Real-time Listings Service

1. Introduction

Thank you for your interest in the Zoopla Property Group (ZPG) Real-time Listings Service.

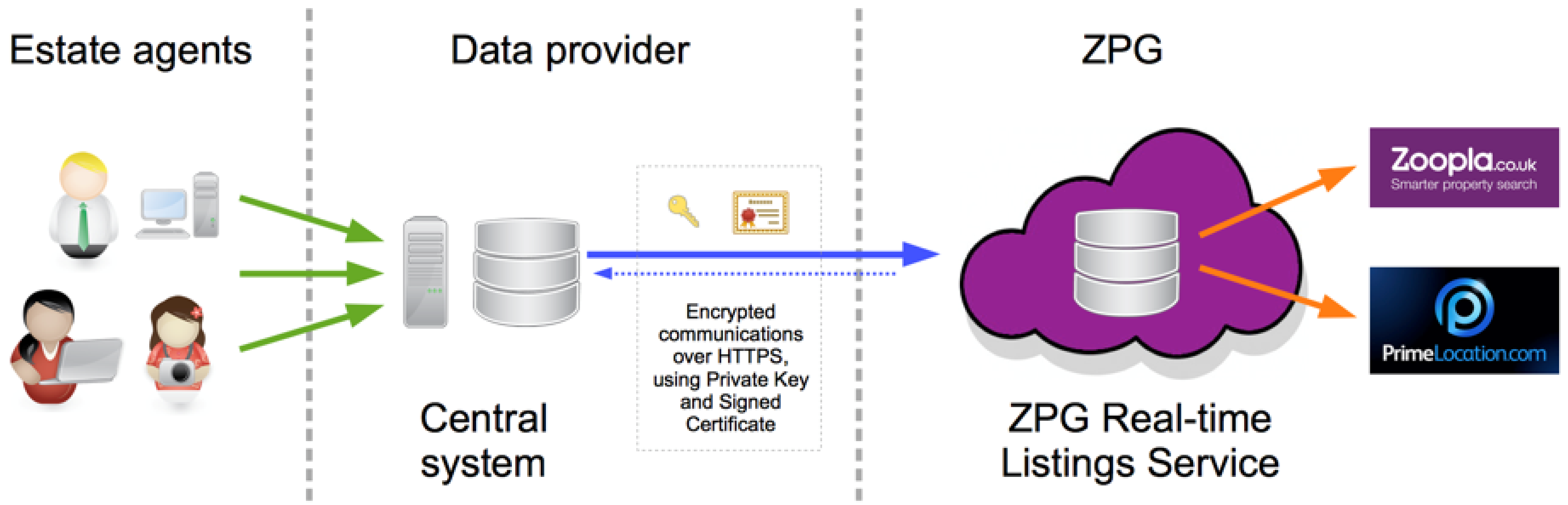
ZPG operates a number of property websites; the principal ones being Zoopla and PrimeLocation. Our mission is to provide the most useful online resources to property consumers, and be the most effective partner to property professionals.

To achieve this, data about available properties needs to be transferred from estate agents' software systems to ZPG. Traditionally the data transfer involves exporting large volumes of data from a database, providing a comprehensive picture of the estate agent's properties at that time. This method has the benefit of being very easy to implement but it is also very slow (due to the amount of data involved) and inefficient (most of that data has not changed since the previous export). These limitations mean that it is usually only performed once a day, overnight.

The level of service that users and estate agents now expect means that the daily bulk transfer of data is no longer suitable; the ZPG Real-time Listings Service is its replacement. It allows estate agents to make changes on their system and have those changes reflected on ZPG's websites within minutes. It achieves this by reducing the amount of data that needs to be transferred: only information about listings which have actually changed needs to be transmitted (i.e. updates are incremental).

What follows is the technical specification of the ZPG Real-time Listings Service so that anyone who wishes to integrate it with their systems may do so.

2. Overview



Estate agents
Data provider
ZPG
Central system
Encrypted communications over HTTPS, using Private Key and Signed Certificate
ZPG Real-time Listings Service
Zoopla.co.uk Smarter property search
PrimeLocation.com

The ZPG Real-time Listings Service is designed to interact with a central data provider. Estate agents manage their properties using either a cloud-based property management solution or locally installed software which logs changes to a central data system. As changes are registered on this central system it sends messages to the ZPG Real-time Listings Service, in order to update ZPG's version of that data, which is displayed on its websites.

2.1 Simple technical overview

The ZPG Real-time Listings Service uses a fairly conventional web service design. Clients communicate with the service using standard HTTP over a secure connection (HTTPS). Each method (action) that the service provides has its own dedicated URL. The data required for the method is sent in a JSON formatted message.

The service runs in two environments: *sandbox* and *live*. The sandbox environment is used for testing your system's messaging interactions; whilst the live environment is for data you intend to be published to our websites. Initially we will provide you with access to the sandbox environment and once you are happy with your implementation you will be granted access to the live environment.

There are two principal entities which are tracked: branches and listings. A branch can have many listings associated with it but a listing can only belong to one branch. Whenever a change to a branch or listing is registered by your system, its full details should be sent to us. We will then replace our record entirely with your new version.

The media content for a listing (e.g.: images; brochures) is not transferred directly via the service. Instead, the content should be hosted on a web server and associated URLs included in a *listing/update* message. We will then download that content from your web server.

2.2 Simple workflow

The ZPG Real-time Listings Service is used in the following way:

1. When information about a branch changes, you send a *branch/update* message to replace the data we have.
2. When information about a listing changes, you send a *listing/update* message to replace the data we have. When sending this message you also include some metadata (*ZPG-Listing-ETag*) which will enable you to detect data synchronisation issues at a later date.
3. When a listing becomes inactive, you send a *listing/delete* message.
4. In order to check that the listing data for a branch has been synchronised properly, you can send a *listing/list* message. This will return information about all of the branch's listings, including the *ZPG-Listing-ETag*s that were previously sent. You can then ascertain if there are any data differences between ZPG's system and yours, and correct any issues by sending *listing/update* and *listing/delete* messages as appropriate.

For a more detailed description of this workflow please see *10. Workflow overview*.

3. Security and authentication

There are two aspects to security: privacy and identity. By using HTTPS we can guarantee both: communications are encrypted (privacy) and both parties are assured that they are communicating directly with the person/system they expect (identity). HTTPS uses standard public-private key-pair cryptography (messages that are encrypted with one key can only be decrypted with the other key) which is wrapped up in an identity exchange mechanism called X.509 Certificates. The set-up process is:

1. You create a public-private key-pair. The *private key* is integral to proving your identity; it should **never** leave your corporate network, nor be given to any third-party, including ZPG. The *public key*, as its name suggests, can be given to anyone.

2. The public key, plus details about your corporate identity (name) are combined to generate a *certificate signing request* file. (Since anyone can perform steps 1 and 2, this cannot be used as a certificate on its own; it needs to be verified and signed by a *certificate authority*. In this case, ZPG will be acting as the certificate authority.)
3. You send your certificate signing request file (in PEM format) to ZPG via our certificate signing request submission page. This page will check that it is both formatted correctly and does not use encryption settings which are no longer secure. Upon acceptance of your certificate signing request, you will be provided with a *verification token* (which looks like "XXXX-XXXX-XXXX") that you should write down and keep safe.
4. ZPG will contact you and verbally confirm that you sent us a certificate signing request and ask for your verification token.
5. Having confirmed your identity and your request, ZPG will sign the certificate signing request and return the resulting *signed certificate* to you.
6. You will need to configure your software so that it can use the *signed certificate* and *private key* when contacting our service, so that both systems can authenticate each other's identity and encrypt communications.

3.1 Example: creating keys and certificate signing request with OpenSSL on Linux

How you generate a public-private key-pair and associated certificate signing request will be dependent on your particular software platform. For reference, this is how you might do it on Linux using *openssl*:

1. Create the public-private key-pair.

```
openssl genrsa -out private.pem 2048
```

This generates a key-pair using the RSA algorithm with keys that are 2048 bits long, which is then output to the file *private.pem*. Note that this file includes everything about the generated key-pair and the public key can be produced from the information it contains. Reminder: *private.pem* should **never** leave your corporate network.

We suggest that when you create your key-pair you do not specify an accompanying password. Otherwise you'll have to provide that password (to decrypt the file) every time your system is restarted. The lack of a password will not affect the security of the connection itself. The example given here does not set a password.

If you are interested, you can look at the contents of *private.pem*:

```
openssl rsa -in private.pem -text
```

2. Create a certificate signing request.

```
openssl req -new -sha256 -key private.pem -out public.csr
```

This creates a new certificate signing request, using the SHA-256 cryptographic hash algorithm, which includes the public key component contained in *private.pem*. You will be prompted to enter details about your company:

- *Country Name*: ISO 3166-1 alpha-2 country code.
- *State or Province Name*: county.
- *Locality Name*: town/city.
- *Organization Name*: company name.
- *Organizational Unit Name*: department (optional).
- *Common Name*: (optional).
- *Email Address*: (optional).
- *A challenge password*: (optional).
- *An optional company name*: (optional).

The certificate signing request is output to file *public.csr* and is the file you would submit to our certificate signing request submission page.

If you are interested, you can look at the contents of *public.csr*:

```
openssl req -in public.csr -text
```

3.2 Example: testing authentication using Curl on Linux

How you configure your system to use the private key and signed certificate to establish an HTTPS connection will be dependent on your particular software platform. Consequently our ability to advise you on how to troubleshoot authentication issues is limited. However, if you use Linux you can test that the private key and signed certificate work together correctly by:

```
echo '{ "branch_reference" : "test" }' |

curl --key /path/to/private.pem --cert /path/to/certificate.crt --data @-

--header "Content-type: application/json;
```

```
profile=http://realtime-listings.webservices.zpg.co.uk/docs/v1.2/schemas/listing/list.json"
https://realtime-listings-api.webservices.zpg.co.uk/sandbox/v1/listing/list
```

which should receive the following response:

```
{
   "status" : "OK",
   "listings" : [],
   "branch_reference" : "test"
}
```

You can test send any of the message examples in a similar way, although you may need to include additional headers (see *4. Messaging*).

