Fonda Spring School Universidade de Aveiro March 6, 2024

Manuel Campagnolo
ISA/ULisboa, CEF, Terra

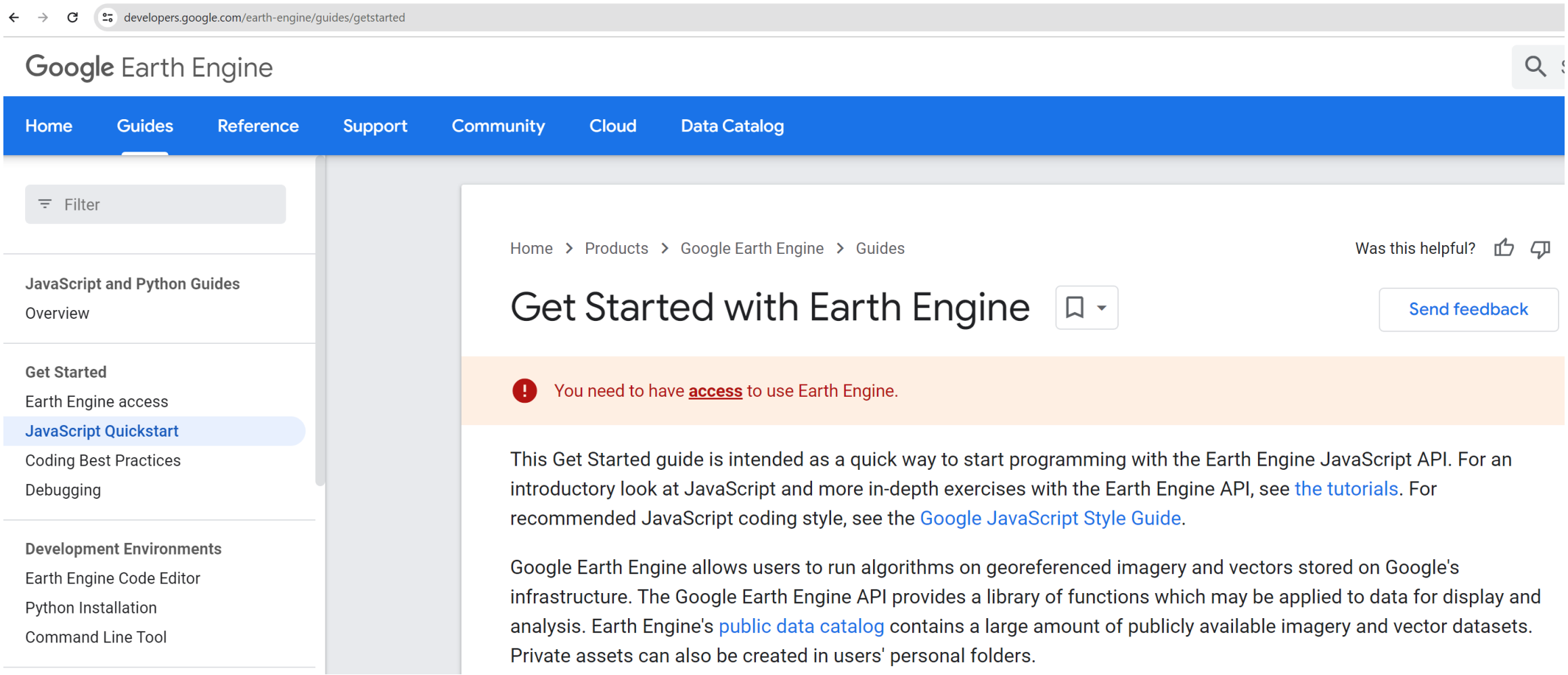
An introduction to Remote Sensing and Land Cover mapping using the Google Earth Engine

