-matched lookup (single record, single asset)

This response may be given to a single record, single asset lookup request and comprises the headers with the `Content-Length` header set to 2 and the body with empty curly braces, i.e. `{}`.

This is the normal response where a search of the ConsumerView database resulted in no match.

5.2.7 200 – Non-matched lookup (0-byte)

This response may be given to a lookup request and comprises the response headers only, i.e. no body and with the `Content-Length` header set to 0 e.g.

```
HTTP/1.1 200
Content-Type: application/json;charset=ISO-8859-1
Content-Length: 0
Date: Tue, 22 Oct 2019 08:51:55 GMT
```

Cause or Resolution	Description
Cause:	This response may be given because of an unknown cause either in the composition or content of the request, or a problem with the API.
Resolution:	Please contact the Helpdesk to report the problem so that it may be investigated further.

5.2.8 200 – Partially-matched lookup (batch record)

If the lookup request results in a non-match for one or more (but not all) consumers in the request, a 200 (OK) response is given but empty curly braces, i.e. `{ }` are returned for that consumer in the response array.

For example, the response below shows that for the first consumer in the request (element 0 in the array) matched at person-level, while for the second consumer (element 1 in the array) no match was found:

```
[{
  "h_age_fine": "07",
  "p_financial_advice_band": "1",
  "p_gender": "0",
  "p_length_of_residency": "11",
  "p_0212_perc_newspaper_main_sun_v1b": "77",
  "Match": "P"
},
{}]
```

5.2.9 200 – Partially-matched lookup (single record, multi-asset)

If the lookup request results in a non-match for one or more assets (but not *all*) in the request, a 200 (OK) response is given but empty curly braces for the content of the `result` object for that asset in the response array, i.e.

```
{
  "result": {},
  "assetId": "10002E"
}
```

For example, the response below shows that for the first two assets in the request (element 0 and 1 in the array) no match was found, while for the third asset (element 2 in the array) a match was found:

```
[
  {
    "result": {},
    "assetId": "10002A"
  },
  {
    "result": {},
    "assetId": "10002B"
  },
  {
    "result": {
      "building_description": "DETACHED HOUSE",
      "property_roof_shape": "PITCHED",
      "property_wall_type": "CAVITY",
      "reception_room_count": "2",
      "property_built_in_period": "1500 TO 1550",
      "bedroom_count": "1",
      "property_wall_material": "GRANITE",
      "property_is_new_build": "FALSE",
      "dwelling_description": "HOUSEBOAT",
      "bathroom_count": "5+",
      "dwelling_type": "HOUSE",
      "building_type": "DETACHED"
    },
    "assetId": "10002C"
  }
]
```

Note: The same rule about which variables and propensities are in the response for a single record, single asset request is also applicable to single record, multi-asset requests.

5.2.10 200 – Successful change password

This response may be given to a successful change password request, with the body of the response containing a single key and Boolean value pair i.e.

```
{
  "success": true
}
```

This is the expected response to a change password request.

5.2.11 200 – Successful log in

This response may be given to a successful log in request, with the body of the response containing a single key and string value pair comprising the token key name and a hexadecimal string e.g.

```
{ "token": "713dbcaa-21f5-4124-a576-176805b7b326" }
```

This is the expected response to a log in request.

5.3 400 – (Bad Request) responses

5.3.1 400 – Bad parameters, invalid

This is a custom response given by the server if it detects a problem with the keys in the lookup request. The body of the custom response contains two key and value pairs:

```
{ "code": 400, "response": "Bad parameters, invalid" }
```

Cause or Resolution	Description
Cause:	A problem was detected with the <code>clientId</code> , <code>token</code> or <code>ssoId</code> keys. This could be because the key is missing from the request or that it has been mis-spelt.
Resolution:	Check that all keys are supplied in the request and that they are spelt correctly.