3.3 Certificate revocation

If at any point you believe that your private key may have been compromised (seen by anyone outside your company) then you should inform ZPG immediately so that we can revoke the associated signed certificate, thus preventing anyone from masquerading as you. Then, repeat the above procedure to generate a new private key and obtain a new signed certificate. You can confirm that your previous signed certificate has been revoked by consulting our Certificate Revocation List.

3.4 Supported TLS versions

The ZPG Real-time Listings Service **only** accepts secure connections using the **TLS 1.2** protocol.

Since security is so important it is worth explaining why this is the case. As previously discussed, security is composed of two parts: identity and privacy. Whilst the data being sent to the Real-time Listings Service is not particularly sensitive (privacy), being assured that the person sending the data is actually who they claim to be (identity) is paramount: no-one should be able to read/change data except its legitimate owner.

(ZPG also expects to launch other services in future. Those services may be dealing with data that is sensitive and should therefore be transmitted in a suitable manner. The Real-time Listings Service is the first of these services, and so we have taken the opportunity to establish best practice standards at the outset, so that partners are familiarised with our security requirements, leading to fewer suprises or problems in future.)

There are a number of industry-standard secure communication protocols - *many of which have become obsolete due to flaws in their design*:

- SSL 1.0, 2.0, 3.0 are all **insecure**. There are known exploits and no-one should be using them any more.
- TLS 1.0 is **insecure**. There are known exploits and no-one should be using it any more.
- TLS 1.1 is believed to be secure but is not the latest version.
- TLS 1.2 is believed to be secure and is the latest version, with support for additional, stronger cipher algorithms than TLS 1.1.

(A summary of cipher and protocol attacks on TLS has been published by the IETF.)

After reviewing library support for TLS across programming languages and operating systems, we determined that, in the vast majority of cases, if TLS 1.1 was supported then TLS 1.2 was too. We therefore feel comfortable that by restricting access to TLS 1.2 we are following best practice guidelines, should not cause undue disruption, and is in the interest of all users of our services.

4. Messaging

The ZPG Real-time Listings Service uses JSON as its messaging format. JSON has a simple structural syntax; is easy to read; is familiar to software developers; and has excellent library support across many software platforms. We have no plans to support XML.

(Note that in JSON nomenclature an object's attributes are called "properties", which is somewhat unfortunate given the industry we operate in. In order to remove any possibility of ambiguity, this documentation refers to "attributes" instead and the only use of the word "properties" in our schemas is as required by the JSON standard, not as the name of a data attribute.)

For the API method you wish to call, you should construct a JSON message which conforms to the schema definition for that method. We recommend you perform a schema validation check before sending us the message so that you can detect and respond to any validation errors earlier in your messaging process. Our service will **always** validate your message against the associated schema and reject those that fail.

The JSON message would then be sent to the method's URL endpoint using an HTTP POST. Having received your message the service will respond with a standard HTTP response status code and provide additional information in the response content, formatted in JSON.

4.1 Method endpoints

The ZPG Real-time Listings Service operates in two environments:

- *sandbox*, for testing: https://realtime-listings**-api**.webservices.zpg.co.uk/**sandbox**/v1/
- *live*, for data destined for our websites: https://realtime-listings**-api**.webservices.zpg.co.uk/**live**/v1/

Having chosen the correct environment, you then append the name of the method you wish to call. For example:

- https://realtime-listings-api.webservices.zpg.co.uk/sandbox/v1/branch/update
- https://realtime-listings-api.webservices.zpg.co.uk/sandbox/v1/listing/update
- https://realtime-listings-api.webservices.zpg.co.uk/sandbox/v1/listing/delete
- https://realtime-listings-api.webservices.zpg.co.uk/sandbox/v1/listing/list

You would then use an HTTP POST to send your JSON message to that endpoint, along with the appropriate HTTP headers.

Reminder: your software will need to have been configured so that it can access both the *signed certificate* and *private key*, discussed in *3. Security and authentication*, when you contact either of these environments. This is so that the secure HTTPS connection can be established.

Please note that the endpoints reference the *major* component of the API version number you wish to use, whilst the JSON schemas also include minor version components, which must be specified in the Content-Type profile.

(You will initially be given access to the sandbox environment to test your system integration. Data sent here will not affect ZPG's websites. When you are happy that your system integration is complete, you will need to contact us to gain access to the live environment. At that time we will then discuss the best way to perform an initial upload of all your branch and listing data, as well as how to migrate any existing clients who currently use an FTP feed.)

Note that some schema versions may not be available in the live environment; for example, during the development and testing of new versions, or because an old version has been retired. Please refer to our current message schemas to see which environments support the API version documented here.

4.2 HTTP request

The JSON message should be sent to the URL endpoint for the method you wish to use, via a POST.

4.3 HTTP headers

4.3.1 Content-Type

The Content-Type should be set to *application/json*, along with a *profile* definition, which is the URL of the specific version of the message schema you are conforming to. e.g.:

```
Content-Type: application/json; profile=https://realtime-listings.webservices.zpg.co.uk/docs/v1.2/schemas/listing/update.json
```

4.3.2 ZPG-Listing-ETag (only required for *listing/update*)

ZPG-Listing-ETag is the mechanism for ensuring that the listings on your system and ours are synchronised. Whenever you send us a *listing/update* message we will replace the data we have with the new version. At a later date you may wish to check whether the data we have is indeed the same as yours. Rather than sending the entire message back to you - which would then require you to compare all the individual attributes to find differences - we instead ask that you provide us with metadata that will allow your system to quickly determine whether the listing data we have is synchronised with yours. This metadata is sent via the *ZPG-Listing-ETag* header with each *listing/update* message, which can later be retrieved via the *listing/list* method.

The *ZPG-Listing-ETag* value is a string that you define (up to 255 characters in length) that indicates the overall state of the listing, as represented by the *listing/update* message. Its value should:

- remain constant for a given *listing/update* message
- change when data within the *listing/update* message is modified
- be unlikely to be duplicated in future

i.e. if something changes on your system that would change the *listing/update* message you would send to us, then the accompanying *ZPG-Listing-ETag* should change to reflect that.

We recommend using a checksum of the canonicalised *listing/update* message. Depending on how your system tracks changes to listing data (and how it is sent to us) an auto-incrementing revision_id, or a last-modified date **may** also be suitable.

The *ZPG-Listing-ETag* you provide will be returned by the *listing/list* method. By comparing its value with the one your system would produce now, you can determine whether it is necessary to refresh the listing on our system. Please see 10.3 Periodically check synchronisation for additional information about this process.

5. Methods

The service supports the following methods:

- *branch/update*: create or update information about a branch (e.g.: their name; address).
- *listing/update*: create or update information about a listing, associated with a branch (e.g.: description; price).
- *listing/delete*: delete a listing from a branch's inventory.
- *listing/list*: retrieve a branch's inventory list.

Please see the current message schemas and example messages for each method.

5.1 *branch/update* method (optional)

This method allows you to describe a branch, to which listings are then associated. The address and other contact details allow us to identify the equivalent branch on our system and map your *branch_reference* to ours.

Note that this method is optional. If you do not use it then the branch information will need to be transferred to ZPG in some other way (typically via an emailed CSV spreadsheet), although this will likely delay the integration of branches with our system.

Message attribute	**Requirements**	**Type**	**Description**	**Example**
branch_reference	Mandatory	string	Your unique identifier for the branch. (We do not generally issue these as it should be the identifier that is native to your system. This makes debugging easier for you and means you do not need to wait for any	"1234"; "kd-789d"

			additional information from us before you can start uploading listing data for a branch.)	
branch_name	Mandatory	string	The name of the branch. This is usually the name of the company and may also include some location information in order to differentiate it from other branches in the company.	"Estate Agent Ltd - Shepherd's Bush"
location	Mandatory	*location object*	Information about the location of the branch, such as its address.	
telephone		string	Telephone number.	"02079460184"; "+1 246-123-4567"
email		string	Email address.	"enquiries@estateagentltd.co.uk"
website		string	The URI-encoded URL for the branch's website, or that of its parent company if it doesn't have one of its own.	"http://www.estateagentltd.co.uk"

5.2 *listing/update* method

This method allows you to describe a listing. It is used for both creation and update purposes and, in either case, the listing should be described in its entirety.

There are a small number of mandatory attributes:

1. *branch_reference*
2. *category*
3. *detailed_description*
4. *life_cycle_status*
5. *listing_reference*
6. *location*
7. *pricing*

8. *property_type*

The remaining attributes are all optional. However, all the additional information you can provide improves our ability to successfully match the listing to a user's search. If you do not have any data for a particular attribute, then do not include it in your message - do **not** default it. This will make it less likely for the message to fail validation rules which use the attribute and you won't accidentally introduce meaning (and consequence) where there was none before.

There are three attributes which define how we will interpret the listings's data:

1. *location:country_code*: "Where is it?" (This can be further reduced to: "Is this a UK or overseas listing?")
2. *category*: "Is this listing for a property which is residential (people will live in it) or commercial (a place of business)?"
3. *pricing:transaction_type*: "What kind of financial transaction will take place: sale or rent?"

Attributes which are nonsensical when considered in the above context will be ignored (for example: *council_tax_band* will be ignored if the *location:country_code* indicates an overseas (non-UK) listing).