Geospatial processing services

public data catalog, compute infrastructure, geospatial APIs

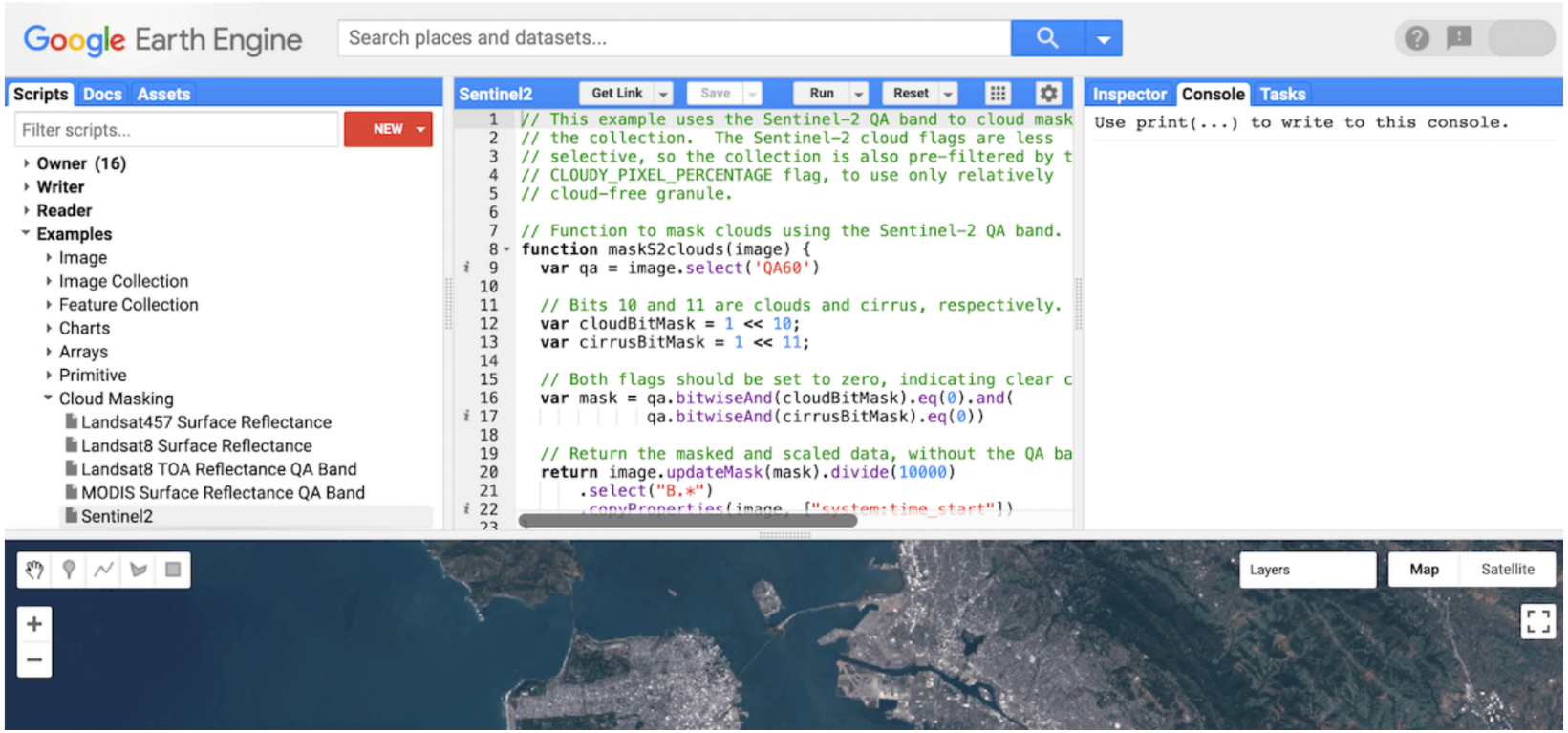
- **Google Earth Engine** (Google Cloud)
- Microsoft Planetary Computer (Azure)
- Amazon Web Services (AWS) GeoSpatial Services
- Copernicus Data Space Ecosystem, mostly for Sentinel imagery
- …

The GEE code editor


developers.google.com/earth-engine/guides/getstarted
Google Earth Engine
Home
Guides
Reference
Support
Community
Cloud
Data Catalog
Filter
JavaScript and Python Guides
Overview
Get Started
Earth Engine access
JavaScript Quickstart
Coding Best Practices
Debugging
Development Environments
Earth Engine Code Editor
Python Installation
Command Line Tool
Home > Products > Google Earth Engine > Guides
Was this helpful?
Get Started with Earth Engine
Send feedback
You need to have access to use Earth Engine.
This Get Started guide is intended as a quick way to start programming with the Earth Engine JavaScript API. For an introductory look at JavaScript and more in-depth exercises with the Earth Engine API, see the tutorials. For recommended JavaScript coding style, see the Google JavaScript Style Guide.
Google Earth Engine allows users to run algorithms on georeferenced imagery and vectors stored on Google's infrastructure. The Google Earth Engine API provides a library of functions which may be applied to data for display and analysis. Earth Engine's public data catalog contains a large amount of publicly available imagery and vector datasets. Private assets can also be created in users' personal folders.

The Code Editor

The Code Editor is an interactive environment for developing Earth Engine applications (Figure 1). The center panel provides a JavaScript code editor. Above the editor are buttons to save the current script, run it, and clear the map. The **Get Link** button generates a unique URL for the script in the address bar. The map in the bottom panel contains the layers added by the script. At the top is a search box for datasets and places. The left panel contains code examples, your saved scripts, a searchable API reference and an asset manager for private data. The right panel has an inspector for querying the map, an output console, and a manager for long-running tasks. The help button ? in the upper right contains links to this Guide and other resources for getting help. Learn more from the Code Editor guide and the Get Help guide.



