Adress Data Autocomplete
Webservice Documentation

5. Mai 2025

Content

1. Why use our Adress Data Autocomplete Webservice? 3
   1.1. Comprehensive Data Source 3
   1.2. Global Accessibility 3
2. Choosing the right method 3
3. Frontend examples 4
   3.1. Hierarchical search method 4
   3.2. Inline search method 6
   3.3. Validate method 7
4. Before you start - requirements 8
   4.1. API Key 8
   4.2. Endpoint of the REST API 8
   4.3. Online Documentation for technical integration 8
   4.4. Secure API Key usage 8
      4.4.1. Risk of exposure 8
      4.4.2. Server-to-Server communication 8
      4.4.3. Authentication and authorization 8
5. Methods 9
   5.1. Hierarchical search 9
   5.2. Inline Search 14
   5.3. Validate 15
6. Geographical Coordinates 16
7. “addinfoColumns” Parameter 17
   7.1. Available additional columns 18
8. Description of returned columns 19

1. Why use our Adress Data Autocomplete Webservice?

Our Adress Data Autocomplete (ADA) Webservice provides address completion and validation for Austrian addresses within your application. This functionality is particularly valuable during registration processes (where our ADA Webservice supports both address field-driven responses and single address field responses) or when you need to validate an unverified address.

1.1. Comprehensive Data Source

- We leverage the most up-to-date information on Austrian addresses, directly maintained by Österreichische Post AG. This makes it the single most comprehensive source of Austrian address data.

- Data quality is crucial for any business process, especially when dealing with valid and accurate addresses. Whether it's successful communication with customers or identifying individuals in your database, reliable address data matters.

1.2. Global Accessibility

- Our ADA Webservice offers global access, eliminating the need for maintaining comparable applications or time-consuming updates.

- Say goodbye to resource-intensive data synchronization applications - our service streamlines the process.