Message attribute	Requirements	Type	Description	Example
accessibility		boolean	Whether the property includes accessibility features for disabled people. We recommend that the accessibility improvements be described in more detail via *feature_list* or *detailed_description*.	true
administration_fees		string	A description of the estate agent's administration fees.	"£10 per person."
annual_business_rates		number	The annual business rate (UK national non-domestic tax rate) applicable to this property.	150.00
areas	Best practice (commercial); Mandatory (when providing *pricing:price_per_unit_area*)	*areas* object	The internal/external areas of the property.	
available_bedrooms		integer	The number of vacant bedrooms being offered in shared accommodation. Note that	2

			if supplying this attribute, *shared_accommodation* **must** be present and *true*, although the relationship is asymmetric. See *shared_accommodation* for more details.	
available_from_date	Best practice (rent)	datetime	The earliest date that a property can be moved into by the new owner/tenant.	"2010-01-31"; "2010-01-31T12:00:00"
basement		boolean	Whether the property has a basement/cellar.	true
bathrooms	Best practice (residential)	integer	The number of bathrooms (or shower rooms) within the property.	1; 3
bills_included		array of enum	Which utility bills are included in the rental price.	["water"]
branch_reference	Mandatory	string	Your unique identifier for the branch, previously sent via *branch/update:branch_reference*.	"1234"; "kd-789d"
burglar_alarm		boolean	Whether the property has a burglar alarm installed.	true
business_for_sale		boolean	Whether the business currently operating at the premises is included as part of the sale.	true
buyer_incentives		array of enum	The list of applicable incentive schemes available to the buyer of the property.	["shared_equity"]
category	Mandatory	enum	Whether the property is commercial or residential.	"commercial"

central_heating		enum	The extent to which a property has central heating.	"full"
chain_free		boolean	Whether the sale of the property is unencumbered by the current owner needing to purchase another property.	true
commercial_use_classes		array of strings	The commercial use classes for which the property is currently approved.	["A3", "D2"]
connected_utilities		array of enum	A list of utilities which are connected to the property.	["water"]
conservatory		boolean	Whether the property has a conservatory.	true
construction_year		integer	The year the property was constructed.	2010; 1701
content	Best practice	array of *content* objects	A list of media content associated with the listing. (These will be displayed in the order provided.)	
council_tax_band		enum	The UK council tax band that the property belongs to.	"A"
decorative_condition		enum	An indication of the general decorative state of the property.	"good"
deposit		number	The tenancy deposit amount. (This should be in the same currency as *pricing:currency_code*.)	500.00
detailed_description	Mandatory	array of *description* objects	The detailed description of the property. Please see *8. Structuring of detailed descriptions* for additional information, including examples of usage.	
display_address		string	The address string to be displayed on the website. Please note that the display of this	"Barker Road, Sutton Coldfield, Birmingham"

			string is subject to our data validation and presentation rules and that this attribute is likely to be deprecated in future versions of this API.	
double_glazing		boolean	Whether the property has double glazed windows.	true
epc_ratings		*epc_ratings* object	The Energy Performance Certificate (EPC) ratings.	
feature_list		array of string	A list of important aspects about the property the agent wishes to highlight; "bullet points". Please see *8. Structuring of detailed descriptions* for more information about how these are displayed.	["Newly carpeted throughout", "Remodelled kitchen"]
fireplace		boolean	Whether the property has a room with a fireplace.	true
fishing_rights		boolean	Whether ownership of the property also grants the owner the right to fish in waters that cross the property's borders.	true
floor_levels		array of enum	Which floors the property occupies in a multi-storey building. Please see our floor labelling convention.	["ground", 1]
floors		integer	The total number of floors that the property occupies.	2
furnished_state	Best practice (residential rent)	enum	The amount of furnishing provided by the landlord.	"part_furnished"
google_street_view		*google_street_view* object	Position and orientation for Google Street View.	
ground_rent		number	The annual ground rent. (This should be in the same currency as *pricing:currency_code*.)	100.00
gym		boolean	Whether the property has access to a private or communal gym.	true
lease_expiry		*lease_expiry* object	When the leasehold tenure expires.	

life_cycle_status	Mandatory	enum	An indication as to where the listing is in its life cycle, from initial availability, through offers being made, to its legal conclusion. For rent: available -> under_offer -> let_agreed -> let. For sale: available -> under_offer -> sold_subject_to_contract -> sold.	"let_agreed"
listed_building_grade		enum	The grade assigned to the property should it be listed as a building of special architectural or historic interest.	"grade_a"
listing_reference	Mandatory	string	Your unique identifier for the listing. **This should be unique across your entire system, not just for its associated branch (i.e. this is the PRIMARY KEY).** If your identifiers are based on those submitted by the agent themselves then you may wish to concatenate, with an unambiguous separator, your branch reference and their listing reference to ensure uniqueness (e.g. "branch_id|agent_listing_id").	"5678"; "dfhd-kjdf-1"
living_rooms	Best practice (residential)	integer	The number of living rooms within the property.	1
location	Mandatory	*location* object	Information about the location of the property, such as its address.	
loft		boolean	Whether the property has a loft/attic.	true
new_home		boolean	Whether the property is newly built and has had no previous occupants.	true
open_day		datetime	The date of the next open-to-all viewing of the property.	"2010-01-31"
outbuildings		boolean	Whether the property has additional outbuildings, such as stables or a greenhouse.	true
outside_space		array of enum	What outside areas are available.	["private_garden"]

parking		array of enum	What parking facilities are available.	["off_street_parking"]
pets_allowed		boolean	Whether the landlord accepts tenants who have pets.	true
porter_security		boolean	Whether the property resides in a complex or building which has an on-site porter/attendant.	true
pricing	Mandatory	*pricing* object	Information about how the listing is priced.	
property_type	Mandatory	enum or string	The broad architectural type of the property. Please note that a particular property_type is only valid within certain contexts.	"maisonette"
rateable_value		number	The estimated open market annual rental value of the premises (in GBP).	15000.00
rental_term		*minimum_contract_length* object or enum	An indication of the duration of the rental contract.	"short_term"
repossession		boolean	Whether the sale is for a repossessed property.	true
retirement		boolean	Whether the property is for retirees-only.	true
sap_rating		integer	The SAP rating for this property.	85
service_charge		*service_charge* object	The service charge applicable to the property.	
serviced		boolean	Whether the premises are serviced.	true
shared_accommodation		boolean	Whether the listing is for shared accommodation (i.e. the listing is for a room, or rooms, within an existing house/flatshare). If this attribute is *true*, we	true

			recommend also supplying the *available_bedrooms* attribute.	
summary_description		string	A brief summary of the property. This is displayed in search results and recommend that it is limited to 255 characters.	"A well decorated house with lots of space with easy access to nearby shops and public transportation."
swimming_pool		boolean	Whether the property has a swimming pool.	true
tenanted		boolean	Whether the property being sold is currently occupied by renting tenants.	true
tenant_eligibility		*tenant_eligibility* object	The eligibility of potential tenant classes that can apply (e.g. students only).	
tennis_court		boolean	Whether the property has a tennis court.	true
tenure		enum	The tenure of the property.	"share_of_freehold"
total_bedrooms	Best practice (residential)	integer	The total number of bedrooms within the property. When describing shared accommodation, see also *available_bedrooms*.	3
utility_room		boolean	Whether the property has a utility room.	true
waterfront		boolean	Whether the property is close by to a large body of water such as the sea, a lake or river.	true
wood_floors		boolean	Whether the property has rooms with exposed wooden floors.	true

5.3 *listing/delete* method

This method allows you to remove a listing from a branch's inventory list.

Message attribute	Requirements	Type	Description	Example
listing_reference	Mandatory	string	Your unique identifier for the listing that you wish to remove, previously sent via *listing/update:listing_reference*.	"1234"; "kd-789d"
deletion_reason		enum	Why the listing is being deleted.	"withdrawn"

5.4 *listing/list* method

Because of the incremental nature of the service it is possible for the data that ZPG has to drift relative to yours (because of network problems or uncaught errors, for example). It is recommended that you periodically check the synchronisation of data between our system and yours. The *listing/list* method allows you retrieve a summary of the listing inventory for a branch and their state.

Message attribute	Requirements	Type	Description	Example
branch_reference	Mandatory	string	Your unique identifier for the branch, previously sent via *listing/update:branch_reference*.	"1234"; "kd-789d"

The response will include the listing inventory for the branch as an array of:

Response attribute	Type	Description	Example
listing_reference	string	Your unique identifier for the listing, previously sent via *listing/update:listing_reference*.	"1234"; "kd-789d"
listing_etag	string	This is the string that you specified in the *ZPG-Listing-ETag* HTTP header when you last called *listing/update*, referring to *listing_reference*. If the value	"18ab1f2e4c8ed62882fa57380c9b7b026413a663"

		is not the same as the one you expect then there is a synchronisation issue and you should call *listing/update* again to refresh the data that ZPG has.
url	string	A link to the listing's detail page. (When using the *sandbox* environment this links to a preview page; when using the *live* environment it is a redirect to the detail page on Zoopla's website.)

Please see *10. Workflow overview* for additional information about how to use this method.

6. Objects

If you include an attribute in your method call that uses an object to represent it, a valid object **must** be supplied. When discussing objects, a constituent attribute described as being *mandatory* is with reference to how to produce a valid object; it does **not** imply that the object, nor the attribute that uses that object, is mandatory when calling a particular method. Please refer to the method itself to see whether a particular attribute is mandatory.

6.1 *area* object

This describes a single area. It is a constituent of a number of other objects. This should not be confused with the *areas* object.

Object attribute	Requirements	Type	Description	Example
value	Mandatory	number	The area value.	54.5
units	Mandatory	enum	The units of measurement used.	"sq_metres"

6.2 *areas* object

The internal/external areas of the property.

Object attribute	Requirements	Type	Description	Example
external		*min_max_area* object	The external area of the property.	
internal	Mandatory (when specifying *pricing:price_per_unit_area*)	*min_max_area* object	The internal area of the property.	

6.3 *content* object

A *content* object specifies a single piece of media content associated with the listing, such as an image or brochure. The *listing/update:content* attribute is an array of these *content* objects, which will be displayed in the order provided. Please see *9. Retrieval of media content* for more information about how the content will be retrieved from the specified URLs.

Object attribute	Requirements	Type	Description	Example
url	Mandatory	string	The URI-encoded URL for the media content to download.	"http://www.estateagentltd.co.uk/properties/images/1234.jpg"
type	Mandatory	enum	The content type.	"image"
caption		string	A short description of the content. This will be displayed alongside the content or, where appropriate, used as the hyperlink text.	"The entrance hall."

6.4 *coordinates* object

This object describes the geographic location of a point on the Earth's surface using the latitude/longitude system. We recommend you store latitude/longitude with a precision of **at least 5 decimal places**, which allows for locating a point to within approximately 1 metre.