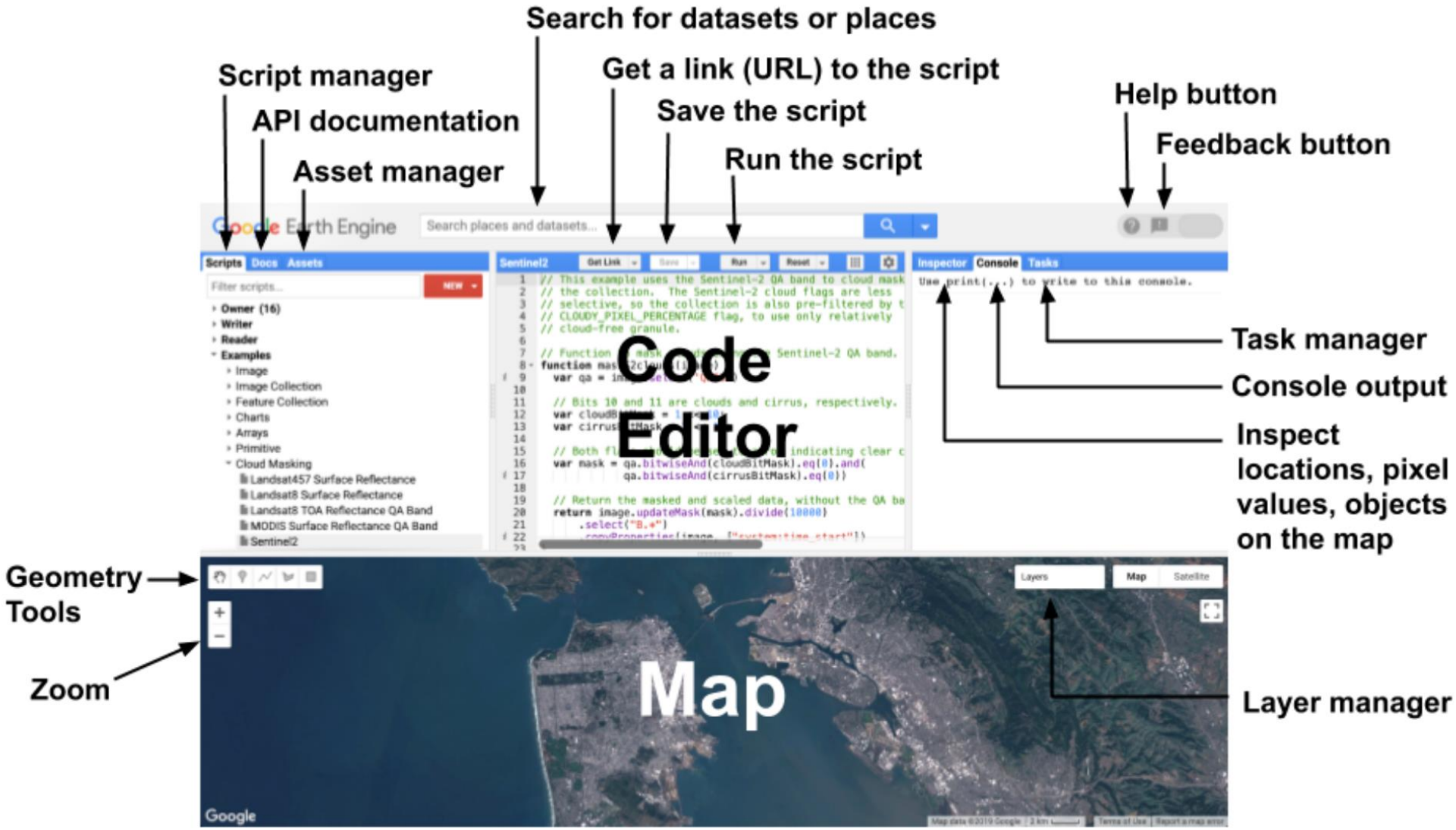


Figure 1. *Diagram of components of the Earth Engine Code Editor at code.earthengine.google.com.*

selected images. The samples introduce you to commonly used methods, such as `filter()`, `clip()`, and `Map.addLayer()`.

Earth Engine data structures

The two most fundamental geographic data structures in Earth Engine are `Image` and `Feature` corresponding to raster and vector data types, respectively. Images are composed of bands and a dictionary of properties. Features are composed of a `Geometry` and a dictionary of properties. A stack of images (e.g. an image time series) is handled by an `ImageCollection`. A collection of features is handled by a `FeatureCollection`. Other fundamental data structures in Earth Engine include `Dictionary`, `List`, `Array`, `Date`, `Number` and `String` (learn more about basic data types from this tutorial. It is important to remember that these are all *server-side* objects and are not manipulated the same way as client-side JavaScript objects are (learn more).

