RedBrick AI Docs

Getting Started

Rapid and collaborative medical data annotation.

RedBrick AI is a purpose-built SaaS application created to help healthcare AI teams annotate medical data more effectively. RedBrick AI offers web-based annotation tools for CTs, MRIs, X-rays, etc., as well as comprehensive project management and quality control tools.

- [x] Request a free trial or product demonstration by contacting us through our website: https://redbrickai.com.

Getting started with guides

Our documentation offers a comprehensive overview of the features and functionality of RedBrick AI. Before diving deep into the documentation, working through our use-case-driven guides to discover RedBrick AI may be useful.

Guide

Segmenting aorta and iliac arteries

This guide showcases how to use RedBrick AI to segment the aorta and iliac arteries in Abdominal Computed Tomography Angiography scans.

Guide

Segment large organs in abdominal CT scans

This guide showcases how to use advanced segmentation tools to segment organs like the liver and kidneys in a abdominal CT scan.

Guide

Volumetric segmentation of necrosis and edema

This guide provides a step-by-step overview of configuring your viewer for a 3D MRI study and performing segmentation of necrosis and edema.

Guide

FDA study for Chest x-ray anomaly detection

Using RedBrick AI for FDA clinical validation studies. In this guide we walk through a hypothetical study for a chest anomaly algorithm.

Guide

Lesion detection in mammogram, DBT and breast MRI

In this guide we will focus on RedBrick AI's features for breast imaging. We'll walk through three projects focusing on different modalities, and the special features the platform has for efficient viewing and annotation.

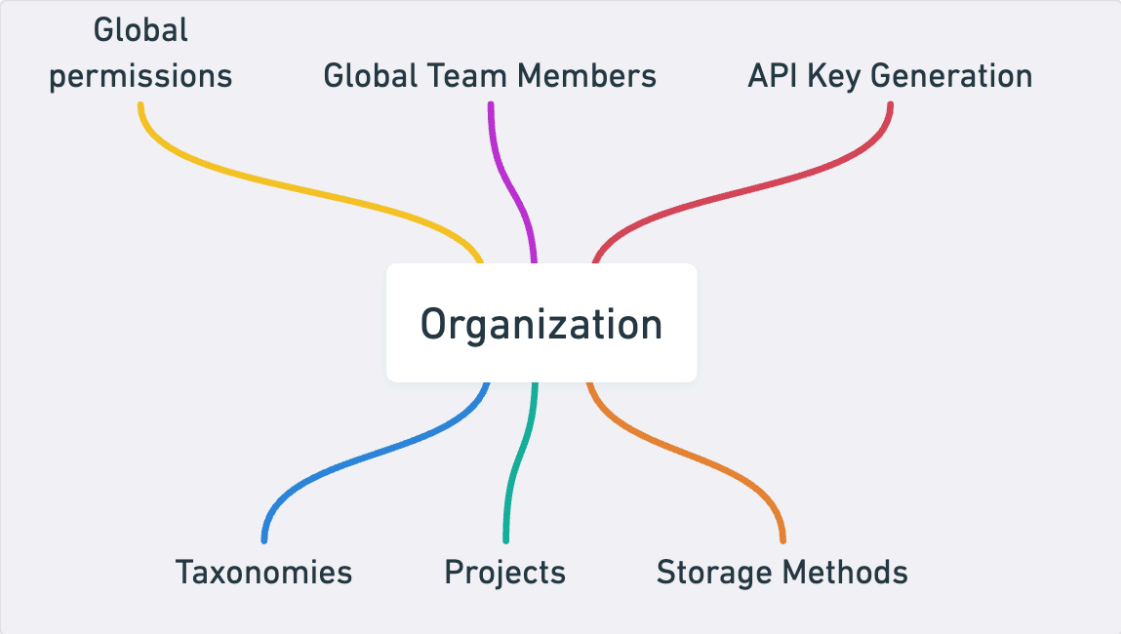
Team & Organization

Organization and Project Roles

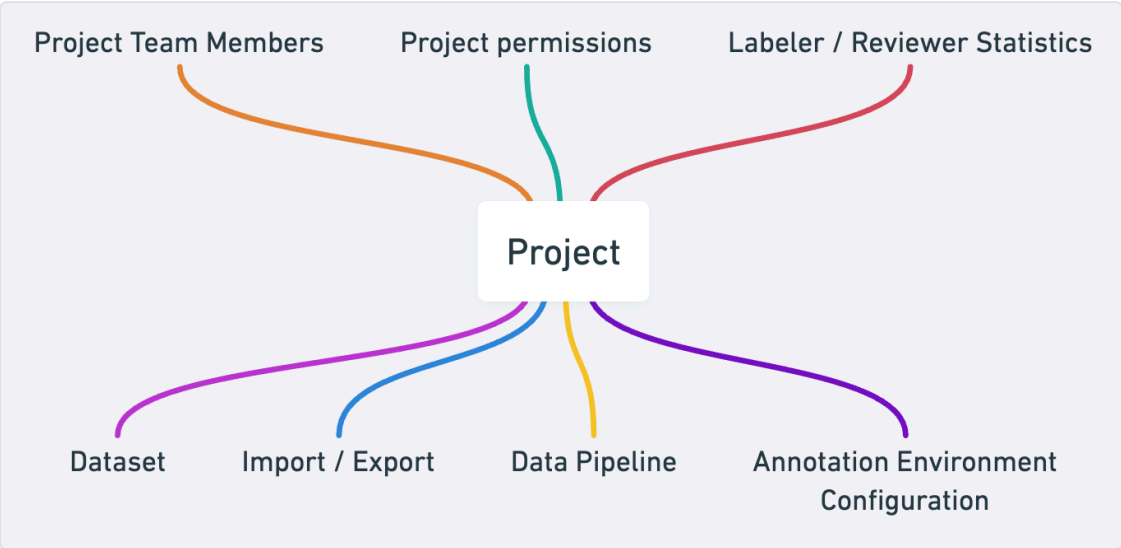
Overview

Your **Organization** is a unique structure that RedBrick AI creates for you and your team.

All of the work you do on RedBrick AI and any resources you use will be housed within your Organization. This includes your team members, your Taxonomies, your Projects, and more.



In RedBrick AI, a **Project** is a workspace to which you can upload data and inside of which you perform annotation work within a pipeline defined by you.



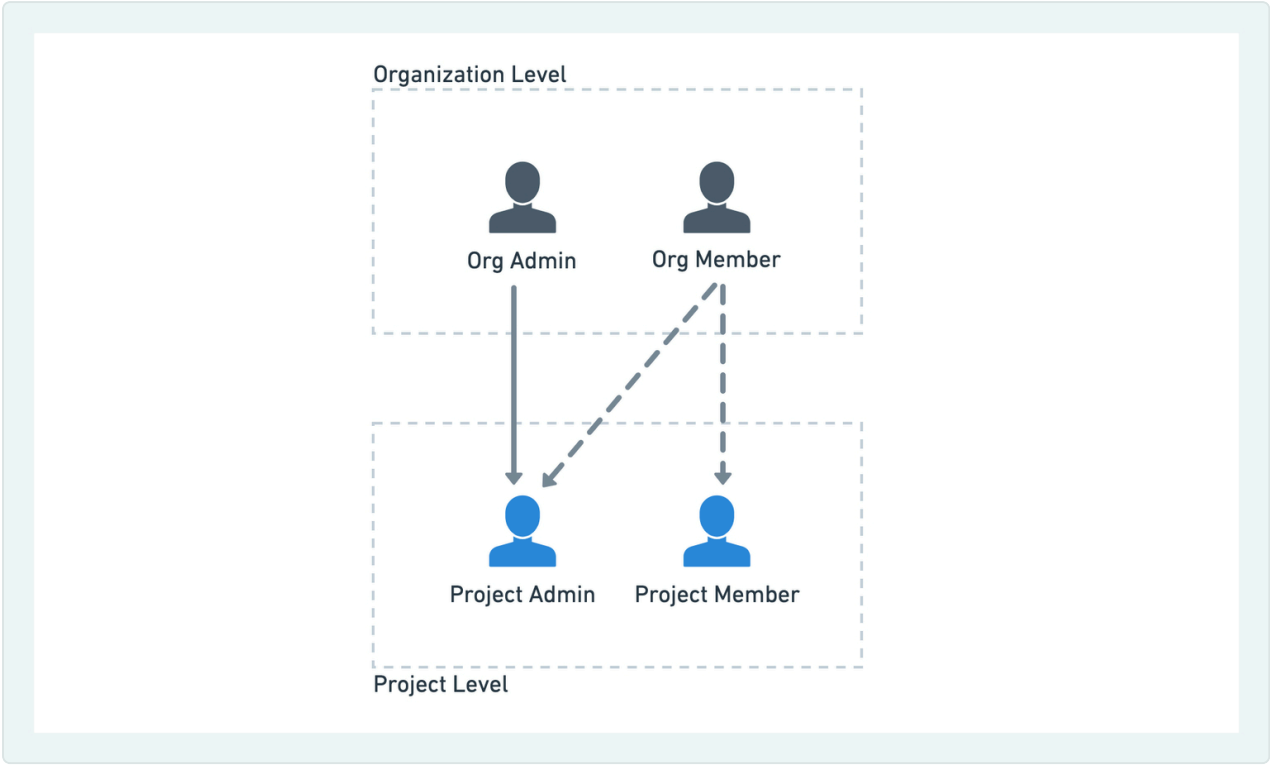
Within a Project, you can:

- upload images, volumes, and segmentation files;
- assign work to your labelers and reviewers on a Task level;
- perform and review annotation work;
- define Project-level permissions for labelers, reviewers, administrators, etc.;
- invite specific members of your team and regulate their access to various Stages;
- view a range of statistics on the quality of your labelers' work, time spent;
- configure custom toolkits and Project-level settings specific to your use case;
- and much, much more!

While all of your team members have to be invited to your Organization in order for them to access RedBrick AI, you can easily configure their permissions based on their Roles.

Roles

RedBrick AI offers role-based access control at two levels - the **Organization level** and the **Project level**. Each of these roles governs what actions a user can perform at the respective level.



Organization-level Roles

While each Organization can only have a single **Org Owner**, there is no limit to the number of **Org Admins** and **Org Members** an Organization can have.

Org Owner	**Organization Level:** Has access to all assets within an Organization; has the ability to create, edit, and delete assets, including the Organization itself. **Project Level:** Org Owners are automatically added to all Projects as Project Admins (see below).
Org Admin	**Organization Level:** Has access to all assets within an Organization; has the ability to create, edit, and delete assets, but not the Organization itself. **Project Level:** Org Admins are automatically added to all Projects as Project Admins (see below).
Org Member	**Organization Level:** cannot create or edit resources at the Organization level. **Project Level:** Org Members are **not** automatically added to any

> Projects, and **must be invited** to a Project by a Project Admin (see below).

Project-level Roles

Role	Permissions
Project Admin	Can perform administrative actions at the Project level, i.e. uploading data, assigning Tasks, editing Project Settings, and viewing Project Overview statistics & other user statistics.
Project Member	Can only annotate/review data (i.e. Tasks) that are assigned to them. Cannot view the activity of any other users.
Project Manager	Can manage Tasks and user permissions. Cannot access Project settings.

Common Role Configurations

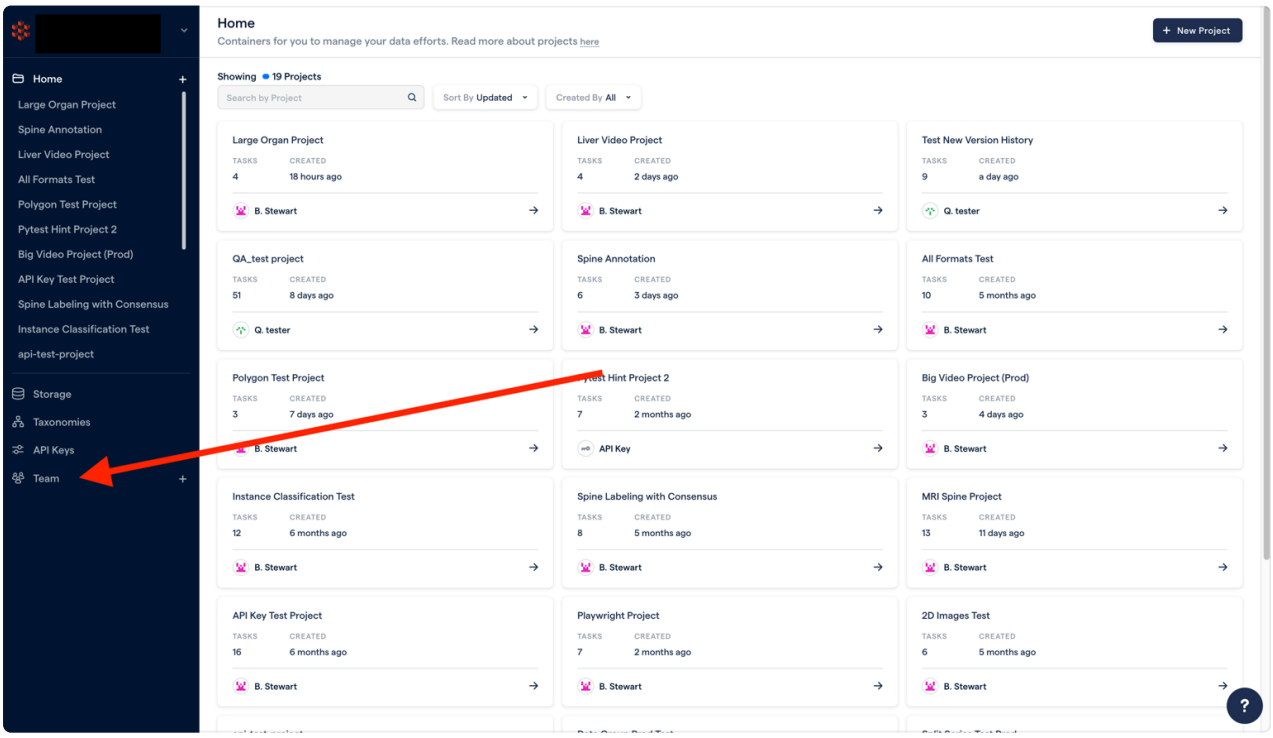
- **Labelers** are often first added to an Organization as Org Members, added to relevant Projects as Project Members, and given access to the Label Stage by a Project Admin.
- **Internal Reviewers** are often first added to an Organization as Org Members and then added to relevant Projects as Project Admins, which gives them Project-wide Admin access.
- **External Reviewers** are often first added to an Organization as Org Members, added to relevant Projects as Project Members, and given access to any relevant Review Stages by a Project Admin.
- **External Project Managers** should be added to an Organization as Org Members and added to relevant Projects as Project Managers.

> If you would like to change your Organization's Org Owner, please reach out to our support team at support@redbrickai.com.

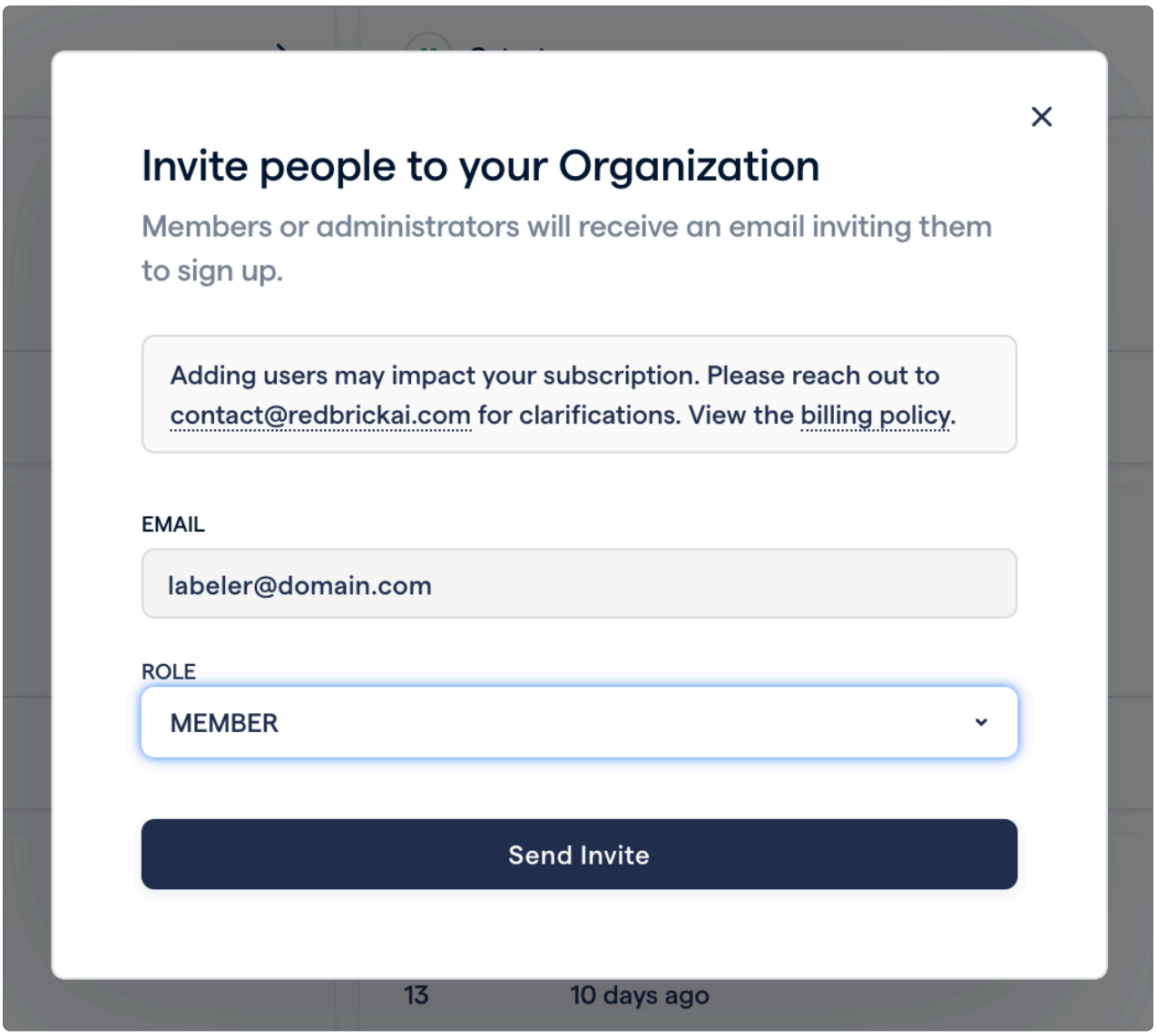
Inviting Your Team

> The following instructions do not apply to Organizations who have implemented SSO.

You can view all the current members of your Organization inside the **Team Tab** on the left sidebar.



To invite a team member to your Organization, navigate to the Team Page, click on "Invite Member", enter their email address and select their Organization-level role (i.e. either Org Admin or Org Member).



Click on "Send Invite" to issue an **email invitation** at the email address that you indicated.

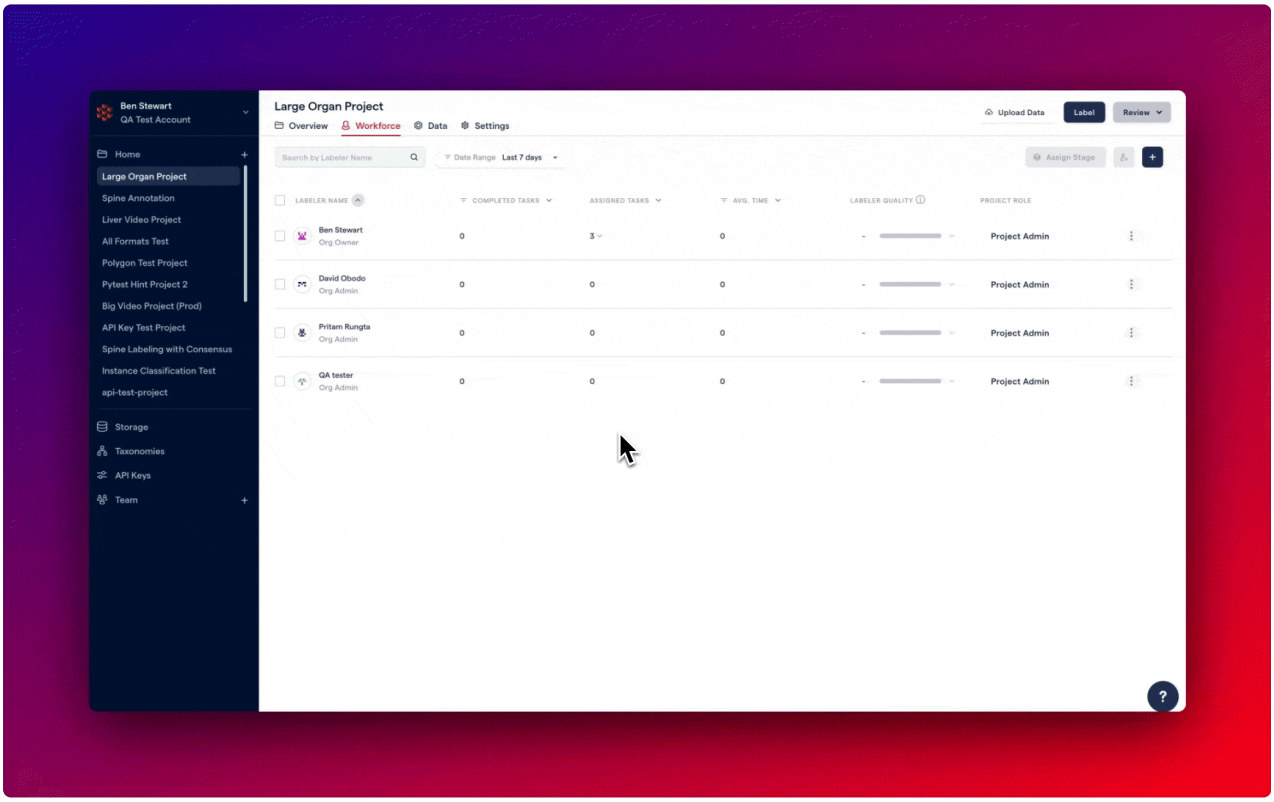
> If you or a member of your team does not receive an email invitation, please check your spam folder. If the invitation is not in Spam, you can accept the invitation directly from the application by doing the following:
>
> Navigate to https://app.redbrickai.com/createaccount and fill in the relevant fields, being sure to use the email address to which the invitation was issued.
>
> After logging in, you should be able to accept the invitation to your Organization.

Once a user has been invited to your Organization, you can invite specific users to individual Projects and manage their Project-level permissions.

Each team member's Project-level permissions can be managed under the Workforce Tab of your Project Dashboard.

- [x] Assigning a Project Member to a particular Stage restricts their access to **only those Tasks that are currently in that specific Stage**. This restriction applies to both manual and automatic task assignment.

Please see the video below for a visual demonstration of how to invite a Project Member to a Project and limit their access to a specific Stage.



Single Sign-on

Overview

RedBrick AI is proud to offer Single Sign-on (SSO) for Enterprise-level clients, which allows your team to effortlessly access our platform using your trusted internal credentials for a seamless end-user experience.

Implementing SSO for your Organization fundamentally alters the way that Users are added to and sign into your Organization on RedBrick AI.

Please see more detailed summaries of both of these processes below.

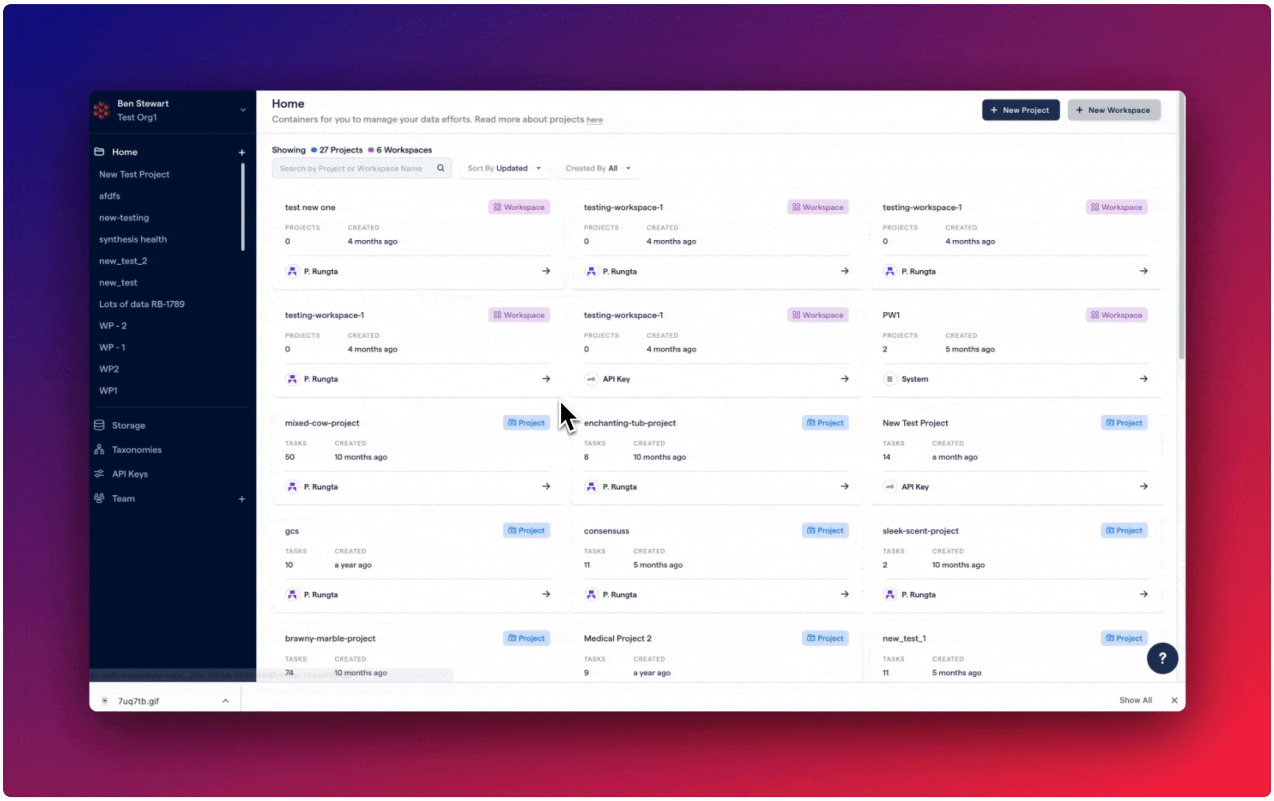
> If you'd like to implement SSO for your organization, please reach out to us at support@redbrickai.com. We'll reach out to you directly to discuss the specific requirements of your integration given your company's existing security architecture.

Inviting Users to a RedBrick Organization with SSO

After SSO has been configured for your Organization (and assuming you have the proper permissions), you can invite team members by first navigating to the Team Page.

The "Invite Users" dialog will allow you to invite colleagues as either an Org Admin or an Org Member, as well as display a URL slug. This URL slug is unique to your Organization and is key to user sign-in, as detailed below.

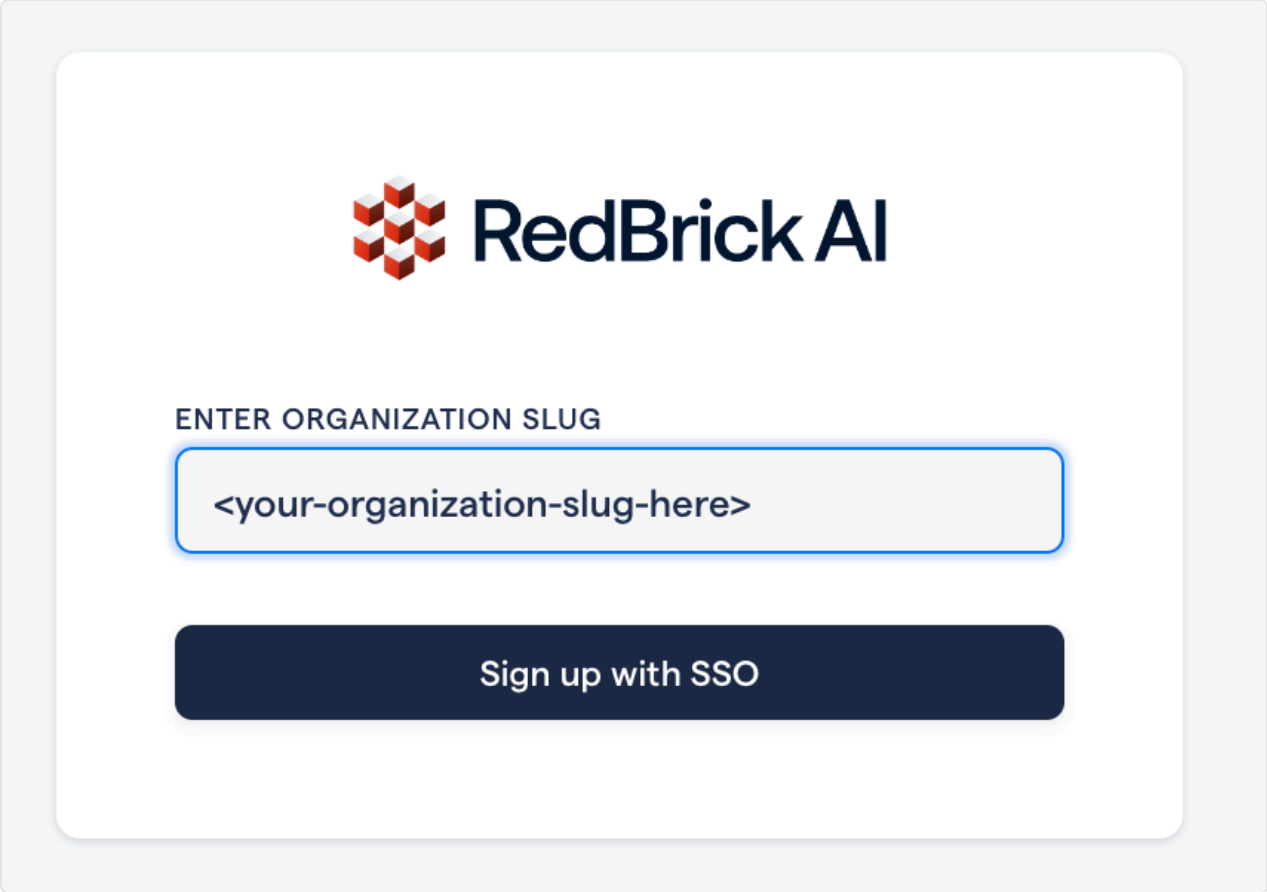
Simply enter the email address of the user you would like to invite to your RedBrick Organization, designate their privilege level and click "Send Invite".