Object attribute	Requirements	Type	Description	Example
latitude	Mandatory	number	The latitude, measured in degrees, of the property.	-90.000000; 54.123456; 90.000000
longitude	Mandatory	number	The longitude, measured in degrees, of the property.	-180.000000; 119.265984; 180.000000

6.5 *description* object

The description is expressed as an array of *description* objects, each of which represent a paragraph or section. For additional information about how to use this object, including examples, please see section *8. Structuring of detailed descriptions*.

Object attribute	Requirements	Type	Description	Example
heading	See below	string	The title of the section.	"Kitchen"
dimensions		*dimensions* object or string	The dimensions of the room, when the section is describing a individual room. (The *dimensions* object is the preferred and recommended representation because the string variant is subject to our data validation and presentation rules and so may not be displayed.)	
text	See below	string	The main text for this section.	

Requirements: you must provide at least one of *heading* or *text* per object, whilst *dimensions* is optional (but contingent on the presence of a *heading*).

6.6 *dimensions* object

Information about the dimensions of a room.

Object attribute	Requirements	Type	Description	Example
length	Mandatory	number	The length of the room.	12
width	Mandatory	number	The width of the room.	10
units	Mandatory	enum	The units of measurement used.	"feet"

This will be displayed as "*length* x *width*" with appropriate unit markers. Traditionally, *length* is greater than *width*, but this is not required.

6.7 *epc_ratings* object

The Energy Performance Certificate (EPC) provides information about the energy efficiency and environmental impact of the property.

Object attribute	Requirements	Type	Description	Example
eer_current_rating		integer	The current Energy Efficiency Rating (EER).	80
eer_potential_rating		integer	The potential Energy Efficiency Rating (EER).	95
eir_current_rating		integer	The current Environmental Impact (CO_2) Rating (EIR).	50
eir_potential_rating		integer	The potential Environmental Impact (CO_2) Rating (EIR).	100

6.8 *google_street_view* object

Google Street View provides users with the ability to virtually explore outdoor areas. This is becoming increasingly valuable to users, who are able to, for example: plan routes to bus stops; see which services are provided by local shops.

Object attribute	Requirements	Type	Description	Example
coordinates	Mandatory	*coordinates* object	The coordinates of the point where StreetView should be initially located.	
heading	Mandatory	number	The angle of rotation to the right of the view, measured in degrees, relative to north. Also known as compass heading, or bearing.	0 (north); 74.5; 90 (east); 360 (north)
pitch	Mandatory	number	The angle of rotation up/down of the view, measured in degrees, relative to the horizon.	-90 (straight down); 5.4; 90 (straight up)

6.9 *lease_expiry* object

The expiry of leasehold tenure can be expressed as the actual expiry date or the number of years remaining on the lease.

Object attribute	Requirements	Type	Description	Example
expiry_date	Mandatory	string	The date when the lease expires. Note that this is a date-like string whose validation is not as restrictive as the *datetime* used for other attributes. It may be any of the following formats: *YYYY-MM-DD*; *YYYY-MM*; *YYYY*.	"2014-01-31"; "2014-01"; "2014"

or

Object attribute	Requirements	Type	Description	Example
years_remaining	Mandatory	integer	The number of years remaining on the lease.	99

6.10 *location* object

Providing an accurate address and location for the property is incredibly important because a large number of existing and future features on our websites depend on it. For example: which council is responsible for the maintaining and taxing the local area; or average travel times between the property and other locations of interest to the user.

Object attribute	Requirements	Type	Description	Example
property_number_or_name	Best practice; mandatory when not providing *street_name*.	string	The public designation of the property.	15; "14A"; "Flat B, 41"; "The Wildings"
street_name	Best practice; mandatory when not providing *property_number_or_name*.	string	The name of the road on which the property is principally adjacent.	"Barker Road"; "Chestnut Street"
locality		string	The familiar name of the area as it is referred to by local residents. This is usually a traditional, historic name and may refer to an aspect of the area which has ceased to exist.	"Sutton Coldfield"; "North Beach"
town_or_city	Mandatory	string	The nearest large urban area to the property. This is usually included in the property's postal address and sometimes referred to as "post town". Note that this does not imply that the property resides within the boundaries of this urban area (for example, it might be in an outlying village - which would be referenced via the *locality*).	"Birmingham"; "San Francisco"

county		string	The largest territorial area division within the country which the property resides in. (Synonymous with e.g.: province; principality.)	"West Midlands"; "California"
postal_code	Mandatory (UK)	string	The postal area code issued by the primary postal service in the country. For example, for the UK, this would be Royal Mail's postcode; for the US, the United States Postal Service's ZIP code.	"B19 4JY"; "94112"
country_code	Mandatory	string	The ISO 3166-2 (preferred) or ISO 3166-1 alpha-2 country code.	"GB-BIR"; "US"
coordinates	Best practice	*coordinates* object	The geographic location of the property.	
paf_address		*paf_address* object	Royal Mail's address reference.	
paf_udprn		string	Royal Mail's Unique Delivery Point Reference Number (UDPRN).	"00001234"

Note that there is a hierarchical order to the address attributes, so in order of increasing size:

- *property_number_or_name*
- *street_name*
- *locality*
- *town_or_city*
- *county*
- *country_code*

If your software does not currently support the separation of the *property_number_or_name* from *street_name* (i.e. they are stored as a single string value; for example: "29 Acacia Road"), then please supply that string via *street_name*.

6.11 *minimum_contract_length* object

The *rental_term* attribute can be specified with a *minimum_contract_length* object or, if the contract length is not currently known, by an enum that roughly indicates its likely duration (e.g. "short_term").

Object attribute	Requirements	Type	Description	Example

minimum_length	Mandatory	number	The minimum duration of the contract.	6
units	Mandatory	enum	The units of measurement used.	"months"

6.12 *min_max_area* object

The *min_max_area* object defines an area range. Where only one is specified, we will regard them both as having the same value - i.e. a fixed area.

Object attribute	**Requirements**	**Type**	**Description**	**Example**
minimum		*area* object	The minimum area.	
maximum		*area* object	The maximum area.	

6.13 *paf_address* object

Royal Mail's addressing scheme is known as Postcode Address File (PAF). Many third-party systems use this to facilitate the selection of an address from an initial postcode. Per Royal Mail's specification: "Each address on PAF® has an eight-digit number associated with it - the Address Key. This number in conjunction with the Organisation Key and Postcode Type identifies the address."

Object attribute	**Requirements**	**Type**	**Description**	**Example**
address_key	Mandatory	string	The 8-digit Address Key.	"02341509"
organisation_key	Mandatory	string	The 8-digit Organisation Key.	"00000000"; "00001150"

postcode_type	Mandatory	enum	The Postcode Type.	"L"; "S"

Note that Royal Mail's UDPRN (see *location:paf_udprn*) is generally preferred to *paf_address* because it is more stable with respect to changes in property utilisation (e.g. commercial becoming residential).

6.14 *price_per_unit_area* object

This object describes a price which is provided as a function of floor area.

Object attribute	**Requirements**	**Type**	**Description**	**Example**
price	Mandatory	number	The per unit area price.	100.00
units	Mandatory	enum	The units of measurement used.	"sq_feet"

6.15 *pricing* object

The *pricing* object's structure is determined by whether its component attribute *transaction_type* indicates that the listing is for *sale* or *rent*. Please note that we do not currently support multiple pricing, nor multiple transaction_types, for a single listing. If you do have multiple pricing models, perhaps due to a range of incentive schemes being available, then you should supply the non-subsidised price in the *pricing* object and indicate available incentive plans via *listing/update:buyer_incentives*.

Object attribute	**Requirements**	**Type**	**Description**	**Example**
transaction_type	Mandatory	enum	The type of financial transaction for this listing.	"sale"

currency_code	Mandatory	string	The ISO 4217 currency code for the stated price. Other monetary attributes (e.g. deposit) also use this as their currency.	"GBP"; "JPY"
price	Mandatory (residential)	number	The non-subsidised price of the listing. Note that the price should be quoted in its natural currency (i.e. the currency the vendor/landlord has specified), which is usually the official currency of the country that the property resides in. To be clear: this value should **not** be the result of a currency conversion.	100000.00; 250
price_per_unit_area	Best practice (commercial)	*price_per_unit_area* object	The price quoted as a function of floor area. This is generally only used with commercial properties.	
rent_frequency	Mandatory (rent)	enum	The frequency with which rent is required to be paid.	"per_week"
price_qualifier		enum	Additional information about the price which requires emphasis, usually for legal reasons.	"guide_price" (auction); "fixed_price" (Scotland)
auction		boolean	Whether the sale will be transacted by an auction.	true

Because pricing is so important there are a number of schema-enforced dependencies for some of the *pricing* object's attributes. In particular:

- When *transaction_type* = "rent", *rent_frequency* **must** be specified.
- When providing a *price_per_unit_area* you **must** also provide an *areas* attribute with an *internal* area.
- *price_qualifier* = "non_quoting" is **only** valid for *location:country_code* = "gb" and *category* = "commercial". When stating *price_qualifier* = "non_quoting" then you **must not** provide *pricing/price* **nor** *pricing/price_per_unit_area*.

Note that you do **not** have to supply both *price* and *price_per_unit_area*, although that is allowed. Do not calculate one price attribute from the other.

6.16 *service_charge* object

This object describes any ongoing service charges which are applicable. Note that you can only specify one of *per_unit_area_units* or *frequency*.

Object attribute	Requirements	Type	Description	Example
charge		number	The service charge amount.	100
per_unit_area_units		enum	When the charge is a function of unit area (usually for commercial properties), the units of area measurement.	"sq_feet"
frequency		enum	When the charge is a fixed amount (usually for residential properties), the frequency with which the service charge is required to be paid.	"per_year"

6.17 *tenant_eligibility* object

This allows you to specify the eligibility of various classifications of applicant for a tenancy. Note that you do not have to specify eligibility for all classifications.

Note: as of April 2019, the *dss* attribute in this object is ignored. Please see "Zoopla to end 'No DSS' wording in rental adverts" for more details.

