getrets-php-sdk

A simple "no framework" php wrapper for the GetRETS® API from timitek (http://www.timitek.com).

How To Use

```
include "GetRETS.php";
$getRets = new GetRETS($customerKey);
$listings = $getRets->getListing()->searchByKeyword($preparedKeywords);
```

Listing

```
(new GetRETS($customerKey))->getListing();
```

This is the main entry point for retrieving **cached** listings. Using this entry point will pull listing data that has been **pre-fetched** from your MLS.

Advantages

- Faster retrieval of data *(The retrieval of data does not have to be negotiated and translated with your MLS RETS provider)*
- More advanced searching *(Since it's pre-fetched data we aren't limited to DMQL queries or other limitations by the RETS provider)*
- Works even when the MLS is down for maintenance

Disadvantages

- Data is not 100% live *(We are constantly polling the MLS for new data but it could be an hour or so before new listings show up, and we refresh each listing every 24 hours)*

1. SearchByKeyword

```
(new GetRETS($customerKey))->getListing()->searchByKeyword($preparedKeywords);
```

A simple search that will retrieve listings by a keyword search.

Parameters

keyword - Keywords to search on

Returns

An array of **CondensedListing**'s

```
[
  {
    "id": "string",
    "listingSourceURLSlug": "string",
    "listingTypeURLSlug": "string",
    "listingID": "string",
    "listingSource": 1,
    "listingType": 1,
    "address": "string",
    "baths": 0,
    "beds": 0,
    "listPrice": "string",
    "rawListPrice": 0,
    "providedBy": "string",
    "squareFeet": 0,
    "lot": "string",
    "acres": "string"
  }
]
```

2. Search

```
(new GetRETS($customerKey))->getListing()->search($keywords, $maxPrice, $minPrice,
$includeResidential, $includeLand, $includeCommercial);
```

A more advanced search that retrieves listings constrained by the optional parameters.

Parameters

keyword - Keywords to search on

maxPrice - *(optional)* The maximum listing price

minPrice - *(optional)* The minimum listing price

includeResidential - *(optional)* Include residential listings

includeLand - *(optional)* Include land listings

includeCommercial - *(optional)* Include commercial listings

Note - If you don't set any of the *include* parameters, all will be assumed as set.

Returns

An array of **CondensedListing**'s

```
[
  {
    "id": "string",
    "listingSourceURLSlug": "string",
    "listingTypeURLSlug": "string",
    "listingID": "string",
    "listingSource": 1,
    "listingType": 1,
    "address": "string",
    "baths": 0,
    "beds": 0,
    "listPrice": "string",
    "rawListPrice": 0,
    "providedBy": "string",
    "squareFeet": 0,
    "lot": "string",
    "acres": "string"
  }
]
```

3. Listing Details

```
(new GetRETS($customerKey))->getListing()->details($listingSource, $listingType, $listingId);
```

Retrieves the more specific / non condensed details for a listing. You will typically use the values returned from search functions as the parameters.

Parameters

listingSource - A string representation of the MLS listing source (see FeedsModels.Models.enumListingSource)

listingType - A string representation of the listing type such as residential, land etc.. (see FeedsModels.Models.enumListingType)

listingId - The unique ID for the listing to retrieve the listing for

Returns

A single **Listing**

```
{
  "description": "string",
  "features": [
    "string"
```

```
  ],
  "photoCount": 0,
  "id": "string",
  "listingSourceURLSlug": "string",
  "listingTypeURLSlug": "string",
  "listingID": "string",
  "listingSource": 1,
  "listingType": 1,
  "address": "string",
  "baths": 0,
  "beds": 0,
  "listPrice": "string",
  "rawListPrice": 0,
  "providedBy": "string",
  "squareFeet": 0,
  "lot": "string",
  "acres": "string"
}
```

4. Images

```
(new GetRETS($customerKey))->getListing()->imageUrl($listingSource, $listingType, $listingId,
$photoId, $width = null, $height = null);
```

Retrieves an image(s) associated with a specific listing.

Special Note - While the width and height parameters are optional, using them to specify an appropriate image size will increase the speed in which your site renders by lowering the need to download a full size image.

Also, fetching the first photo ($photoId) is a suggested strategy for displaying a thumbnail image.

Parameters

listingSource - A string representation of the MLS listing source (see FeedsModels.Models.enumListingSource)

listingType - A string representation of the listing type such as residential, land etc.. (see FeedsModels.Models.enumListingType)

listingId - The unique ID for the listing to retrieve the listing for

photoId - A zero based index for the photo to retrieve *(see the photoCount that is returned in the listing details)*.

width - The width to be used for resizing the photo

height - The height to be used for resizing the photo

Returns

A URL for the image specified

RETSListing

```
(new GetRETS($customerKey))->getRETSListing();
```

This is the main entry point for retrieving **live** listing data from the MLS via RETS.