Account Creation with SSO

After an Org Admin has extended an invitation, the user should receive a message in their email inbox containing a button that redirects the user to RedBrick AI to create an account (or, if they have already created an account, to sign in to your RedBrick Organization).

After clicking on the invitation link, the user will be prompted to confirm their Organization's unique URL slug.

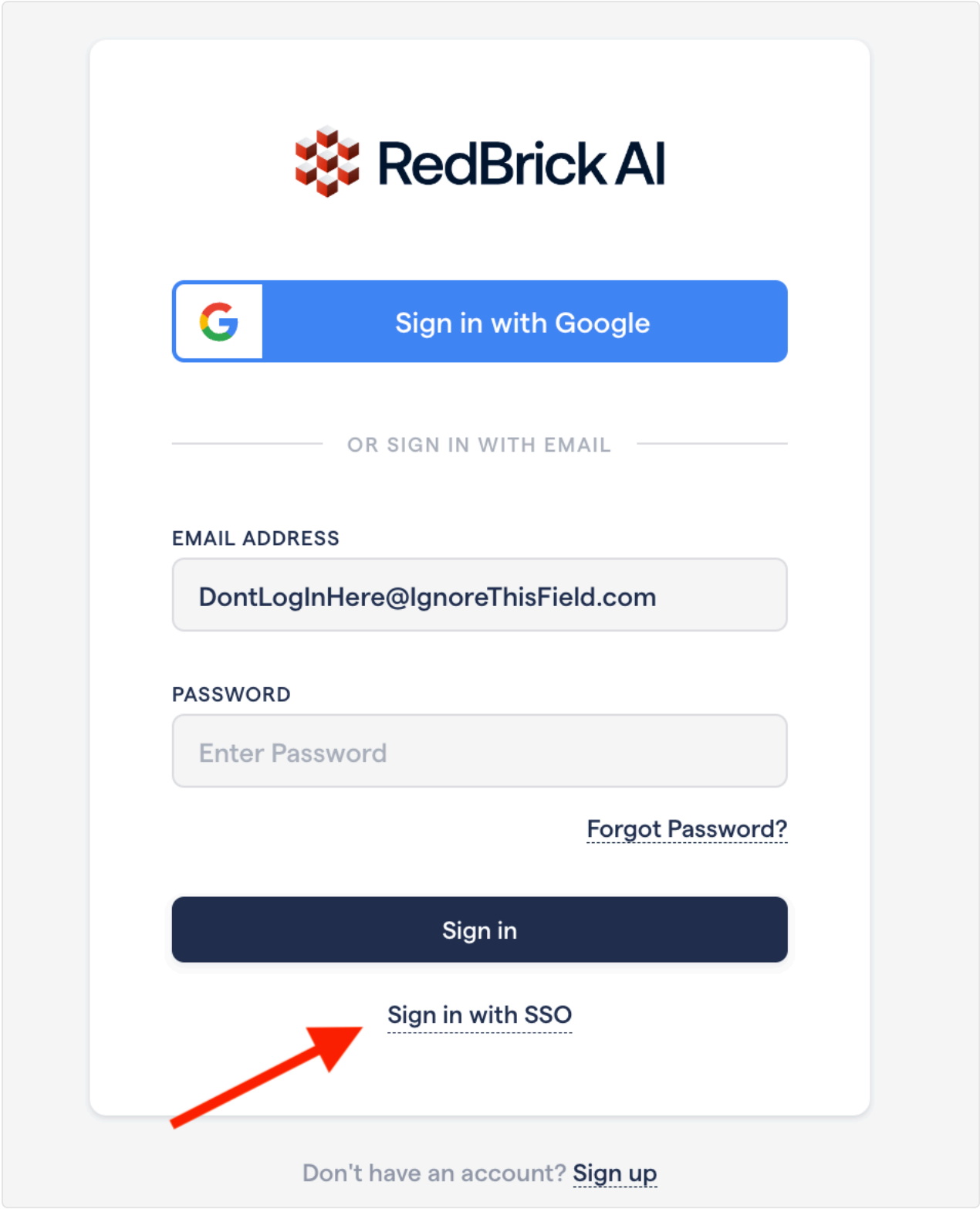


After clicking on "Sign up with SSO", the user will be redirected to an external identity provider, where they will have to complete a successful login. If the login is successful, a RedBrick AI account will be generated for the user.

After a RedBrick AI account has been generated for the user, a confirmation code will be issued to their email address. Once the user enters that confirmation code into the necessary field, they will be able to join their team's RedBrick Organization and begin work.

Signing In with SSO

Once a user has created an account on RedBrick AI and successfully joined an Organization that has SSO enabled, they can easily sign in by clicking on "Sign in with SSO" on the login page.



After clicking on "Sign in with SSO", the user will be prompted to enter their Organization's unique URL slug. Enter the slug and click on "Sign in with SSO" to proceed to the RedBrick UI.

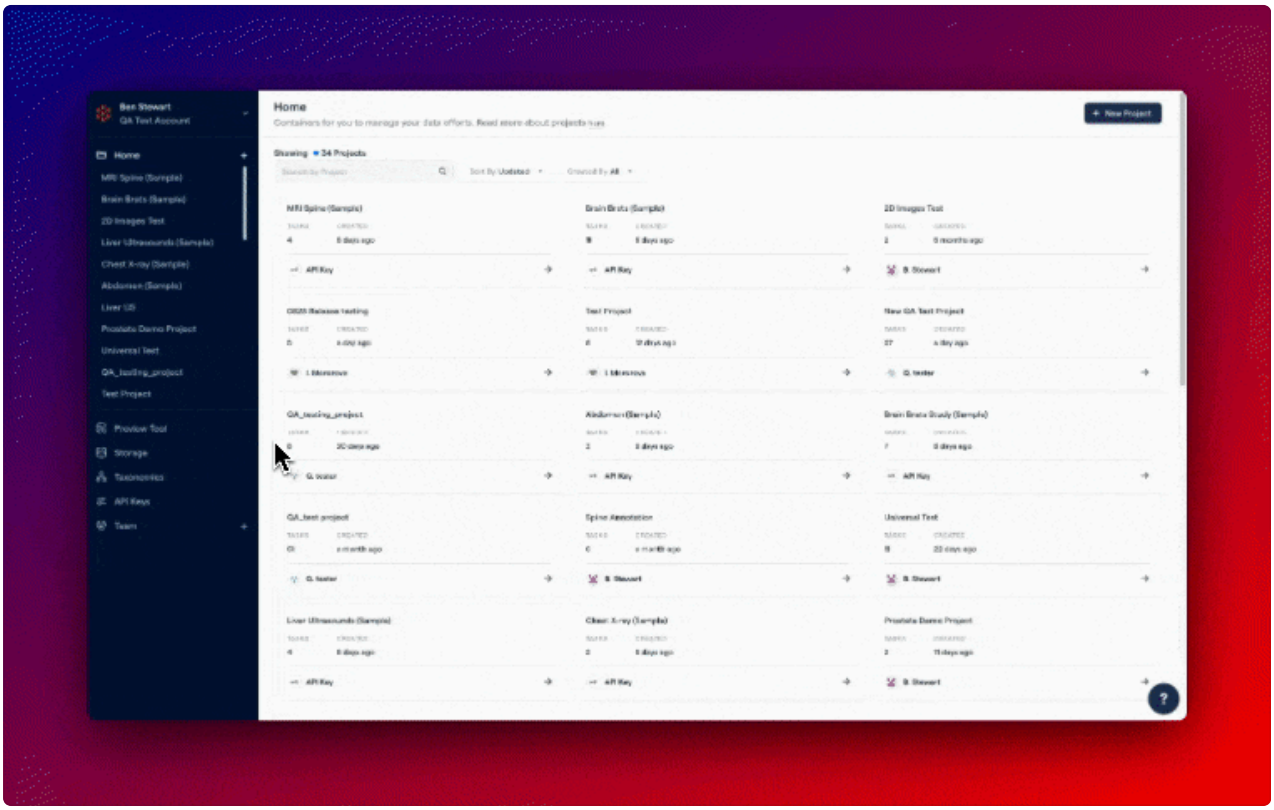
Importing Data

Preview Mode

RedBrick AI's Preview Mode allows users to see how their images, volumes, and annotations are displayed within RedBrick AI's Annotation Tool without having to create a Project or integrate a third-party Storage Method.

To visualize your data (and, optionally, your annotations) in Preview Mode:

1. Click on **Preview Mode** in the left hand toolbar of the **Home Page**;
2. Within **Preview Mode**, select the image(s) you would like to display or simply drag and drop them into the corresponding left hand window;
3. (Optional) select the annotation file you would like to display or simply drag and drop it into the corresponding right hand window;
4. Click on **View Data;**
5. Use the **Manage Files** button (top right hand corner) and **Replace** buttons (in the dialog menu) to swap out new files as needed;



Uploading a spine MRI scan and annotations to Preview Mode

If you are uploading a segmentation file, Preview Mode will automatically map the annotations and display them as unique Instances in the left hand toolbar.

Please note that images and volumes can be manipulated in Preview Mode just as they would be in the standard Annotation Tool. The following functions represent a non-exhaustive list of features available in Preview Mode:

- Windowing Settings
- Thresholding Settings
- MPR Layout
- Command Bar (and commands such as "toggle Linear Pixel Interpolation", "toggle permanent crosshairs", etc.)
- Oblique Planes
- Horizontal and Vertical Flipping
- Viewport Maximization and Minimization;
- ...and more!

If you are uploading a segmentation file that contains **non-segmentation annotations** (e.g. length measurements, bounding boxes, etc.), the non-segmentation annotations will **not** display in Preview Mode.

Direct Data Upload

The Direct Upload functionality allows users to upload their image data directly to RedBrick AI's servers. We recommend using Direct Upload if you're working with a small dataset or want to do some light experimentation with RedBrick's toolset.

> All image data that is directly uploaded to RedBrick AI's servers will also be stored there. If you'd rather not have your image data hosted on our servers, we recommend integrating your storage.

RedBrick AI supports a variety of different image formats:

1. DICOM - .dcm, .ima, .dicom, .dicm
2. NIfTI - .nii, .nii.gz
3. Videos - .mp4, .mov, .avi
4. RGB Images - .png, .jpeg, .jpg, .bmp
5. NRRD - .nrrd

> If you require additional support for a file format that is not present in this list, please reach out to us at support@redbrickai.com.

First, open a project that will serve as the destination for your upload. Then, click on **Upload Data** on the top-right of the dashboard.

1. Select the type of data you want to upload.

   Please note that **you can only upload one type of image data** (DICOM, NIfTI, etc.) **at a time**, and **each data type has its own folder structure**.
2. With DICOM & NIfTI volume data, you can also choose to group your data by study. Selecting **Group by Study** allows you to upload multiple scans as a single task. This is useful when you want to view or annotate multiple images (i.e. a full study) at once.

Folder Structure

DICOM 3D Volume

Upload all instances of a DICOM series to a destination folder. If you're only uploading a single series, you can do so without designating a folder.

NIfTI 3D Volume

Individual NIfTI files are uploaded as separate tasks. If you'd like to group your tasks by study, place all of the NIfTI files correlating to a specific task/study in a separate folder.

Image 2D

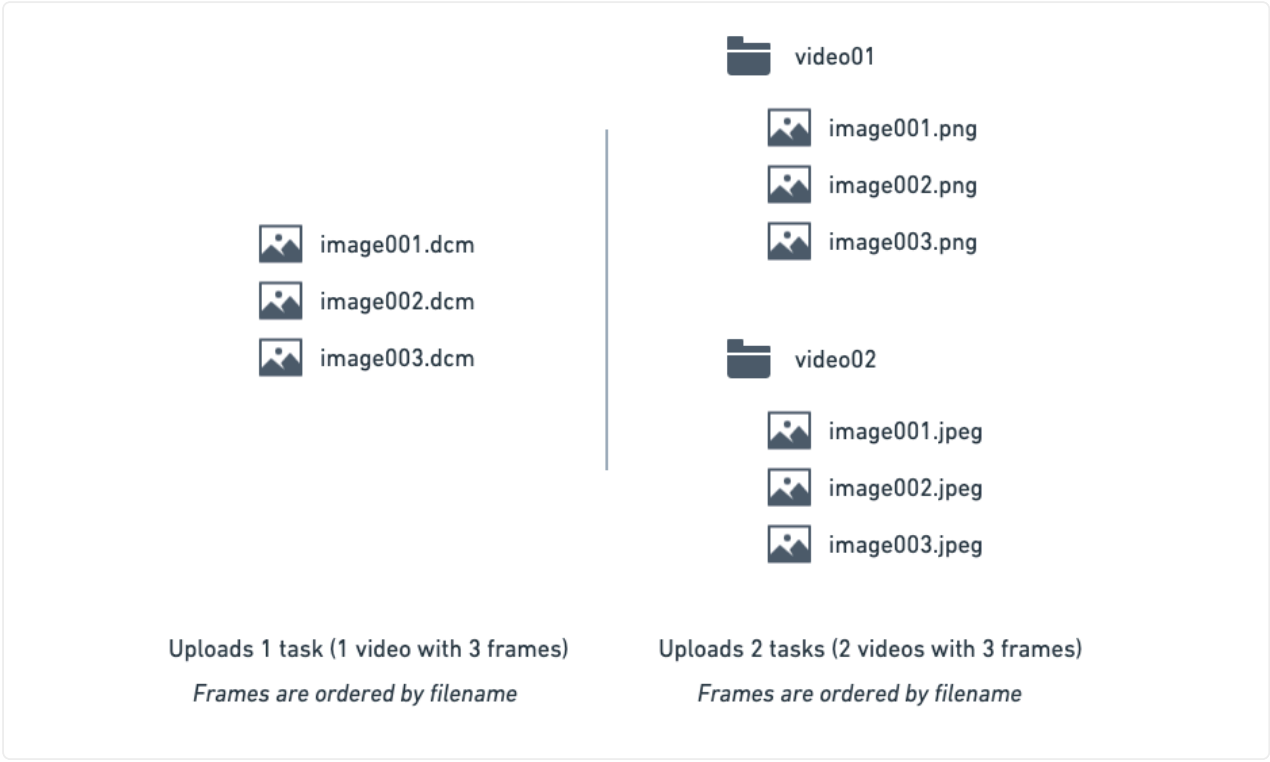
Individual 2D images are uploaded as individual tasks. If you'd like to create a study task with 2D images, please use your external storage or upload data using the CLI.

Video Files

Individual 2D videos are uploaded as individual tasks. If you'd like to create a study task with 2D videos, please use your external storage or upload data using the CLI.

Video Frames

All video frames in a folder are uploaded as a single task and sorted by file name.



Uploads 1 task (1 video with 3 frames)

Frames are ordered by filename

Uploads 2 tasks (2 videos with 3 frames)

Frames are ordered by filename

Import Cloud Data

RedBrick AI supports integration with a range of cloud providers, allowing you to make use of its suite of tools without having to upload your data directly to our servers.

While the steps and configurations required for uploading data from cloud storage vary depending on your provider, the overall procedure remains the same.

1. Configure your cloud storage - AWS s3, Google Cloud Storage, Azure Blob Storage
2. Create and upload an Items List - your Items List communicates the location of your data in your cloud storage to RedBrick AI's platform.

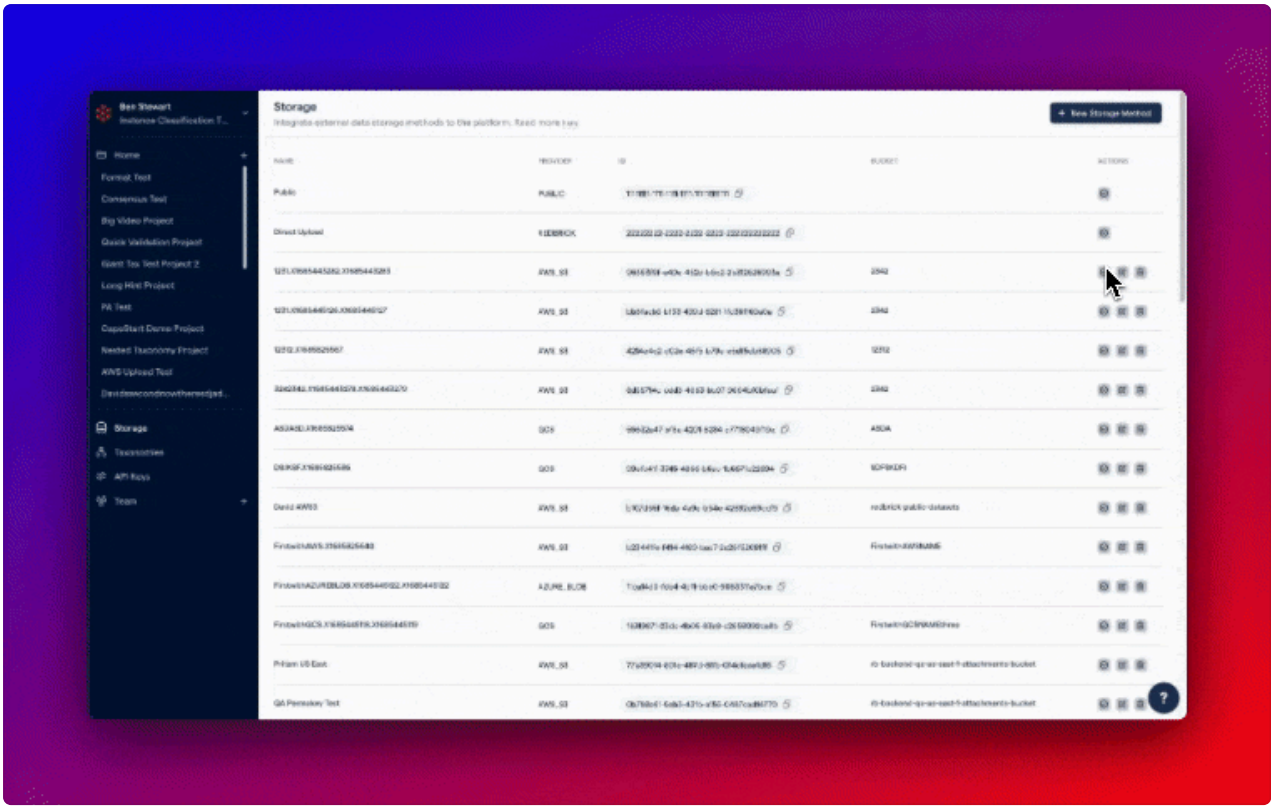
Configuring Cloud Storage

RedBrick AI currently supports the following external storage methods:

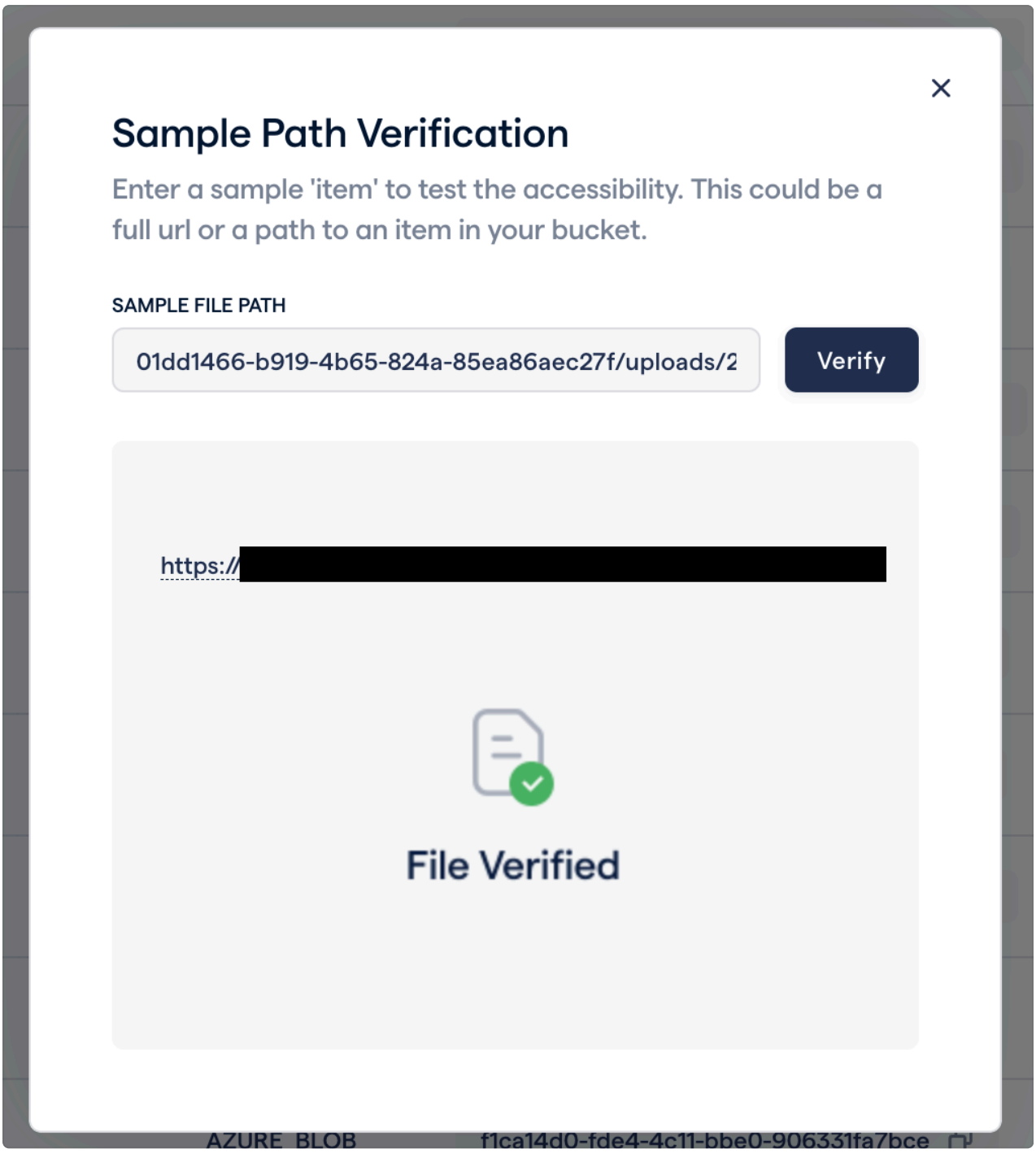
- AWS S3 Buckets
- Google Cloud Storage
- Azure Blob Storage
- Public - this includes data stored on your computer, as well as data stored on any public server accessible via URL

After opening the Storage tab in the lefthand sidebar, click on **New Storage Method** to configure your new Storage Method for further use with RedBrick AI. After inputting the required credentials, click on **Create Storage** to integrate your cloud storage with RedBrick AI.

If you are using AWS, GCS, or Azure for third-party storage, you can also make use of the **Verify** action to ensure that your Storage Method has been successfully configured.



If your Storage Method has been configured correctly, pasting a sample file path to one of your visual assets into the field will display the following:



The **Sample File Path** field expects a file path starting at your bucket, not a full URL.

✕ **Don't Do:** https://s3.region-1.amazonaws.com/redbrick-bucket/project-2/brain-mri.dcm

✓ **Do:** redbrick-bucket/project-2/brain-mri.dcm

Once your Storage Method has been integrated, you can move on to creating an Items List.

Items List

Overview

Your Items List is a JSON file that points the RedBrick AI platform to the data in your external storage and allows you to selectively import data points that can be annotated as single units of work.

> Jargon Alert! Single units of work (i.e. a single row on the Data Page) are referred to as **Tasks** within RedBrick AI.

Your Items List can be uploaded to RedBrick from the Project Dashboard or by using the SDK. Each entry in your Items List will be created as a separate Task, and you can find detailed explanations of each key in our Format Reference.

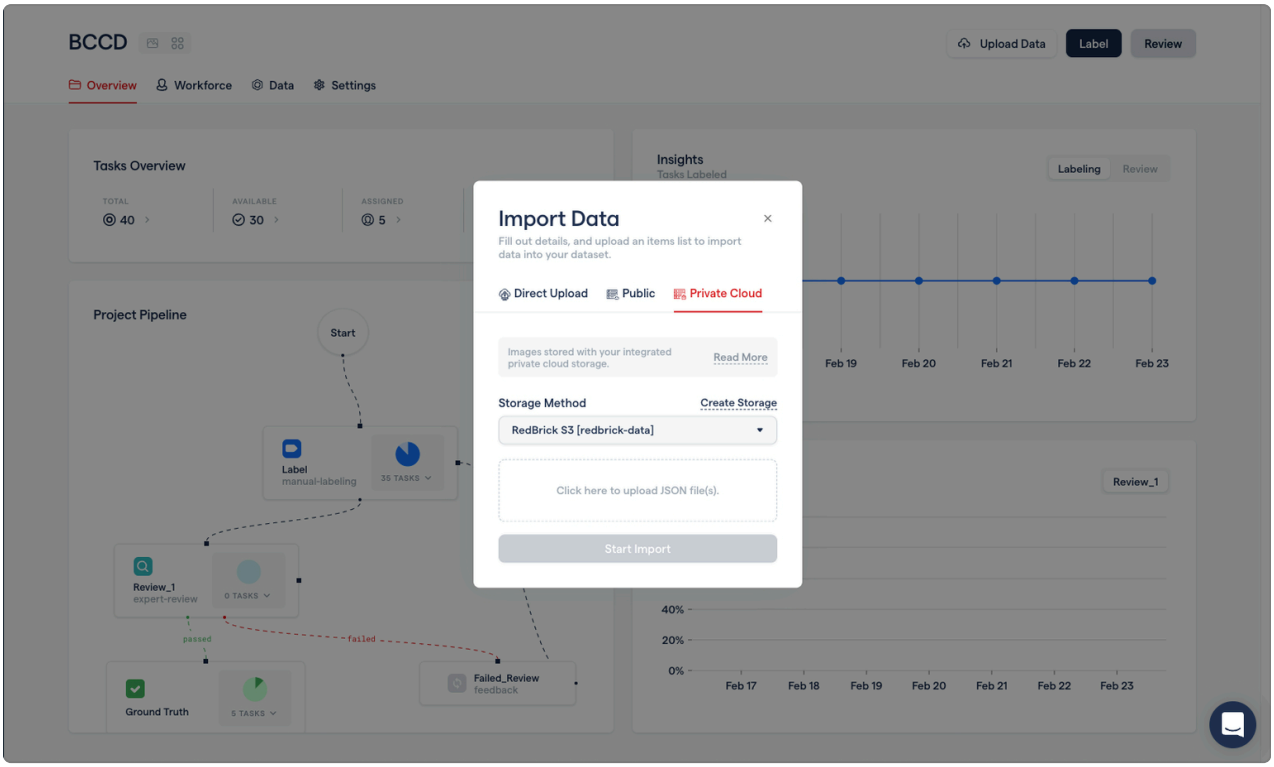
Creating & Configuring an Items List

The format of your Items List depends on both the type of cloud storage you have integrated with RedBrick AI and the type of data you are uploading.

For solution-specific instructions regarding how to format your Items List with AWS S3, GCS, Azure Blob Storage, or a Public source, please refer to the corresponding configuration guide.

Importing Data

Once you have integrated your third-party Storage Method (where applicable) and generated an Items Path, all that you have to do is click on **Import Data** within your Project and upload the Items Path.