Object attribute	Requirements	Type	Description	Example
dss		enum	The eligibility for applicants who are receiving Housing Benefit.	"excluded"
students		enum	The eligibility for applicants who are students.	"only"

7. Responses

After processing your message we will send you a response that includes a standard HTTP status code and a JSON object in the content body that provides more detailed feedback.

7.1 Success

A response with a *200 OK* HTTP status code indicates a successfully processed message. For convenience the JSON response object includes the primary identifiers provided in the original message. The methods respond as follows:

7.1.1 branch/update

```
{
    "status" : "OK",
    "branch_reference" : "<message's branch_reference>",
    "new_branch" : true
}
```

If this was the first time that we have received a message about *branch_reference* then *new_branch* will be *true*, otherwise *false*.

7.1.2 listing/update

```
{
    "status" : "OK",
    "listing_reference" : "<message's listing_reference>",
    "listing_etag" : "<message's ZPG-Listing-ETag header>",
    "url" : "<URL for this listing>",
    "new_listing" : true
}
```

If this was the first time that we have received a message about *listing_reference* then *new_listing* will be *true*, otherwise *false*.

In both *sandbox* and *live* environments the response contains a *url* attribute:

- in the *sandbox* environment, this is a link to a web page where you can preview how the listing will be displayed on our websites.
- in the *live* environment, this is a link which, once your listing has been processed and published, will redirect to the detail page of this listing on Zoopla.

7.1.3 listing/delete

A *listing/delete* message will return a *200 OK* if the *listing_reference* is no longer active on our system. If your message deleted *listing_reference* then you will receive:

```
{
    "status" : "OK",
    "listing_reference" : "<message's listing_reference>"
}
```

whereas if the *listing_reference* was already inactive, or previously unknown to us, then you will receive:

```
{
    "status" : "UNKNOWN",
    "listing_reference" : "<message's listing_reference>"
}
```

The effect is the same but you may wish to investigate these cases since this may be indicative of a synchronisation issue.

7.1.4 listing/list

listing/list allows you to retrieve listing inventory information for a branch:

```
{
    "status" : "OK",
    "branch_reference" : "<message's branch_reference>",
    "listings" : [
        {
            "listing_reference" : "<listing_reference 1>",
            "listing_etag" : "<ZPG-Listing-ETag 1>",
            "url" : "<URL for this listing>"
        },
        {
            "listing_reference" : "<listing_reference 2>",
            "listing_etag" : "<ZPG-Listing-ETag 2>",
            "url" : "<URL for this listing>"
        },
        ...
    ]
}
```

The *listing_etag* attribute contains the value of the *ZPG-Listing-ETag* you provided with your last *listing/update* message relating to *listing_reference*. Using this value you can check the synchronicity of the listing's data between ZPG's system and yours.

The *url* attribute behaves in exactly the same way as that described for the *listing/update* response.

If *branch_reference* has no listings then it will not be treated as an error and an empty list will be returned:

```
{
    "status" : "OK",
    "branch_reference" : "<message's branch_reference>",
    "listings" : []
}
```

7.2 Errors

If there was a problem with the message then the response will include an HTTP status code in the *4xx* range. The accompanying JSON response object will include more detailed information about the error to help you diagnose and fix it.

An error response will always include the following attributes:

Response attribute	Type	Description
error_name	string	A unique string identifier for the error.
error_advice	string	A description of the error and some advice as to how to fix it.

If there are problems parsing your message as JSON, there will also be:

Response attribute	Type	Description
request_content	string	The content body of your request (which should have been a JSON object).

json_validation	string	The reason(s) why *request_content* could not be parsed as a JSON object.

If there are problems determining which schema or method should be used, there will also be:

Response attribute	**Type**	**Description**
method	string	The method your message was sent to.
profile	string	The JSON schema you declared your message would conform to.

If there are problems validating the message against its associated JSON schema, there will also be:

Response attribute	**Type**	**Description**
errors	array of JSON objects	The errors that were generated when validating your message against the schema.
schema	string	The JSON schema your message was validated against.
status	string	This will be set to "FAILURE". (See also the *status* attribute returned by successful responses.)

For example, if you were to send a correctly formatted message to the incorrect method endpoint you would receive:

```
{
    "error_name" : "schema_method_mismatch",
    "error_advice" : "The schema referenced in the Content-Type profile does not match the method endpoint where the message was sent. Please check that you are using the correct message schema and that the message is being sent to the correct endpoint."
    "method" : "/sandbox/v1/branch/update",
    "profile" : "http://realtime-listings.webservices.zpg.co.uk/docs/v1.2/schemas/listing/update.json"
}
```

7.2.1 Referencing of JSON attributes

The most common errors relate to problems with the JSON message itself (for example, invalid data or a missing mandatory attribute). The JSON validator will reference attributes of interest via a hierarchical path, with "#/" denoting the root level of the object:

```
#/category


{

    "category" : "residential"

}

#/location/coordinates/latitude


{

    "location" : {

        "coordinates" : {

            "latitude" : 120.659815,

            "longitude" : -7.846114

        }

    }

}
```

Note that, in JSON, array elements are referenced using a zero-based index:

```
#/an_array/1


{

    "an_array" : [

        "element 0 => #/an_array/0",

        "element 1 => #/an_array/1",
        ...
    ]
}
```

7.2.2 JSON schema validation

The JSON validator's error messages should be self-explanatory when describing problems about an individual attribute and its dependencies.

The JSON validator will attempt to validate your message against the schema by interpreting the listing's data in all possible contexts. If it is unable to find a successful match then it will return all the errors it encountered. Recall that there are three principal attributes that are used to interpret the listing's data (#/location/country_code, #/category, #/pricing/transaction_type) so, naturally, there are some schema-enforced rules related to these attributes. If one of these attributes has a problem then it may trigger multiple errors (each produced by considering the message in a different context), which may make it appear as though there are more problems than there actually are, so we recommend you investigate errors for these attributes first. For example, consider the following *pricing* object:

```
{
    "pricing" : {
        "transaction_type" : "rent",
        "currency_code" : "GBP",
        "price" : 10000
    },
    ...
}
```

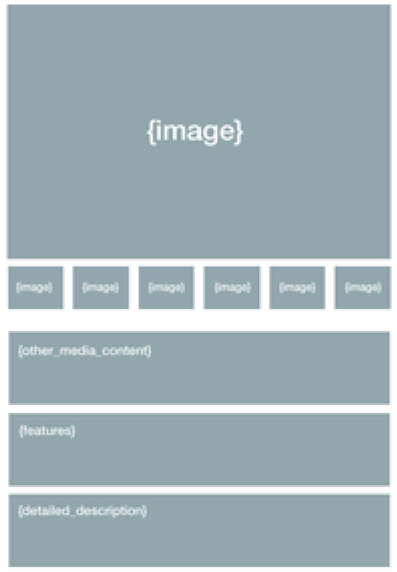
This produces two errors:

```
{
   "error_name" : "json_does_not_validate",
   "error_advice" : "The JSON message does not conform with the schema. Please check the 'errors' array for details."
   "errors" : [
      {
         "message" : "'rent_frequency' is a required property",
         "path" : "#/pricing"
      },
      {
         "message" : "'rent' is not one of ['sale']",
         "path" : "#/pricing/transaction_type"
      }
   ],
   "schema" : "http://realtime-listings.webservices.zpg.co.uk/docs/v1.2/schemas/listing/update.json",
   "status" : "FAILURE"
}
```

The validator has attempted to interpret the message as a *rent* listing and discovered that the required *rent_frequency* attribute is missing. Having failed to validate as a *rent* listing, the validator then attempted to interpret the message as a *sale* listing instead. That failed too because the *transaction_type* is incorrect for a *sale* listing. In order to fix this error the sender would have to decide what the correct *transaction_type* was and then either change the *transaction_type* or provide the *rent_frequency*.

8. Structuring of detailed descriptions

Providing a comprehensive description of the listing is very important: it is the estate agent's primary opportunity to convince the user that this property is exactly what they are looking for. Our detail page is divided into the following sections:



- Images: see *9. Retrieval of media content*.
- Other media content: see *9. Retrieval of media content*.
- Features: a simple bulleted list of important or interesting aspects of the property. These are specified via *listing/update:feature_list*.
- Detailed description: the topic for the rest of this section.

The description should provide an overview of the property and, if relevant, its constituent rooms. Presentation is also very important, from both structural and visual perspectives, as it allows the user to quickly find information about the property that they are specifically interested in. Unfortunately there is no standard as to how to structure descriptions, so our *detailed_description* attribute has been designed to be as flexible as possible, whilst providing a framework to encourage good practice.

The *detailed_description* is an array of *description* objects, each of which can have the following attributes:

- *heading*
- *dimensions*
- *text*

These attributes populate a template with this structure:

heading (dimensions)
    text

You must provide at least one of *heading* or *text* per object, whilst *dimensions* is optional (but contingent on the presence of a *heading*). The *heading* will be styled appropriately. By specifying a sequence of these objects, each using various combinations of *header*, *dimensions* and *text*, a wide variety of complex layouts can be achieved. A number of common scenarios are discussed below.

The *text* can include HTML. However, we only support a limited number of HTML tags in order to prevent unforeseen and undesirable interactions with our own HTML layout and styling. We currently only support:

- `<br>` (line break)
- `<p>` (paragraph)
- `<strong>` or `<b>` (highlighted)
- `<em>` or `<i>` (emphasised)
- `<u>` (underlined)
- `<ul>` and `<li>` (unordered list)

Note that any external links and contact details will be removed from the description text.

8.1 Example: a single block of text

If your system only stores a single block of descriptive text then you would use one object with a single *text* attribute:

```
"detailed_description" : [

    {

        "text" : "This is a single block of text."

    }

]
```

    This is a single block of text.

Because there is no information about the structure of the text when provided this way, the author may need to embed HTML tags in order to achieve the layout they desire.

8.2 Example: sections

If your descriptions are split into sections then you would use multiple objects, one per section, specifying a *heading* for each:

```
"detailed_description" : [
    {
        "heading" : "Section one",
        "text" : "The text for section one."
    },
    {
        "heading" : "Section two",
        "text" : "The text for section two."
    }
]
```

Section one
The text for section one.