Finding data and filtering

Google Earth Engine

Search

English

Filter

JavaScript and Python Guides

Overview

Get Started

Earth Engine access

JavaScript Quickstart

Coding Best Practices

Debugging

Development Environments

Earth Engine Code Editor

Python Installation

Command Line Tool

API Tutorials

Overview

Introduction to JavaScript for Earth

Finding images, image collections and feature collections

Images, image collections, and feature collections are discoverable by searching the Earth Engine Data Catalog. For example, entering 'Landsat 8' into the search field results in a list of raster datasets. (The complete listing of Earth Engine datasets is at the Earth Engine Data Catalog). Click on the dataset name to get a brief description, information about the temporal availability, data provider and collection ID. Click the **Import** button to automatically create an **Imports** section at the top of your script with a variable for this collection.

Alternatively, copy the collection ID and paste it into your code. For example, choose the Tier 1 TOA result of the 'Landsat 8' search and copy the ID as follows:

Code Editor (JavaScript)

```
var collection = ee.ImageCollection('LANDSAT/LC08/C02/T1_TOA');
```

Filtering and Sorting

It is often necessary to filter a collection by space and/or time in order to limit the number of results. For example, consider the task of sorting the Landsat 8 scene collection in order to find a cloud-free scene for San Francisco. First, it's necessary to define the region of interest. A point is often useful for that. Activate the **Inspector** tab on the right side of the Code Editor and click near the center of your area of interest, copy the coordinates from the **Inspector** tab, then construct a `Point` using:

Code Editor (JavaScript)

```
var point = ee.Geometry.Point(-122.262, 37.8719);
```

Construct start and end dates:

Code Editor (JavaScript)

```
var start = ee.Date('2014-06-01');
var finish = ee.Date('2014-10-01');
```

Filter the Landsat 8 collection using the point and the dates, then sort using a metadata property (discovered during inspection of the Landsat 8 scene metadata):

Code Editor (JavaScript)

```
var filteredCollection = ee.ImageCollection('LANDSAT/LC08/C02/T1_TOA')
  .filterBounds(point)
  .filterDate(start, finish)
  .sort('CLOUD_COVER', true);
```

construction code is written for you in the imports section of the Code Editor. You can also use a personal asset ID as shown in this doc.

Get an `ee.Image` from an `ee.ImageCollection`

The standard way to get an image out of a collection is to filter the collection, with filters in order of decreasing specificity. For example, to get an image out of the Sentinel-2 surface reflectance collection:

Code Editor (JavaScript) Colab (Python)

```
var first = ee.ImageCollection('COPERNICUS/S2_SR')
                .filterBounds(ee.Geometry.Point(-70.48, 43.3631))
                .filterDate('2019-01-01', '2019-12-31')
                .sort('CLOUDY_PIXEL_PERCENTAGE')
                .first();
Map.centerObject(first, 11);
Map.addLayer(first, {bands: ['B4', 'B3', 'B2'], min: 0, max: 2000}, 'first');
```

Note that the sort is *after* the filters. Avoid sorting the entire collection.

Visualizing images

developers.google.com/earth-engine/tutorials/tutorial_api_02

Google Earth Engine

Home | Guides | Reference | Support | Community | Cloud | Data Catalog

Filter

- Coding Best Practices
- Debugging

Development Environments

- Earth Engine Code Editor
- Python Installation
- Command Line Tool

API Tutorials

- Overview
- Introduction to JavaScript for Earth Engine
- The Earth Engine API
  - Introduction
  - Visualizing Images and Image Bands
  - Computations using Images
  - Image Collections
  - Compositing, Masking, and Mosaicking
  - NDVI, Mapping a Function over a Collection, Quality Mosaicking
  - Exporting Charts and Images

Home > Products > Google Earth Engine > Guides

Was this helpful?

Visualizing Images and Image Bands

Send feedback

Now that you're ready to begin writing Earth Engine JavaScript, start by copying the following code into the Code Editor:

Code Editor (JavaScript)

```
// Instantiate an image with the Image constructor.
var image = ee.Image('CGIAR/SRTM90_V4');

// Zoom to a location.
Map.setCenter(-112.8598, 36.2841, 9); // Center on the Grand Canyon.

// Display the image on the map.
Map.addLayer(image);
```

Click the **Run** button at the top of the Code Editor and observe that a very gray image appears on the map. Don't worry, you'll make it look better soon. If the syntax of any part of this example in unfamiliar, be sure to review the JavaScript for Earth Engine tutorial.

Google Earth Engine

Search

Home | Guides | Reference | Support | Community | Cloud | Data Catalog

Filter

- Objects and Methods Overview
- Image
  - Image Overview
  - Image Visualization
  - Image Information and Metadata
  - Mathematical Operations
  - Relational, Conditional and Boolean Operations
  - Convolutions
  - Morphological Operations
  - Gradients
  - Edge Detection
  - Spectral Transformations
  - Texture
  - Object-based Methods
  - Cumulative Cost Mapping
  - Registering Images
- ImageCollection
- Geometry
- Feature & FeatureCollection
- FeatureView
- Reducer
- Join

Home > Products > Google Earth Engine > Guides

Was this helpful?