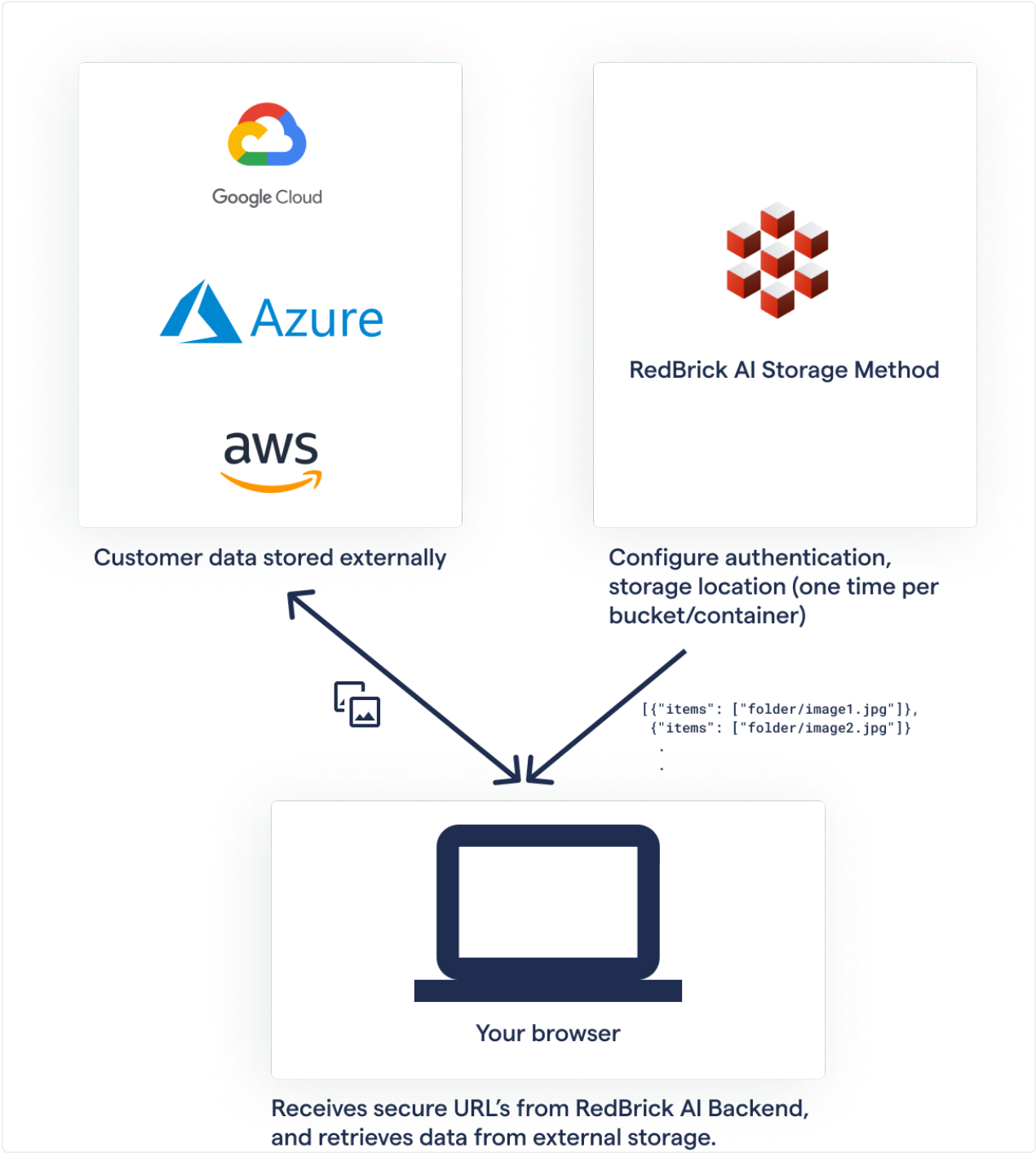


Clicking on Import Data will allow you to upload your Items List

Congratulations! Now you're ready to begin working on RedBrick AI!

Data Privacy

When using an external storage method with RedBrick AI, all of your data is transferred directly from your storage method to your browser.



Your raw data will never be routed through RedBrick AI's servers, downloaded, or duplicated (unless specifically requested for certain features - please review our Privacy Policy).

Supported Data Formats

RedBrick AI supports a variety of different image and volume formats:

1. DICOM - .dcm, .ima, .dicom, .dicm
2. NIfTI - .nii, .nii.gz
3. Videos - .mp4, .mov, .avi
4. RGB Images - .png, .jpeg, .jpg, .bmp
5. NRRD - .nrrd

If you require additional support for a file format that is not present in this list, please reach out to us at support@redbrickai.com.

Configuring AWS s3

This section covers how to prepare your Amazon S3 storage to import data into the RedBrick AI platform. After following the instructions in this section, you will be able to create an Amazon S3 'storage method' on the RedBrick platform to connect your S3 bucket to your RedBrick account.

Signing Up for AWS

The first step to preparing data storage on Amazon S3, is to sign up for an AWS account on https://aws.amazon.com.

Create an S3 Bucket

> Skip creating a bucket if you want to use an existing bucket

Once you are logged in to your AWS account, head over to the Amazon S3 console.

- Create a new bucket, give it a unique name (this will be needed later).
- Select the region of the bucket - we recommend keeping it close to your physical location for the best data transfer experience.
- We recommend **blocking all public access to your s3 bucket**, and giving the RedBrick AI interface access through IAM credentials.
- You can leave all other settings as default.

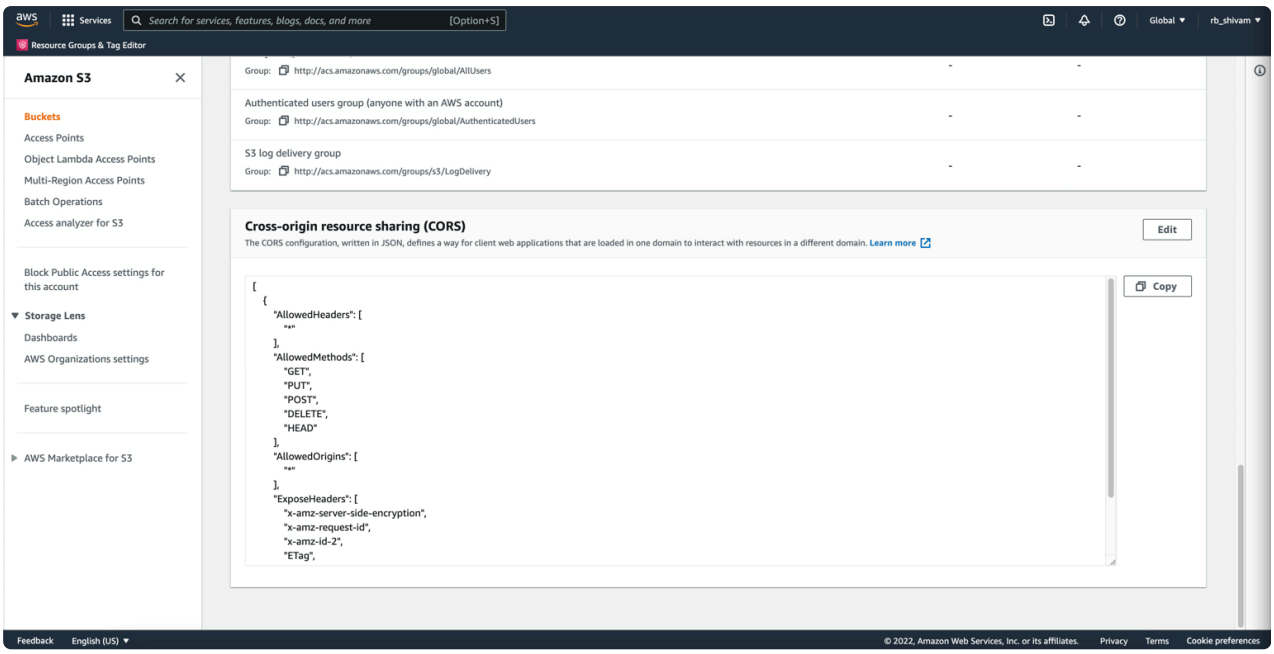
  After creating your S3 bucket, **upload your images** through the User Interface, or use the AWS CLI for large amounts of data.

S3 Bucket Settings

To ensure your data is private and secured, RedBrick uses pre-signed URL's to render data in browsers. To allow RedBrick to use pre-signed URL's to serve data, you need to define a CORS policy on the S3 bucket. Here is the AWS S3 documentation on CORS.

To set the proper CORS policy, go to the **Permissions tab** in your S3 bucket. Under Permissions select the CORS configuration and copy paste the following block of code.

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "PUT",
            "POST",
            "DELETE",
            "GET"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [
            "x-amz-server-side-encryption",
            "x-amz-request-id",
            "x-amz-id-2"
        ],
        "MaxAgeSeconds": 3000
    }
]
```



CORS Permissions on s3 bucket console

Access and Secret Keys

If your S3 Bucket blocks public access to the data, you will have to create an IAM user to allow RedBrick to securely access the data in your S3 bucket. AWS IAM enables you to manage access to your AWS services. You can read about IAM in the AWS documentation.

To create an IAM user from the AWS console, follow these steps:

1. Sign in to the AWS Management Console and open the IAM console
2. In the IAM console navigation pane, choose Users and then choose Add user.

Add User

1. Type the user name for the new user.
2. Select Programmatic Access

Permissions

- Click on Attach existing policies directly
-

- Create a new policy
- **Paste the following block of JSON inside the JSON tab**. Remember to replace `<your_s3_bucket_name_here>` with the name of the S3 bucket that has your data. You can modify the Resource path for added security or specificity.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RedBrickLabelingReadOnly",
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::<your_s3_bucket_name_here>/*"
    }
  ]
}
```

> If your bucket will be used as an annotation storage bucket, you need to add PUT object access as well.

- Review your policy and create it.
- Head back to your IAM user creation and attach the policy you just created.

> **Granular Permissions**
>
> You can configure your S3 bucket to give RedBrick AI access to particular data points inside your s3 bucket by modifying the "Resource" section in the file above.

Create the user

- Create the user and **download Access and Secret key .csv**
- Store the CSV file with your keys carefully.

Items Path

Once you've created your AWS Storage method on RedBrick AI, you have to upload an items list to your projects to import specific datapoints. Please have a look at the items list documentation for a overview of the format for the JSON file.

For data stored in an AWS s3 bucket, the `items` path needs to be formatted as follows:

```
"root-folder/sub-folder/datapoint.dcm"
```

Where `root-folder` is inside the AWS s3 bucket storage method.

Programmatically Generate Items List For S3

If you don't have a standard naming convention for your files inside your s3 bucket, or you're not sure which files are in the s3 bucket, you can use the AWS CLI to enumerate a list of all the objects inside a bucket. Using this list, you can programmatically generate an items list.

Installing and Configuring the AWS CLI

Have a look at the AWS documentation for installing the CLI. For mac users, running `brew install awscli` is the easiest way to install the AWS CLI.

Once you have the AWS CLI installed, you need to configure the cli with your AWS root account Access Key and Secret Key to give permissions to AWS CLI. After installing AWS CLI, do the following.

```
$ aws configure
AWS Access Key ID [None]: <your_access_key>
AWS Secret Access Key [None]: <your_secret_key>
Default region name [None]:
Default output format [None]:
```

After filling out your keys, you can leave the last two fields empty.

Now that your AWS CLI is configured, you can list out the objects inside your s3 bucket by running the following command.

```
$ aws s3 ls s3://<your_bucket_name>
```

After listing out all the images inside the s3 bucket (or a folder in the bucket), you can save the output to a txt file and write a simple script to convert that output into a JSON file in the format of the Items List covered above.

Configuring Azure Blob

Create a Storage Account

> If you already have a storage account, *skip* this section, and head directly to the next section.

An Azure storage account contains all of your Azure Storage data objects, you can find the official documentation here.

1. Sign in to your Azure portal.
2. On the left portal menu, select **Storage Accounts** to list all of your storage accounts.
3. On the **Storage Accounts** page, click **create**.

Basics Tab

You need to fill out the required fields under the basics tab, please refer the table below as a quick guide.

Section	Field	Description
Project details	Subscription	Select the subscription for the new storage account.
Project details	Resource group	Create a new resource group for this storage account, or select an existing one. For more information, see Resource groups.
Instance details	Storage account name	Choose a unique name for your storage account.
Instance details	Region	Select the appropriate region for your storage account. For more information, see Regions and Availability Zones in Azure.

Section	Field	Description
Instance details	Performance	Select your desired level of performance, or choose the default option.
Instance details	Redundancy	Select your desired redundancy configuration.

Other Settings

To configure other advanced settings on your storage account, head to advanced tabs, otherwise you can continue with default settings.

Review + create

When you navigate to the Review + create tab, Azure runs validation on the storage account settings that you have chosen. If validation passes, you can **proceed to create the storage account**.

Create a Container

> If you already have a container, please skip to the next section.

Containers organizes a set of blobs, your Azure storage account can have an unlimited number of containers. To create a container, head to your Azure portal:

1. Head to the **Storage Accounts** page from the left portal menu, and select the **Storage Account** you want to create your container in.
2. On the left menu of the Storage Account, scroll to the **Data Storage** section, then select **Containers.**
3. Create a container by clicking on the **+ Container** button.
4. Type in a **name** for the container, and set the **level of public access** to the container (we recommend **Private**)

Get your Connection String

1. Navigate to your **Storage Account** on your Azure portal.
2. In the left menu of the Storage Account, scroll to **Security + Networking**, and select **Access Keys.**
3. On the Access Keys page, click on **Show keys** at the top, and **copy one of the connection strings**

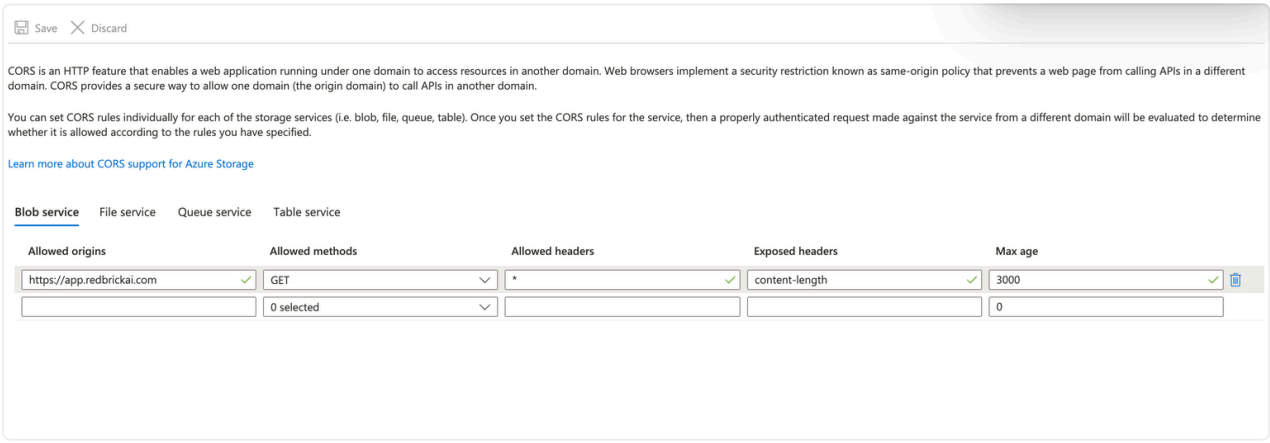
Create a RedBrick Storage Method

Head over to your RedBrick AI Account:

1. Click on the **Storage Method** tab on the left sidebar, and **Create New Storage Method.**
2. In the creation dialog, select **Azure Blob** as the storage type and enter your **connection string,** and **storage account name.**

Enable CORS on your Storage Account

To ensure your data is private and secured, RedBrick uses signed URL's to render data in browsers. To allow RedBrick to use signed URL's to serve data, you need to enable CORS on the Storage Account. This can be done from your **Storage Account -> Settings -> Resource Sharing (CORS).** We recommend the following CORS policy:



Verify your Azure connection

Once you've added your Azure storage method on RedBrick AI, you can verify the connection by doing the following:

1. First upload an image to your container within your azure storage account (e.g. `image.png`)
2. Head to the Storage Method page on RedBrick AI, and click on the **verify** button of the storage method you just created.
3. Paste the unique path of your blob, which will be in the following format: `container_name/blob_path` . So if you uploaded `image.png` within the sub-folder `images` in your container `image-container` , your path would be `image-container/images/image.png` .
4. If the connection was successful, you should see the image appear once you verify.

Items Path

Once you've created your Azure Storage method on RedBrick AI, you have to upload an items list to your projects to import specific datapoints. Please have a look at the items list documentation for a overview of the format for the JSON file.

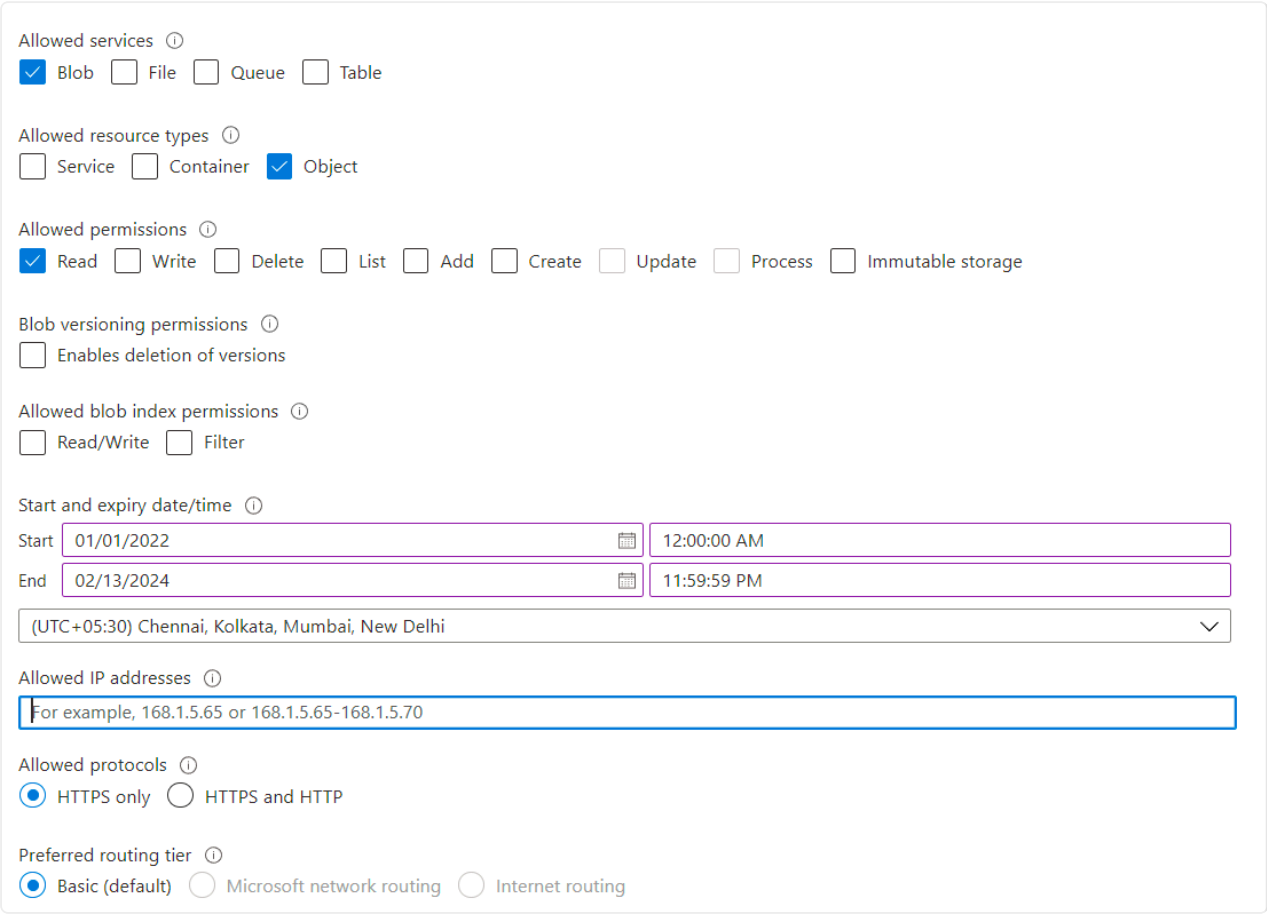
For data stored in an Azure container, the `items` path needs to be formatted as follows:

```
"container-name/root-folder/sub-folder/image.png"
```

Where `container-name` is inside the Storage Account.

Shared Access Signature

You can also used a "Shared Access Signature" URL for enabling access to your bucket through RedBrick AI.



Example access configuration

- Expiry should be until the time you want to have access to your data through RedBrick AI, with this time expires you will lose access to your data through RedBrick AI and will have to update the configuration. We recommend giving this at least a few years, you can always cancel the access later.
- IP address (optional) could be the user's permanent network address range
- Only Read permissions are necessary

Upload Items

When you want to upload data that is in a connected storage method to the RedBrick AI platform you will do this as an Items List. Depending on which level of permissions you want to restrict access to with your SAS URL, you will need to create your "items" differently. These items tell RedBrick AI where to find your data.

Permissions	Sample item
Connection string and Service level SAS:	"container/folder/item.jpg"

Permissions	Sample item
Container level SAS:	"folder/item.jpg"
Blob level SAS: (not recommended)	""

After creating your storage method integration, we recommend you test the way you generate items using the "verify" feature. This will perform pre-signing and check if your browser is able to fetch the image from your bucket.

Configuring GCS

This section covers how to prepare your GCS storage to import data into the RedBrick AI platform. After following the instructions in this section, you will be able to create an GCS 'storage method' on the RedBrick platform to connect your GCS bucket to your RedBrick account.

Signing up for Google Cloud Platform

The first step to preparing data storage on GCP is to create a account on GCP.

Create a bucket within a project

Once you have your GCP account and a project within your account, a bucket and upload your data within the bucket either through the UI or CLI. You can leave all the settings during creation as the default. After creating your bucket, upload your data into the bucket.

Create a Service Account

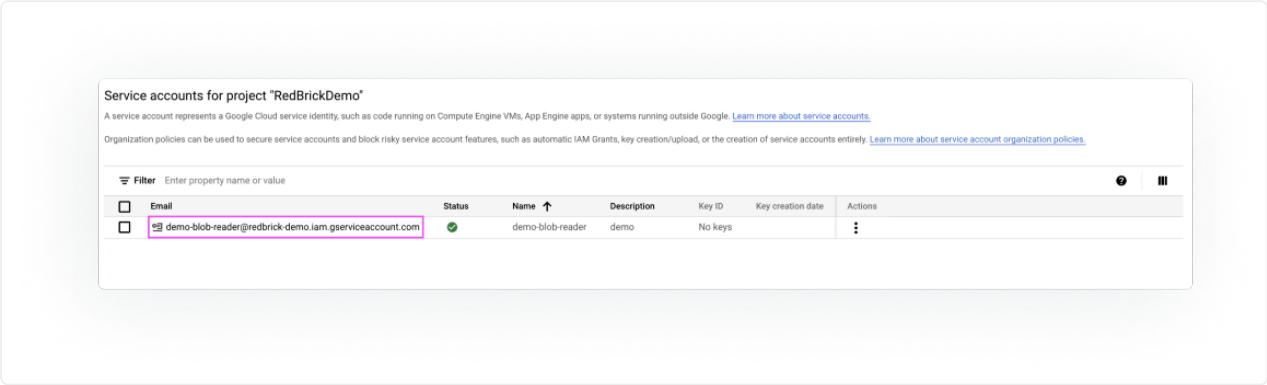
A service account is a special kind of account used by an application or a virtual machine (VM) instance, not a person. Applications use service accounts to make authorized API calls, authorized as either the service account itself, or as Google Workspace or Cloud Identity users through domain-wide delegation.

1. In the Cloud Console, go to the Service Accounts page.
2. Select the appropriate project.
3. Click Create service account.
4. Enter a service account name to display in the Cloud Console.

   For eg, `gcs-blob-reader`

The Cloud Console generates a service account ID based on this name. Edit the ID if necessary. You cannot change the ID later.

5. *Optional:* Enter a description of the service account.
6. Click Create and continue to the next step.
7. Add the following two IAM roles to grant to the service account on the project.
    1. **Storage Object Viewer** (For reading the blobs from the GCS bucket)
    2. **Service Account Token Creator** (For pre-signing the blobs)
    3. If your bucket will be used as an annotation storage bucket, you need to give RedBrick AI access to PUT files as well.
8. Once done adding roles, click Continue.
9. Click Done to finish creating the service account.
10. Note down the email id of the newly created service account.



Steps to give Bucket access to Service Account

1. In the Cloud Console, go to the Bucket Browser page.
2. Click on the more actions button (three dots) at the right of the necessary bucket.
3. Click on `Edit Bucket Permission`.
4. Click on `ADD PRINCIPAL`.
5. Add the email address of the service account created in the above step (step 10).
6. Add following two roles

    Storage Legacy Bucket Reader

Storage Legacy Object Reader

7. Click on save.

Steps to create Service Account Key (JSON)

To use a service account from outside of Google Cloud, such as on other platforms or on-premises, you must first establish the identity of the service account. Public/private key pairs provide a secure way of accomplishing this goal. When you create a service account key, the public portion is stored on Google Cloud, while the private portion is available only to you.

1. In the Cloud Console, go to the Service Accounts page.
2. Click the email address of the service account that we created in the above step.
3. Click the Keys tab.
4. Click the Add key drop-down menu, then select Create new key.
5. Select JSON as the Key type and click Create.
6. Clicking Create downloads a service account key file. After you download the key file, you cannot download it again.

The downloaded key has the following format, where private-key is the private portion of the public/private key pair:

```
{
  "type": "service_account",
  "project_id": "project-id",
  "private_key_id": "key-id",
  "private_key": "-----BEGIN PRIVATE KEY-----\nprivate-key\n-----END PRIV
  "client_email": "service-account-email",
  "client_id": "client-id",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://accounts.google.com/o/oauth2/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/ce
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x
}
```

Make sure to store the key file securely, because it can be used to authenticate as your service account. You can move and rename this file however you would like.

Enable the Identity and Access Management (IAM) API

1. In the Cloud Console, go to the API Library page.
2. Select the appropriate project.
3. Search for **Identity and Access Management (IAM) API** and click on it.
4. Click on `Enable`

> **Use this downloaded service account JSON key to create `GCS` store in the Redbrick app.**

Configuring CORS

The final step in preparing your GCS bucket for use with RedBrick AI is enabling CORS on the bucket. Cross-Origin Resource Sharing allows the RedBrick AI application (with the domain https://app.redbrickai.com/) to make requests to your bucket.

Please visit the following Google Cloud documentation for enabling CORS on a bucket - https://cloud.google.com/storage/docs/configuring-cors.

You will want to use the following CORS JSON configuration file:

```
[
    {
      "origin": ["https://app.redbrickai.com"],
      "method": ["GET"],
      "responseHeader": ["Content-Type"],
      "maxAgeSeconds": 3600
    }
]
```

- [x] If you're also storing your annotations in your GCS bucket, be sure to add "PUT" to your method configuration array (e.g. "`["GET", "PUT"]`").

Items Path

Once you've created your Google Storage method on RedBrick AI, you have to upload an items list to your projects to import specific datapoints. Please have a look at the items list documentation for an overview of the format for the JSON file.

For data stored in an GCS bucket, the `items` path needs to be formatted as follows:

```
"root-folder/sub-folder/datapoint.dcm"
```

Where `root-folder` is inside the GCS bucket storage method.

Configuring AltaDB

This section covers how to prepare your AltaDB storage to import data into the RedBrick AI platform. After following the instructions in this section, you will be able to create an AltaDB 'storage method' and import Data into RedBrick AI.

Signing up on AltaDB Platform

The first step is to create an account on AltaDB.

Dataset Creation & Importing Data

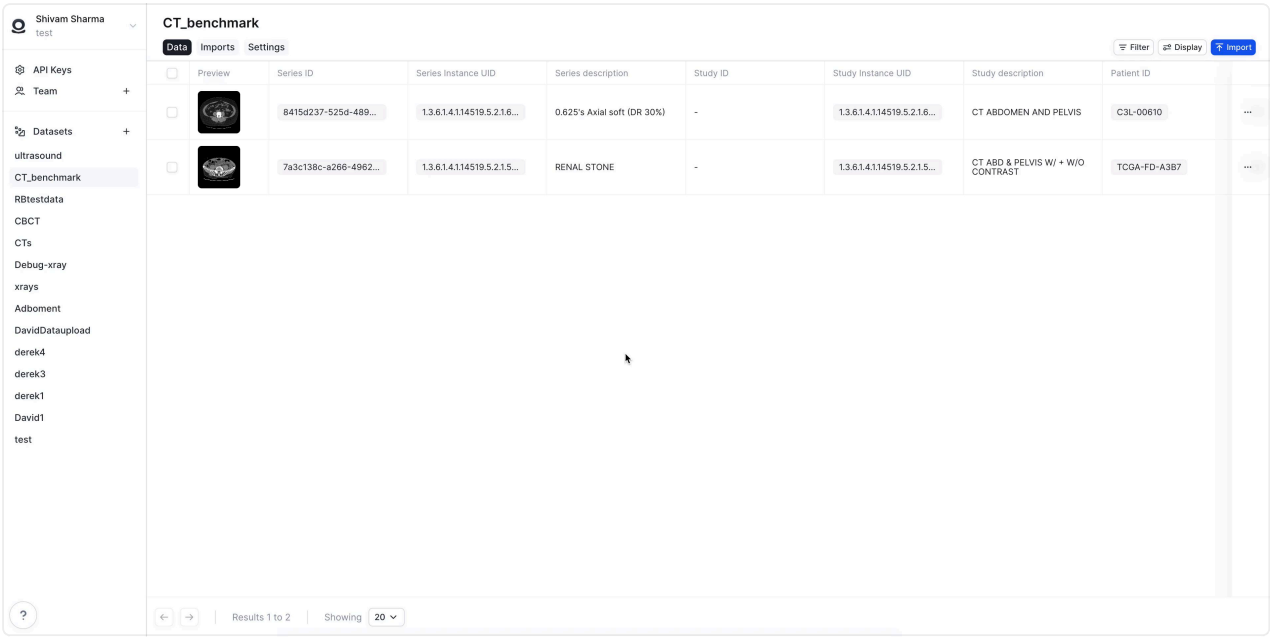
1. Create a dataset.
2. Click on Import data, and drag & drop your DICOM files into the data importer. **AltaDB v0.0 only supports direct upload through the browser**.
3. Once the upload is complete, AltaDB will index the DICOM headers and re-compress the DICOM pixel data with state-of-the-art compression. This process will take a few minutes for 10 series.
4. You can monitor the status of your imports inside the "Import" tab.