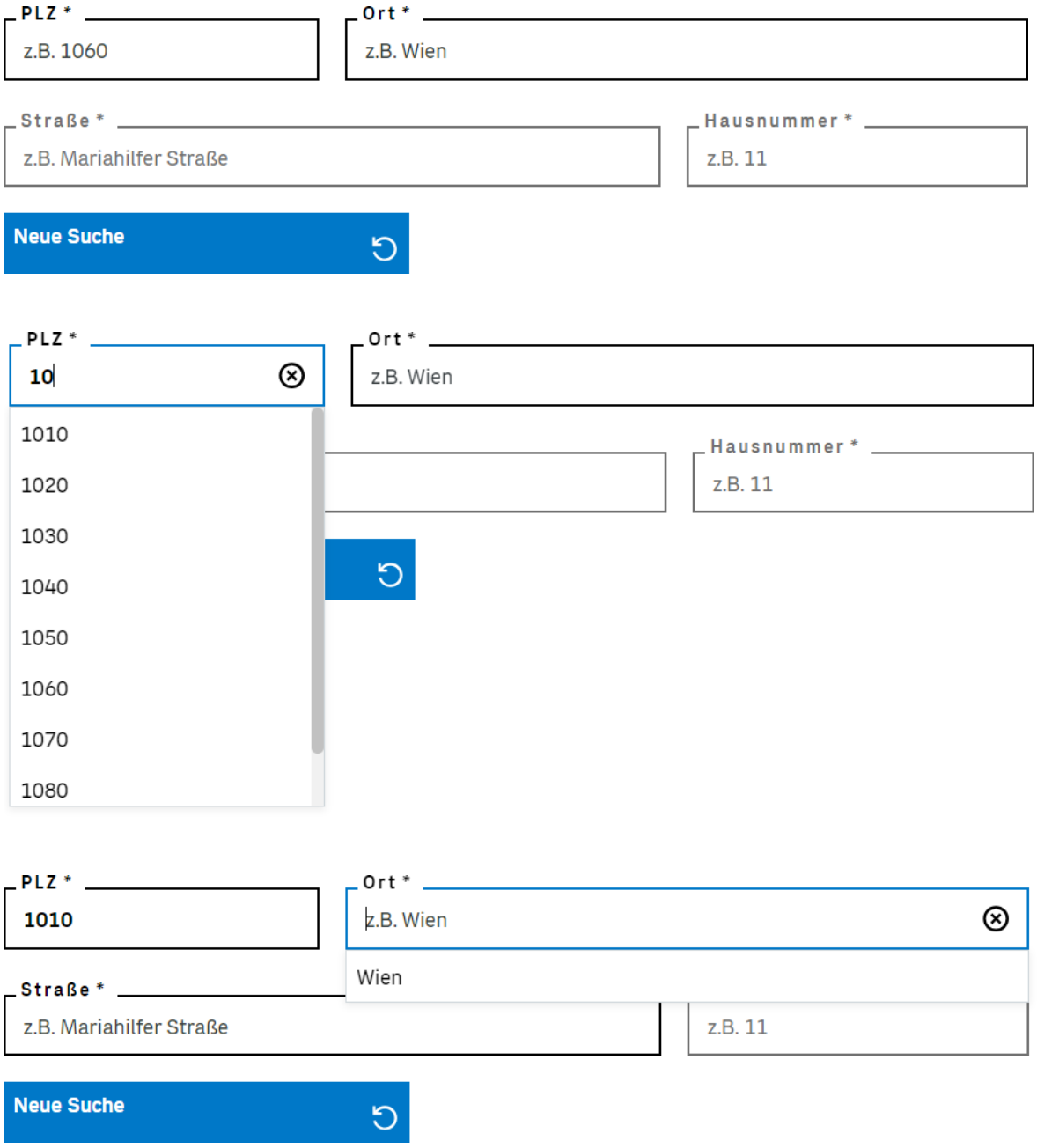
2. Choosing the right method

Our ADA Webservice is built on a REST API, making it straightforward to use and integrate.

It offers three different methods for obtaining correct addresses: If your customer is in a registration process and can confirm the address, we recommend the integration of either the "Hierarchical Search" method or the "Inline Search" method. If you already have a complete address and no chance to get the address confirmed by your customer, we recommend the "Validate" method.