Image Visualization

Send feedback

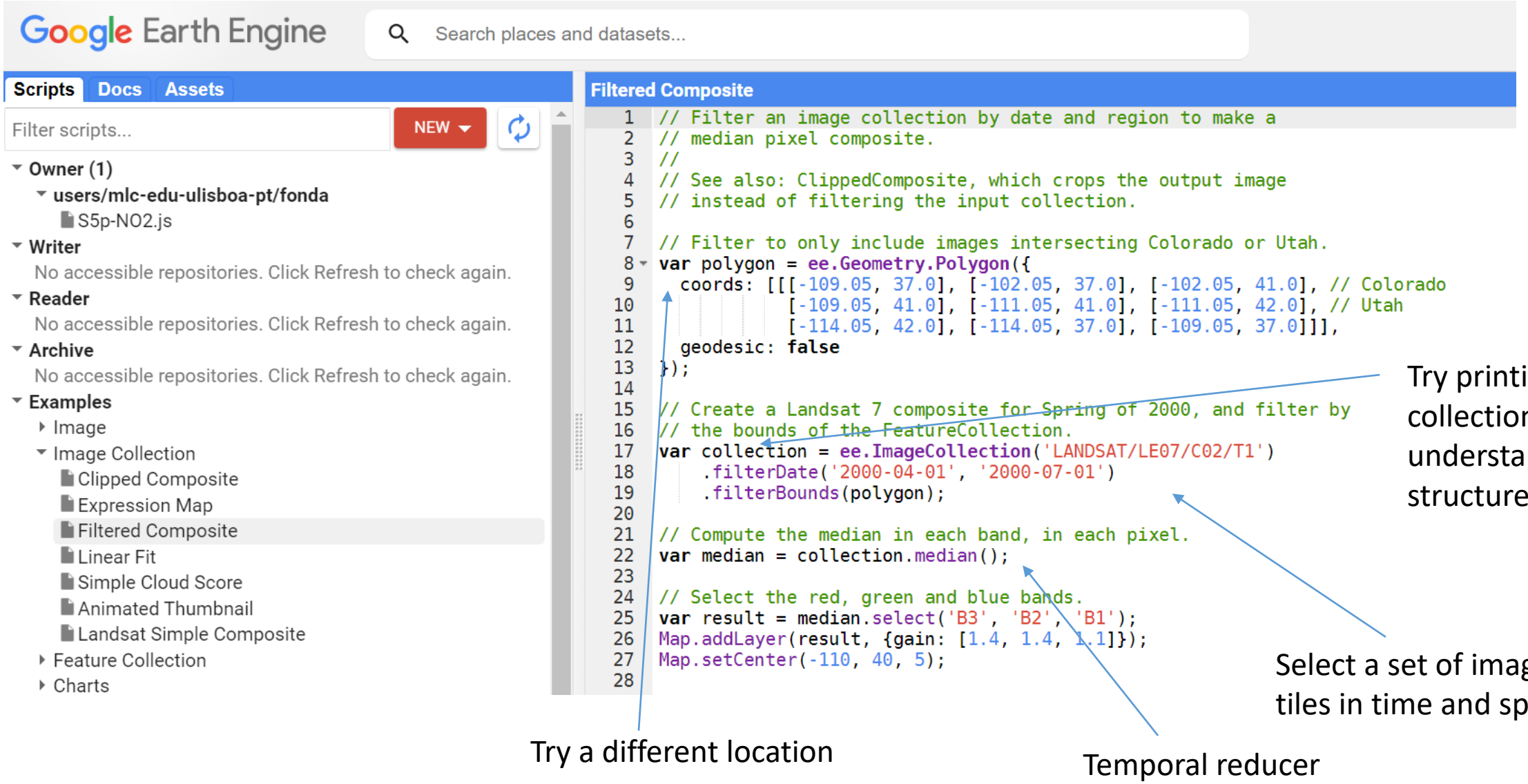
Run in Google Colab | View source on GitHub

The are a number of `ee.Image` methods that produce RGB visual representations of image data, for example: `visualize()`, `getThumbURL()`, `getMap()`, `getMapId()` (used in Colab Folium map display) and, `Map.addLayer()` (used in Code Editor map display, not available for Python). By default these methods assign the first three bands to red, green and blue, respectively. The default stretch is based on the type of data in the bands (e.g. floats are stretched in [0, 1], 16-bit data are stretched to the full range of possible values), which may or may not be suitable. To achieve desired visualization effects, you can provide visualization parameters:

Image visualization parameters

Parameter	Description	Type
bands	Comma-delimited list of three band names to be mapped to RGB	list
min	Value(s) to map to 0	number or list of three numbers, one for each band
max	Value(s) to map to 255	number or list of three numbers, one for each band