Section two
The text for section two.

Note that sections do not necessarily have to have a heading, in which case they will appear as paragraphs. A common use-case is to have an un-headed introductory paragraph, followed by headed sections:

```
"detailed_description" : [
    {
        "text" : "Introductory overview of the property."
    },
    {
        "heading" : "Section one",
        "text" : "The text for section one."
    },
    {
        "heading" : "Section two",
        "text" : "The text for section two."
    }
]
```

Introductory overview of the property.

Section one
The text for section one.

Section two
The text for section two.

8.3 Example: sections for individual rooms

The most common (residential) use-case is an overview paragraph followed by sections that describe individual rooms. This is exactly the same as the sectioned description discussed previously but with the addition of the *dimensions* attribute:

```
"detailed_description" : [
    {
        "text" : "Introductory overview of the property."
    },
    {
        "heading": "Room one",
        "dimensions": {
            "length": 12.2,
            "width": 10,
            "units": "metres"
        },
        "text" : "Information about room one."
    },
    {
        "heading": "Room two",
        "dimensions": "10m x 8.2m",
        "text": "Information about room two."
    }
]
```

Introductory overview of the property.

Room one (12.2m x 10.0m)
Information about room one.

Room two (10m x 8.2m)
Information about room two.

If you do not have a description for a room (although that's not recommended) then you can just specify a heading in order to maintain consistency of presentation:

```
"detailed_description" : [
    {
        "text" : "Introductory overview of the property."
    },
    {
        "heading": "Master bedroom",
        "dimensions": {
            "length": 20.1,
            "width": 15.2,
            "units": "feet"
        },
        "text" : "A large bedroom with lots of storage space and an outlook over the valley."
    },
    {
        "heading" : "Bedroom",
        "dimensions" : "10' x 8'"
    },
    {
        "heading" : "Bedroom",
        "dimensions" : "9' x 9'"
    },
    {
        "heading" : "Kitchen",
        "text" : "The kitchen was recently refitted."
    }
]
```

Introductory overview of the property.

Master bedroom (20.1' x 15.2')
A large bedroom with lots of storage space and an outlook over the valley.

Bedroom (10' x 8')

Bedroom (9' x 9')

Kitchen
The kitchen was recently refitted.

Note that in the above example the dimensions of the kitchen are missing. This is permissible but not good practice when discussing individual rooms.

9. Retrieval of media content

A listing's content is not transferred to ZPG via the Real-time Listings Service directly. Instead the *listing/update* method includes a *content* attribute for you to specify a list of URLs from where that content can be obtained. It is likely that you already have a web server hosting this content, so this mechanism allows you to leverage your existing infrastructure to make the transfer of these assets more efficient. We support web servers using either HTTP or HTTPS.

Upon receipt of a *listing/update* message, we will request all the URLs specified, with appropriate HTTP headers to make the request conditional. So the first time a particular URL is mentioned we will download that content. For subsequent *listing/update* messages which mention that URL, we will modify our request to include the first of the following HTTP headers that we have an accompanying value for:

- *If-None-Match* with the value from the HTTP *ETag* response header your web server returned when we previously requested the URL.
- *If-Modified-Since* with the date from the HTTP *Last-Modified* response header your web server returned when we previously requested the URL.

In this way we will only download that content again when it changes. We recommend that your system support ETags in addition to last-modified dates. Whilst the generation of the ETag is not defined by the HTTP specification, many systems use a hashing algorithm (e.g.: MD5; SHA-1) since they are only generated from the binary data itself (so file system changes do not affect it), and the likelihood of two files producing the same hash is very small. This provides a high level of confidence that when the hash changes, the content is different; and vice versa.

Your web server probably behaves correctly when sent these headers already, particularly if you are using a commercial content delivery network. However, we nevertheless recommend you check its behaviour to be sure - especially since web browsers use the same mechanism to maintain their local data cache, so a misconfiguration could be enhancing your network traffic significantly.

9.1 Downloading your content

Our download process will identify itself as:

```
User-Agent: Zoopla Property Group automated download
```

If your content will be hosted on a private web server and therefore wish to use firewall rules to restrict access, please whitelist the following IPs:

- 31.221.120.234
- 52.16.166.50
- 52.18.106.77
- 52.18.124.138
- 54.77.228.44
- 54.171.72.173
- 54.171.126.99
- 54.229.57.122
- 54.229.75.214

9.2 MIME types

We only support web-safe formats for content. Acceptable MIME types are:

- application/pdf
- image/gif
- image/jpeg
- image/png

9.3 Images

Images can strongly influence a user's impression of a listing so it is important to provide good quality images which are representative of the property.

- Please make available the original image (or the largest downscaled variant of the original if that is not possible). When resizing images for use on our websites and mobile apps, we **only** downscale because upscaling results in inferior quality images. Be careful not to accidentally submit URLs for image thumbnails.
- The first image should ideally be of the exterior of the property.
- We recommend including images for features and rooms which are highlighted in the description (e.g. a conservatory, or a recently remodelled kitchen).
- Do not send agent logos or placeholder images. These will removed from our websites.
- We recommend images have a size of at least 1280x960. Please note that we provide users access to original images and they should display larger than the default view on our listing detail page. Large images are particularly important for mobile devices when viewed full screen. For reference:

Mobile device	**Screen resolution**
iPhone 3 (and previous generations)	480 x 320
iPhone 4	960 x 640
iPhone 5 (all variants)	1136 x 640
iPhone 6	1334 x 750
iPhone 6 Plus	1920 x 1080
Google Nexus 5	1920 x 1080
HTC One	1920 x 1080
Samsung Galaxy 5	1920 x 1080
iPad Mini	1024 x 768

iPad Retina display	2048 x 1536
iPad Air	2048 x 1536

9.4 Floor plans

Floor plans are also of great interest to our users when attempting to visualise the property. It is important to provide high resolution images for floor plans because otherwise text can become illegible.

10. Workflow overview

This section describes the workflow for how the ZPG Real-time Listings Service should be used and how to ensure that the data ZPG has is synchronised with your own.

10.1 Initial upload

10.1.1 A new estate agent is added to your system

For each branch that the estate agent owns, you send a *branch/update*.

10.1.2 Upload listing data

For each added branch, find the set of listings that they are responsible for. For each listing, send a *listing/update* message.

Recall that *listing/update* requires that it be accompanied by a *ZPG-Listing-ETag* HTTP header. The value of *ZPG-Listing-ETag* should be an identifier on your system which allows you to accurately determine that when two such values are different, the overall state of the listing is also different. Depending on how you track changes to your data (and how it is represented in a *listing/update* message) an auto-incrementing revision counter, or possibly a last-modified date **may** be suitable. However, we would recommend using a checksum based on the *listing/update* message itself:

- Canonicalise the JSON *listing/update* message. (This is the default for many JSON software libraries but some require you explicitly perform that action, or possibly set a configuration flag to enable it.) This process ensures that the message is represented in a consistent fashion, regardless of the order in which you populated attributes within the message.
- Produce a checksum hash (e.g.: MD5; SHA-1) of the canonicalised JSON message. Store this value on your system and send it as the *ZPG-Listing-ETag*.

10.2 Incremental changes

10.2.1 Amendments are made to a listing on your system

Send a *listing/update* message which details the complete state of the updated listing.

10.2.2 A listing ceases to be valid

Send a *listing/delete* message to delete it from our system.

10.3 Periodically check synchronisation

Because of the incremental nature of the updating process, should a message be lost (e.g. due to a networking problem), then the data that ZPG has will be different to yours and will need correcting. We therefore recommend that you periodically, once a week say, check the synchronisation of data:

10.3.1 Ask about the state of a branch's listings on ZPG's system

For each branch you maintain, send a *listing/list* request. In the response, for each of the branch's listings, we will provide you with your *listing_reference* and the *ZPG-Listing-ETag* that we last received for that listing.

10.3.2 Add listings

If *listing/list* is missing a particular listing, use *listing/update* to create it.

10.3.3 Remove listings

If *listing/list* includes a listing that is no longer valid, use *listing/delete* to remove it.

10.3.4 Update listings

For each listing that is common to both systems, compare the value of the *ZPG-Listing-ETag* with that of the relevant field in your system:

- If the values are identical then the data is the same on both systems and you do not need to do anything. This should be the normal case.
- If the values are different then ZPG's data is stale and you should use *listing/update* to refresh it.

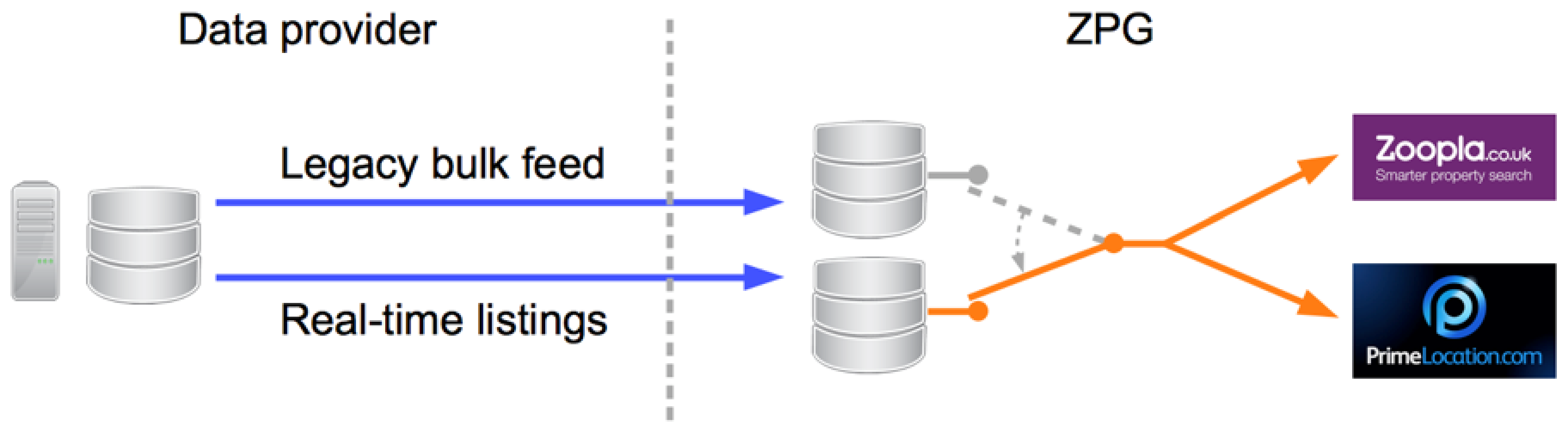
11. Integration

11.1 Transitioning from an existing bulk data feed

If you currently send ZPG a bulk data feed then it may be useful to understand how the transition between that feed and the ZPG Real-time Listings Service will be handled before reading about how to build your implementation.