3. Frontend examples

3.1. Hierarchical search method



PLZ *
1010

Ort *
z.B. Wien

Straße *
Mari

Hausnummer *
z.B. 11

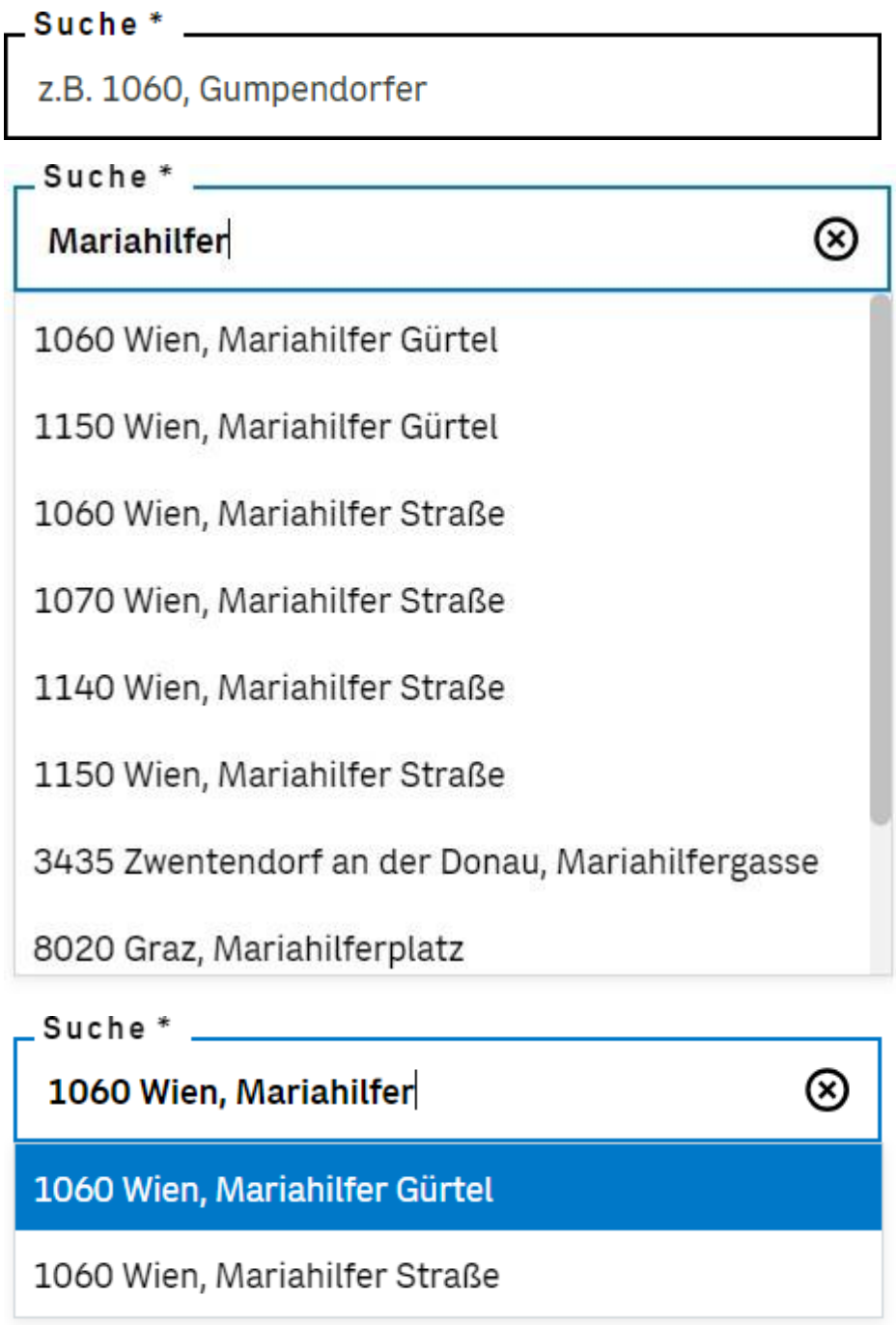
Maria-Theresien-Platz

Maria-Theresien-Straße

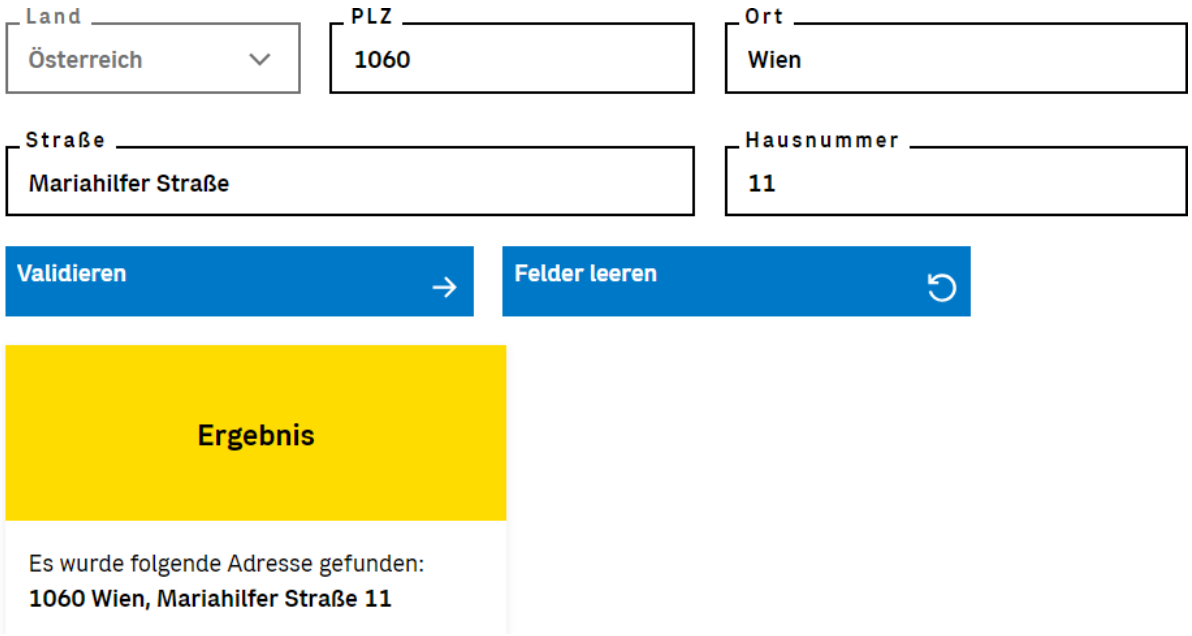
Marienstiege

Passage Mariahilfer Straße

3.2. Inline search method



3.3. Validate method



4. Before you start - requirements

4.1. API Key

Please contact us at adressdata@post.at to receive a valid API key required for consuming the REST API methods.

4.2. Endpoint of the REST API

- Production: https://api.post.at/ada/api

4.3. Online Documentation for technical integration

- Swagger JSON: https://transfer.post.at
  - User: ADA_Swagger
  - Password: ngqMR1fd

4.4. Secure API Key usage

To prevent API key hijacking, it is crucial to follow best practices for secure communication. We recommend using server-to-server communication rather than sending credentials directly from your client application.

4.4.1. Risk of exposure

- Transmitting your credentials (such as API keys) directly in a client application exposes them to potential attackers.
- Anyone with access to the client-side code could read and reuse your credentials, compromising security.

4.4.2. Server-to-Server communication

- Implement your own intermediary API (server-side) that communicates with our ADA REST API.
- Your intermediary API acts as a gatekeeper, holding your API key securely.
- When a client application creates requests, it interacts with your intermediary API, not directly with our ADA REST API.

4.4.3. Authentication and authorization

- Your intermediary API authenticates and authorizes requests using your API key.
- It ensures that only authorized clients (your applications) can access the ADA REST API.