5.3.2 400 – Invalid input

This is a custom response given by the server if it detects a problem with the keys or format of the JSON for a login request or a change password request. The body of the custom response contains two key and value pairs:

```
{ "code": 400, "response": "Invalid input" }
```

Cause or Resolution	Description
Cause:	A problem was detected with the keys or formatting of the JSON in the request.
Resolution:	Review the keys and format of the JSON being sent out in the request and fix any errors.

5.4 401 – (Unauthorized) responses

5.4.1 401 – Asset Not Found.

This is a custom response given by the server if it detects that the value of the `assetId` supplied in the lookup request does not exist. The body of the custom response contains two key and value pairs:

```
{"code":401,"response":"Asset Not Found."}
```

Note: This response is specific to an asset. This means that for a multi-asset request the response text given above would appear in the `result` object for the appropriate asset and within a 200 (OK) response.

Cause or Resolution	Description
Cause:	The value <code>assetId</code> given in the lookup request could not be found by the server.
Resolution:	Check the value of the <code>assetId</code> matches one of those given in the registration e-mail, and correct if necessary. If the value of the <code>assetId</code> does match the one given in the registration e-mail, contact the Helpdesk to report the problem and receive the correct asset ID.

5.4.2 401 – Invalid token

This is a custom response given by the server if it detects a problem with the `token` supplied in the lookup request. The body of the custom response contains two key and value pairs:

```
{"code":401,"response":"Invalid token"}
```

Cause or Resolution	Description
Cause:	A problem was detected with the value of the <code>token</code> parameter or key. This could be that the value of the <code>token</code> has been quoted incorrectly in the lookup request or more likely that the token is more than 30 minutes old.
Resolution:	If the token is more than 30 minutes old, check that its value in the lookup request is correct. You should also ensure that the code for your log in request is in its own thread. If you do not do this, then any request which is received by your application will invalidate any current token, which will cause any search request which use it to fail. If the token is more than 30 minutes old, log in to the ConsumerView API to obtain a new <code>token</code>.

5.4.3 401 – Unauthorized Asset.

This is a custom response given by the server if it detects that the client is not authorised to use the requested asset given in the lookup request. The body of the custom response contains two key and value pairs:

```
{"code":401,"response":"Unauthorized Asset."}
```

Note: This response is specific to an asset. This means that for a multi-asset request the response text given above would appear in the `result` object for the appropriate asset and within a 200 (OK) response.

Cause or Resolution	Description
Cause:	The value of the <code>assetId</code> given in the lookup request is not authorised to be used by the client.
	Check the value given in the <code>assetId</code> matches one of those given in the registration e-mail, and correct if necessary.
Resolution:	If the value of the <code>assetId</code> does match the one given in the registration e-mail, contact the Helpdesk to report the problem and receive the correct asset ID.

5.4.4 401 – Unauthorized Client

This is a custom response given by the server if it detects a problem with the credentials given in a lookup request. The body of the custom response contains two key and value pairs:

```
{"code":401,"response":"Unauthorized Client"}
```

Cause or Resolution	Description
Cause:	A problem was detected with the credentials in a lookup request where the value of the <code>ssoId</code> or <code>clientId</code> key may be incorrect.
	Check the value of the <code>ssoId</code> matches the username in the registration e-mail.
Resolution:	The value of the <code>clientId</code> key must match the value given in the registration e-mail to the client. Check the value of the key against the one in the e-mail.
	If the response is still given, contact the Helpdesk.

5.4.5 401 – Unauthorized Client, null token

This is a custom response given by the server if it detects a problem with the credentials given in log in request, so no token was returned. The body of the custom response contains two key and value pairs:

```
{"code":401,"response":"Unauthorized Client, null token"}
```

Cause or Resolution	Description
Cause:	A problem was detected with the credentials given in a log in request where the value of the <code>password</code> or the <code>userId</code> key may be incorrect.
Resolution:	The value of the <code>userId</code> key must match that given by the username in the registration e-mail. The value and case of <code>password</code> key must match the current password which is used to log in to Plexus applications. Check the value of both keys and correct as necessary. Ensure that value of the <code>password</code> key matches the latest password, particularly if it was recently changed by someone else in your organisation.