> Processing an import will take several minutes to completely index all DICOM headers and re-compress pixel data with state-of-the-art compression

Viewing the Data

You can preview your images by clicking on the thumbnail. The simple viewer offers a view of the imaging axis and supports a few key functions:

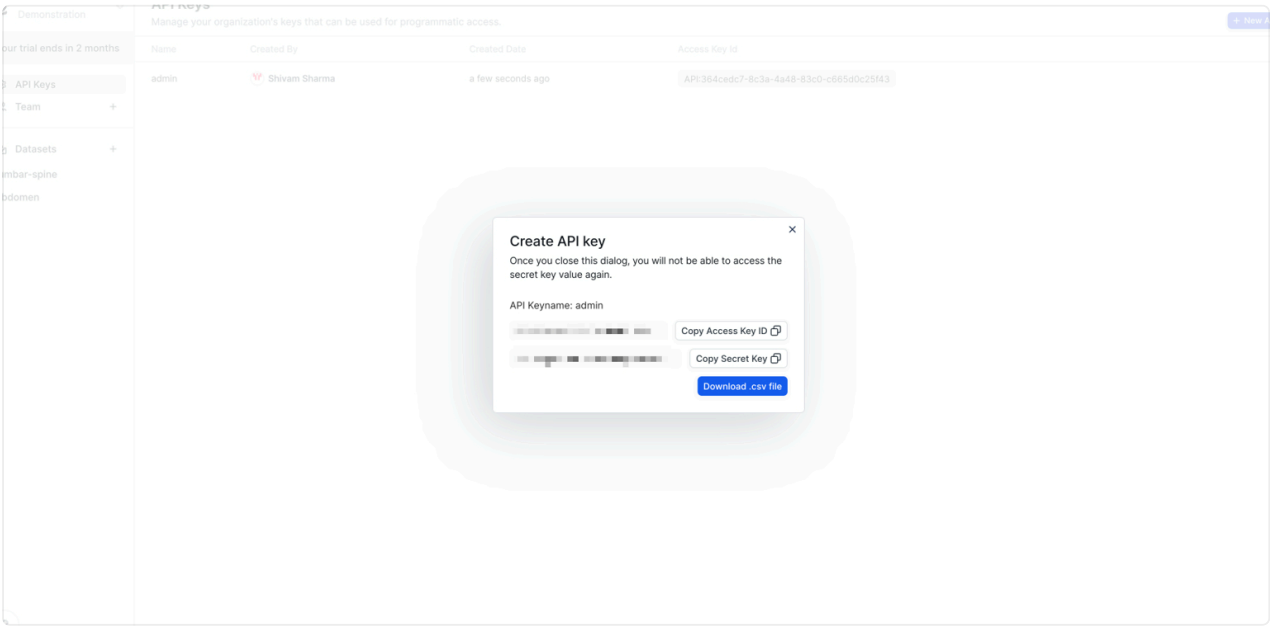
1. Change slice: Mouse scroll.
2. Windowing: Right-click drag.
3. Zoom: Left-click drag.
4. Pan: Middle mouse drag.



Integrating with RedBrick AI

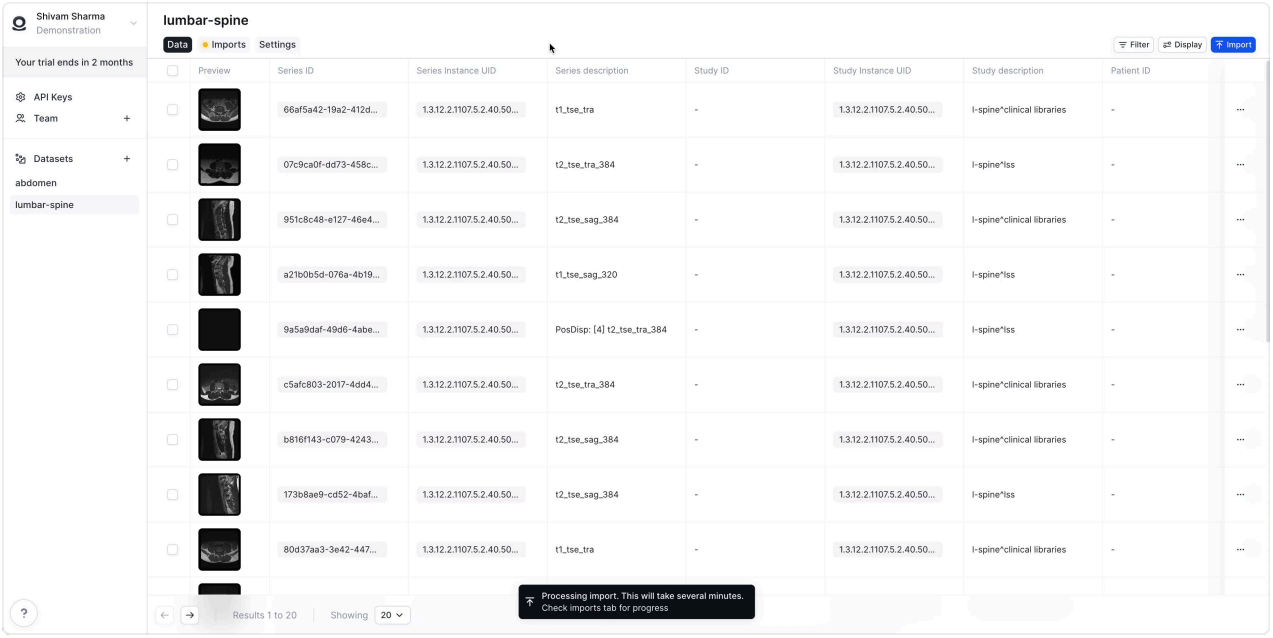
Create API key on AltaDB

To Create a new API key, Go to API keys in the left menu sidebar and click on "New API key" button on the top right corner. Once created, note down the "access" and "secret" keys.



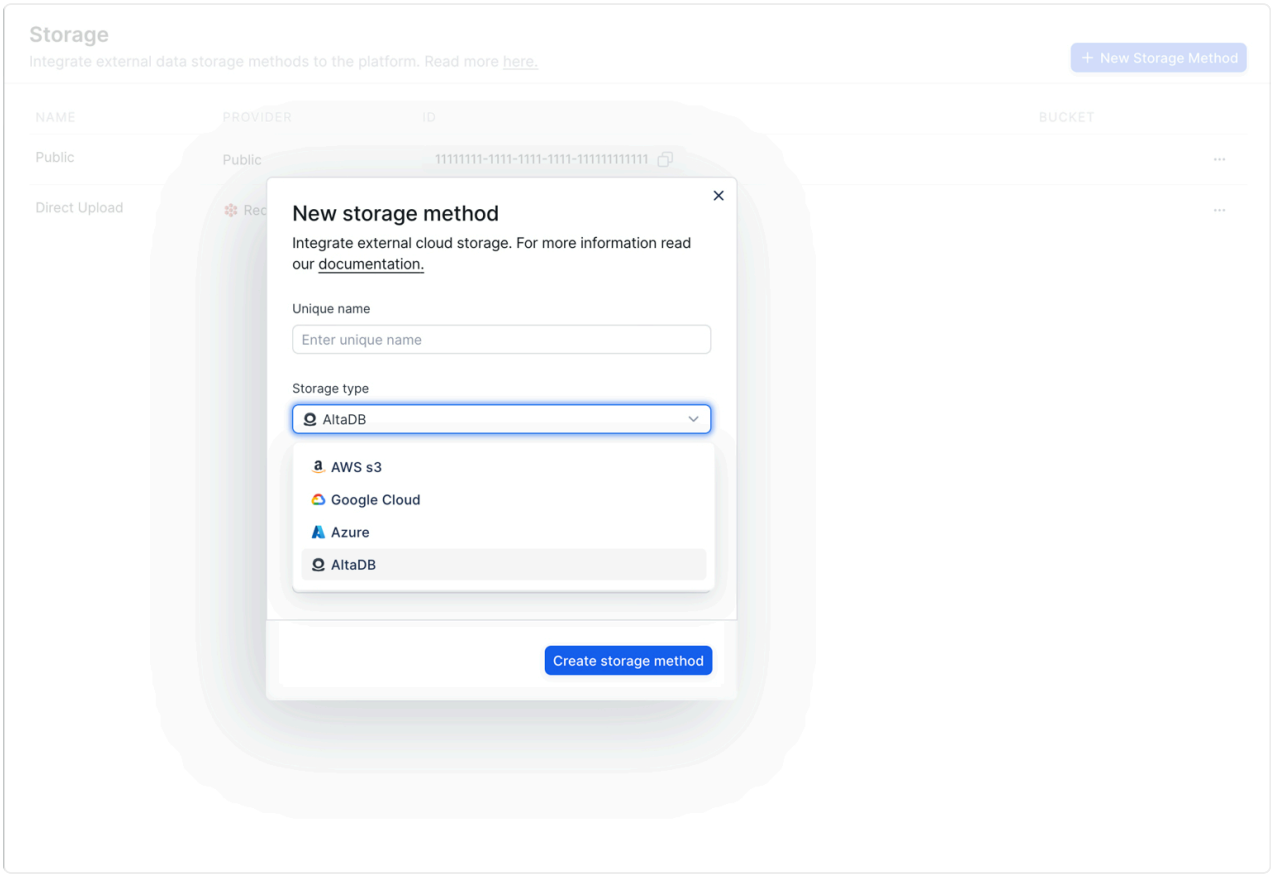
Download the JSON from AltaDB

In your AltaDB dataset, select the series you want to import into RedBrick AI and click on the "Download RedBrick AI JSON button".



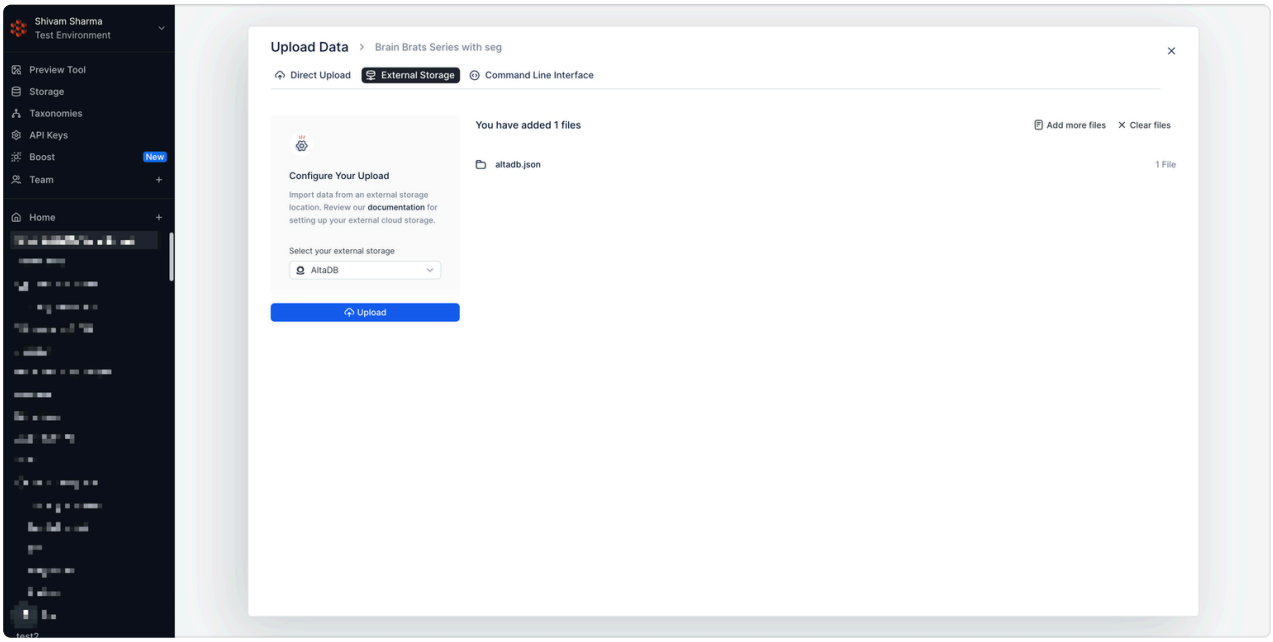
Create AltaDB storage method on RedBrick AI

Next step is to create an AltaDB storage method in your RedBrick AI account. You will need your AltaDB Access and Secret keys to setup this storage method.



Importing Data into RedBrick AI

To import data into RedBrick AI, Upload the JSON file you downloaded from AltaDB and Select AltaDB from the dropdown in the external storage method.



Creating an Items List

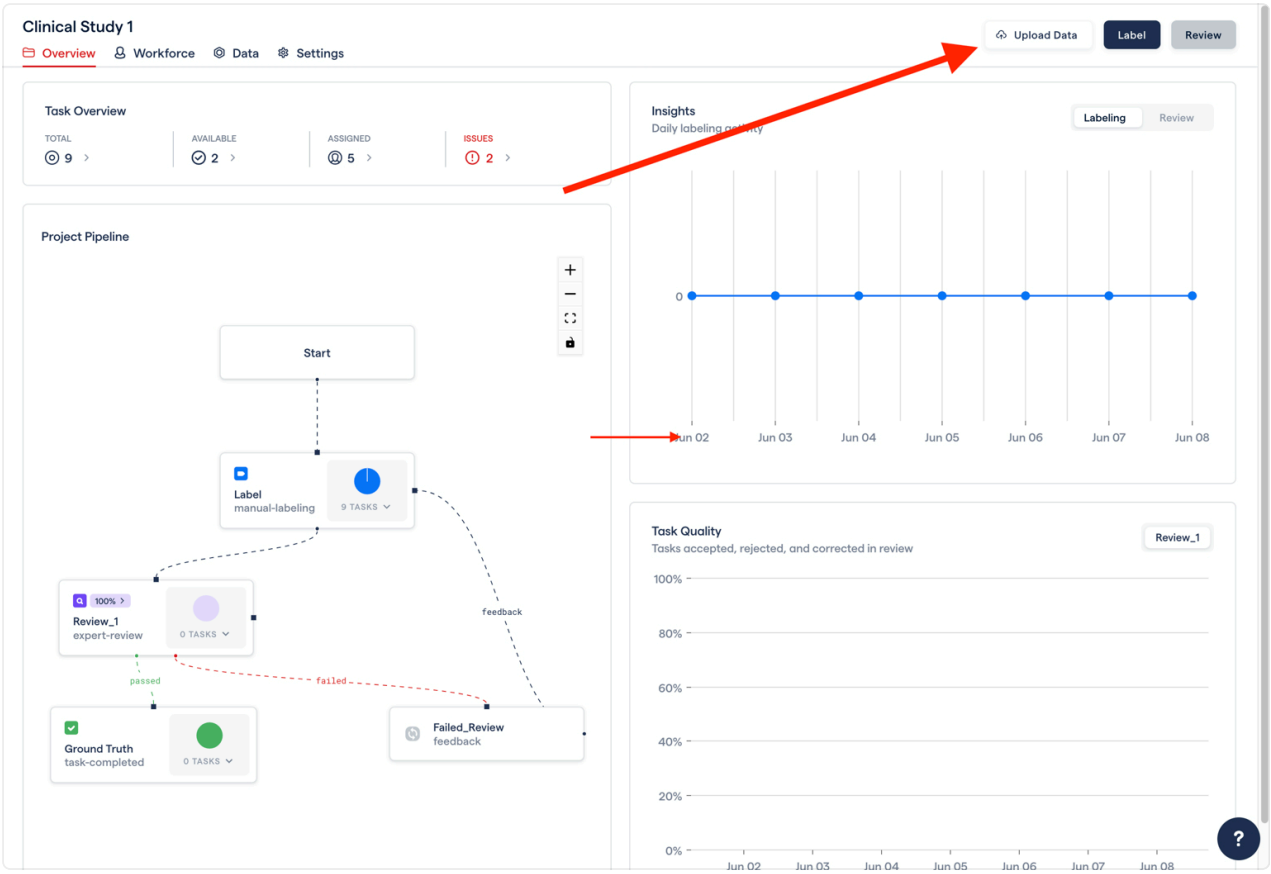
Overview

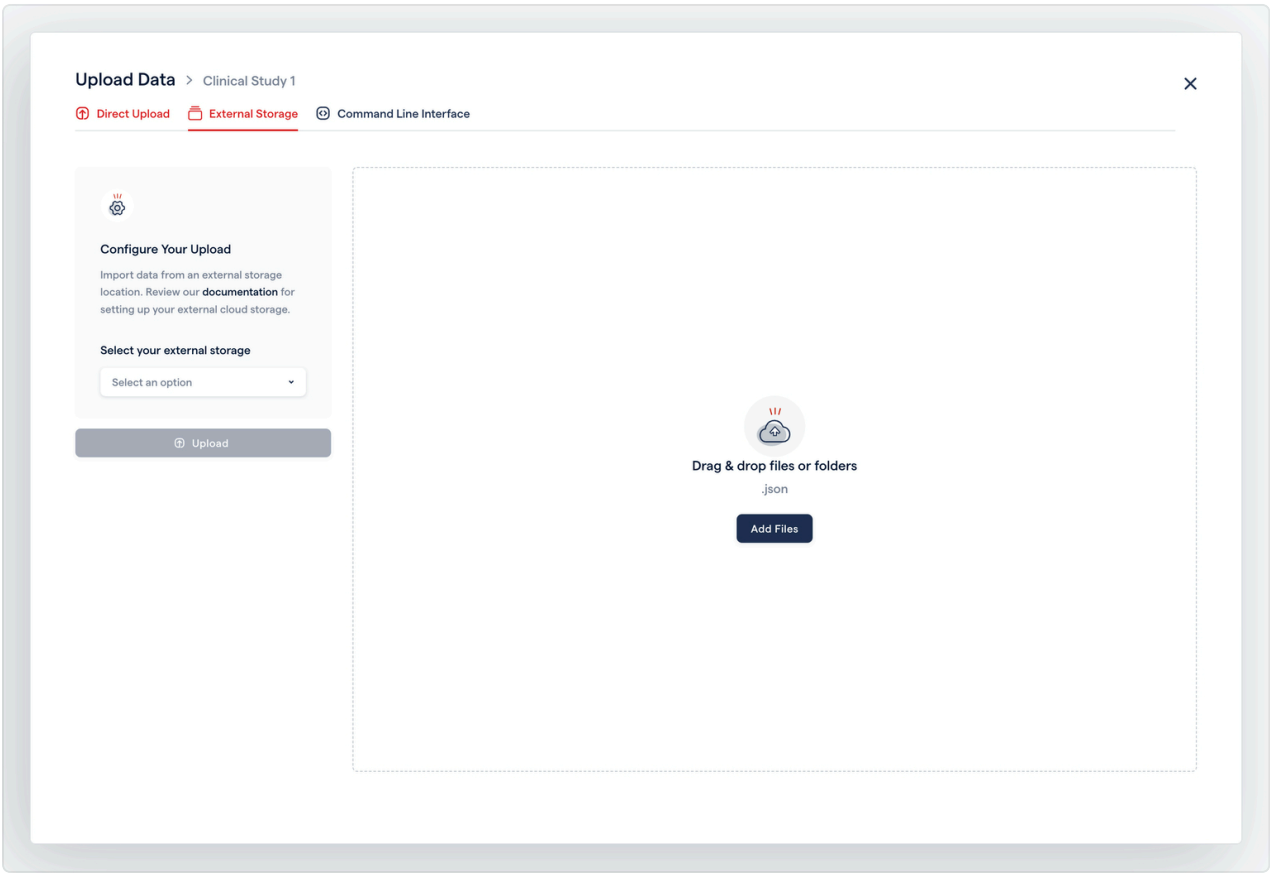
An Items List is a JSON file that points the RedBrick AI platform to the data in your external storage and allows you to selectively import data points. The format of your Items List depends on both the type of cloud storage you have integrated with RedBrick AI and the type of data you are uploading.

For solution-specific instructions regarding how to format your Items List with AWS S3, GCS, or Azure Blob Storage, please refer to the corresponding configuration guide.

It's important to note that each entry in your Items List **will be created as a separate Task**, which can then be annotated as a single unit. You can find detailed explanations of each key in our Format Reference.

After creating your JSON Items List, you can upload it from RedBrick AI's Project Dashboard or by using the SDK.





Select a Storage Method and upload your Items List

Please note that there is no need to create an Items List when using Direct Upload.

Example Items Lists

The example below contains fields relevant to image-only uploads.

```
type Items = Task[];

interface Task {
    // A unique, user-defined ID
    // After import, you can search tasks using this field.
    name: string;

    // You can upload a single series, or an entire study (array of series
    series: Series[];
}

interface Series {
    // Filepath/URL's of all the instances in a single series.
    items: string[] | string;
    name: string;
}
```

The `items` entry enumerates the file paths referencing your data in your cloud storage. Depending on the storage method, this file path may be relative to your bucket name or the root folder in your bucket. Please reference the relevant documentation to verify the format of the Items List for each of RedBrick AI's supported storage methods:

- AWS S3 Items List
- Azure Blob Items List
- GCS Items List

Example Item Lists by Format

3D DICOM

3D DICOM Study

This Items List will upload a single Task containing two Series.

```
[
  {
    "name": "study001",
    "series": [
      {
        "items": [
          "study001/series001/001.dcm",
          "study001/series001/002.dcm",
          "study001/series001/003.dcm"
        ]
      },
      {
        "items": [
          "study001/series002/001.dcm",
          "study001/series002/002.dcm",
          "study001/series002/003.dcm"
        ]
      }
    ]
  }
]
```

3D DICOM Series

This Items List will upload two Tasks, each containing a single Series.

```
[
  {
    "name": "series001",
    "series": [
      {
        "items": [
          "series001/001.dcm",
          "series001/002.dcm",
          "series001/003.dcm"
        ]
      }
    ]
  },
  {
    "name": "series002",
    "series": [
      {
        "items": [
          "series002/001.dcm",
          "series002/002.dcm",
          "series002/003.dcm"
        ]
      }
    ]
  }
]
```

NIfTI

> Please note that `items` must be a single string for NIfTI uploads.

NIfTI Series

This Items List will upload a single Task containing two Series.

```
[
  {
    "name": "study001",
    "series": [
      {
        "items": "series001.nii"
      },
      {
        "items": "series002.nii"
      }
    ]
  }
]
```

NIfTI Study

This Items List will upload two Tasks, each containing one Series.

```
[
  {
    "name": "series1",
    "series": [
      {
        "items": "series001.nii"
      }
    ]
  },
  {
    "name": "series2",
    "series": [
      {
        "items": "series002.nii"
      }
    ]
  }
]
```

2D Image

2D Image Study

This Items List will upload a single Task containing two images.

```
[
  {
    "name": "patient1",
    "series": [
      {
        "items": "scan1.dcm"
      },
      {
        "items": "scan2.dcm"
      }
    ]
  }
]
```

2D Image Series

This Items List will upload two Tasks, each containing a single image.

```
[
  {
    "name": "patient1",
    "series": [
      {
        "items": "scan.dcm"
      }
    ]
  },
  {
    "name": "patient2",
    "series": [
      {
        "items": "scan.dcm"
      }
    ]
  }
]
```

Video Frames

The frames must be in the correct order in the `items` array.

Video Frames Study

This Items List will upload a single Task with two videos, where each video contains three frames.

```
[
  {
    "name": "study001",
    "series": [
      {
        "items": [
          "study001/series001/001.png",
          "study001/series001/002.png",
          "study001/series001/003.png"
        ]
      },
      {
        "items": [
          "study001/series002/001.png",
          "study001/series002/002.png",
          "study001/series002/003.png"
        ]
      }
    ]
  }
]
```

Video Frames Series

This Items List will upload two Tasks, each containing a single video with three frames.

```
[
  {
    "name": "video1",
    "series": [
      {
        "items": [
          "001.png",
          "002.png",
          "003.png"
        ]
      }
    ]
  },
  {
    "name": "video2",
    "series": [
      {
        "items": [
          "001.png",
          "002.png",
          "003.png"
        ]
      }
    ]
  }
]
```

Automatically Split Study

In some use cases, you can rely on RedBrick AI to split your Study into a list of Series. This can be especially useful if you do not follow a strict naming convention for your studies.

You can upload a single Series or multiple Series per task using the simplified Study-Level format.

Please note that any Tasks uploaded using this format will only be automatically split **after** a user opens the Task in the Annotation Tool.

```
type Items = Task[];

interface Task {
  // A unique, user-defined ID
  // After import, you can search tasks using this field.
  name: String;

  // You can upload a single series, or an entire study (array of .dcm fi
  // The items array will automatically be split into individual series.
  items: String[]
}
```

DICOM

The Items List below will create a single task containing one or more series. RedBrick AI will parse the DICOM files on the client side and automatically split this list of `.dcm` into one or more series (depending on the DICOM headers).

```
[
  {
    "name": "study001",
    "items": [
      "bbfa85feb36f.dcm",
      "d4a49634cd4c.dcm",
      "eed2e7462ba5.dcm",
      "45455dd0e45b.dcm"
    ]
  }
]
```

NIfTI

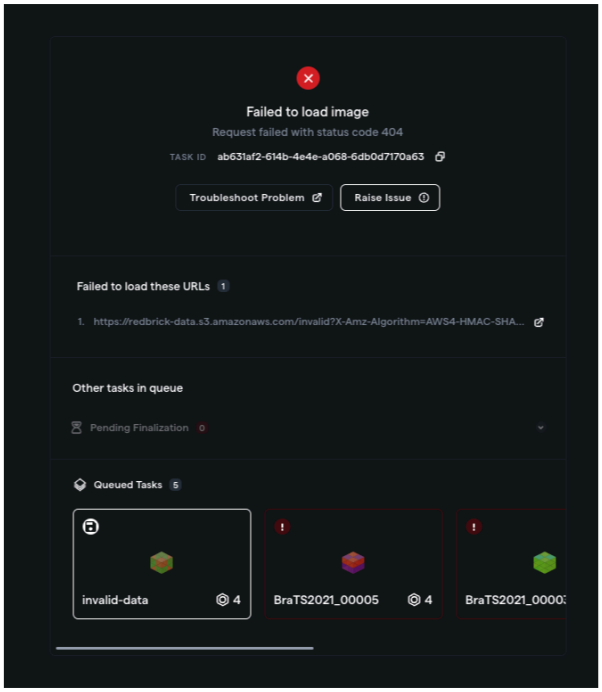
The Items List below will create a single task containing exactly two series. RedBrick AI will assume each of the NIfTI files in the `items` array is an individual series.

```
[
  {
    "name": "study001",
    "items": [
      "bbfa85feb36f.nii.gz",
      "d4a49634cd4c.nii"
    ]
  }
]
```

This format is the same as the Legacy Items List format (pre-July 2022)

Troubleshooting

This section covers troubleshooting issues with data loading in the Annotation Tool. If there is an error in loading your data, you will encounter something very similar to the following:



A typical 404 error

General Troubleshooting Walkthrough with Example

Data can fail to load for a variety of reasons, from network issues to invalid file pathing to incompatible image headers and more.

Let's walk through some basic steps using the example above that will help you determine how to best troubleshoot an error.

1. **Check for specific information** - you may have a specific error message under **Failed to load image**, such as in the example above - "Request failed with status code 404". The error messages that RedBrick AI generates for you can provide valuable insight into why your data is not loading.

In this example, a 404 error, or a "Not Found" error, likely means you have an issue with your external storage integration or internet connection.

2. **Use the context you have** - RedBrick AI displays a 404 error when you attempt to open an image/volume that RedBrick AI can't find, which typically implies that:
    1. There is an issue with your **file path**, i.e. you're asking RedBrick to open something that doesn't actually exist in that specific location;
    2. There is an issue with the **image/volume itself,** i.e. you're asking RedBrick to access a file that may have been renamed, moved, deleted, etc.;
    3. There is an issue with your **network**, i.e. your internet connection was disrupted or failed while you were trying to load the image/volume;
    4. There is an issue with your **integration**, i.e. there's something wrong with the configuration of your external storage.