Advantages

- The data is queried immediately from the MLS RETS server

Disadvantages

- A bit slower than the cached method. *(The data has be translated to DMQL and retreived from a 3rd party server)*.
- Keyword searches have less "*fuzzy logic*" applied to them as we are limited to only searching via the available DMQL classes as defined by

the MLS.

- If your MLS is down for maintenance, results can not be retrieved.

Special Note - All of the same functions used for fetching data from the cached data are applicable to this API controller as well, as the exist with the same signatures, only they will go directly to the RETS server.

5. Execute DMQL

```
(new GetRETS($customerKey))->getRETSListing()->executeDMQL($query, $feedName, $listingType);
```

This is a powerful function that will execute raw DMQL against the RETS MLS server and will return the results as a serialized object.

Special Note - These results will not be returned in a translated fashion similiar to the other listing detail searches. These results are in the format as returned from the MLS RETS server. If you wish to retrieve listings in a **translated** format use getListingsByDMQL.

Parameters

query - The DMQL to be executed against the MLS RETS server

feedName - The name of the feed to run the query against

listingType - A string representation of the listing type such as residential, land etc.. (see FeedsModels.Models.enumListingType)

Returns

An enveloped response with the success or failure of the query, as well as the raw serialized results that were fetched. These serialized results will be different for each feedName and listingType.

```
{
  "success": true,
  "code": 0,
  "message": "string",
  "data": [
    {
      "className": "string"
    }
  ]
}
```

6. Get Translated Listings by DMQL

```
(new GetRETS($customerKey))->getRETSListing()->getListingsByDMQL($query, $feedName, $listingType);
```

This is a powerful function that will execute raw DMQL against the RETS MLS server and will return the results as a serialized object. It is similar to executeDMQL, however this function will **translate** data to be in the same format as returned by other methods that retrieve listing details.

Parameters

query - The DMQL to be executed against the MLS RETS server

feedName - The name of the feed to run the query against

listingType - A string representation of the listing type such as residential, land etc.. (see FeedsModels.Models.enumListingType)

Returns

An enveloped response with the success or failure of the query, as well as the raw serialized results that were fetched.

```
{
  "success": true,
  "code": 0,
  "message": "string",
```

```
  "data": [
    {
      "description": "string",
      "features": [
        "string"
      ],
      "photoCount": 0,
      "id": "string",
      "listingSourceURLSlug": "string",
      "listingTypeURLSlug": "string",
      "listingID": "string",
      "listingSource": 1,
      "listingType": 1,
      "address": "string",
      "baths": 0,
      "beds": 0,
      "listPrice": "string",
      "rawListPrice": 0,
      "providedBy": "string",
      "squareFeet": 0,
      "lot": "string",
      "acres": "string"
    }
  ]
}
```


Geocoding

```
(new GetRETS($customerKey))->getGeocoding();
```

This controller is a planned area of growth to provide more advanced geo-spatial style searching for listing data. For the time being, it is used for parsing keywords into more geocoded data to be used for searching.

If you provide a google geocode key to be associated with your account, you can use these methods.

7. Parse Google Results

```
(new GetRETS($customerKey))->getGeocoding()->parseGoogleResults($googleResults);
```

This function will parse the results returned from googles service and translate them into a consistent format more suitable for searching the listing data.

Parameters

googleResults - Results from Google's geocoder.geocode


```
[
  {
    "address_components": [
      {
        "long_name": "string",
        "short_name": "string",
        "types": [
          "string"
        ]
      }
    ],
    "formatted_address": "string",
    "geometry": {
      "bounds": {
        "south": 0,
        "west": 0,
        "north": 0,
        "east": 0
      },
      "location": {
        "lat": 0,
        "lng": 0
```

```
      },
      "location_type": "string",
      "viewport": {
        "south": 0,
        "west": 0,
        "north": 0,
        "east": 0
      }
    },
    "place_id": "string",
    "types": [
      "string"
    ]
  }
]
```

Returns

Data translated as **AddressDetail**'s.

```
[
  {
    "streetNumber": "string",
    "street": "string",
    "city": "string",
    "county": "string",
    "state": "string",
    "stateAbbreviation": "string",
    "country": "string",
    "postalCode": "string",
    "latitude": 0,
    "longitude": 0,
    "formattedAddress": "string"
  }
]
```

8. Parse Google Results

```
(new GetRETS($customerKey))->getGeocoding()->googleGeocode($address);
```

This function will take a keyword and run it through Google's geocoding service and return the translated results.

Parameters

address - A free form text to geocode (The expectation is that this is a possible address)

Returns

Data translated as **AddressDetail**'s.

```
[
  {
    "streetNumber": "string",
    "street": "string",
    "city": "string",
    "county": "string",
    "state": "string",
    "stateAbbreviation": "string",
    "country": "string",
    "postalCode": "string",
    "latitude": 0,
    "longitude": 0,
    "formattedAddress": "string"
  }
]
```

Further Reading

More information on the API itself can be found at the Swagger UI (http://getrets.net/swagger/).