5.4.6 401 – Unsuccessful, bad token or new password invalid

This is a custom response given by the server if it detects a problem with the change password request. The body of the custom response contains two key and value pairs:

```
{"code":401,"response":"Unsuccessful, bad token or new password invalid"}
```

Cause or Resolution	Description
Cause:	A problem was detected with the <code>token</code> or the value of <code>newpassword</code> in a change password request. This could be that the value of the <code>token</code> has been quoted incorrectly in the request or more likely that the token is more than 30 minutes old. It could also be that the value of <code>newpassword</code> does not meet the password requirements.
Resolution:	If the token is more than 30 minutes old, check that its value in the request is correct. If the token is more than 30 minutes old, log in to the ConsumerView API to obtain a new token. Check the value of the new password and confirm that it meets the requirements given in Subsection 2.2.1.

5.5 404 – (Not Found) response

This is a standard response given by the server if it does not recognise the endpoint or URL supplied in a request. The response body is supplied as an HTML encoded page.

Cause or Resolution	Description
Cause:	The URL or endpoint specified in a request was not matched to any resource by the server and so a standard error was returned.
Resolution:	Check the spelling of the log in request URL; the change password request URL; or the lookup request URL and confirm that it matches the ones specified in this document, i.e. for a log in request: <pre>https://[domain_name]/overture/login</pre> or for a change password request: <pre>https://[domain_name]/overture/changepassword</pre> or for a lookup request: <pre>https://[domain_name]/overture/lookup</pre>
Note: See Subsection 1.5.1 for details of the domain names could be used for the <code>[domain_name]</code> placeholder.	

5.6 405 – (Method Not Allowed) response

This is a standard response given by the server if request uses an HTTP method not supported by the API endpoint. The response body is supplied as an HTML encoded page.

Cause or Resolution	Description
Cause:	The HTTP method used in the request is not supported by the API endpoint. Currently API endpoints only support the POST method. Any other method such as GET are not supported.
Resolution:	Ensure that you only use the POST method in the requests.

5.7 417 – (Expectation Failed) responses

5.7.1 417 – Batch limit exceeded

This is a custom response given by the server if it detects that there are more than 5,000 records in a batch lookup request. The body of the custom response contains two key and value pairs:

```
{"code":417,"response":"Batch limit exceeded"}
```

Cause or Resolution	Description
Cause:	There were more than 5,000 records in the batch lookup request.
Resolution:	Reduce the number of records in the batch lookup to 5,000 or less, and then retry.

5.7.2 417 – Incorrect JSON

This is a custom response given by the server if it detects a problem with the JSON format in a login request or lookup request. The body of the custom response contains two key and value pairs:

```
{"code":417,"response":"Incorrect JSON"}
```

Cause or Resolution	Description
Cause:	A problem was detected with the formatting of the JSON in the request.
Resolution:	Review the format or the JSON being sent out in the request and fix the formatting errors.

5.7.3 417 – Missing parameters or invalid

This is a custom response given by the server if it detects a problem with the parameters sent to the endpoint. The body of the custom response contains two key and value pairs:

```
{"code":417,"response":"Missing parameters or invalid"}
```

Cause or Resolution	Description
Cause:	A problem was detected with the parameters supplied in the request to the endpoint. For example, it might be that the endpoint does not understand the value of the parameters which are sent. This response can occur if you attempt to send a multi-asset request to the single asset lookup endpoint.
Resolution:	Confirm that you are using the correct endpoint for the request. Review the value of the parameters sent to the endpoint and correct as necessary.

5.8 429 – (Too Many Requests) response

This is a custom response given to limit the loading on the network. The response may be given for any type of request. If this occurs you will need to wait a specified amount of time dependent on the endpoint before retrying.