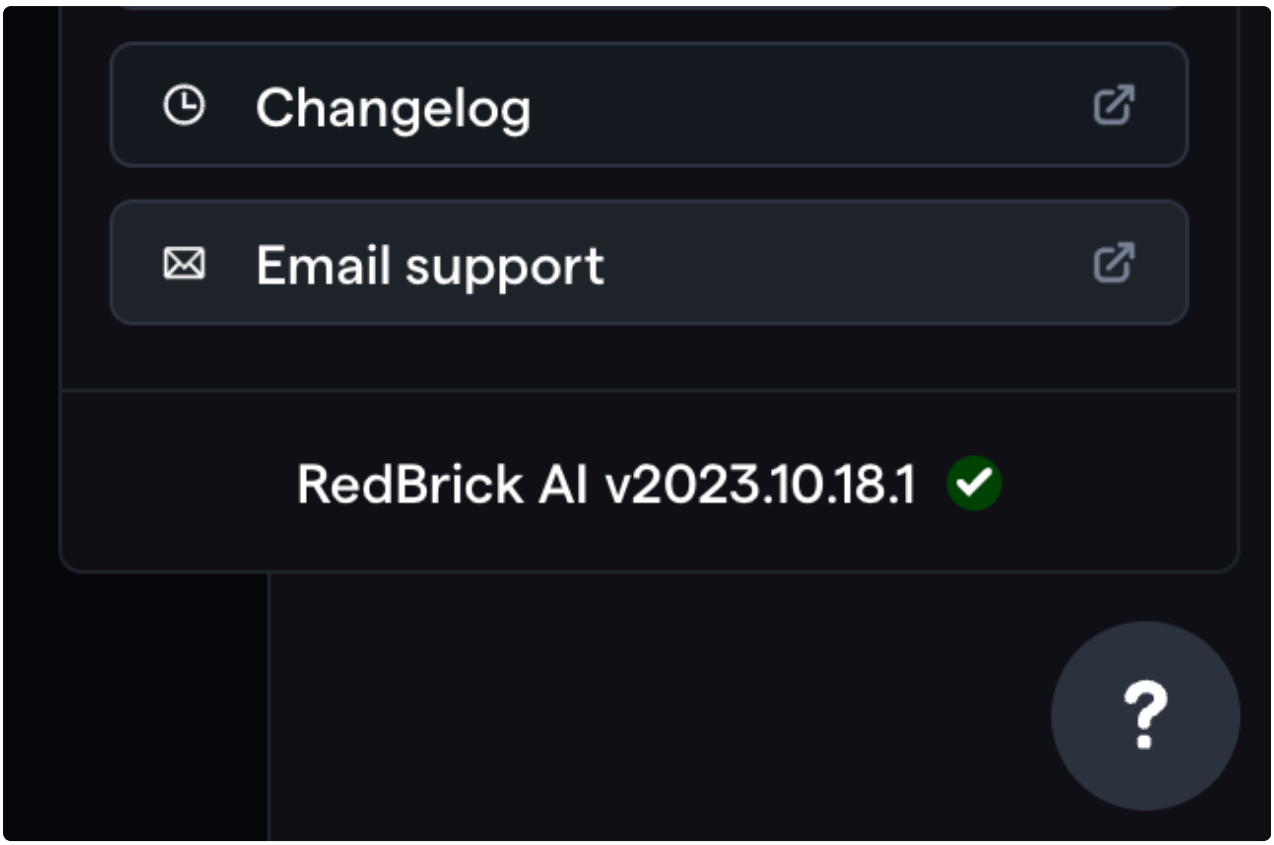
> ✓ Your admins can verify the validity of any Storage Method on the Storage Page.

2. **Escalate appropriately** - in the case of our 404 error, it may be best to reach out to a Team Lead to verify that all files were correctly uploaded to your Project.

Contacting RedBrick AI Support

If you don't have a specific error message to work with or would simply like RedBrick AI's Support Team to investigate, we're always standing by to help!

You can either email us at support@redbrickai.com or click on the Help Button, and then on **Email support**.



Common Information to Include in an Email to RBAI Support

When emailing RedBrick AI's Support Team, please include as much context as you can so that we can get to work right away! Some things that will help often include:

1. **The Task URL** - the URL of the Task that is failing to load. This URL contains the ID of both the Task itself and the Project that the Task is located in;
2. **The Error** - A screenshot of the error message (or a copy/pasted version of the content of the error message);
3. **Additional Context (for technical users)-** screenshots or logs from the Console or Network tabs of Developer Tools, where applicable;

Continuing Work

If you'd simply prefer to continue work on other Tasks, you have several options.

Raise Issue - if you'd like to inform your Admin of an issue within RedBrick AI, you can click on the **Raise Issue** button. This will remove the Task from your Labeling Queue, preventing it from appearing over and over again while you work. You can

learn more about raising an Issue here. Once you raise an Issue, you will automatically be presented with the next Task in your Labeling Queue;

If you don't want to raise an Issue, you can simply navigate to another Task in your queue by either:

1. clicking on another Task **at the bottom of the error screen**, or
2. backing out of the Annotation Tool and selecting another Task.

Other Situational Troubleshooting Tips

Error Message: Failed to load these URLs

If you're using an external Storage Method and can see URLs listed under the "Failed to load these URLs" message, try opening the link in your browser.

If the browser loads a screen with an error message, there is likely an **issue with your Storage Method integration** (possibly your bucket name or access keys).

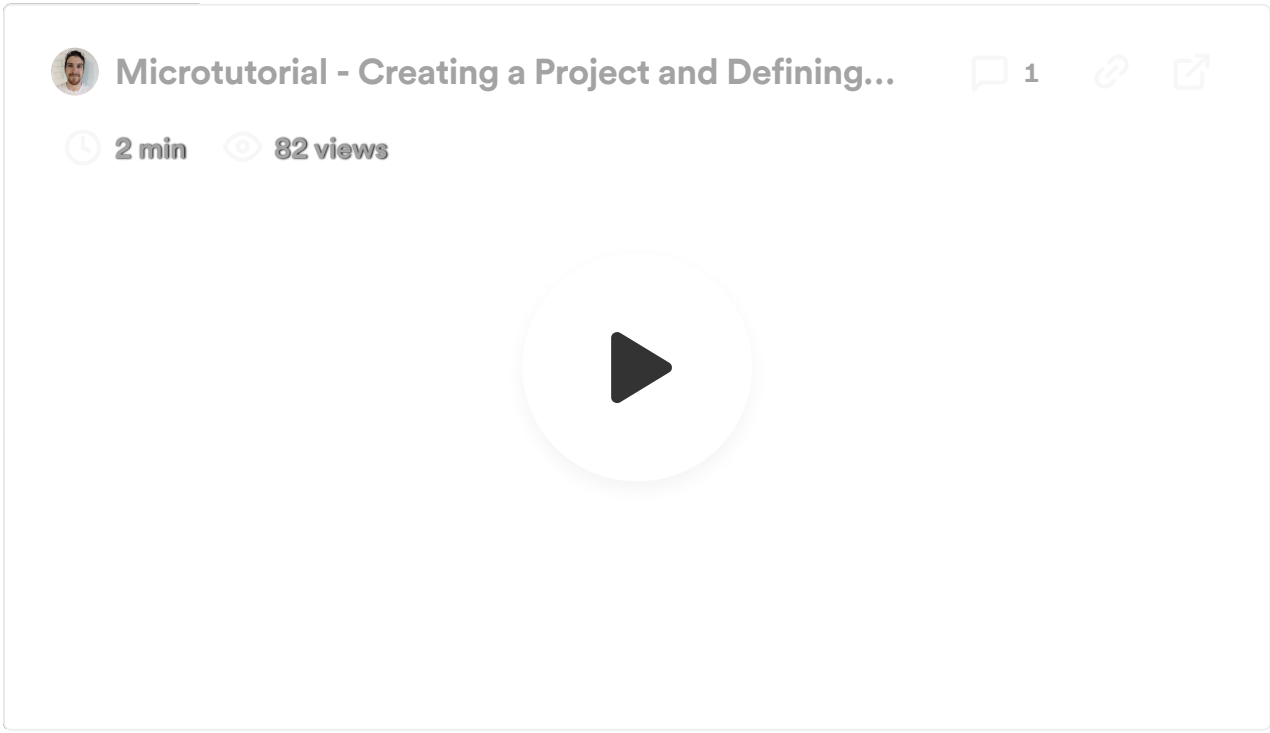
CORS Configuration Error

If you are seeing a data file is being downloaded but not displayed, it's likely that your team **has not not enabled CORS on your storage method**. If CORS is not enabled on your storage method, your browser will not be able to load the image (as it comes from a different origin*).*

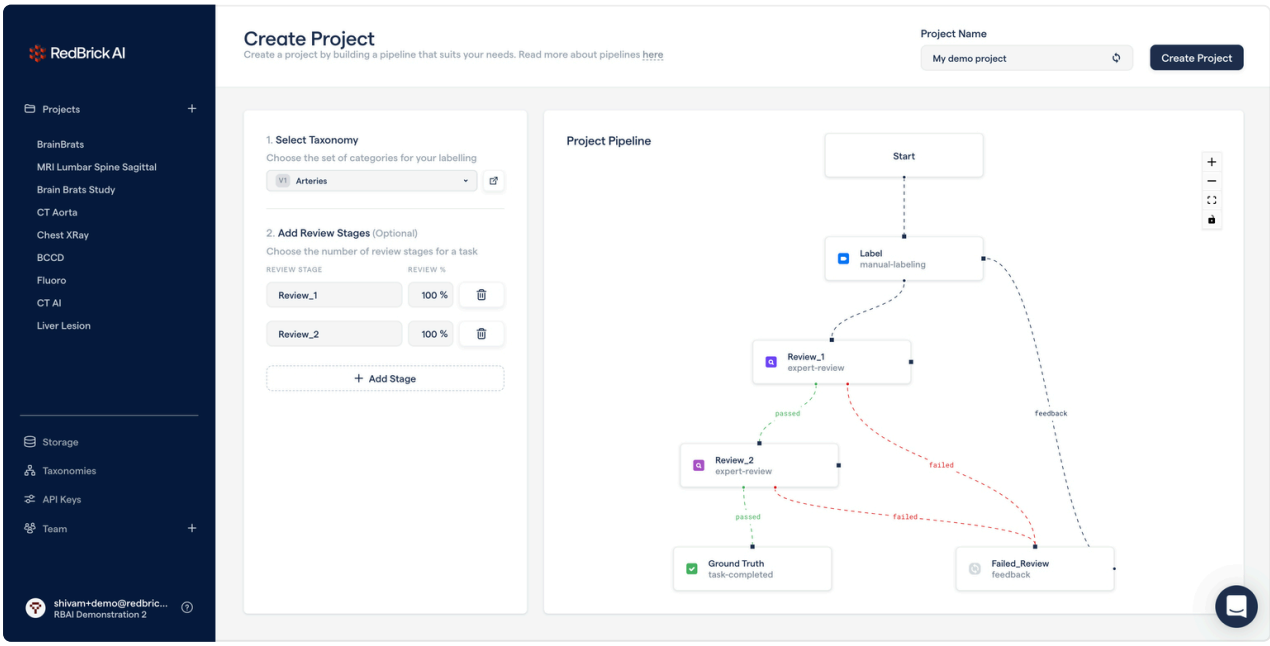
Projects

Get Started with a Project

Please see the following micro-tutorial for a visual walkthrough of how to create a Project in RedBrick AI.



Creating a Project: Step by Step



Create project page

To create a new Project, press the **Create Project** button on the Projects page;

Step 1: Specify a Unique Project Name

Each Project has a unique name that can be changed at any time within Project Settings;

Step 2: Select the Taxonomy for your Project

Taxonomies are stored at the Organization level, which allows you to reuse a single labeling Taxonomy across Projects.

> Selecting a Taxonomy for a Project is a permanent choice that cannot be undone. However, you can edit your Taxonomy after using it in a Project.

Step 3: Define your Project Workflow

Project workflows allow you to explicitly define a series of annotation and review stages in your project.

You can also configure the **number of review stages** and the **review percentage** (i.e. randomly select a subset of data for pseudo-random review) in your workflow.

Once your data has successfully passed through all stages of your workflow, it will reach the **Ground Truth** Stage.

A common workflow included a single Label Stage followed by two Review Stages where different sets of reviewers validate the annotations performed.

> When a reviewer rejects a Task in a Review Stage, it is sent back to the Label Stage and automatically assigned to the same Labeler who annotated it the first time.

> The number of Review Stages cannot be modified after a workflow has been created. You can, however, disable a Review Stage by setting its Review Percentage to 0% in your **Project Settings**.

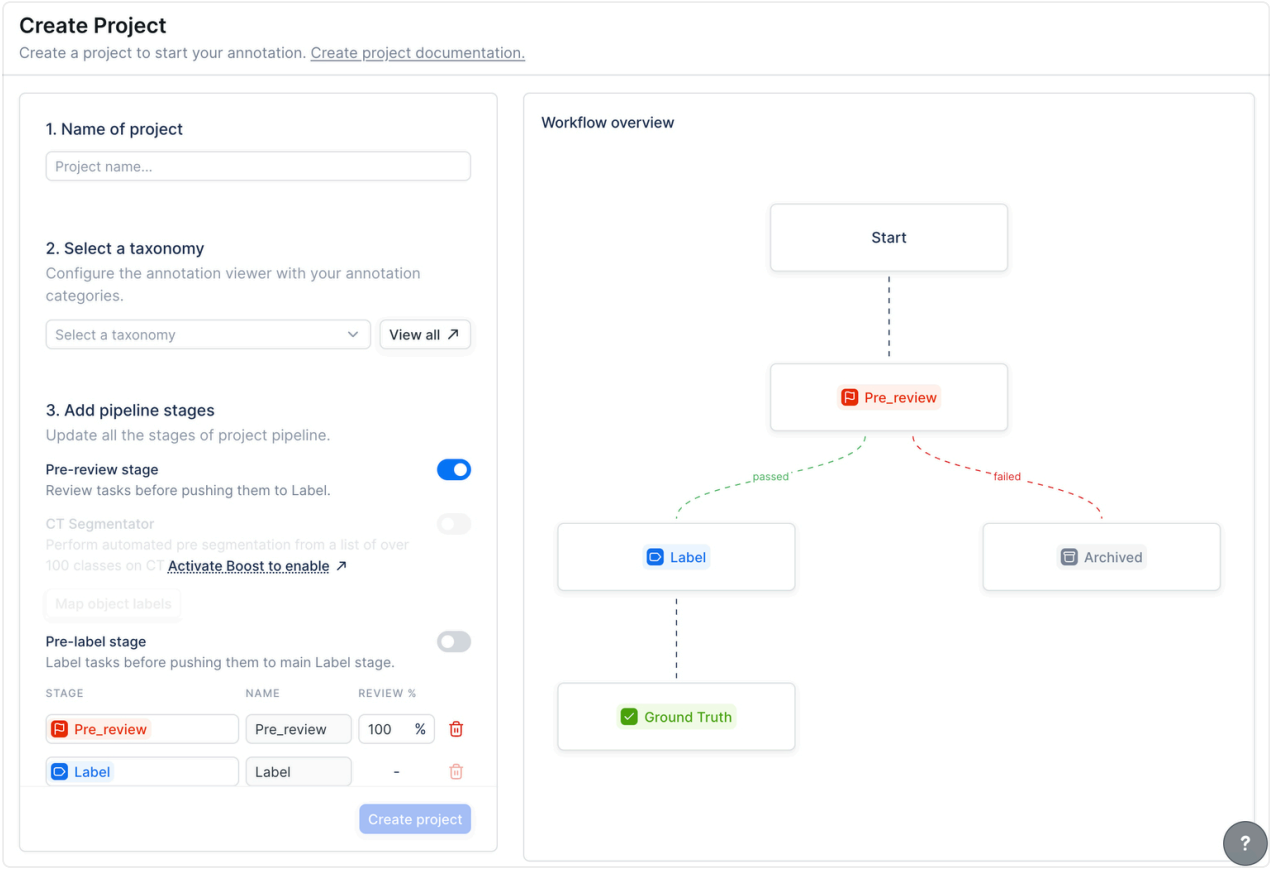
Pre-review and Pre-Label stage

Pre-review stage

The Pre-review stage enables you to examine tasks for quality control or other purposes before they proceed to the labeling stage.

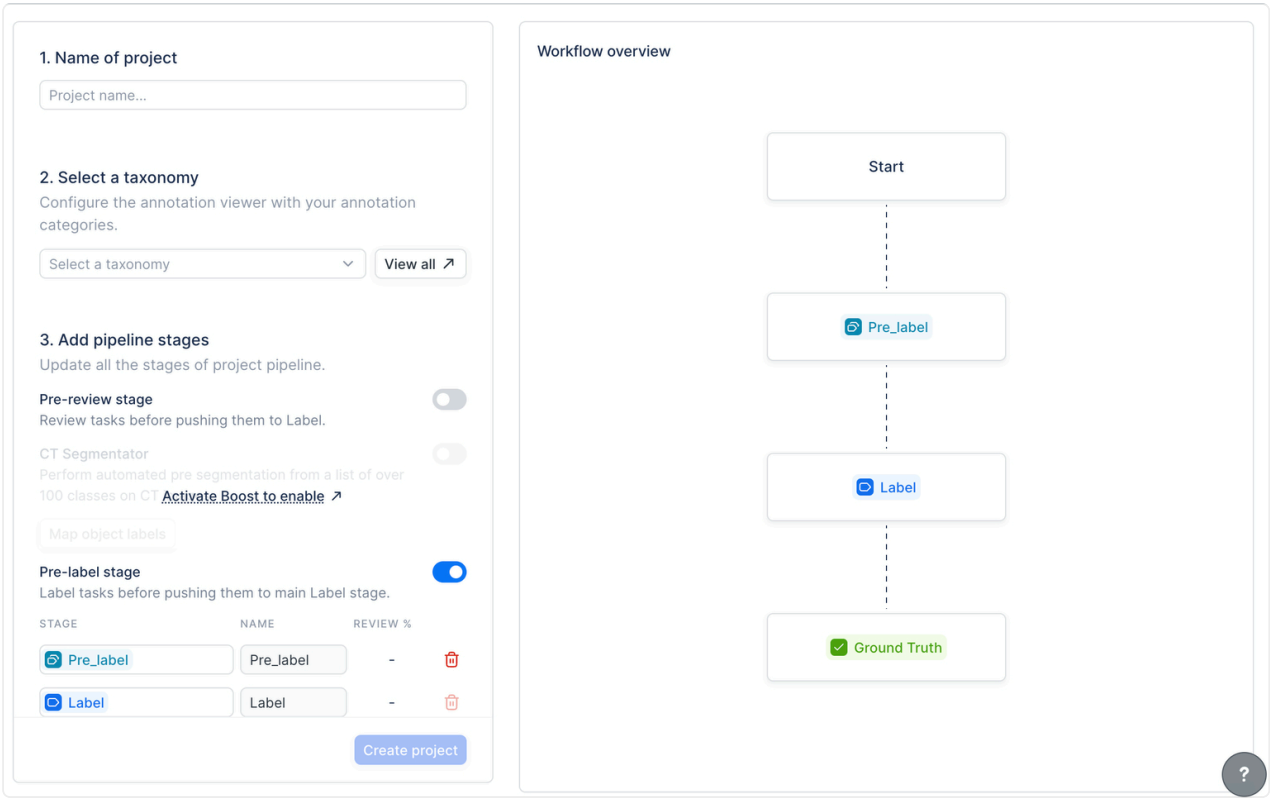
To enable the pre-review stage for your project, follow these steps:

- Begin by creating a new project.
- Turn on the pre-review stage by clicking the toggle button.
- Once activated, the pre-review stage will be added to the workflow.
- After all the stages are added, click the "Create Project" button.
- Navigate to the "Data" tab, and begin reviewing tasks. You can then either accept or reject them as needed.



Pre-Label stage

The pre-label stage enables you to label and classify tasks before they are moved to the labeling stage. This stage simplifies the process, especially for complex projects.



To enable the pre-label stage for your project, follow these steps:

- Begin by creating a new project.
- Turn on the pre-label stage by clicking the toggle button.
- Once activated, the pre-label stage will be added to the workflow.
- After all the stages are added, click the "Create Project" button.
- Navigate to the "Data" tab, and begin labelling the tasks.

Additional Project Setup

Once you have successfully created a Project, you will most likely want to perform the following actions to add collaborators and data and modify Project settings.

Adding Labelers to your Project

By default, all **Org Admins** have comprehensive access to all Projects. As with any elevated role, we recommend taking care when designating fellow team members as

Org Admins.

Generally speaking, it's advisable to add Labelers and Reviewers to your Organization as **Org Members**. This grants them access to your Organization on the RedBrick AI Web Application, but limits their access and permissions.

Org Admins have granular control over the Project-level access of all **Org Members**.

In order to add Labelers or Reviews to individual Projects, we recommend the following:

1. On the Team Page, invite the Labeler/Reviewer as an **Org Member (MEMBER)**.
2. After the Labeler/Reviewer accepts their invitation and creates an account, navigate to the Project you'd like to add them to and open the Workforce Tab. From there, you can invite the Labeler/Reviewer to the Project as either a **Project Admin** or **Project Member**. Please note that if your Labeler/Reviewer is a **Project Admin**, they will have access to all Stages of a Project, as well as the Project Settings.
3. If your Labeler/Reviewer is a **Project Member**, you can regulate their access to various Stages limited based on your workflow's needs or regulatory requirements.

For a more in-depth breakdown of the permissions and access privileges associated with each role, please consult the documentation below.

Organization and Project Roles ›

Uploading Data

You can upload data to a Project either through the UI or CLI/SDK. Most teams tend to integrate external storage and host their own data, however, you can also directly upload data to RedBrick AI servers.

If you'd like to upload annotation files alongside your images and/or volumes, you must use our CLI/SDK.

Importing Data & Annotations

Assigning Tasks

Once you have uploaded data to your Project, RedBrick AI will begin automatically assigning Tasks to the users in your Project. You can learn more about the specifics of task assignment by referencing the documentation below.

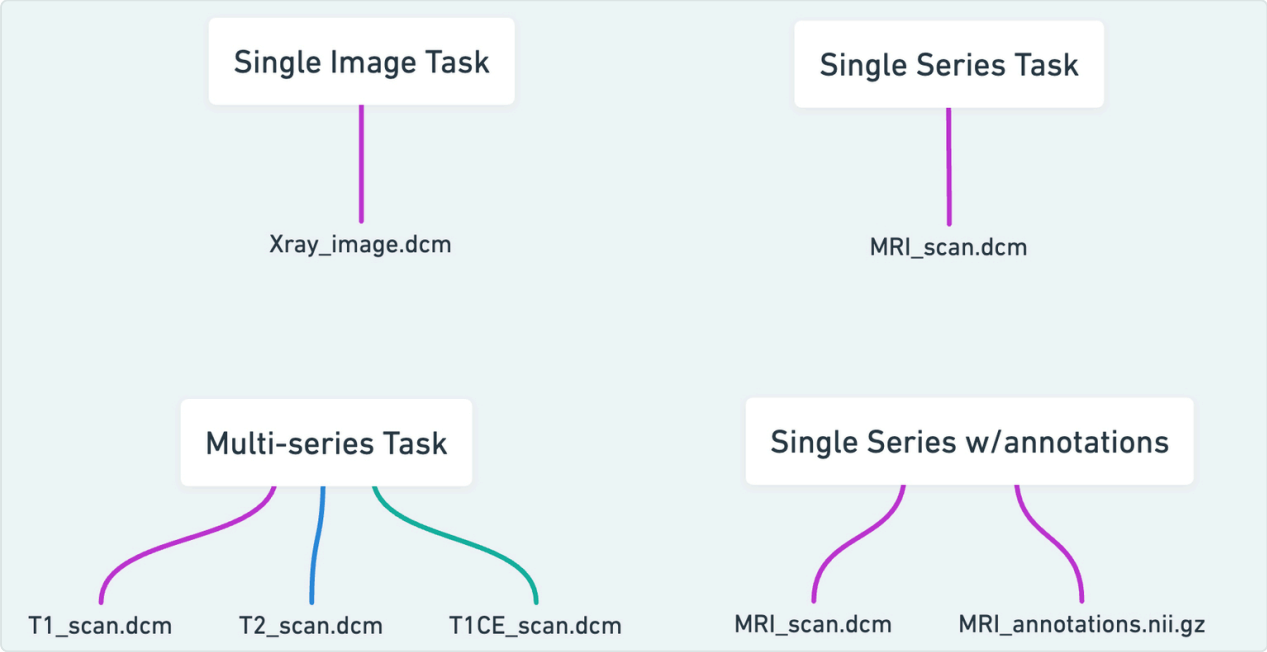
Task Assignment

Task Assignment

What is a Task?

A Task is a unit of work that moves through your project pipeline in RedBrick AI. Tasks can consist of anything - a single image, series, or entire study, and your Labeler works with one Task at a time while annotating.

For example, if you'd like your Labelers to view & annotate an **entire MRI study** comprised of **4 series** together, you should upload the 4 series together as a single Task (see "Multi-series Task" below):



Several examples of valid Tasks

Please see our data import documentation for a more comprehensive overview of how to structure your data imports.

Task Assignment in RedBrick AI

RedBrick AI allows you to delegate work among your team using either automatic or manual task assignment.

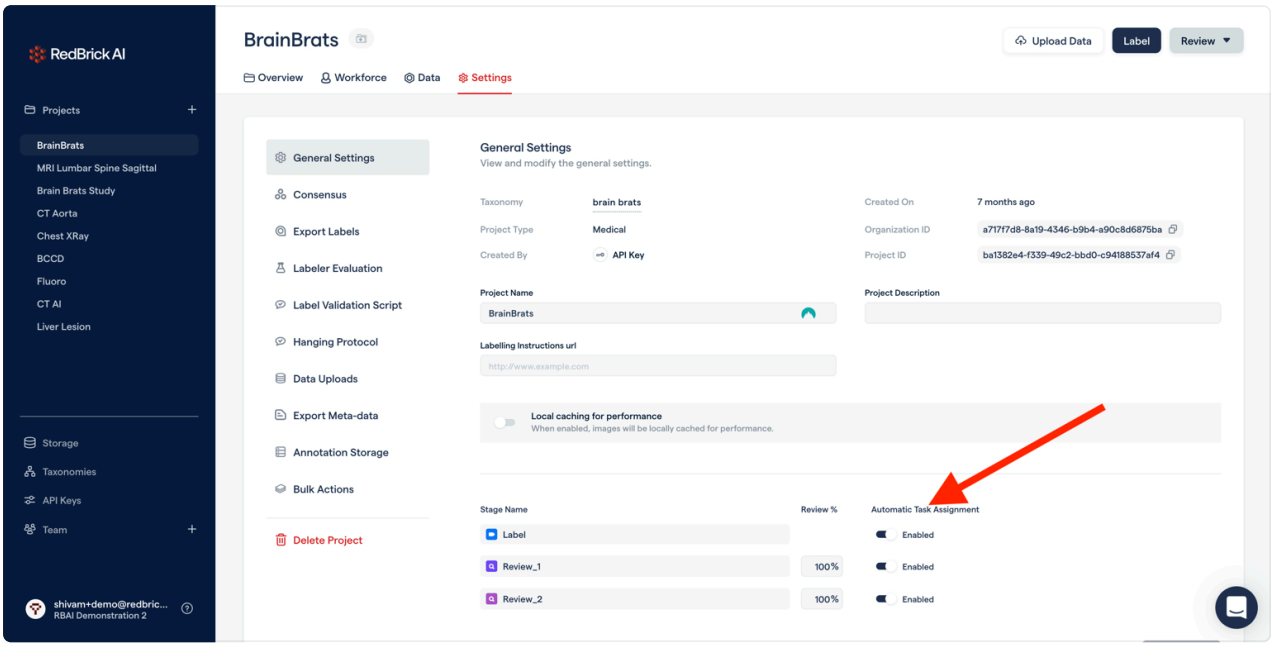
Any Project Admin or Member with relevant permissions can view a Task on RedBrick AI.

Automatic Task Assignment

Automatic Task Assignment is enabled by default upon Project creation. The automatic assignment protocol is a first-come-first-serve system, i.e. it assigns the oldest Tasks to the first annotators that request new Tasks.

Labelers can request new Tasks by clicking on the "Label/Review" buttons on the Project Dashboard.

> You can disable Automatic Task Assignment in **Project Settings -> General Settings.**



Manual Assignment

Admins can also override the automatic assignment protocol and manually assign Tasks to users from the **Data Page**.

> RedBrick AI will not automatically re-assign Tasks that have been manually assigned.

Programmatically Assigning Tasks

You can programmatically assign tasks by prescribing the assignment during data upload as part of your Items List or using the `assign_tasks()` method of our SDK.

Assigning Tasks on Upload

You can use the `preAssign` field in the to assign a Task you are uploading to a specific user(s) at each Stage.

For example, the snippet below will assign `study_001` to `annotator@email.com` in the Label Stage. Once the annotation is complete, the Task will be queued in `Review_1` and `reviewer@email.com` will be assigned as the Reviewer.

```
[
    {
        "name": "study_001",
        "preAssign": {
            "Label": "annotator@email.com",
            "Review_1": "reviewer@email.com"
        },
        "series": [
            {
                "items": "ImageFile.extension",
            }
        ]
    }
]
```

> Always double check that your Stage Names (i.e., Label, Review_1, etc.) and user emails have been input correctly.
>
> Also, when preassigning Tasks, all emails must be associated with an existing Project Member.

Assigning Tasks after Upload

You can use the assign_tasks() method to designate task assignment using the SDK. Please see our SDK Documentation for further details.

