UnsplashExplorer ver 1.0

SearchPhotosAsync

Get a single page of photo results for a query.

```
public Task<UnsplashMultiplePhotosRequestResult> SearchPhotosAsync(string
query, string collections = null, int page = 1, int perPage = 10,
UnsplashPhotoOrientation orientation = UnsplashPhotoOrientation.Any)
```

Parameter	Description
query *(string)*	Search query
collections *(string)*	Unsplash public collection ID('s) to narrow search. If multiple, comma-separated.
page *(int)*	Page number to retrieve.
perPage *(int)*	Number of items per page. Default: 10, Min: 1, Max: 30
orientation *(UnsplashPhotoOrientation)*	Filter search results by photo orientation. Available: *Any, Landscape, Portrait, Squarish*

Example Usage

```
UnsplashExplorer.Main.SearchPhotosAsync("watermelon", perPage: 9,
                                        orientation:
UnsplashPhotoOrientation.Landscape)
  .ContinueWith(t => {
    if(t.IsCanceled){
      print("Query canceled");
    }else if(t.IsFaulted){
      Debug.LogError($"Failed to load search results: {t.Exception}");
    }else{
      DisplayResults(t.Result.results);
    }
  }, TaskScheduler.FromCurrentSynchronizationContext()) // to run on main
thread
  .LogExceptions(); // log any exceptions in last task to Unity console
```

GetRandomPhotoAsync

Retrieve a random photo, given optional filters.

```
public Task<UnsplashPhoto> GetRandomPhotoAsync(bool onlyFeatured = false,
string query = null, string collections = null, string user = null,
UnsplashPhotoOrientation orientation = UnsplashPhotoOrientation.Any)
```

Parameter	Description
onlyFeatured *(bool)*	Limit selection to featured photos.
query *(string)*	Limit selection to photos matching a search term.
collections *(string)*	Unsplash public collection ID('s) to filter selection. If multiple, comma-separated.
user *(string)*	Limit selection to a single user.
orientation *(UnsplashPhotoOrientation)*	Filter search results by photo orientation. Available: *Any, Landscape, Portrait, Squarish*

Example Usage

```
UnsplashExplorer.Main.GetRandomPhotoAsync(onlyFeatured:true)
  .ContinueWith(t => {
    if(t.IsCanceled){
      return;
    }else if(t.IsFaulted){
      Debug.LogError($"Failed to get random photo: {t.Exception}");
    }else{
      LoadPhoto(t.Result);
    }
  }, TaskScheduler.FromCurrentSynchronizationContext()) // to run on main
thread
  .LogExceptions(); // log any exceptions in last task to Unity console
```

Downloading `UnsplashPhoto`

```
void LoadPhoto(UnsplashPhoto photo){
  new UnsplashDownloader().DownloadPhotoAsync(photo, null,
UnsplashPhotoSize.Regular).ContinueWith(t => {
    if(t.IsCanceled){
      print("Photo download canceled");
    }else if(t.IsFaulted){
      Debug.LogError($"Failed to download photo: {t.Exception}");
    }else{
      var tex = t.Result;
      rawImage.texture = tex;
      rawImage.GetComponent<AspectRatioFitter>().aspectRatio =
(float)tex.width/tex.height;
    }
  }, TaskScheduler.FromCurrentSynchronizationContext()) // to run on main
thread
    .LogExceptions(); // log any exceptions in last task to Unity console
}
```