-8.67


Google Earth Engine
Search places and datasets...
Scripts
Docs
Assets
Filter scripts...
NEW
Owner (1)
users/mlc-edu-ulisboa-pt/fonda
S5p-NO2.js
Writer
No accessible repositories. Click Refresh to check again.
Reader
No accessible repositories. Click Refresh to check again.
Archive
No accessible repositories. Click Refresh to check again.
Examples
Image
Image Collection
Clipped Composite
Expression Map
Filtered Composite
Linear Fit
Simple Cloud Score
Animated Thumbnail
Landsat Simple Composite
Feature Collection
Charts
Filtered Composite
1 // Filter an image collection by date and region to make a
2 // median pixel composite.
3 //
4 // See also: ClippedComposite, which crops the output image
5 // instead of filtering the input collection.
6
7 // Filter to only include images intersecting Colorado or Utah.
8 var polygon = ee.Geometry.Polygon({
9 coords: [[[-109.05, 37.0], [-102.05, 37.0], [-102.05, 41.0], // Colorado
10 [-109.05, 41.0], [-111.05, 41.0], [-111.05, 42.0], // Utah
11 [-114.05, 42.0], [-114.05, 37.0], [-109.05, 37.0]]],
12 geodesic: false
13 });
14
15 // Create a Landsat 7 composite for Spring of 2000, and filter by
16 // the bounds of the FeatureCollection.
17 var collection = ee.ImageCollection('LANDSAT/LE07/C02/T1')
18 .filterDate('2000-04-01', '2000-07-01')
19 .filterBounds(polygon);
20
21 // Compute the median in each band, in each pixel.
22 var median = collection.median();
23
24 // Select the red, green and blue bands.
25 var result = median.select('B3', 'B2', 'B1');
26 Map.addLayer(result, {gain: [1.4, 1.4, 1.1]});
27 Map.setCenter(-110, 40, 5);
28
Try printing the collection to understand its structure
Select a set of image tiles in time and space
Try a different location
Temporal reducer

Charts

Chart Overview

The Earth Engine JavaScript Code Editor seamlessly integrates with Google Charts for convenient tabular data visualization via `ui.Chart` functions. Charts can be displayed interactively in the Code Editor console, `ui.Panel` widgets, and in stand-alone browser tabs.

Caution: the `ui.Chart` widget is available for the JavaScript Code Editor API only.

`DataTable` charts

Earth Engine uses the Google Visualization API to support charting. The API accepts a `DataTable`, which is a 2-D table where rows are observations and columns are observation attributes. All charts in Earth Engine are derived from a `DataTable`; the `ui.Chart` widget allows you to supply a `DataTable` directly. It affords the greatest opportunity for chart customization, but may be less convenient than methods for charting specific Earth Engine objects (see the following section). Learn more about creating charts from a `DataTable`:

Earth Engine object charts

The `ui.Chart` widget provides helper methods to construct a `DataTable` and render charts from `Image`, `ImageCollection` `Feature`, `FeatureCollection`, `Array`, and `List` objects. Each function accepts a specific data type and includes methods for reducing the data to tabular format in a variety of arrangements that dictate data assignment to chart series and axes.

Visit the following links to learn how to generate a chart for each data type:

- `Feature` charting
- `FeatureCollection` charting
- `Image` charting
- `ImageCollection` charting
- `Array` charting
- `List` charting

Chart types

Google Earth Engine

Search

Home | Guides | Reference | Support | Community | Cloud | Data Catalog

Filter

- Objects and Methods
- Objects and Methods Overview
- Image
- ImageCollection
- Geometry
- Feature & FeatureCollection
- FeatureView
- Reducer
- Join
- Array
- Chart
  - Chart Overview
  - Feature and FeatureCollection Charts
  - Image Charts
  - ImageCollection Charts
  - Array and List Charts
  - DataTable Charts
  - Chart Styling
- Asset Management

Home > Products > Google Earth Engine > Guides

Was this helpful?

ImageCollection Charts

Send feedback

The `ui.Chart.image` module contains a set of functions for rendering charts from the results of spatiotemporal reduction of images within an `ImageCollection`. The choice of function dictates the arrangement of data in the chart, i.e., what defines x- and y-axis values and what defines the series. Use the following function descriptions and examples to determine the best function for your purpose.

Chart functions

Use the following plot diagrams as a visual guide to understand how each function arranges spatiotemporal image collection reduction results in a chart; i.e., what elements define x values, y values, and series. Note that `ui.Chart.image.doySeries*` functions take two reducers: one for region reduction (`regionReducer`) and another for intra-annual coincident day-of-year reduction (`yearReducer`). Examples in the following sections use `ee.Reducer.mean()` as the argument for both of these parameters.

`ui.Chart.image.series`

Image date is plotted along the x-axis according to the `system:time_start` property. Series are defined by image bands. Y-axis values are the reduction of images, by date, for a single region.