- By keeping the API key server-side, you maintain control over its usage.

5. Methods

To interact with our API, ensure that all requests include a valid API key as a parameter in the header.

If you require more detailed information about the address result, you can request additional columns. These columns can be controlled using the "addinfoColumns" Parameter within the header request. For a comprehensive understanding of the possible columns that can be returned, please refer to the "Description of returned columns" section.

Keep in mind that all methods, except the "Validate" method, include a "limit" parameter. This parameter allows you to control the maximum number of records returned (with a maximum limit of 100).

5.1. Hierarchical search

This method requires four input fields, each serving a different API method:

- **/PostCode**
  - Parameters
    - postCode
    - limit (max. records returned)
  - Purpose: Delivers valid postcode/city combinations and filterId

- **/City**
  - Parameters
    - postCode
    - city
    - limit (max. records returned)
  - Purpose: Delivers valid postcode/city combinations and filterId

- **/Street**
  - Parameters
    - streetName
    - filterId
    - limit (max. records returned)
  - Purpose: Delivers a list of valid postcode/cityName/subCityName/streetName combinations and filterId

- **/HouseNumber**
  - Parameters
    - houseNumber
    - filterId
    - limit (max. records returned)
  - Purpose: Delivers a list of valid addresses

1st and/or 2nd Step *(/PostCode* and/or */City)*

First, you need to find a valid postcode/city combination to obtain a filterId, which is necessary for subsequent search methods. To achieve this, use the */PostCode* and/or */City* method. At least one parameter of these methods must be non-empty to receive a valid response.