The body of the custom response contains three JSON key and value pairs:

```
{
  "code":429,
  "response":"Limit Reached for [endpoint], try again in
[time_ms]",
  "remaining":[time_ms]
}
```

Cause or Resolution	Description
Cause:	The server has detected that the loading on the network is too high.
Resolution:	You should write code to deal with this response, and waiting the specified time remaining before retrying the request.

5.9 500 – (Internal Server Error) responses

5.9.1 500 – Internal Server Error

This is a standard response given by the server if an unexpected and severe problem occurs. The response body is supplied as a HTML encoded page.

Cause or Resolution	Description
Cause:	An invalid or missing <code>assetId</code> given in the lookup request could not be processed by the server and so a standard error was returned.
Resolution:	Check the value of the <code>assetId</code> matches one of those given in the registration e-mail, and correct if necessary. Contact the Helpdesk to report this problem for further investigation.

5.9.2 500 – Problem with authentication

This is a custom response given by the server if there was an issue with a login request or a change password request. The body of the custom response contains two key and value pairs:

```
{"code":500,"response":"Problem with authentication"}
```

Cause or Resolution	Description
Cause:	There was an issue with logging in or changing the password because of a communication issue with the Plexus Administration server.
Resolution:	Try again. If the issue is still present, contact the Helpdesk to report this problem for further investigation.

5.9.3 500 – Problem with login

This is a custom response given by the server if there was an issue retrieving a token from the Plexus Administration server. The body of the custom response contains two key and value pairs:

```
{"code":500,"response":"Problem with login"}
```

Cause or Resolution	Description
Cause:	There was an issue retrieving a token from the Plexus Administration server.
Resolution:	Try again. If the issue is still present, contact the Helpdesk to report this problem for further investigation.

5.10 503 – (Service Unavailable) responses

5.10.1 503 – Internal refresh in progress

This is a custom response given by the server if a refresh was in progress when the request was made. The body of the custom response contains two key and value pairs:

```
{"code":503,"response":"Internal refresh in progress"}
```

Cause or Resolution	Description
Cause:	There was temporary problem with the server in the cluster which received the request.
Resolution:	Try the request again as it is likely to succeed when the refresh operation has been completed. If the request still fails, try again in 10 minutes and if it still fails report the problem to the Helpdesk.

5.10.2 503 – Server error

This is a custom response given by the server if a serious server error occurs when a request is made. The body of the custom response contains two key and value pairs:

```
{"code":503,"response":"Server error"}
```

Cause or Resolution	Description
Cause:	The cause of these errors are unknown, but the system would be closely monitored by Experian for any of these failures.
Resolution:	Report the problem to the Helpdesk, and the status of a fix.

A Dummy data

You can use the dummy data given in the table below to test the API with your asset.

Data type	Value	Match level
Name and address	YAK PORTER 62 GRAPEFRUITS LANE XX5 3NG	Person, Household or Postcode (Note 1)
Personal email address	Godzilla@GrapeFruitInternets.org	Person
Mobile number	07123456789	Person
Household email address	garfield@GrapeFruitInternets.org	Person
Household match key	999999999999999991	Household (Note 2)
Family match key	999999999999999992	Household (Note 3)
Person match key	9909909992	Person (Note 4)
Individual match key	999999999999999993	Person (Note 5)

Notes:

- 1) The actual level depends on which parts of the name and address are used in the search. If all parts are used then a match at person level is given. However, if only the postcode is used then a match at postcode level is given.
- 2) The dummy household match key (`key_household`) is associated with the address of YAK PORTER above.
- 3) The dummy family match key (`key_family`) is associated with the PORTER family living at the above address.
- 4) The dummy person match key (`key_person`) is associated with YAK PORTER living at the above address.
- 5) The dummy individual match key (`key_individual`) is associated with YAK PORTER living at the above address. In this case the key is unique because there are no other people living at the same address with the same surname and same initial.