Labeling Queue

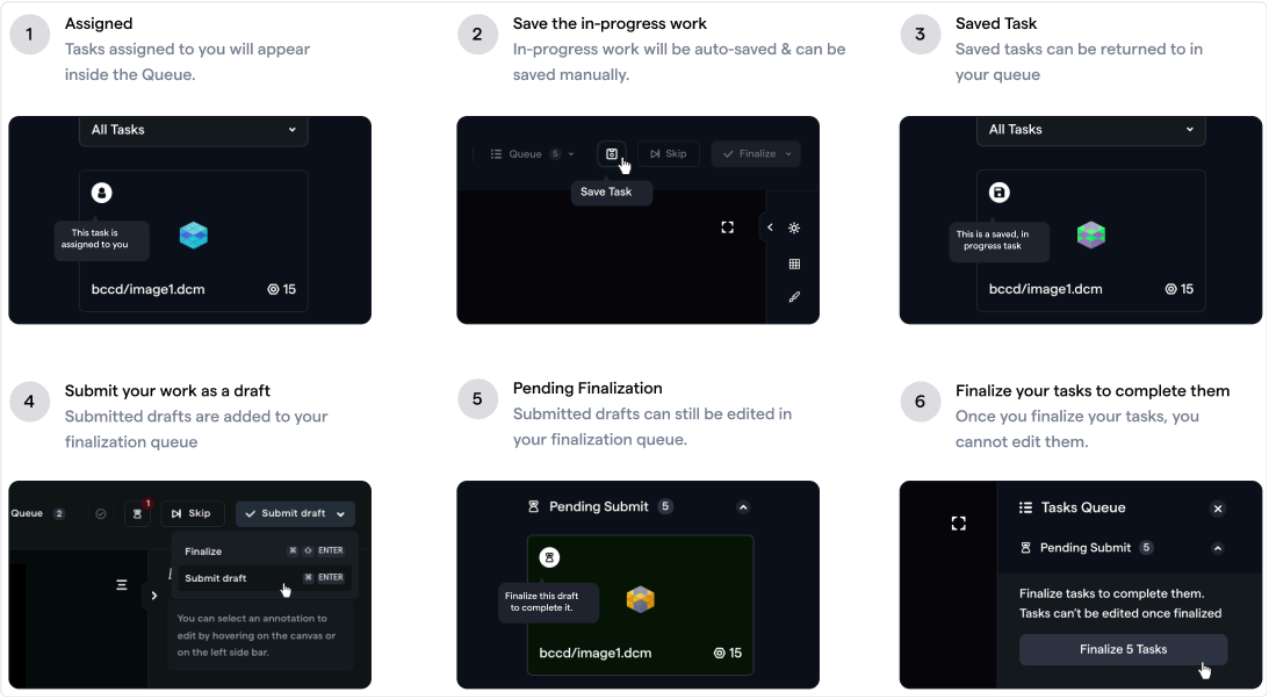
Once a Task is assigned to a user, it is added to their Labeling Queue. You can view your labeling queue in two ways.

1. **From the Data Page:**
   On the Data Page, you can filter existing Tasks by **Queued for Labeling/Review** and then by Tasks assigned to you.
2. **In the Annotation Tool:**
   The Labeling Queue can be expanded/retracted by clicking on the corresponding button in the top bar of the Annotation Tool.

While in your Queue, a Task can be in a few different states depending on the status of the annotation:

1. **Assigned**
   Tasks that you have not worked on yet will be displayed as Assigned.
2. **Saved**
   Once you save your in-progress annotation (either manually or through auto-save), the Task will show as saved.
3. **Pending Finalization**
   Once you are done with the annotation, you can **Submit a Draft**. All drafts that have been submitted will **still be in your Labeling Queue** pending finalization. You must finalize the draft to complete it and send it to the next stage of the workflow.
4. **Skipped**
   If you encounter a Task that you would like to complete at a later time, you can skip it to send it to the end of your Labeling Queue.

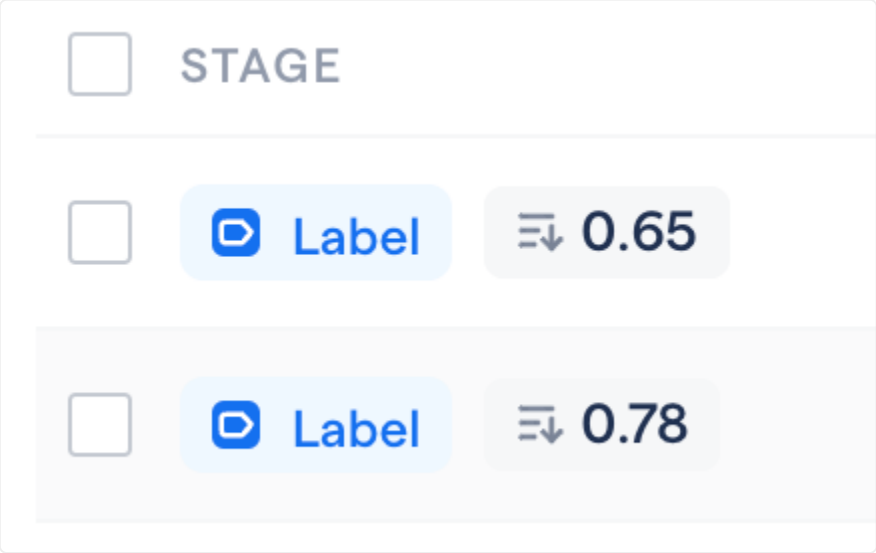
The diagram below is a visual guide to the flows associated with completing Tasks in your Labeling Queue, including associated actions and Task states.



Guide to submitting Tasks in your Labeling Queue

Task Prioritization

RedBrick AI allows you to designate specific Tasks as **prioritized**, which elevates them to the top of your Labeling Queue.



Two Tasks with priority scores

Task Priority is reflected in the Web Application in the following ways:

1. Task Priority is visible in the Data Page when sorting by **Queued for Labeling/Review** or **Recently Labeled/Reviewed** - this logic applies to all Stages except for Ground Truth.
2. Task Priority will persist throughout Raising an Issue and/or Rejecting a Task at any Stage.
3. Task Priority will be visible in the Annotation Tool when viewing the Queue
4. Tasks that are Assigned and Prioritized will occupy a higher position in the queue than Tasks that are Unassigned and Prioritized.

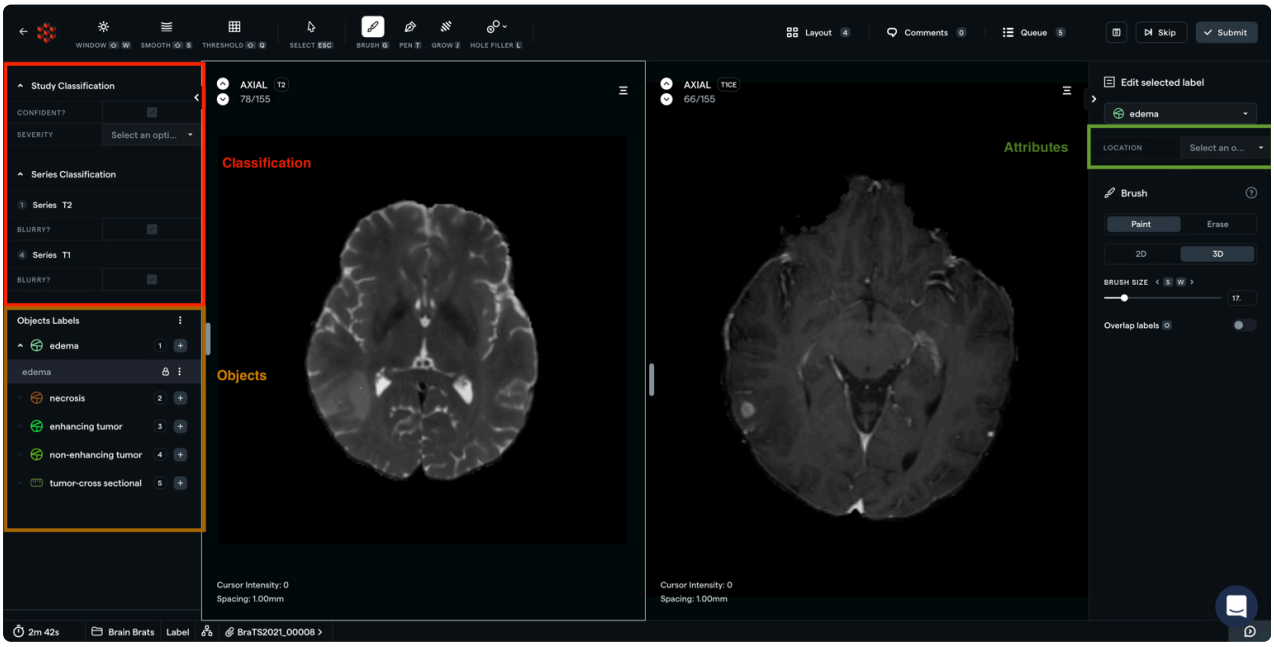
As seen in the snippet below, you can use the `update_tasks_priority()` method to designate a float between 0 and 1 that reflects the priority of a given Task (where 1 is the highest priority and 0 is the lowest).

```
tasks =
    [
        {
            # High Priority Task
            "taskId": "2716057",
            "priority": 0.95
        },
        {
            # Mid Priority Task
            "taskId": "BU221729",
            "priority": 0.50
        },
        {
            # Low Priority Task
            "taskId": "8675309",
            "priority": 0.32
        }
]

project.labeling.update_tasks_priority(
    stage_name="Label",
    tasks=tasks
)
```

For the truly brave, our Prioritization API supports up to the billionth place for floats.

Taxonomies

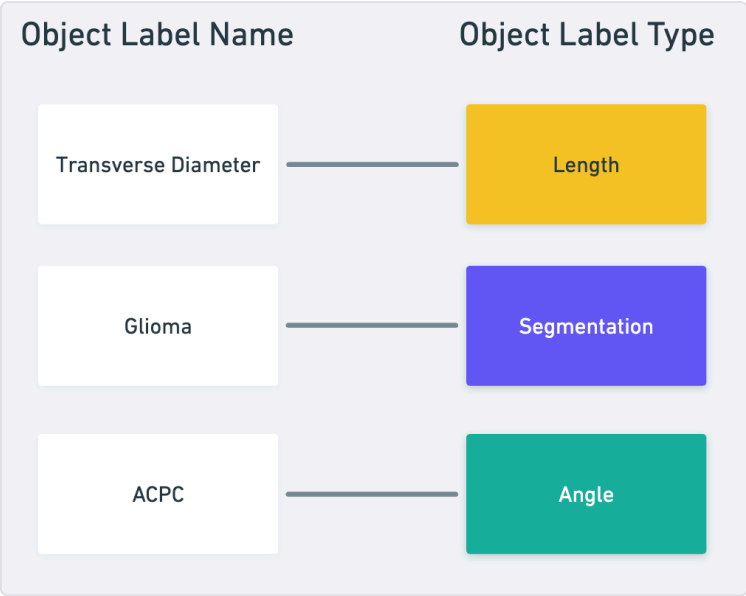
Taxonomies allow you to define the structures that you'd like to annotate in your project and apply them to your data quickly and accurately. They ensure all annotations follow a structured schema which is automatically imported to the left hand sidebar of RedBrick AI's Annotation Tool.



Object Label Types

Object Labels are the structures that your team will annotate on RedBrick AI.

When creating your Taxonomy, you must define a Label **Type** (e.g. "Segmentation") and a **Name** (e.g. "Edema") for each Object Label.



RedBrick AI supports the following Object Label Types:

Object Label Type	2D Image	3D Image	2D Video
Segmentation	✅	✅	
Landmarks	✅	✅	✅
Angle Measurement	✅	✅	
Length Measurement	✅	✅	
Bounding Box	✅	✅	✅
Ellipse	✅	✅	
Polygon	✅		✅
Polyline	✅		✅
Cuboid		✅	

Object Label Attributes

Attributes allow you to add a deeper level of classification to your Object Labels. Attributes are commonly used to collect more information about a particular object

(e.g. "True/False" for an Object titled "tumor malignancy"). RedBrick AI offers the following Attribute Types:

Attribute Type	Description
Boolean	A checkbox that can be either True or False
Select	A dropdown that can be a single value from a list of predefined values
Multi-select	A dropdown that can have multiple values from a list of predefined values
Textfield	A text input that can record free form text

Classifications

Classifications are data attributes that can be affixed to studies, individual Series, or individual video frames.

Just like Object Label Attributes, Classifications can be **Booleans**, **Single Selects**, **Multi-selects**, or **Text** fields, and there is no limit to the number of Classifications you can have in your Taxonomy.

Study-Level Classifications are a classification for an entire Task (e.g. an MRI study consisting of 4 Series).

Series-Level Classifications are applied to a single series (e.g. the T1 sequence from an MRI study).

Instance-Level Classifications are applied to a single frame of a video and are only available for 2D video formats.

Classification Type	2D Image	3D Image	2D Video
Study	✅	✅	✅
Series	✅	✅	✅

Classification Type	2D Image	3D Image	2D Video
Instance			✅

Creating Taxonomies

Taxonomies are created and stored at the Organization level, which allows you to use a single Taxonomy for several Projects.

To create a new Taxonomy in the UI, navigate to the **Taxonomies** page in the left hand side bar of the RedBrick web app and click on **Create Taxonomy**.

Taxonomies can also be created using the `create_taxonomy()` SDK method.

> ✓ All Taxonomies must contain at least one Object Label or Classification in order to be successfully created.

Modifying Taxonomies

Taxonomies can be modified on the **Taxonomies** page of the UI at any time.

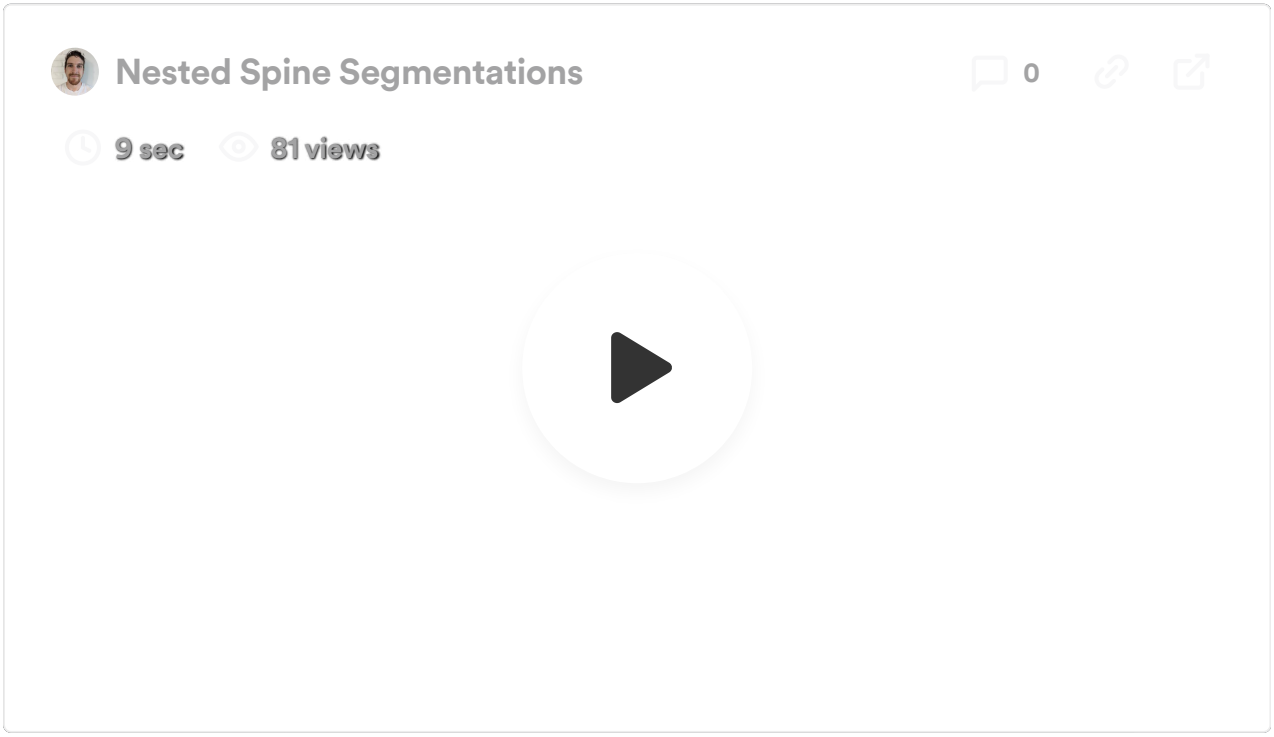
Alternatively, you can use the `update_taxonomy()` SDK method to modify a Taxonomy outside of the UI. Please note the following about modifying existing Taxonomies:

1. the `update_taxonomy()` method overwrites the current Taxonomy in its entirety;
2. If you delete an Object Category, Attribute, or Classification from your Taxonomy, all existing associated annotations will need to be updated;
3. Taxonomies that are being used in Projects cannot be deleted;

Nesting Taxonomy Elements

Taxonomies now support the nesting of Object Labels, Study-Level and Series-Level Classifications both in the UI and via the RedBrick AI Python SDK.

By adding the parents attribute to your Taxonomy, you can create and/or designate Parent Tiers for a given Object Label.



Nested Object Labels in the Annotation Tool

Nesting Object Labels in the UI

In the Taxonomies page, simply click on **Add Folder** and give your folder a name.

You can then easily drag and drop your Objects Labels or Classifications into your folder.

Nesting Object Labels via SDK

You can create Parent Tiers by adding the `parents:[]` attribute to any Object Label, Study-Level, Series-Level, or Instance-Level Classification within your Taxonomy. Parent Tiers are created and assigned from **left to right and in descending order**, which means the first string in `parents:[]` will always be a Tier 1 Parent, the second string will be a Tier 2 Parent, and so on.

For full documentation, please see our Taxonomy Object reference.

For an example of a two-tiered Object Label structure, please see the example code below:

```
org.create_taxonomy_new(
    "Clinical Study 1",

    object_types=
    [
        {
            "category": "Herniated Disc",
            "labelType": "SEGMENTATION",
            "attributes": [],
            "color": "#7FFFD4",
            "classId": 0,
            # Creates the Tier 1 Parent 'Spine Pathologies' and Tier 2 Pa
            "parents": ['Spine Pathologies', 'Disc Pathologies'],
        },
        {
            "category": "Bulging Disc",
            "labelType": "SEGMENTATION",
            "attributes": [],
            "color": "#DEB887",
            "classId": 1,
            "parents": ['Spine Pathologies', 'Disc Pathologies'],
        },
        {
            "category": "Degenerated Disc",
            "labelType": "SEGMENTATION",
            "attributes": [],
            "color": "#00FFFF",
            "classId": 2,
            "parents": ['Spine Pathologies', 'Disc Pathologies'],
        },
        {
            "category": "Vertebral Fracture",
            "labelType": "SEGMENTATION",
            "attributes": [],
            "color": "#FF7F50",
            "classId": 3,
            "parents": ['Spine Pathologies'],
        },
    ],
)
```

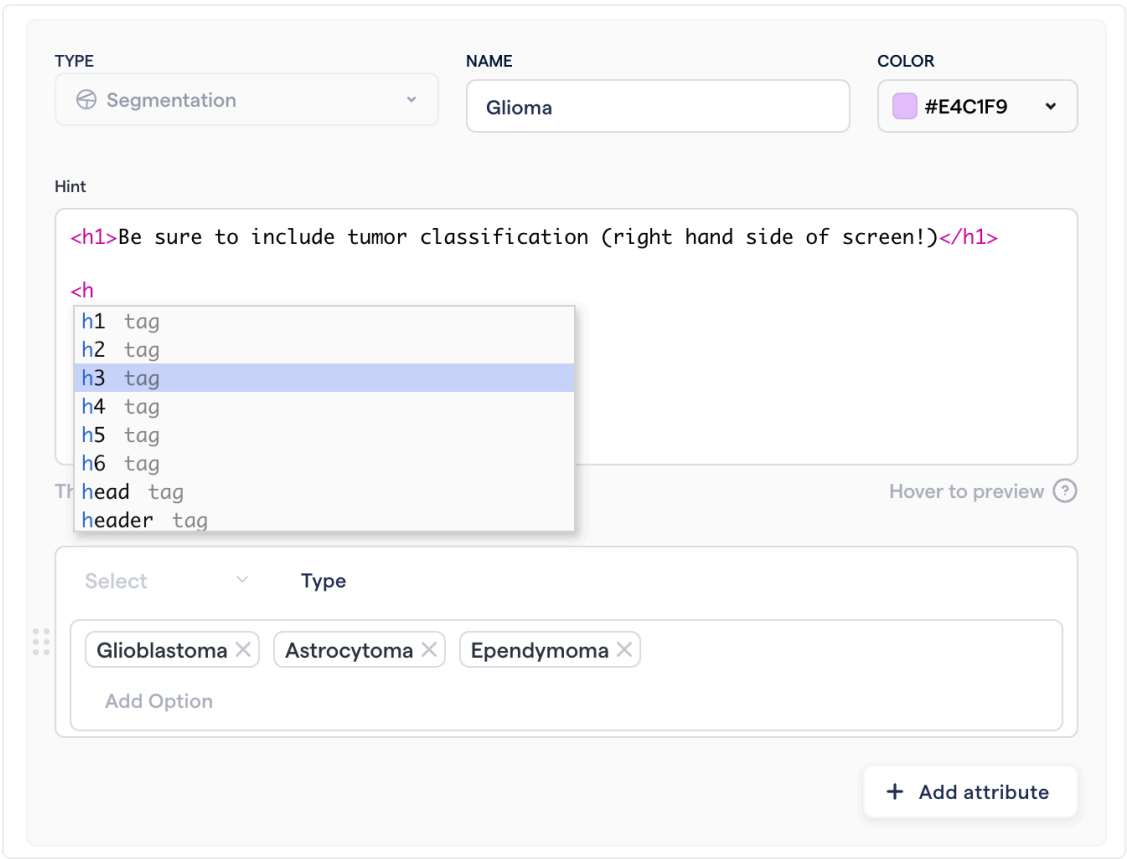
The above Taxonomy will be nested in the Annotation Tool as well. Tiers can also be collapsed or expanded as necessary, allowing you to easily navigate through Label Tiers and save screen space.

HTML Tooltips

RedBrick AI allows users to attach custom HTML tooltips to any Object Label, Study-Level Classification, Series-Level, or Instance-Level Classification. For larger, more complex Taxonomies, these tooltips can be a great form of input for annotators or serve as a record for any internal standards that may be associated with the annotation itself.

Creating HTML Tooltips in the UI

First, open a Taxonomy and click on any Object Label or Classification. The Hint field will then appear, allowing you to copy/paste your HTML into RedBrick or write your own using our intelligent autocomplete feature.



Creating an HTML tooltip in the UI

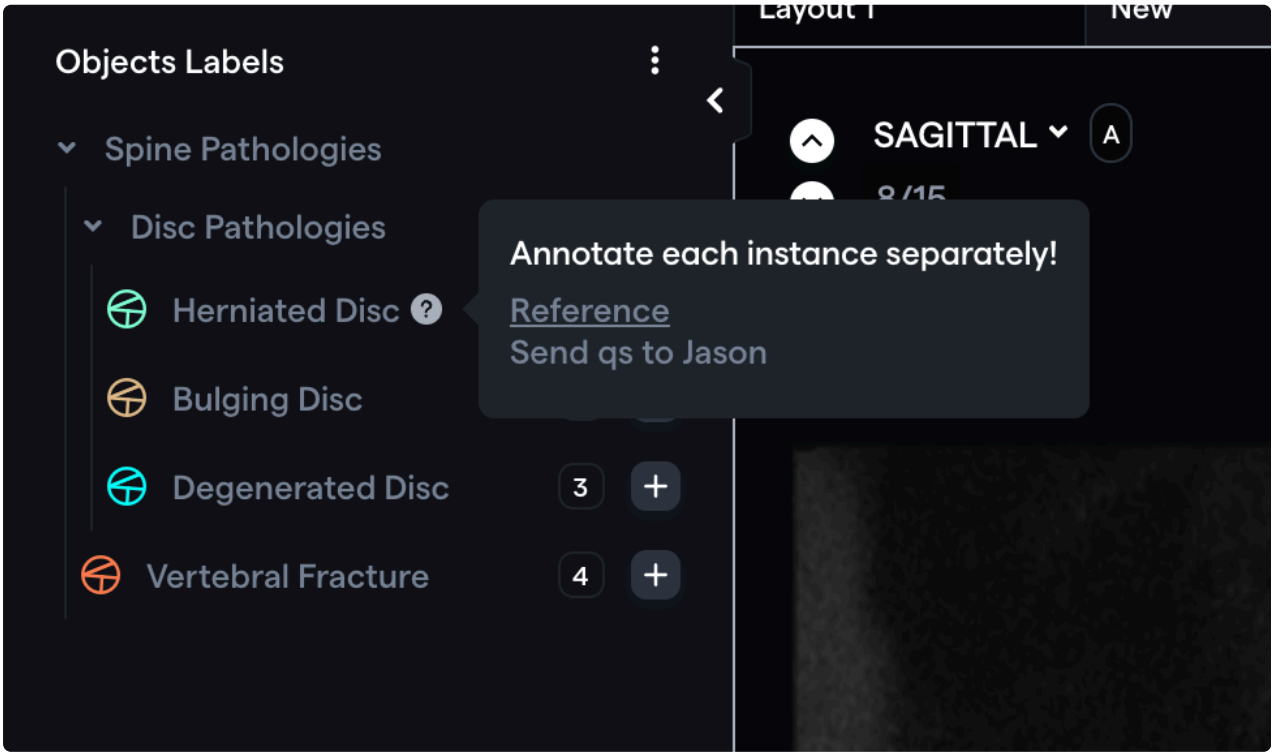
Creating HTML Tooltips via SDK

If you're prefer to create HTML Tooltips with the SDK, you can use the `hint: string` attribute to insert a string of HTML that will display in a tooltip next to an annotation upon hover. Please see the following code for an example of an Object Label with an HTML tooltip.

```
object_types=
    [
        {
           "category": "Herniated Disc",
           "labelType": "SEGMENTATION",
           "attributes": [],
           "color": "#7FFFD4",
           "classId": 0,
           "parents": ['Spine Pathologies', 'Disc Pathologies'],
           "hint": '<h2>Annotate each instance separately!</h2><a href=ht
         },
]
```

> All HTML elements can be included within the `hint: string` attribute, but images must be inserted using `<img src>` and a relevant link.

The above code displays as follows in the Annotation Tool:



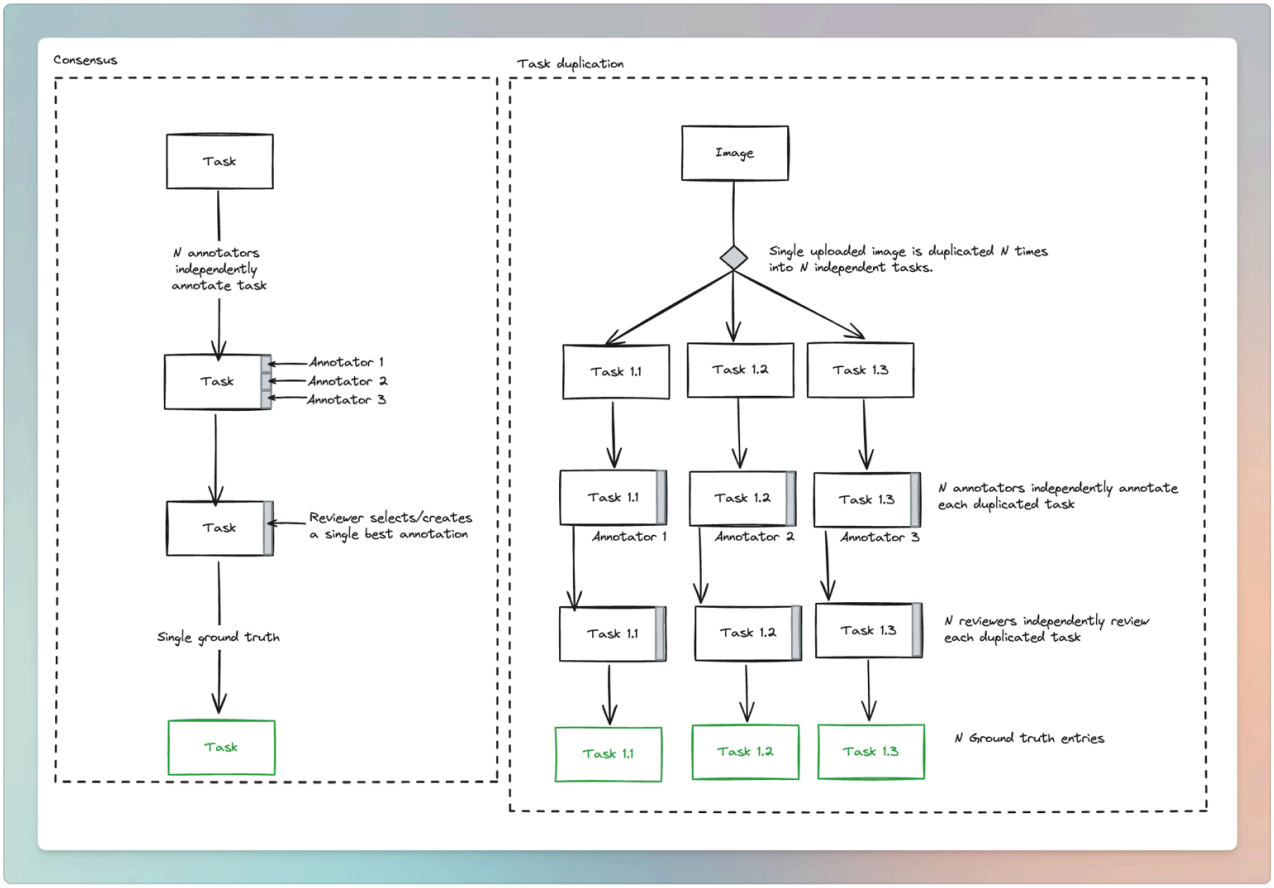
The HTML Tooltip that appears while hovering your cursor over the "?" icon

For security reasons, we do not allow scripts to be executed within HTML Tooltips.

Multiple labeling

RedBrick AI has comprehensive features to help you record multiple annotations per image. There are two core use cases for multiple labeling:

1. **Consensus:** Have multiple labelers annotate a single task and record their *inter-annotator agreement scores,* i.e., measure the overlap between their annotations. The output of consensus is a single set of ground truth.
2. **Task duplication:** Have multiple annotators annotate a single image and generate N unique ground truth records for a single image.