Data feeds received by ZPG are kept separate from each other: your data never interacts with anyone else's (nor theirs with yours) and should you send us multiple feeds they will not interact with each other either. We then choose which subsets of these data feeds will be shown on our websites (on a per-branch basis).

You should therefore design your system so that you can update ZPG via your bulk data feed and the Real-time Listings Service **concurrently**, thus maintaining two equivalent data sets simultaneously:



Once your Real-time Listings Service integration is complete and running smoothly, ZPG can then simply switch its websites over from using one data set to the other. This mechanism preserves your data's integrity; simplifies testing; makes it easier to QA attribute coverage and equivalence between both systems; and allows for a quick transition to having real-time behaviour for all branches simultaneously - if desired. (It also provides the security of a viable rollback option in the unlikely event that a major problem occurs during the migration.)

11.2 Test environment

11.2.1 Sandbox environment and API

This is an exact copy of the *live* environment but is isolated from our public websites. Any test messages you send here will only affect the data you previously sent to the *sandbox* environment; you cannot "break" anything. There are no restrictions on the number of messages you can send or listings you can store. Indeed, as part of your final testing you should maintain a copy of all your listing data in the *sandbox* environment (just as you would do in the *live* environment) in order to demonstrate that the system is mature and suitable for sending data to our public websites in real-time.

11.2.2 Preview listing web page

You can more easily visualise a particular listing's data via our listing preview page. Please note that whilst the preview is styled in a similar fashion to our websites it is only intended for illustrative purposes. The URL for a particular listing's preview page can be found in the responses from *listing/update* and *listing/list*.

The preview page will indicate whether we will be able to download the associated image and other content URLs, and for any images we could not access an "awaiting photos" placeholder will be shown. Please see *9. Retrieval of media content* for more information about how we will retrieve your content and the set of IPs our download requests will originate from.

11.3 Building your implementation

Whilst you do not have to follow this exactly, we suggest:

1. Check that your current image-hosting web servers respect *If-None-Match* and *If-Modified-Since* headers (see: *9. Retrieval of media content*). If they don't then this could be having a negative impact on your infrastructure already.
2. Obtain a *signed certificate* to allow HTTPS access to the ZPG Real-time Listings Service (see: *3. Security and authentication*).
3. Write code to communicate with the ZPG Real-time Listings Service. This can be facilitated by initially sending the example messages, rather than writing your own.
4. Write code to generate and validate *branch/update* and *listing/delete* messages. These have relatively simple message formats which will make it easier for you to interpret schema validation errors.
5. Write code to generate and validate *listing/update* messages. (You can preview how we will display that message by using the link provided in the *url* attribute of *listing/update*'s response.)
6. Determine if your system has a suitable field to use with *ZPG-Listing-ETag*. If there is nothing suitable then implement a hashing system as suggested in *10. Workflow overview*.
7. Having uploaded some listings, you can then use *listing/list* to see what was uploaded and can then implement your synchronisation mechanism, such as the one discussed in *10. Workflow overview*.

11.4 Testing

Once you have a working system, it is important to thoroughly test it:

1. Check that the *ZPG-Listing-ETag* is not using a temporary placeholder value and that it behaves correctly.

2. Send actual listing data. During initial development you may have used our example messages, or written your own, which may not cover all the use-cases present in your actual data. We therefore recommend you send increasingly larger sets of real data to uncover any remaining issues:
    - A small random set of listings.
    - All listings for a branch.
    - All listings for a random set of branches.
    - All listings on your system.

   Check that these display correctly on their preview pages (see the *url* attribute in the responses for *listing/update* and *listing/list*) and that images could be downloaded by us.

3. If you are able to upload all your listings without encountering any errors, then you should maintain those listings for a period of time (we suggest a week or two) just as you would in the live environment. In other words, perform a "dry run":
    - Send *listing/update* messages for all listings on your system. Find and remove previously sent test listings by utilising *listing/list* and *listing/delete*. (These steps are most easily achieved by running your synchronisation process.)
    - Send *listing/update* and *listing/delete* messages in real-time as agents edit their data.
    - Regularly check the synchronisation of data between your system and ours via *listing/list*. We suggest you also check the preview pages for a random sample of listings too in order to ensure that the data is indeed synchronised correctly.

11.5 Migrating to the live environment

It is expected that your system will communicate with the sandbox environment using actual data on a continual basis for at least a week, in order to prove that the combined system is stable, produces no unexpected errors and has no synchronisation issues. When you are happy that this is the case and it is therefore suitable to become the sole means of sending us listing data, please let us know so that you can be granted access to the live environment. The migration procedure is very similar to that used when testing in the sandbox environment:

- Provide us with the details of all your branches. Please contact us if you do not wish to supply them via the *branch/update* method.
- Send *listing/update* messages for all listings on your system to the *live* environment. (This is most easily achieved by running your synchronisation process. Please let us know when this initial upload has been completed.)
- Send *listing/update* and *listing/delete* messages in real-time as agents edit their data.
- Periodically check the synchronisation of data between your system and ours via *listing/list*.

Because the updating process will already have been shown to work correctly over an extended period in the sandbox environment, there should be no difficulties when using the live environment.

If you currently send ZPG a bulk data feed, please continue sending this to us - even after you start sending real-time updates to the live environment. Once your initial upload of data to the live environment has completed, please let us know. We will then change our branch configuration so that we stop using the listing data in your old bulk data feed and start using the data sent via the ZPG Real-time Listings Service instead. When we complete this branch migration process, we will inform you so that you can then terminate your old bulk data feed.

12. Conclusion

We hope that having read this documentation you have decided to integrate the ZPG Real-time Listings Service with your system.

If you have any questions about this documentation, schemas or the service in general, please contact our Support Team.

Thank you for your commitment to improving the quality of service that our mutual clients and users will receive by integrating your system with the ZPG Real-time Listings Service.

Appendix 1. Additional notes on attributes

A1.1 Boolean attributes

The majority of our boolean attributes are optional. You should only include a boolean attribute where you are confident that the value is correct; do not set it to a default value. The reason for this is that you may convey a meaning that you did not intend. For example if you were to default *pets_allowed* to:

- *true*: potential tenants will complain if the landlord then rejects their application.
- *false*: the landlord will receive fewer enquiries even though they might consider tenants with pets.

Omitting boolean attributes where you do not know the correct value also has the benefit of reducing message sizes and making them easier to read when debugging.

A1.2 Datetime attributes

Dates should be expressed using standard ISO format: *YYYY-MM-DD*. Where appropriate, a time component may be included by appending the date with a literal T character separator, followed by the time as *hh:mm:ss*. Note that the time component should use the local time zone of the property. For example:

- 2014-01-31
- 2014-01-31T12:34:56

A1.3 Free-text string attributes

The quality of text supplied in free-text string attributes can vary considerably. In order to improve the user experience on our websites we may modify the display of these strings in order to present a consistent style and aid comprehension (e.g.: capitalisation; removal of HTML tags). If the quality of the text is particularly poor then, in some circumstances, we may choose not to display it at all (e.g. invalid data for the attribute). Care should therefore be taken at the initial data-entry stage to ensure that the correct attribute is being populated and that free-text intended to be read by the public is maintained appropriately (e.g.: spelling; grammar).

Our schemas use a regular expression to ensure that free-text attributes:

- are neither empty nor only consist of whitespace
- have no leading whitespace
- have no trailing whitespace

Note that carriage returns (`\r`) and newlines (`\n`) are considered whitespace.

These rules are enforced by the schema via a regular expression, which will be included in any validation errors:

```
does not match '^\\S(|(.|\\n)*\\S)\\Z'
```

If you encounter this error you may find it easier to re-read the validation rules above in order to correct the text.

Please refer to 7.2 Errors for more information on error responses.

A1.4 Monetary attributes

All costs/prices should be quoted in the currency specified by *pricing:currency_code*.

A1.5 *display_address*

display_address is an obsolete attribute since it can be constructed from the dedicated address attributes: "*street_name*, *locality*, *town_or_city*". ZPG's websites only use address strings and we therefore expect to deprecate the use of *display_address* in future. We recommend software providers encourage their clients to populate address attributes correctly and provide them with a dedicated field for *property_number_or_name* so that unique property address information will not be included in a constructed address.

A1.6 *floor_levels*

The layout of floors within a multi-storey building is defined as follows:

- *penthouse* (top floor)
- ... (increasing integer values)
- *2*
- *1* (above street level)
- *ground* (at street level)
- *basement* (below street level)

A1.7 *property_type*

There are a lot of terms for describing house types. The distinctions between some of these types are sometimes unnecessary (for example: apartment vs flat) and broader groupings that just convey the core aspects of the architecture are preferred. Consequently we only support a relatively limited number of property types. When deciding which property type to use, you should choose the most applicable property type that we support. If you believe that there is a significant difference between your property type and those that we support, then send your property type as a readable string. In such cases we will display property type as "Property", but the string you supplied will be considered during our periodic review of property types to ensure that we are supporting the needs of our users adequately. (Please note that our supported property_type enums use all-lowercase characters; alternative case-variants will be displayed as "Property".)

Some property_types in your system may be conveyed by the appropriate use of other attributes. For example:

- a "duplex" is a *property_type* = "flat" with *floors* = 2
- a "triplex" is a *property_type* = "flat" with *floors* = 3
- a "penthouse" is a *property_type* = "flat" with *floor_levels* = ["penthouse"]

When *category* = "residential", our schema will reject *property_type* = "studio" if *total_bedrooms* is also present and has a value greater than **1**, as we cannot determine which value is inaccurate; see 7.2.2 JSON schema validation for more information about how we reject ambiguous messages.