Note: If you intend to use the */PostCode* method, ensure that at least one numeric character is supplied; otherwise, no result will be returned.

Example: */PostCode?postCode=10&limit=5*


```
{
    "addresses": [
        {
            "postCode": 1010,
            "cityName": "Wien",
            "filterId": 10102568,
            "type": 0,
            "score": 90
        },
        {
            "postCode": 1020,
            "cityName": "Wien",
            "filterId": 10202568,
            "type": 0,
            "score": 90
        },
```


This method returns all postcode/city combinations for postcodes starting with the value "10". The returned *filterId* can be used in the following */Street* method.

If you require more control over input filtering, consider using the */City* method instead. The */City* method allows you to search for either the city-/sub-city name or the postcode, or both.

Reminder: The */PostCode* method only includes city names but no sub-cities. If you want to include sub-cities as well, use the */City* method.

Each result includes a score value that indicates the match quality. This value ranges from 0 to 100 and reflects how accurately the search word matches the returned address. A value of 100 indicates a perfect match (e.g., "1010" in the example above), while "90" suggests that your search word starts with the result but is not complete.

3rd Step *(/Street)*

Now that you have the previously obtained *filterId*, you can proceed with searching for street names.

Example: */Street?streetName=gra&filterId=10102568&limit=3*


```
{
    "addresses": [
        {
            "postCode": 1010,
            "cityName": "Wien",
            "streetName": "Graben",
            "filterId": 1010256820828,
            "type": 0,
            "score": 90
        },
        {
            "postCode": 1010,
            "cityName": "Wien",
            "streetName": "Grashofgasse",
            "filterId": 1010256821106,
            "type": 0,
            "score": 90
        },
```


In this example, we used "gra" to retrieve all streets that start with the provided word snippet. If you don't specify a *streetName*, you will receive all streets associated with this *filterId*. The search is not case sensitive.

The returned *filterId* from this method can be used to search for house numbers associated with the selected street.

4th Step (*/HouseNumber*)

Once again, using the previously obtained *filterId*, you can now search for house numbers associated with the given street.

Example: */HouseNumber?houseNumber=10&filterId=101025682082&limit=3*


```
{
    "addresses": [
        {
            "postCode": 1010,
            "cityName": "Wien",
            "subCityName": "Wien,Innere Stadt",
            "streetName": "Graben",
            "houseNumber": "10",
            "pac": 101254198,
            "filterId": 0,
            "type": 2,
            "score": 100
        }
    ]
}
```


In this example, we searched for the house number "10" and received exactly one address with a score of "100". This score indicates that the address is a perfect match, precisely what you were looking for.

If you provide no house number, you will receive a list of all available house numbers in the street (up to a maximum of 100 entries).

Additionally, this method provides the following additional information:

- *pac*: The post address code (PAC), used by Österreichische Post AG to uniquely identify an address.
- *type*: The type of the address
  - "2" = main address (valid address used for delivery)
  - "3" = identical address ("ident"; valid alias for the main address)
  - "4" = historical address (address that has changed over time and is no longer valid)

Type 3 and 4 always contain the *mainAddress.pac* field which points to the *main address* (recommended for use).

If you include the *"addInfoColumns"*-header in this method, you can receive further details as well:


```
{
    "addresses": [
        {
            "postCode": 1010,
            "cityName": "Wien",
            "subCityName": "Wien,Innere Stadt",
            "streetName": "Graben",
            "houseNumber": "10",
            "pac": 101254198,
            "filterId": 0,
            "shortStreetName": "Graben",
            "shortHouseNumber": "10",
            "fullHouseNumber": "10",
            "province": "Wien",
            "district": "Wien  1.,Innere Stadt",
            "municipality": "Wien",
            "districtId": 900,
            "municipalityId": 90001,
            "districtAuthorityId": 90101075,
            "officialStreetId": 901563,
            "doorNumberRequired": false,
            "type": 2,
            "score": 100
        }
    ]
}
```


Available parameters can be found here: [*"*addinfoColumns" Parameter](#).