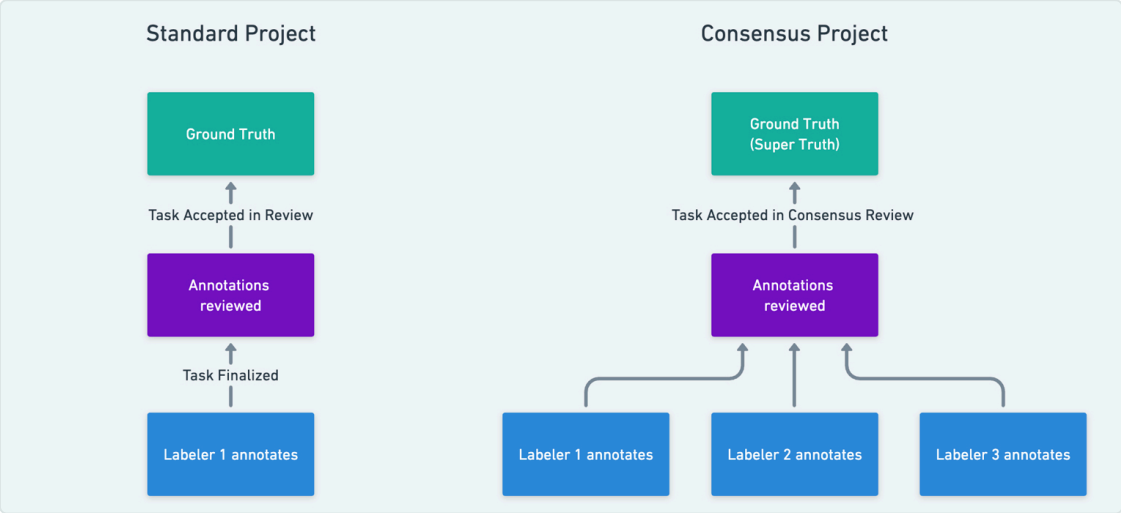
Task duplication vs. consensus

Consensus

Consensus provides you with both a quantitative measure of annotation quality (by means of an Inter-Annotator Agreement Score) and the opportunity to create higher-quality annotations by combining the opinions of multiple annotators.

How Does Consensus Work?

With Consensus enabled, **multiple annotators can be required to label each Task** in the Label Stage. Each individual annotator will only see an empty Task and will not be able to view the annotations done by the other annotators.



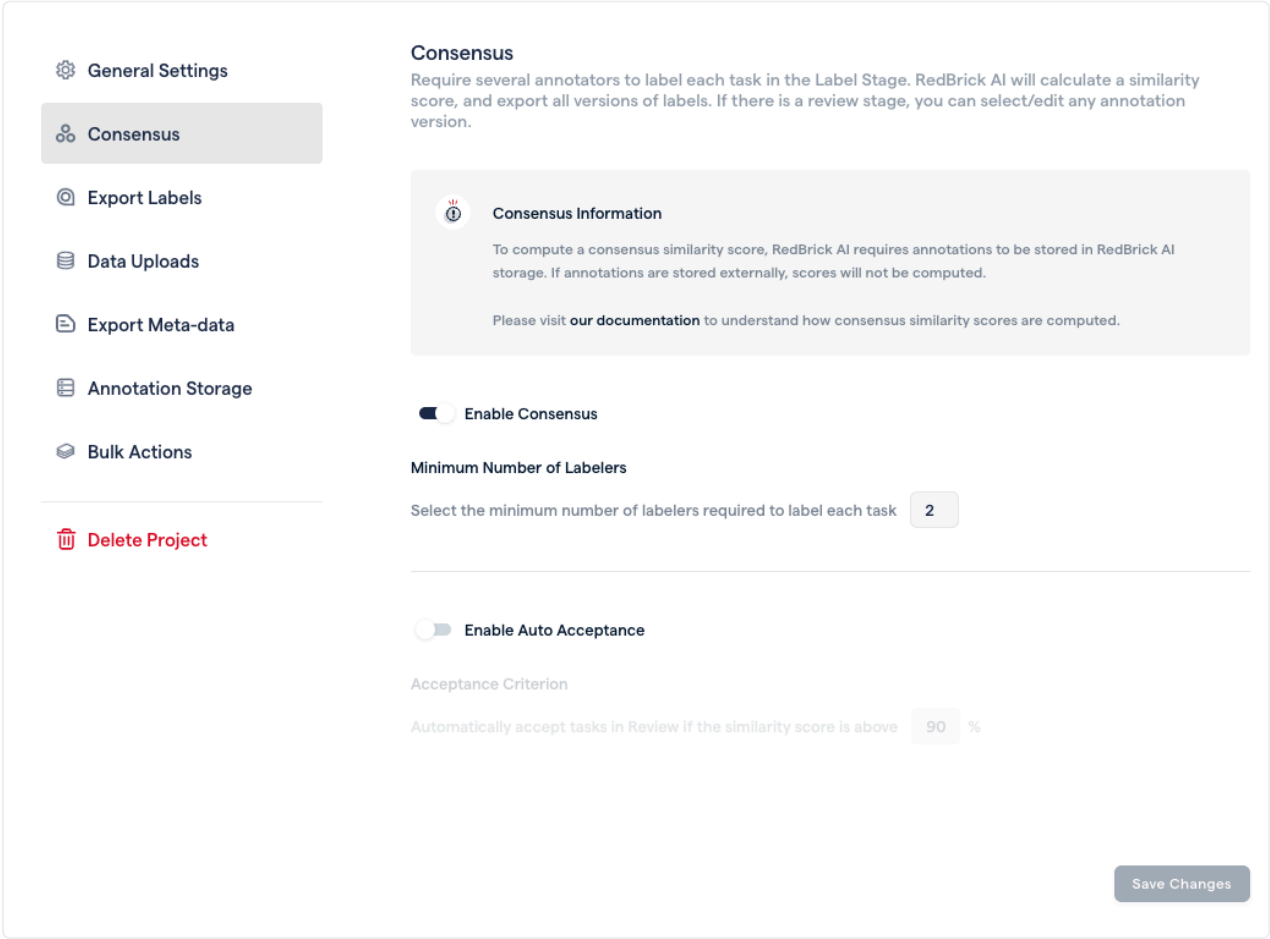
Comparison between standard flow and Consensus flow with 3 labelers

Once all the annotators have completed the Task, RedBrick AI will calculate an **Inter-Annotator Agreement Score** between the annotations. Please reference the following documentation for more information on how we calculate these scores.

The Inter-Annotator Agreement Score is a quantitative measure of quality that can help you select the best set of annotations created by your annotators. It also gives reviewers the ability to arbitrate between the opinions of multiple annotators before generating a single, high-quality Ground Truth.

Enabling Consensus

You can enable Consensus by navigating to Project Settings. Once enabled, you will be required to select a **minimum number of labelers** that will be required to annotate each Task. If your project has a Review Stage, you can **enable auto-acceptance** to automatically accept Tasks whose agreement scores are higher than the specified threshold.

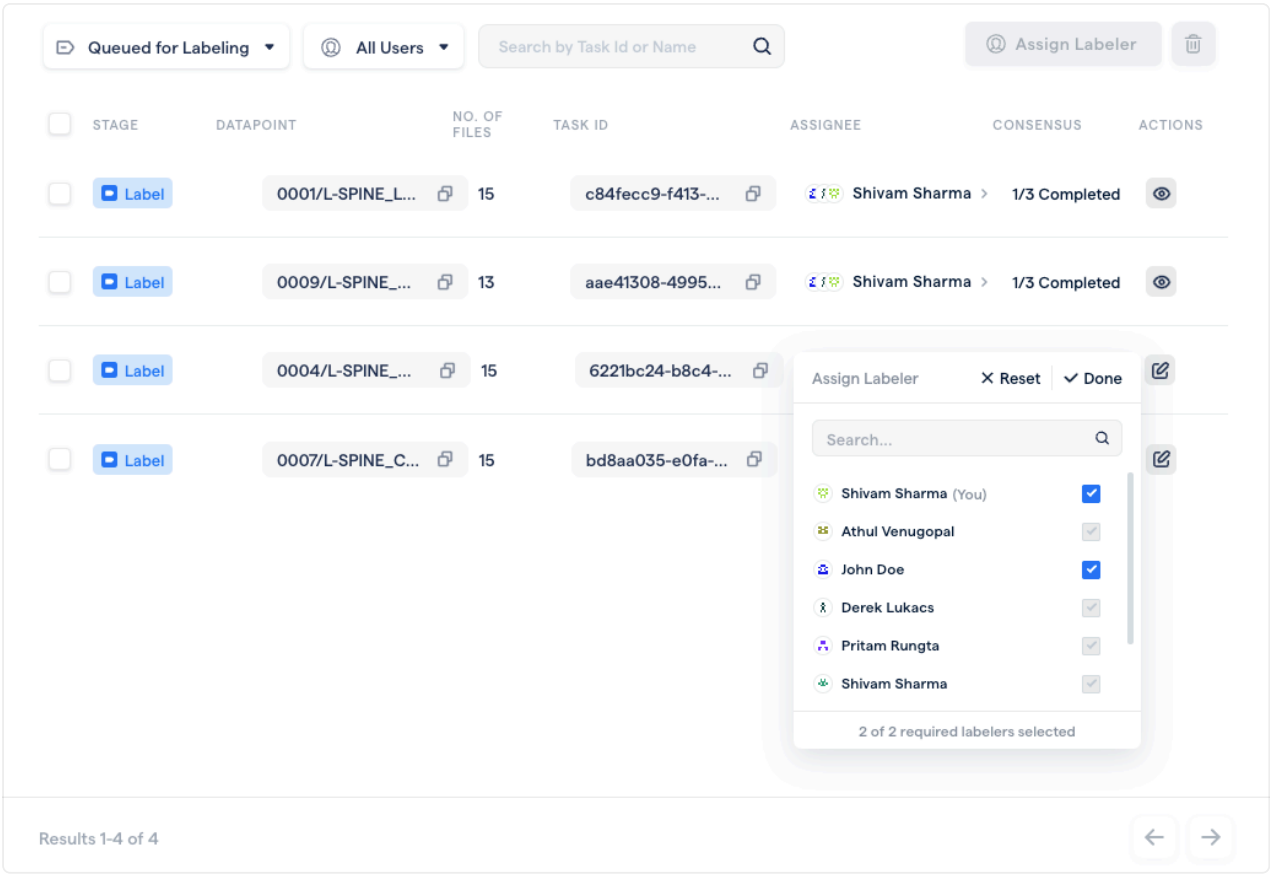


Consensus Project Settings

Assigning Tasks to Multiple Users

RedBrick AI has an **automatic assignment protocol** that will automatically assign multiple users to a Task. As annotators request Tasks by clicking on the **Label** button in the top right of the Dashboard, RedBrick AI will automatically assign available Tasks by prioritizing those that are already in progress/or assigned to other users.

Alternatively, you can **manually override any Task on the Data page**. When Consensus is enabled, the **Assign** dropdown will allow you to select multiple users.



Manual Multi-Assignment

> You can *manually* assign more than the required number of labelers. The automatic assignment protocol will only assign up to the number of required labelers, but you can manually assign as many as you'd like.

Inter-Annotator Agreement

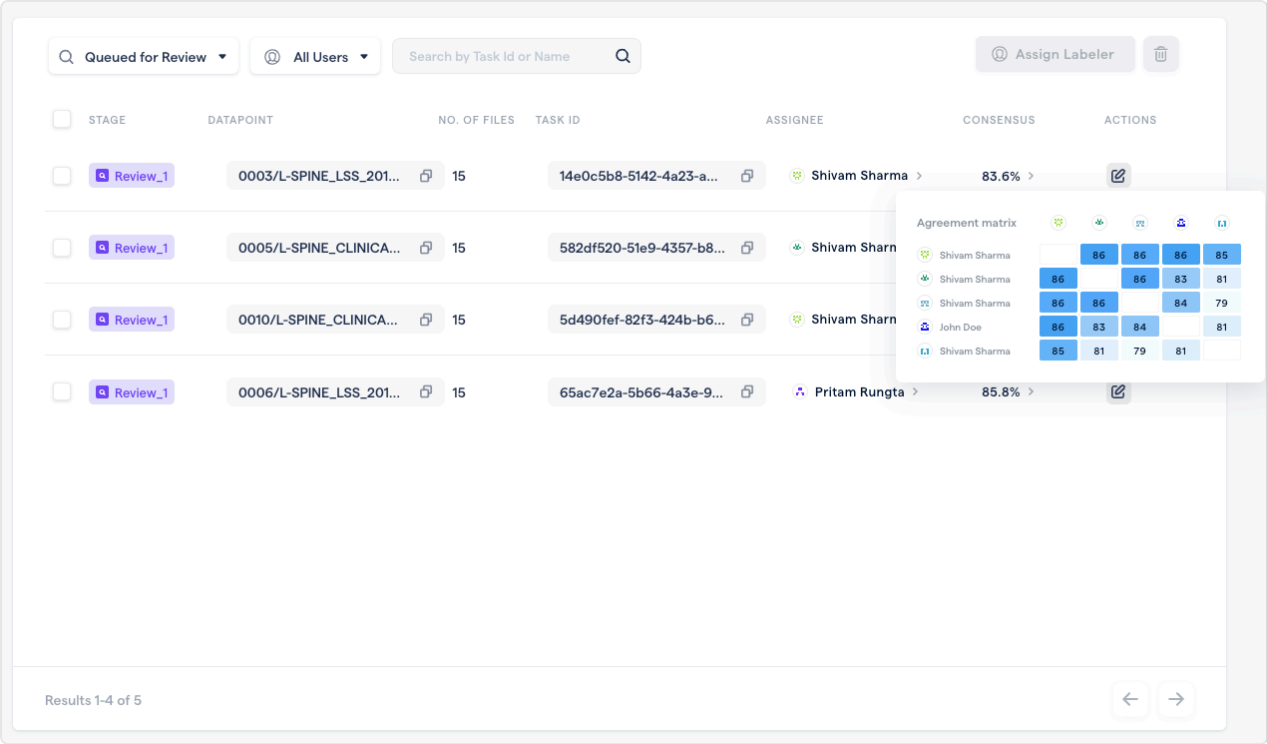
Once all assigned annotators have completed a Task, RedBrick AI will generate an Inter-Annotator Agreement Score, which is calculated by comparing each labeler's annotations with those of every other labeler and averaging the pairs of scores.

	User 1	User 2	User 3
User 1		`Score(U1,U2)`	`Score(U1,U3)`
User 2	`Score(U2,U1)`		`Score(U2,U3)`
User 3	`Score(U3,U1)`	`Score(U3,U2)`	

`Agreement = Average( Score(U1,U2) , Score(U1,U3) , Score(U2,U3) ).`

The type of comparison function used to calculate the `Score` depends on the type of data and annotations you and your team are working with. Please reference the following documentation to read more about how RedBrick calculates Inter-Annotator Agreement.

Agreement calculation



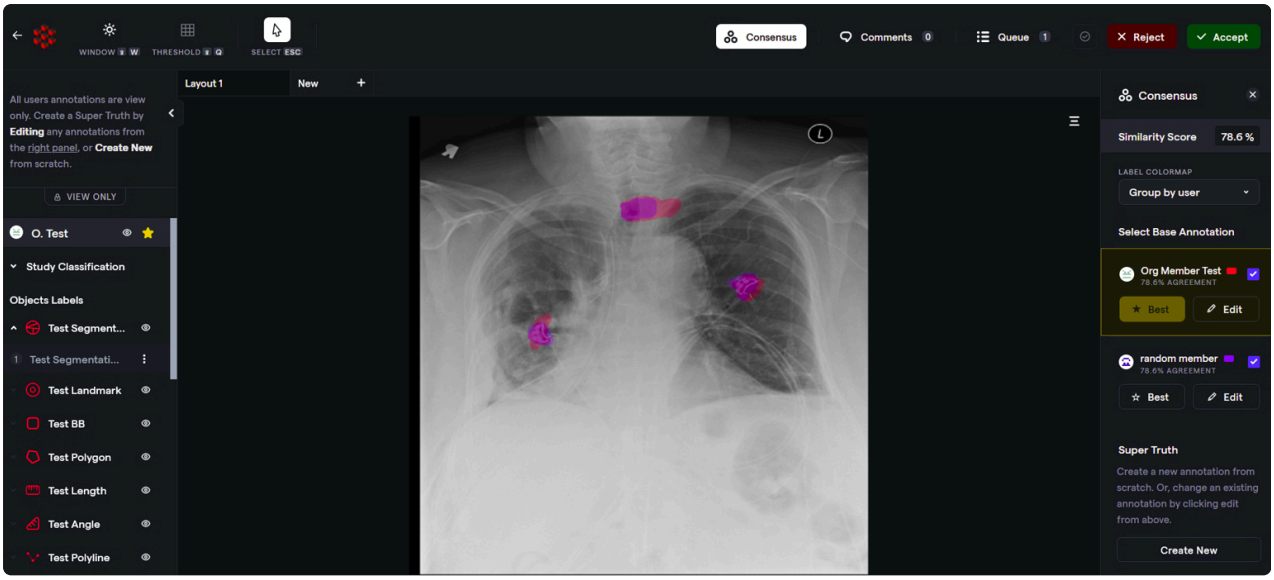
Inter-Annotator Agreement for Tasks queued in Review

Review Stage Absent

If there is no Review Stage after the Label Stage, the set of annotations with the highest Agreement Score (with respect to other annotations) will be selected and stored in Ground Truth. This is the set of annotations that will be exported by default, but you can also export all versions of the annotations.

Review Stage Present

When a Review Stage is present, all annotations will be displayed in the Editor. The list of all users that have annotated the Task is located on the right hand Consensus Panel. By default, annotations are color-coded **by user**, but they can be grouped **by category**.



A Task in Review with Consensus Enabled

Best Annotations and Super Truth

The reviewer's primary task is to analyze the multiple sets of annotations generated by the labelers and produce a single set that will be saved and pushed to the next

Stage. In RedBrick AI's Editor, this single set of annotations is referred to as the **Best Annotations**.

> By default, RedBrick AI selects the set of annotations with the highest Inter-Annotator Score as the Best Annotations.

Reviewers can view, hide/show, etc. all sets of annotations in a Task. This allows the reviewer to analyze the work done by the labelers and select the set of annotations that they consider to be of the highest quality.

If a reviewer is satisfied with an existing annotation set, they can simply designate it as Best Annotations and accept the Task.

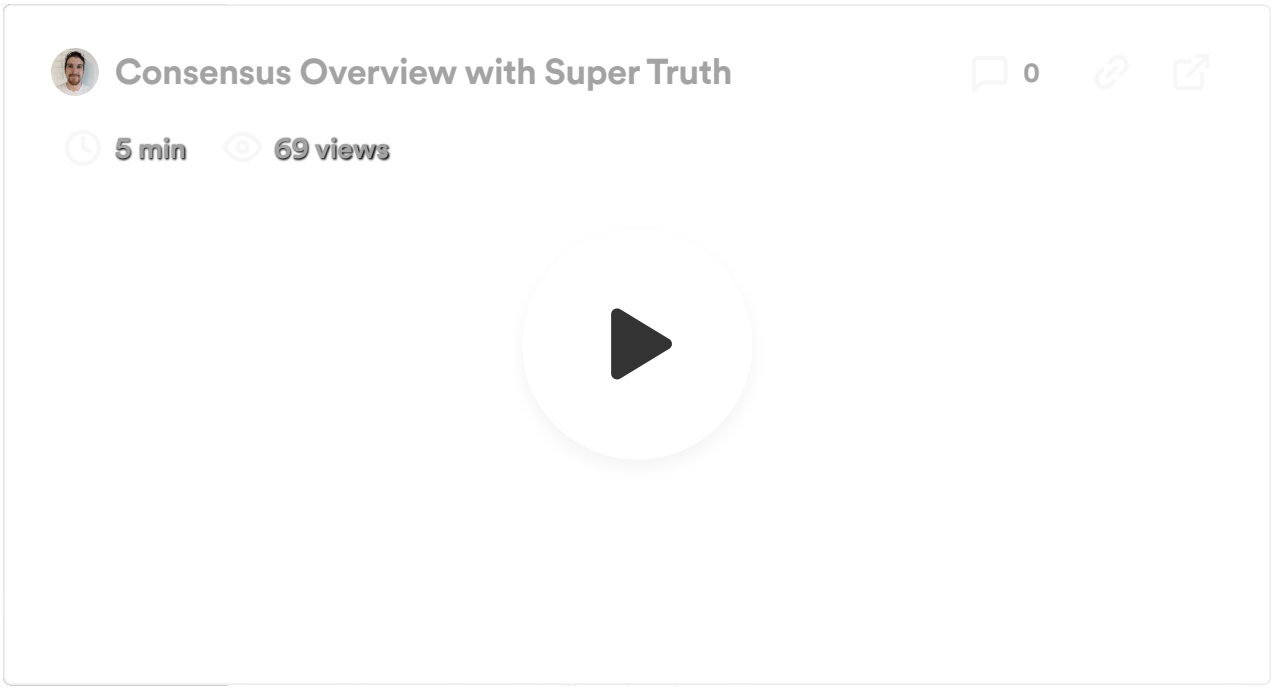
If a reviewer wishes to make changes to an existing set of annotations or start completely from scratch, they can either click on the **Edit** button under a user in the right hand panel or click on **Create New** under Super Truth.

Doing so will create a novel set of annotations known as a **Super Truth** and automatically designate the set as Best. The reviewer can then annotate the Task as they see fit.

> **Only Super Truth Annotations can be edited!**
>
> All other annotations in the Review Stage are View Only.

Once a reviewer is satisfied with the current Best Annotations, they can accept the Task. This saves the Best Annotations and ascribes only that set to the Task. All other annotations are also saved and are available on export. If the reviewer rejects the Task, **all labelers will be required to re-annotate the task**.

The video below contains a brief walkthrough of how you can use Consensus in both your Project and the Editor.



Exporting Consensus Annotations

If a task has gone through Consensus, you will get access to all versions of the annotations done by all users. You will also have access to additional metadata like the annotation similarity scores. You can export the data using the following CLI command inside your project directory:

```
redbrick export
```

Please view the format reference for an overview of the exported format.

If you want to export only a single version of the annotations (i.e. the labeler with the best annotations or the base annotations qualified in Review), you can run the following command:

```
redbrick export --no-consensus
```

Agreement calculation

This section of the documentation will cover how RedBrick AI calculates inter-annotator agreement between two users.

For two sets of labels, annotation instances are matched up by category. For the same category, instances are matched up by selecting pairs that maximize the overall agreement score. For two instances of the same category, RedBrick AI uses the following similarity functions

Bounding box, Polygon, and Pixel Segmentation

RedBrick AI uses IOU for these annotation types. For two annotations A and B IOU is defined by:

$$IOU = \frac{A \cup B}{A \cap B}$$

Landmarks

For landmarks/keypoints, RedBrick AI uses a normalized Root Mean Squared Error (RMSE) to compute similarity, where similarity is $Similarity = 1 - RMSE$.

$$MSE = \frac{1}{n}\sum_{i}^{n}(P_i - \hat{P}_i)^2$$
$$RMSE = \sqrt{MSE}$$

Where n is the number of components of the point (2 for 2D, 3 for 3D), and $P_i, \hat{P}_i$ are normalized components (by width, height, depth of the image) of the two points.

Length Measurements

Comparisons of length measurements are done by comparing the two sets of points (using the technique covered above) that define the length line.

Angle Measurements

For angle measurements, the vectors between each arm of the angle measurement are compared. The two angles comparing both sets of measurement arms are computed. The similarity score is then defined by:

$$Similarity = 1 - \frac{\theta_1 + \theta_2}{2\pi}$$

Where θ_1, θ_2 are the angles between the two sets of measurement arms.

Classification

For classification labels, the agreement is binary. If the chosen category and attributes match, the consensus score will be 100%, otherwise, it will be 0%.

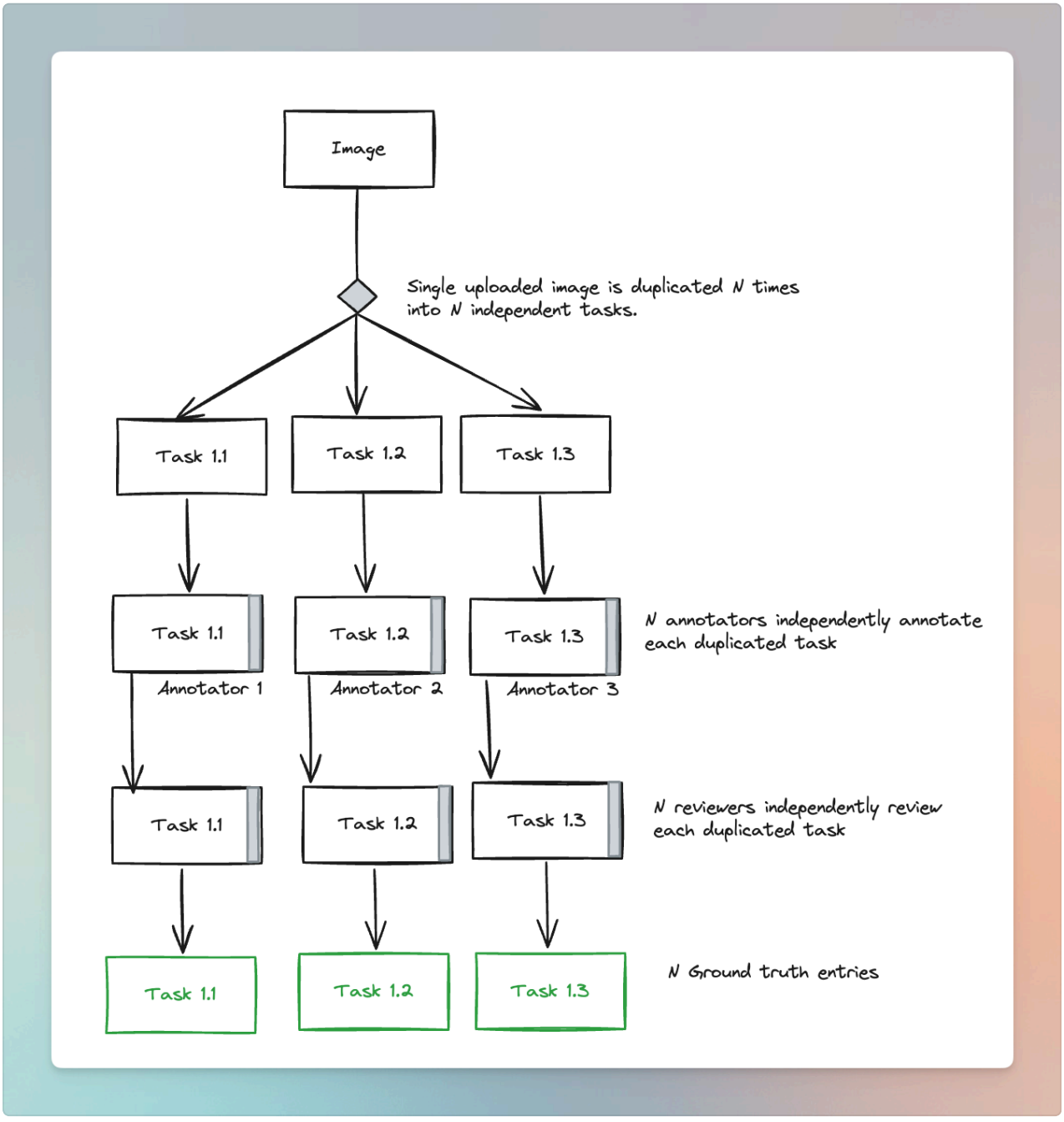
Generating a single score

To generate a single score between two sets of labels, a series of averages are computed.

1. Scores of matching annotations instances of the same category are averaged, to generate a single score per category.
2. Scores are then averaged per category.
3. Scores are then averaged per label type to generate a single score per label type.
4. For videos, scores are calculated per frame and averaged to generate a single score per sequence.
5. For multi-series studies, scores are averaged by volume to generate a single score per study.