Creating new images and bands

Mathematical Operations

Send feedback

Run in Google Colab

View source on GitHub

Image math can be performed using operators like `add()` and `subtract()`, but for complex computations with more than a couple of terms, the `expression()` function provides a good alternative. See the following sections for more information on operators and expressions.

Operators

Math operators perform basic arithmetic operations on image bands. They take two inputs: either two images or one image and a constant term, which is interpreted as a single-band constant image with no masked pixels. Operations are performed per pixel for each band.

Filter

- ee.DateRange
- ee.Dictionary
- ee.ErrorMargin
- ee.Feature
- ee.FeatureCollection
- ee.Filter
- ee.Geometry
- ee.Image
  - ee.Image
  - abs
  - acos
  - add
  - addBands
  - and
  - arrayAccum
  - arrayArgmax
  - arrayCat
  - arrayDimensions
  - arrayDotProduct
  - arrayFlatten
  - arrayGet
  - arrayLength
  - arrayLengths

Home > Products > Google Earth Engine > Reference

Was this helpful?

ee.Image.addBands

Send feedback

Returns an image containing all bands copied from the first input and selected bands from the second input, optionally overwriting bands in the first image with the same name. The new image has the metadata and footprint from the first input image.

Usage	Returns
`Image.addBands(srcImg, names, overwrite)`	Image

Argument	Type	Details
this: `dstImg`	Image	An image into which to copy bands.
`srcImg`	Image	An image containing bands to copy.
`names`	*List, default: null*	*Optional list of band names to copy. If names is omitted, all bands from srcImg will be copied over.*
`overwrite`	*Boolean, default: false*	*If true, bands from `srcImg` will override bands with the same names in `dstImg`. Otherwise the new band will be renamed with a numerical suffix (`foo` to `foo_1` unless `foo_1` exists, then `foo_2` unless it exists, etc).*

Exporting data

Google Earth Engine

Search

Home | Guides | Reference | Support | Community | Cloud | Data Catalog

Filter

- Image Manifest Upload
- Importing Table Data
- Table Manifest Upload

Exporting Data

- Overview
- Exporting Images
- Exporting Table and Vector Data
- Exporting Video and Animations
- Exporting Map Tiles
- Exporting to BigQuery
- Extracting Image Data Programmatically

Home > Products > Google Earth Engine > Guides

Was this helpful?

Exporting Data

Send feedback

You can export images, map tiles, tables and video from Earth Engine. The exports can be sent to your Google Drive account, to Google Cloud Storage or to a new Earth Engine asset.

To use Google Cloud Storage (a fee-based service), you'll need to set up a project, enable billing for the project, and create a storage bucket. See the Cloud Storage Quickstart page for instructions. See this guide for information on storage bucket naming. Data exported to a Cloud Storage bucket will have the bucket's default object Access Control List (ACL). You must have write permission for the specified bucket.

Choose an option from the side menu to learn more about batch exporting images and tables, extracting image data programmatically, and more.

`crsTransform` parameters for full control of the grid.

to Drive

To export an image to your Drive account, use `Export.image.toDrive()`. For example, to export portions of a Landsat image, define a region to export, then call `Export.image.toDrive()`:

Code Editor (JavaScript) | Colab (Python)

```
// Export the image, specifying the CRS, transform, and region.
Export.image.toDrive({
  image: landsat,
  description: 'imageToDriveExample_transform',
  crs: projection.crs,
  crsTransform: projection.transform,
  region: geometry
});
```

Export.image.toDrive(image, *description*, *folder*, *fileNamePrefix*, *dimensions*, *region*, *scale*, *crs*, *crsTransform*, *maxPixels*, *shardSize*, *fileDimensions*, *skipEmptyTiles*, *fileFormat*, *formatOptions*, *priority*)

Creates a batch task to export an Image as a raster to Drive. Tasks can be started from the Tasks tab. "crsTransform", "scale", and "dimensions" are mutually exclusive.