Note that the display of the *property_type* is also dependent on the associated *country_code* and *category* of the listing. The schema does **not** enforce these rules. If an inappropriate *property_type* is specified for a particular *country_code-category* combination then it will be displayed as "Property" (residential) or "Commercial Property" (commercial). The list of supported *property_types* and their validity is as follows:

property_type	Display string	UK residential	Overseas residential	UK commercial	Overseas commercial
barn_conversion	Barn conversion	✓	✓		
block_of_flats	Block of flats	✓	✓	✓	✓
bungalow	Bungalow	✓	✓		
business_park	Business park			✓	✓
chalet	Chalet		✓		
chateau	Château		✓		
cottage	Cottage	✓	✓		

country_house	Country house	✓	✓		
detached	Detached house	✓	✓		
detached_bungalow	Detached bungalow	✓	✓		
end_terrace	End terrace house	✓	✓		
equestrian	Equestrian property	✓	✓		
farm	Farm	✓	✓	✓	✓
farmhouse	Farmhouse	✓	✓		
finca	Finca		✓		
flat	Flat	✓	✓		
hotel	Hotel/guest house			✓	✓
houseboat	Houseboat	✓	✓		
industrial	Industrial			✓	✓
land	Land	✓	✓	✓	✓
leisure	Leisure/hospitality			✓	✓
light_industrial	Light industrial			✓	✓

link_detached	Link-detached house	✓	✓		
lodge	Lodge	✓	✓		
longere	Longère		✓		
maisonette	Maisonette	✓	✓		
mews	Mews house	✓	✓		
office	Office			✓	✓
park_home	Mobile/park home	✓	✓		
parking	Parking/garage	✓	✓	✓	✓
pub_bar	Pub/bar			✓	✓
restaurant	Restaurant/cafe			✓	✓
retail	Retail premises			✓	✓
riad	Riad		✓		
semi_detached	Semi-detached house	✓	✓		
semi_detached_bungalow	Semi-detached bungalow	✓	✓		
studio	Studio	✓	✓		

terraced	Terraced house	✓	✓		
terraced_bungalow	Terraced bungalow	✓	✓		
town_house	Town house	✓	✓		
villa	Villa		✓		
warehouse	Warehouse			✓	✓

Appendix 2. Message examples

- Latest version.
- 2015-05-13: v1.0
  - Promoted version 0.1 to 1.0.
- 2014-11-11: v0.1
  - Initial set of message examples.

Appendix 3. Schema changelog

- Latest version.

Date	Version	Schema(s) changed	Change description

2017-06-14v1.2	• *branch/update* • *listing/update*	• Ensured the constraints within the *location* object in the *branch/update* schema matched those in the *listing/update* schema. • Added support for *equity_loan* to *buyer_incentives*. • Removed constraint on the maximum value of all *epc_ratings* object attributes, and *sap_rating*. • Updated the JSON schema draft v4 link on the schemas page.
2015-08-10v1.1	• *listing/update*	• Added support for *per_person_per_week* frequency to *service_charge:frequency* and *pricing:rent_frequency*. • Implemented constraints for *pricing:price_qualifier* = "non_quoting" when *location:country_code* = "gb" and *category* = "commercial".
2015-05-13v1.0	• *branch/update* • *listing/delete* • *listing/list* • *listing/update*	Promoted version 0.1 to 1.0.
2015-05-08v0.1	• *listing/update*	Changed the constraints concerning *location:property_number_or_name* and *location:street_name* so that at least one must be provided.
2015-03-19v0.1	• *listing/update*	Changed the constraints surrounding *shared_accommodation* and *available_bedrooms*.
2015-03-19v0.1	• *listing/update*	Corrected the spelling of *shared_accommodation*.
2015-03-13v0.1	• *listing/update*	Fixed a bug with how *total_bedrooms* and *property_type* = "studio" interact.
2015-01-07v0.1	• *listing/update*	Added *sap_rating* attribute.
2014-12-11v0.1	• *listing/update*	Fixed constraints for residential listings so that *total_bedrooms* and *bathrooms* are not mandatory, as per the documentation.
2014-12-10v0.1	• *branch/update* • *listing/delete* • *listing/list* • *listing/update*	Updated regular expression validation for free-text attributes.
2014-11-21v0.1	• *branch/update*	Updated regular expression validation for attributes containing URL data so they are consistent: no whitespace is allowed. Where necessary, whitespace and other meta characters should be URI-encoded.

2014-11-14v0.1	• *listing/update*	Updated the list of supported *property_types* so they are consistent with the documentation.
2014-11-10v0.1	• *listing/update*	• Fixed typo in attribute name *administration_fees* • Added *document* to the list of valid *type* values for a *content* object.
2014-09-12v0.1	• *branch/update* • *listing/delete* • *listing/list* • *listing/update*	Initial draft.

Appendix 4. Documentation changelog

- Latest version.

- 2019-05-20: v1.2
    - Added a note explaining that the *dss* attribute in the tenant eligibility object is now ignored.

- 2017-06-14: v1.2
    - Ensured the constraints within the *location* object in the *branch/update* schema matched those in the *listing/update* schema.
    - Added support for *equity_loan* to *buyer_incentives*.

- 2015-08-20: v1.1
    - Added section 3.4 Supported TLS versions. This describes which security protocols we support, and why.

- 2015-08-10: v1.1
    - Added support for "per_person_per_week" to *pricing:rent_frequency* and *service_charge:frequency*.
    - Added support for "non_quoting" to *pricing:price_qualifier*, and documented its relationship with *pricing:price* and *pricing:price_per_unit_area*.

- 2015-07-29: v1.0
    - Correcting the documented data type for *pricing:price_per_unit_area* which uses a *price_per_unit_area* object.

- 2015-06-17: v1.0
    - Clarifying that the *ZPG-Listing-ETag* should be representative of the state of the associated *listing/update* message.
- 2015-06-11: v1.0
    - Modified detailed_description examples to accurately show how headers with dimensions will be rendered.
- 2015-05-13: v1.0
    - Promoted version 0.1 to 1.0.
- 2015-05-07: v0.1
    - Added section 11.1 Transitioning from an existing bulk data feed. This explains how data feeds are handled by ZPG and how this facilitates a quick transition to having real-time behaviour for all branches simultaneously.
- 2015-05-08: v0.1
    - Described the new constraints for *street_name* and *property_number_or_name* in 6.10 *location* object.
- 2015-05-05: v0.1
    - Included sample error output in A1.3 Free-text string attributes.
- 2015-04-15: v0.1
    - Explained the availability of schema versions between environments in 4.1 Method endpoints.
- 2015-04-14: v0.1
    - Improving description of how the method endpoints are constructed.
- 2015-04-09: v0.1
    - Correcting the documented data type for *listing/update*'s *available_bedrooms* and *bathrooms* attributes: integer.
- 2015-03-31: v0.1
    - Reworked the schema changelog to include which schemas were affected by each change.
- 2015-03-30: v0.1
    - Explained how placeholder images are used in 11.2.2 Preview listing web page.
- 2015-03-19: v0.1

    - Explained the relationship between *available_bedrooms* and *shared_accommodation* in the *listing/update* method.

- 2015-03-13: v0.1
    - Explained the validation of *property_type* = "studio" in *A1.7 property_type*.

- 2015-03-10: v0.1
    - Summarising the flow of data to the ZPG Real-time Listings Service in 2. Overview.
    - Summarising the components of our test environment in 11. Integration.

- 2015-03-03: v0.1
    - Clarifying that the role of *ZPG-Listing-ETag* only relates to checking data synchronicity. It does not affect how we interpret the *listing/update* message, which will always replace the previous version.

- 2015-02-27: v0.1
    - Documenting all the possible attributes that might be returned in an error response.

- 2015-02-19: v0.1
    - Added paragraph to *11.5 Migrating to the live environment* discussing how the transition from a bulk data feed to the ZPG Real-time Listings Service will be handled.

- 2015-02-12: v0.1
    - Added section *2.1 Simple workflow* which provides the reader with a better context for understanding the rest of the document.
    - Added section *11. Integration* which discusses:
        - a suggested sequence of steps to build your implementation of the service (this was previously discussed in the Conclusion)
        - testing methodology
        - migrating to the live environment
    - Defining the expected behaviour of *ZPG-Listing-ETag* more clearly.
    - Clarifying that *listing/update:property_type* should be supplied as an all-lowercase string (*A1.7 property_type*).

- 2015-01-27: v0.1
    - Re-ordered appendices and added names to the menu.
    - Added link to example messages from section *5. Methods*.

- 2015-01-23: v0.1

- *preview_url* has been renamed *url*, and this attribute is now present in the successful responses for *listing/list* and *listing/update* in both *sandbox* and *live* environments.

- 2015-01-07: v0.1
  - Correcting documentation for the *listing/update:content* attribute to indicate that it is an array of *content* objects. Also reiterating that the display order is determined by the ordering of those objects within the array.
  - Added new attribute *sap_rating* to the *listing/update* method.

- 2014-12-12: v0.1
  - Adding information about the attribute requirements for the *description* object. Previously this information was only discussed in *8. Structuring of detailed descriptions*.

- 2014-12-10: v0.1
  - Added section *9.1 Downloading your content*.
  - Added note regarding validation rules to the Free-text attributes section.

- 2014-12-02: v0.1
  - Added link to JSON schema software implementations in section *4. Messaging*.

- 2014-11-21: v0.1
  - The facility to preview how a listing will be displayed on our websites is now available. A listing-specific link is provided via the *preview_url* attribute in the responses for *listing/update* and *listing/list*.

- 2014-11-19: v0.1
  - Added section *3.2 Example: testing authentication using Curl on Linux*.
  - Corrected the *listing/list* response example when *branch_reference* has no listings.

- 2014-11-11: v0.1
  - New appendix section for message examples.

- 2014-11-10: v0.1
  - Additional information about responses.
  - Added *document* to the list of valid *type* values for a *content* object.

- 2014-09-12: v0.1

    - Initial draft.