5.2. Inline Search

This method is simple, fast, and user-friendly. Customers can start typing and receive a list of potential results until they find the correct address.

Using a single input field, this method returns the same objects as the other search methods. No *filterId* is necessary here. If the search query is accurate enough, the PAC (post address code) is directly included in the result.

We handle all common address formats. Whether the user starts typing with the street or the postcode, our system accommodates both. However, please note that we do not support the US format where you start with the house number. To address this, consider setting a placeholder to guide users toward the expected input format.

Example: */Inline?query=graben 10, wien&limit=5*


```
{
    "addresses": [
        {
            "postCode": 1010,
            "cityName": "Wien",
            "streetName": "Graben",
            "houseNumber": "10",
            "pac": 101254198,
            "type": 2,
            "score": 100
        },
        {
            "postCode": 1170,
            "cityName": "Wien",
            "streetName": "Grabengasse",
            "houseNumber": "10",
            "pac": 101389858,
            "type": 2,
            "score": 98
        },
```


If the address is not suggested in the search result, but the customer insists that the entered address is valid, you can use the "Validate" method after the Inline Search to verify that a known address has been entered.

5.3. Validate

If you already have a complete address that needs validation, use this method. Unlike autocomplete, it doesn't suggest addresses, but it checks for typing errors. Additionally, this method employs a fuzzy approach, increasing the likelihood of returning the desired address even if it's spelled incorrectly.

To use this method, ensure that all columns are set (postcode, city name, street name, and house number). Send this information as a JSON request body to the API.

Example: */Validate*

Request Body


```
{
    "postCode": "1000",
    "cityName": "Wein",
    "streetName": "Grahben",
    "houseNumber": "10"
}
```


Result


```
{
    "addresses": [
        {
            "postCode": 1010,
            "cityName": "Wien",
            "subCityName": "Wien,Innere Stadt",
            "streetName": "Graben",
            "houseNumber": "10",
            "pac": 101254198,
            "filterId": 0,
            "type": 2,
            "score": 90
        }
    ]
}
```


As demonstrated above, this method can still validate an address even if it has been written incorrectly, if the next possible match is sufficiently accurate. The score value provides an indication of the reliability of the result.

Keep in mind that this method returns only one result if a potential match is found, and no result if no accurate match is possible.

6. Geographical Coordinates

If you already have a PAC (Post Address Code), which is unique for the address, you can request the geocoordinate´s for it.
Since geocoding is a downstream process, it is possible that you receive a "Not-Found" result at this point despite a valid PAC. But 99.8% of all addresses are geocoded. Please consider this case in your implementation.
You will get a "NotFound" result if you submit an invalid PAC like 999999999 and a “BadRequest” result if the PAC is in an invalid range (valid range is between 100000000-999999999).

The following geo coordinate projections are available, and are returned in the response:

- MGI Austria Lambert – EPSG:31287
- LAEA_ETRS89
- WGS84
- GWM

A “z” coordinate is also provided and corresponds to the meters above sea level.

Example: */GeoCoordinates?pac=102375226*


```
{
    "pac": 102375226,
    "mgi": {
        "x": 595226.3125,
        "y": 422763.65625
    },
    "laea_etrs89": {
        "x": 4766314,
        "y": 2746911
    },
    "wgs84": {
        "x": 15.933728218078613,
        "y": 47.67497253417969
    },
    "gwm": {
        "x": 1773734.5,
        "y": 6052951
    },
    "z": 447.966
}
```

7. “addinfoColumns” Parameter

The “addInfoColumns” parameter determines the additional information included in the final address. It also allows you to control the types of addresses you will receive. For instance, you can specify whether you want to receive historical addresses as well or focus solely on current valid addresses.