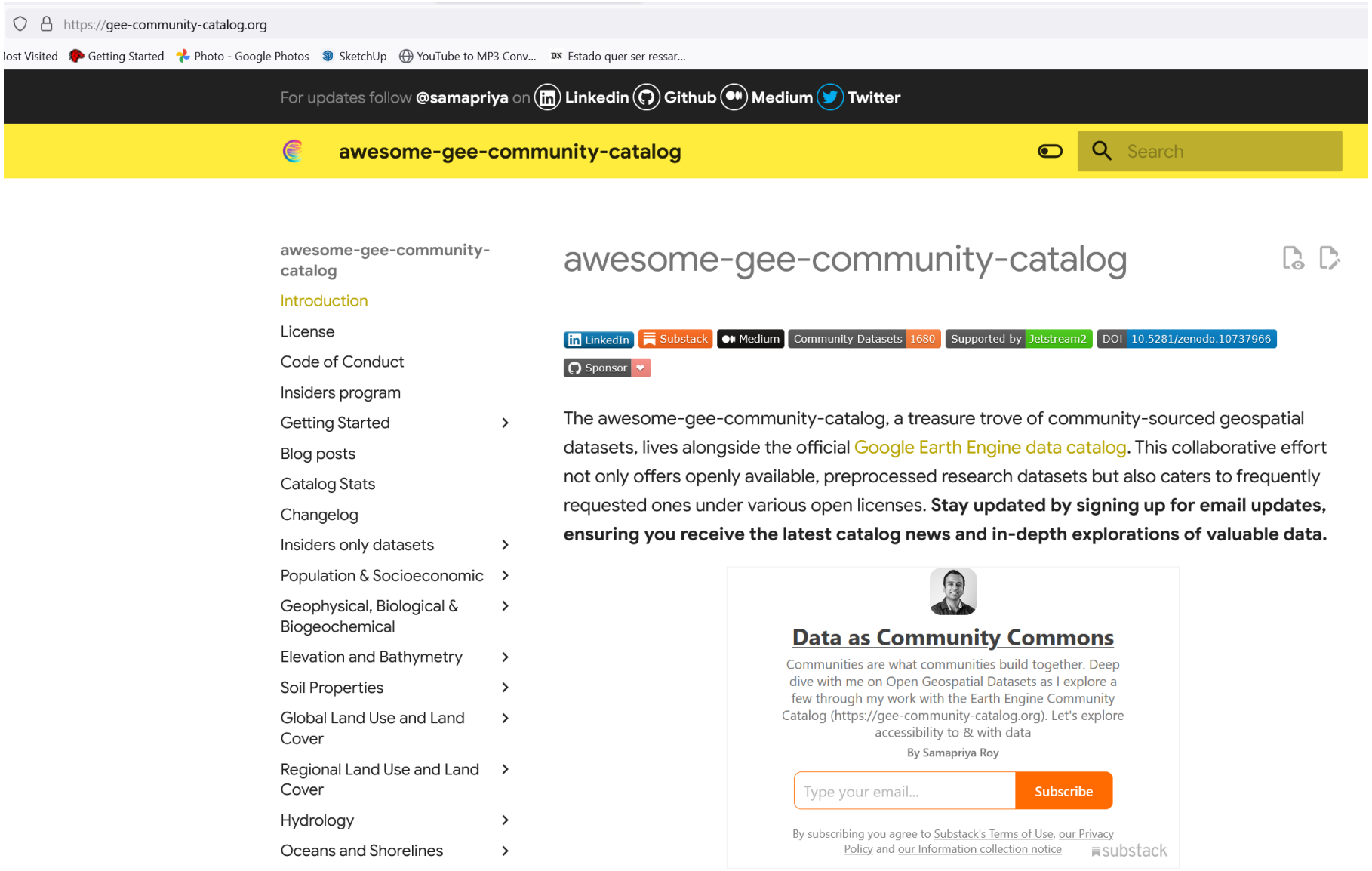
Arguments:

- image (Image):
The image to export.

- description (String, optional):
A human-readable name of the task. May contain letters, numbers, -, _ (no spaces). Defaults to "myExportImageTask".

- folder (String, optional):

Community catalog


https://gee-community-catalog.org
lost Visited
Getting Started
Photo - Google Photos
SketchUp
YouTube to MP3 Conv...
Estado quer ser ressar...
For updates follow @samapriya on Linkedin Github Medium Twitter
awesome-gee-community-catalog
Search
awesome-gee-community-catalog
Introduction
License
Code of Conduct
Insiders program
Getting Started
Blog posts
Catalog Stats
Changelog
Insiders only datasets
Population & Socioeconomic
Geophysical, Biological & Biogeochemical
Elevation and Bathymetry
Soil Properties
Global Land Use and Land Cover
Regional Land Use and Land Cover
Hydrology
Oceans and Shorelines
Agriculture, Vegetation and
awesome-gee-community-catalog
LinkedIn
Substack
Medium
Community Datasets 1680
Supported by Jetstream2
DOI 10.5281/zenodo.10737966
Sponsor
The awesome-gee-community-catalog, a treasure trove of community-sourced geospatial datasets, lives alongside the official Google Earth Engine data catalog. This collaborative effort not only offers openly available, preprocessed research datasets but also caters to frequently requested ones under various open licenses. Stay updated by signing up for email updates, ensuring you receive the latest catalog news and in-depth explorations of valuable data.
Data as Community Commons
Communities are what communities build together. Deep dive with me on Open Geospatial Datasets as I explore a few through my work with the Earth Engine Community Catalog (https://gee-community-catalog.org). Let's explore accessibility to & with data
By Samapriya Roy
Type your email...
Subscribe
By subscribing you agree to Substack's Terms of Use, our Privacy Policy and our Information collection notice
substack