Example for a Default Historical Address


```
{
    "addresses": [
        {
            "postCode": 1160,
            "cityName": "Wien",
            "streetName": "Brunnenmarkt",
            "houseNumber": "Stand 152",
            "pac": 105512980,
            "filterId": 278928737,
            "type": 4,
            "score": 50,
            "mainAddress": {
                "pac": 105210072,
                "filterId": 0,
                "type": 2
            }
        }
    ]
}
```


Example for a Historical Address with the addInfoColumns-Key “WMA”


```
        {
            "postCode": 1160,
            "cityName": "Wien",
            "streetName": "Brunnenmarkt",
            "houseNumber": "Stand 152",
            "pac": 105512980,
            "filterId": 278928737,
            "type": 4,
            "score": 50,
            "mainAddress": {
                "postCode": 1160,
                "cityName": "Wien",
                "subCityName": "Wien,Ottakring",
                "streetName": "Schellhammergasse",
                "houseNumber": "18c",
                "pac": 105210072,
                "filterId": 0,
                "type": 2
            }
        }
```

When specifying columns for additional information, you have two options: single columns or groups of columns. If you use the *ALL* selector, all other columns will be ignored, and the complete set of information will be returned.

Remember to provide the column codes as a comma-separated string.

Example: *"addInfoColumns": "RG,DID,DNR"*

- adds region fields *Province*, *District* and *Municipality*
- adds *District ID*
- adds *Door number required for delivery*

7.1. Available additional columns

- IH - Ignore historical addresses (returns only valid addresses, omitting any historical ones)

- WMA - With main address details (if you receive a historical address or identical address, you get additional information for the main address, which is the current address). Identical addresses can be addressed. Historical addresses cannot be addressed. In such cases, always use the current address instead.

- AV - Additional values (group)
  - SSN - Short street name
  - SHN - Short house number
  - FHN - Full house number

- RG - Regions (group)
  - PRV - Province
  - DST - District
  - MUN - Municipality

- ID - Identifiers (group)
  - DID - District ID
  - MID - Municipality ID
  - DAI - District authority ID
  - SID - Street ID (official street ID from "Statistik Austria")

- DI - Delivery information (group)
  - DNR - Door number required for delivery

- ALL - All fields are returned, if available

8. Description of returned columns

postCode	4 digits long post code/zip code
cityName	name of city
subCityName	name of sub-city
streetName	street name
houseNumber	house number
pac	unique post address code (PAC) for this address
filterId	ID for hierarchical search methods
shortStreetName	street name (max. 23 characters)
shortHouseNumber	suffix names are replaced with slashes (using this house number type instead of the regular houseNumber can lead to ambiguity)
fullHouseNumber	suffix names are fully spelled out
province [i]	province name – e.g.: "Steiermark" for Styria
district [i]	political district name – e.g.: "Graz (Stadt)"
municipality [i]	name of the municipality
districtId [i]	unique ID for district (3 digits)
municipalityId [i]	unique ID for municipality (5 digits; starts with same 3 digits as districtId)
districtAuthorityId [i] [ii]	unique ID for district authority (8 digits; starts with same 5 digits as municipalityId)
officialStreetId [i]	unique ID for physical street (6 digits)
doorNumberRequired	Boolean flag - if this flag is true, we recommend requesting the house number from the customer to get a deliverable address (necessary if the address has more than 3 parties)
type	address type • 2 = main address • 3 = identical address ("ident"; valid alias for the main address) • 4 = historical address (address that has changed over time and is no longer valid)
score	score of the hit, between 0 and 100

[i] Source: „Statistik Austria"

[ii] A district authority ID refers to a counting district ("Zählsprengel" - ZSP). These districts represent the smallest areas for which statistical data is collected separately.

Österreichische Post AG
post.at
Firmensitz: Wien | Firmenbuchnummer: 180219d
Firmenbuchgericht: Handelsgericht Wien
UID: ATU 46674503

Informationen zum Datenschutz finden Sie unter post.at/datenschutz