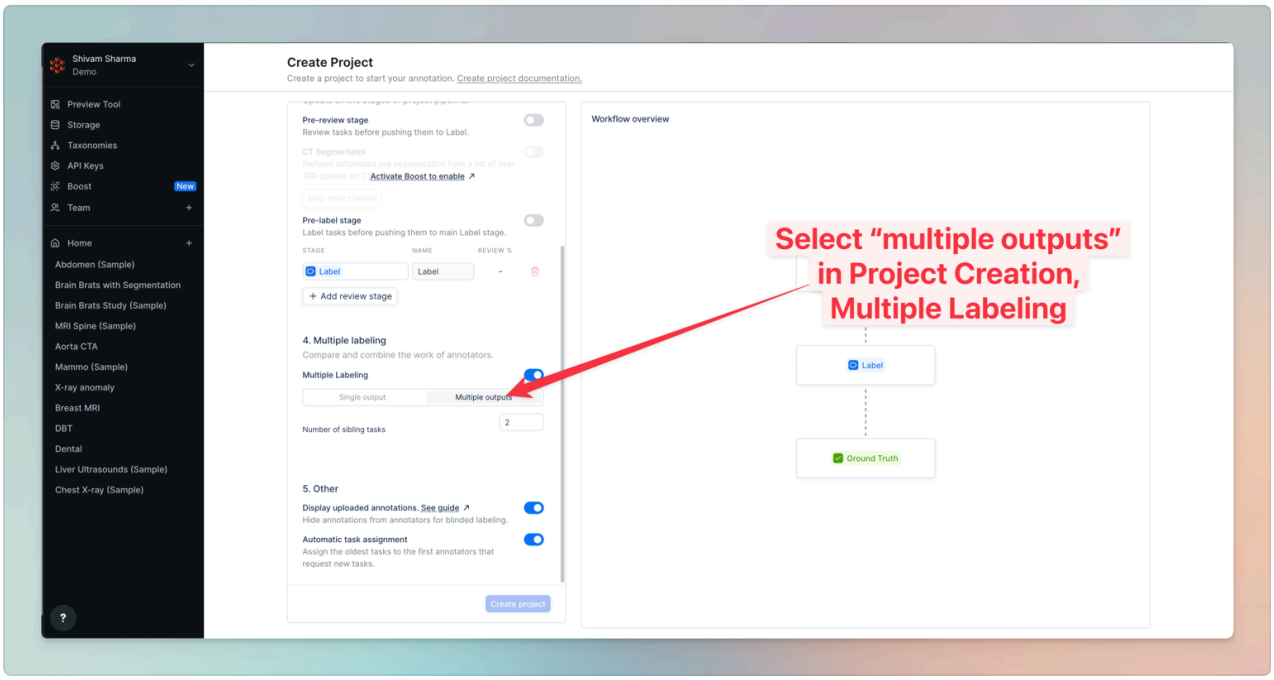
Task duplication

Task duplication is a project workflow setting that allows you to create N tasks from a single image so that N different labelers can annotate the image N times.

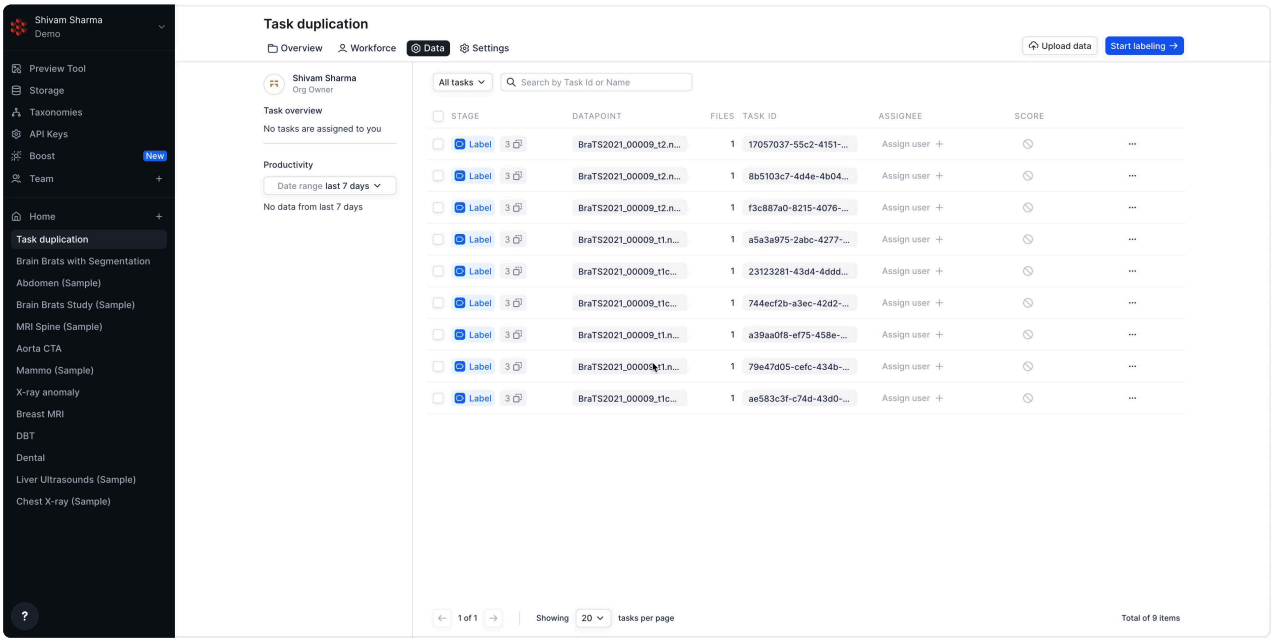


Create a Project with Task Duplication

While creating a project, in the **Multiple Labeling** section, select **Multiple Outputs** and then select the number of "sibling tasks", i.e., the number of times a task will be duplicated.

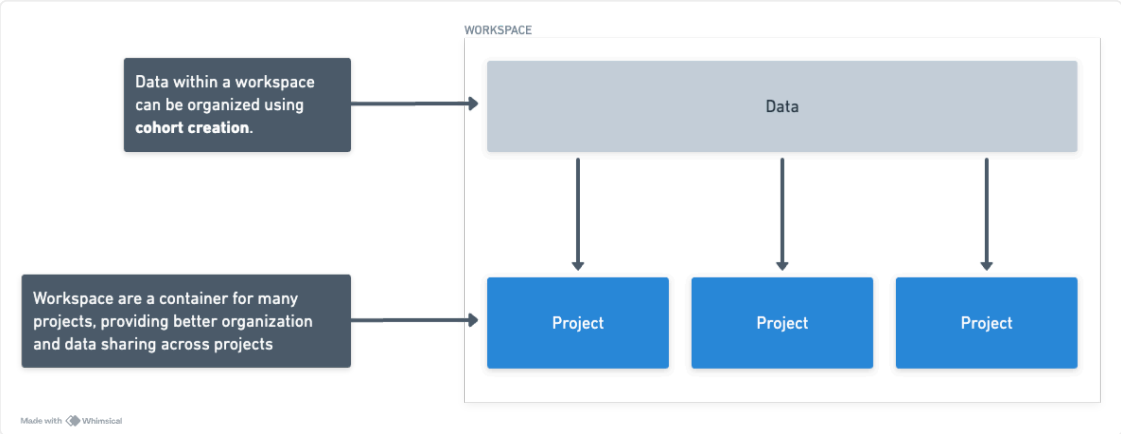


Once you upload tasks to your project, you will notice *N x the number of images* uploaded to the project. Each distinct sibling task can be assigned to a different annotator. You can view all sibling tasks together using the "view sibling" task shortcut.

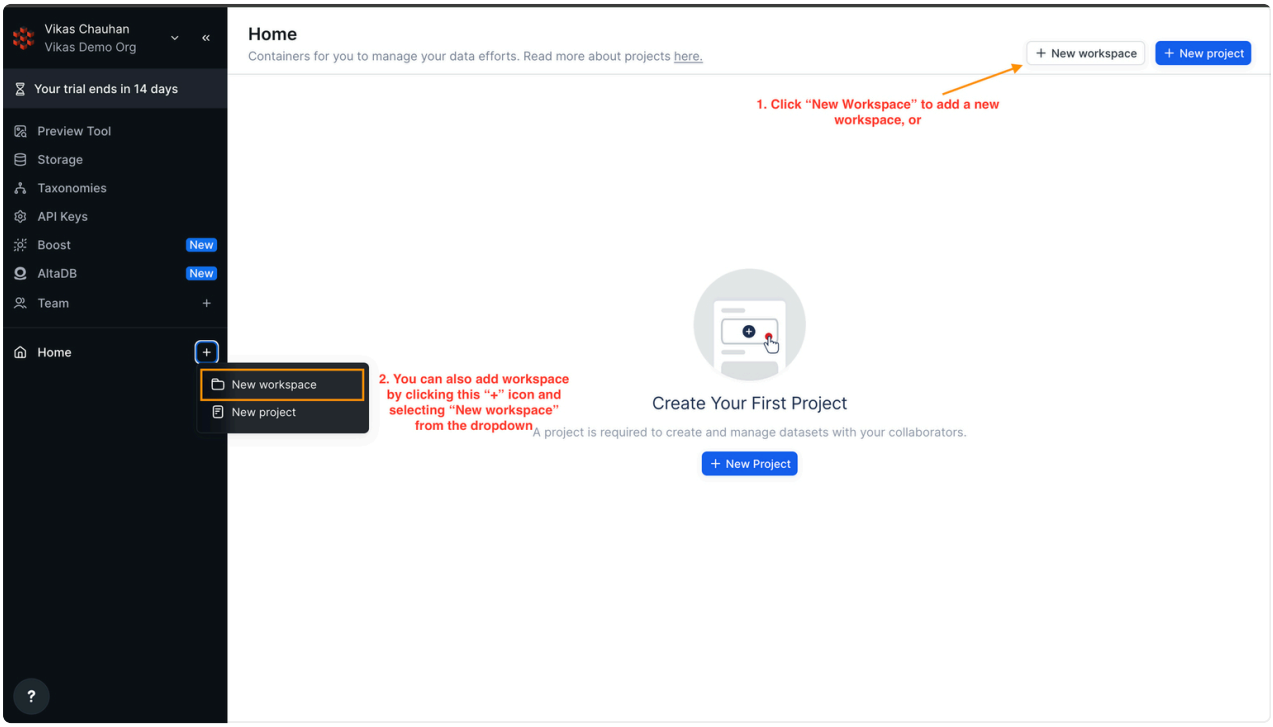


Get Started with Workspace

A workspace is a container in RedBrick AI that brings together data and multiple projects that share that data.



Creating Workspace: Step by Step



To create a new workspace, Press the "New workspace" button on the top right corner or you can click on the "+" Icon on the left-hand menu bar and select New workspace from the dropdown.

Step 1: Specify a Unique Workspace name

Once created, you won't be able to change the name of the workspace.

Step 2: Uploading Data to workspace

Once the workspace is created, the next step is to upload the data. Uploading the data in the workspace is similar to how you upload the data in the project.

If you'd like to upload annotation files alongside your images and/or volumes, you must use our CLI/SDK.

Import Data & Annotations ›

Step 3: Creating Project Inside Workspace

Next step is to create a project inside the newly created workspace, To create a new project click on the New Project button the top right corner.

Get Started with a Project ›

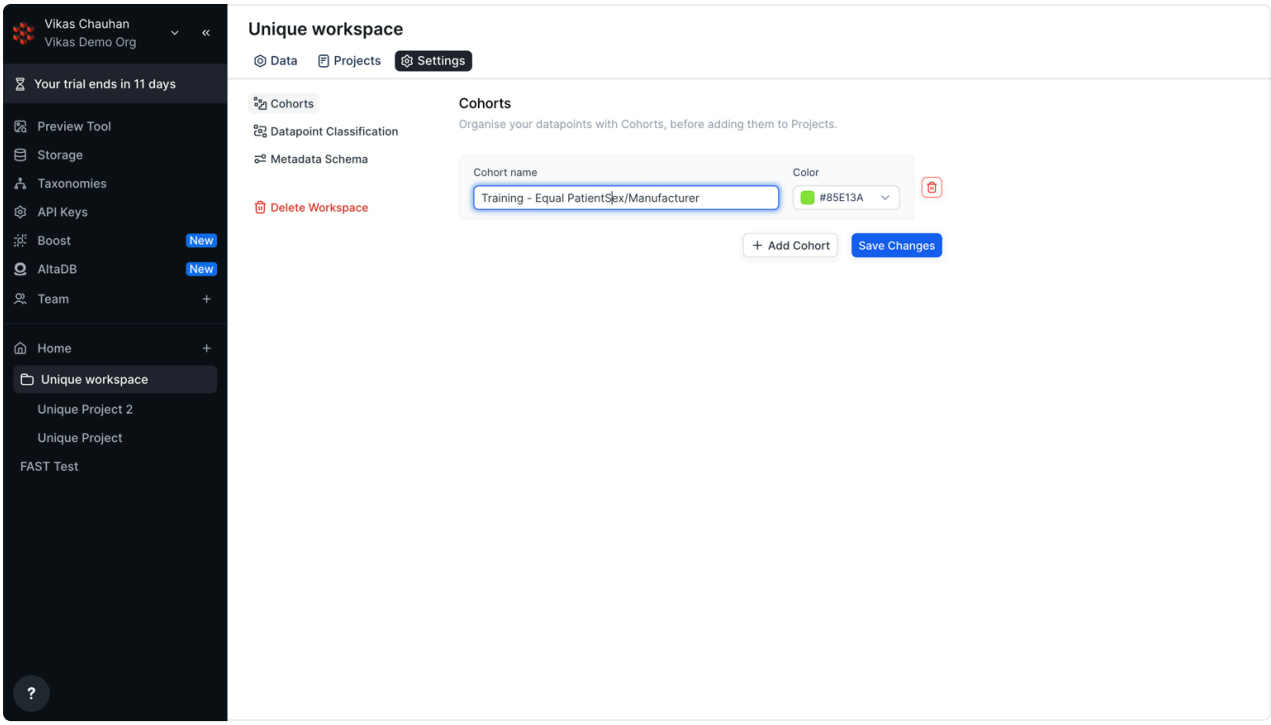
Cohort Creation

RedBrick AI cohort creation feature allows you to upload, index, share, ensure reproducibility, and control the quality of your studies before you send them for annotation.

Create a New Cohort

To create a new cohort, follow these steps:

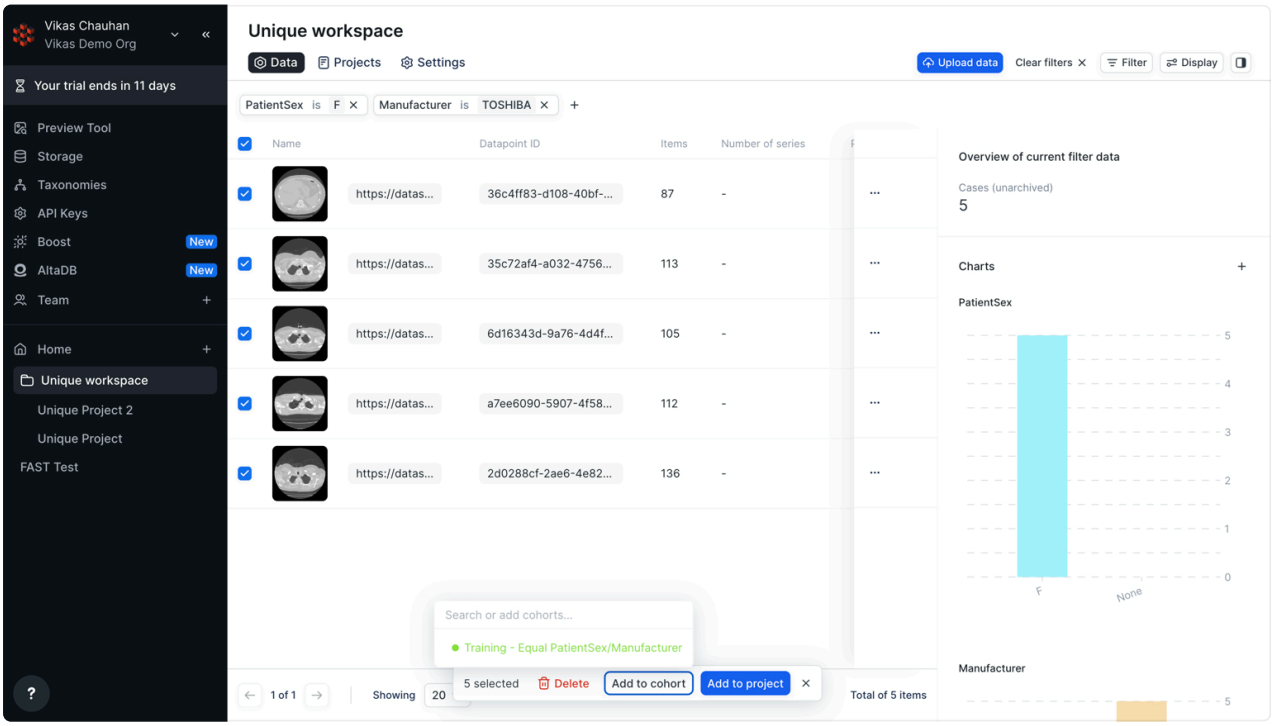
1. Navigate to the "Settings" section.
2. Click on "Cohort" in the left menu bar.
3. Enter the name of the new cohort.
4. Select a color code if you're adding multiple cohorts to help differentiate them and click on Save Changes.



Adding Datapoints to a Cohort

After creating a cohort, follow these steps to add filtered data points:

1. Navigate to the "Data" section.
2. Click on the "Filter" button located in the top right corner.
3. Select the desired data filter to apply.
4. Once the filter is applied, select the filtered data points.
5. Click on the "Add to cohort" button.
6. Choose the cohort to which you want to add the data points.



Alternatively, you can use the SDK method to add Datapoints to your cohorts.

Adding Cohort to a Project

Once data points are added to a cohort, you can add this cohort to a project by following these steps:

1. Click on the "Filter" button located in the top right corner.
2. Filter for the cohort you want to add to the project.
3. Select all the filtered data points.
4. Click on the "Add to Project" button at the bottom of the screen.
5. Choose the desired project from the list.

Datapoint Classification

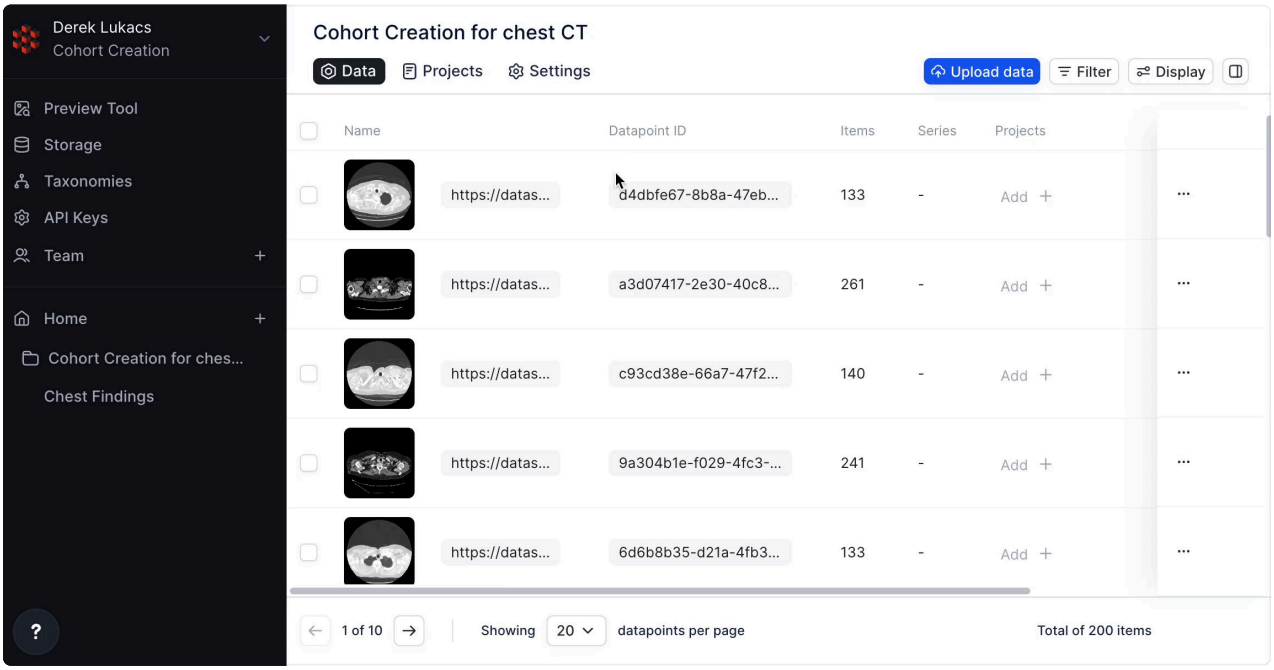
To manually classify data, you can use the Datapoint classification schema in the workspace settings. The classifications can be various types, such as True or False, Selection, Multi-Selection, or Textfield. Once modified, these classifications can be used for searching and filtering. Classification can be done either from the table or while previewing images.

Creating Datapoint Classification

To create a new data point classification, follow these steps:

1. Navigate to "Settings."
2. Go to "Datapoint Classification."
3. Choose from the four types of classifications: True or False, Single Select, Multi-Select, and Textfield.
4. Add the new classification.
5. Click on "Save Changes."

Adding Classification to Datapoints



Once Classification is created, you can add the classification from table or while previewing images.

Configuring Metadata Schema

Metadata Schema allows you to configure the workspace according to the metadata schema extracted from a CSV file or the DICOM headers. This schema is highly flexible and not strictly enforced on the uploaded data, providing maximum adaptability to your data format and allowing the schema to evolve.

Metadata schemas support the following data types for your metadata:

- NUMBER = "number"
- STRING = "string"
- DATETIME = "datetime"
- ENUM = "enum"

Importing Metadata to RedBrick AI

```python
# A script for converting from CSV to a JSON file for uploading to RedBri
import csv
import json


def csv_to_list_of_dicts(filename):
    with open(filename, "r") as file:
        reader = csv.DictReader(file)
        data = list(reader)
    return data


# Use the function
cases = csv_to_list_of_dicts("tags_and_paths.csv")


def item_to_redbrick_usable_url(item: str) -> str:
    """
    Convert the item stored in the csv file to something that RedBrick ca

    This will vary depending on where your images are stored. In this cas
    at a public url. Yours is probably stored in an S3 bucket and your pa

    Check docs.redbrickai.com for more information.
    """
    return "https://datasets.redbrickai.com/chest_ct_lidc_idri/" + item


upload_format = []
for case_ in cases:
    # ['LIDC-IDRI-0195/1-102.dcm', 'LIDC-IDRI-0195/1-103.dcm', ...]
    items = case_["items"]

    # parse the way items were stored in the csv file
    items = json.loads(items.replace("'", '"'))
    del case_["items"]

    metadata = case_

    upload_format.append(
        {
            "items": [item_to_redbrick_usable_url(item) for item in items
            "metaData": metadata,
        }
    )


# Write to a JSON file
```

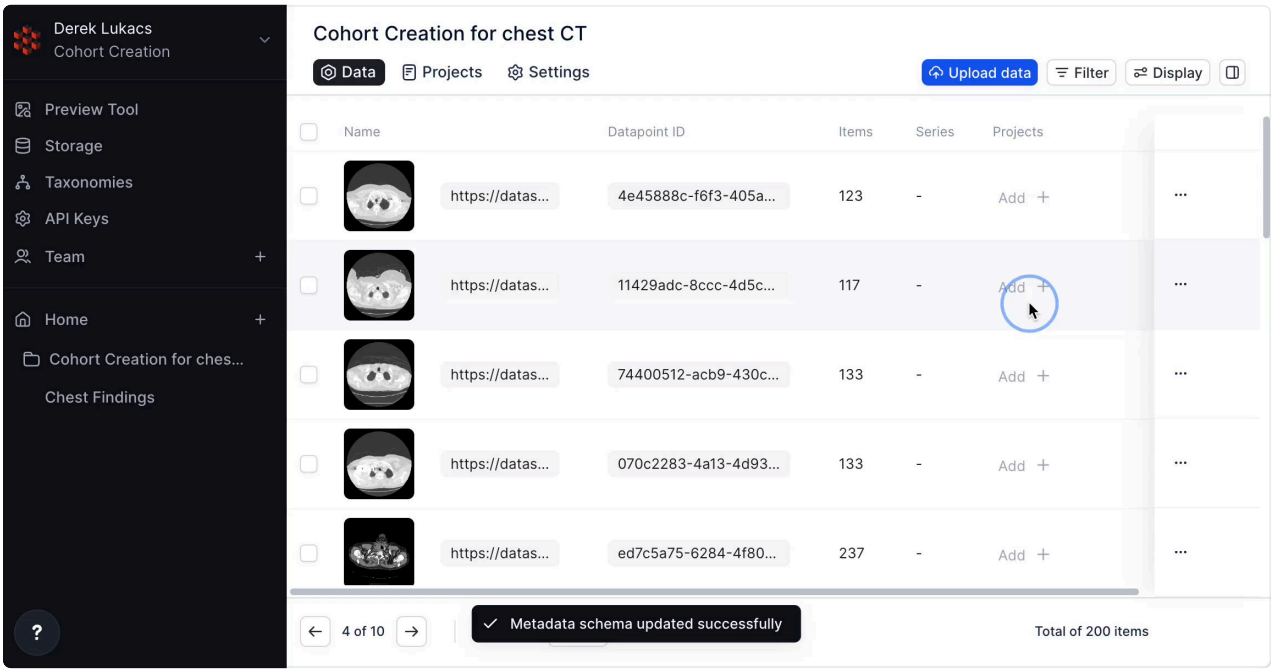
```
with open("upload_format.json", "w+") as file:
    json.dump(upload_format, file, indent=2)
```

This will produce a JSON file in the following format:


```
[
  {
    "items": [
      "https://datasets.redbrickai.com/chest_ct_lidc_idri/LIDC-IDRI-0125/
      ...
    ],
    "metaData": {
      "PatientID": "ABC-125",
      "StudyDate": "20000101",
      "StudyTime": "",
      "AccessionNumber": "",
      "Modality": "CT",
      "Manufacturer": "GE MEDICAL SYSTEMS",
      "StudyDescription": "",
      "SeriesDescription": "",
      "PatientName": "",
      "PatientBirthDate": "",
      "PatientSex": "",
      "BodyPartExamined": "CHEST",
      "SliceThickness": "1.250000",
      "KVP": "120",
      "DistanceSourceToDetector": "949.075012",
      "DistanceSourceToPatient": "541.000000",
      "GantryDetectorTilt": "0.000000",
      "TableHeight": "156.500000",
      "RotationDirection": "CW",
      "XRayTubeCurrent": "400",
      "CountryOfResidence": "",
      "PatientIdentityRemoved": "YES",
      "PatientPosition": "FFS"
    }
  },
  ...
]
```


Before creating the metadata schema in RedBrick AI, we recommend that you import metadata into RedBrick AI.

Creating Metadata Schema in RedBrick AI



Once you've imported the metadata, you can create Cohorts based on your custom metadata schema and then send those Cohorts to Annotate.

To create a Metadata schema, follow these steps:

1. Go to "Settings."
2. Click on "Metadata Schema."
3. Choose from the four schema types: Date, Number, Enum, and Textfield.
4. Create the desired schema.

After creating the schema, you can use it as follows:

1. Go to "Data."
2. Click on the "Filter" button in the top right corner.
3. Apply the filter to the metadata you want to view, sort, or add to a cohort.
4. Once filtered, select all the filtered data points.
5. Add the selected data points to a cohort or a project.

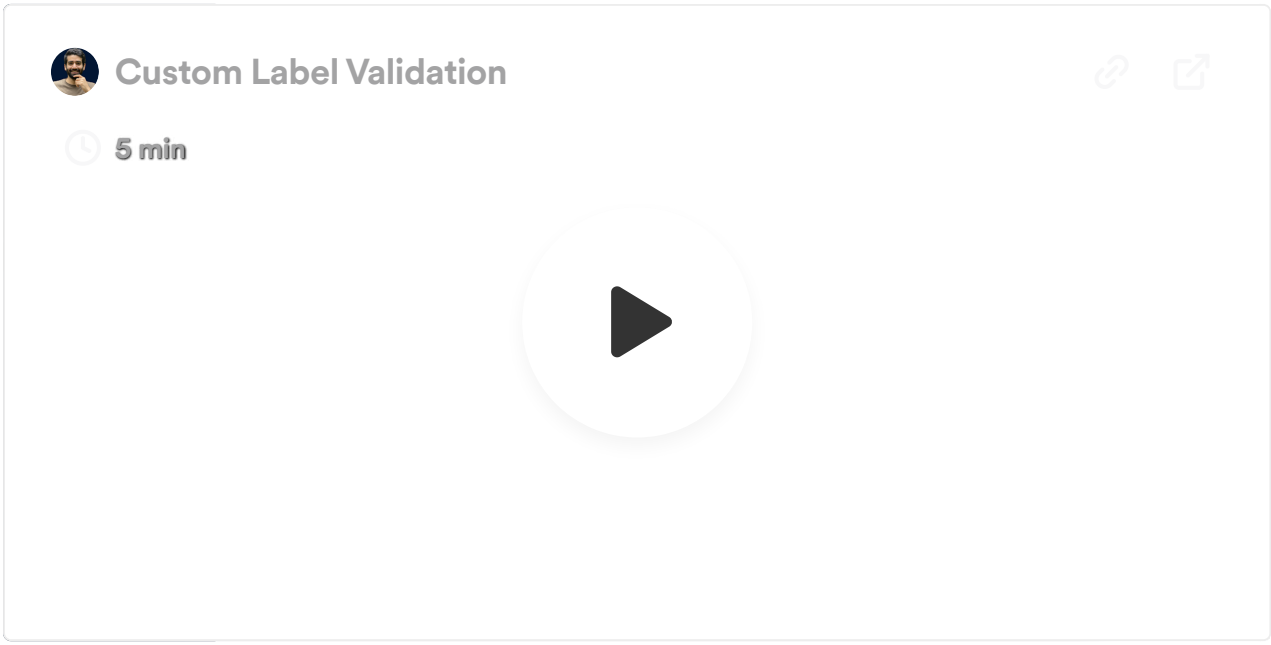
Custom Label Validation

You can define a custom Javascript script that can continuously compare annotations to a schema/set of rules and inform annotators (in real-time) of any mistakes in their annotations. You can also prevent annotators from submitting tasks when your validation script finds errors.

For most annotation projects, there is a schema/rule-set/taxonomy that annotators must follow. A large portion of errors in annotation projects is due to oversights/slips during labeling in adhering to the schema.

For example, the annotator must segment a tumor and fill out a few related attributes if a tumor is found. A common error can be forgetting to fill out all attributes when the tumor is present. Post-processing scripts usually reveal these errors.

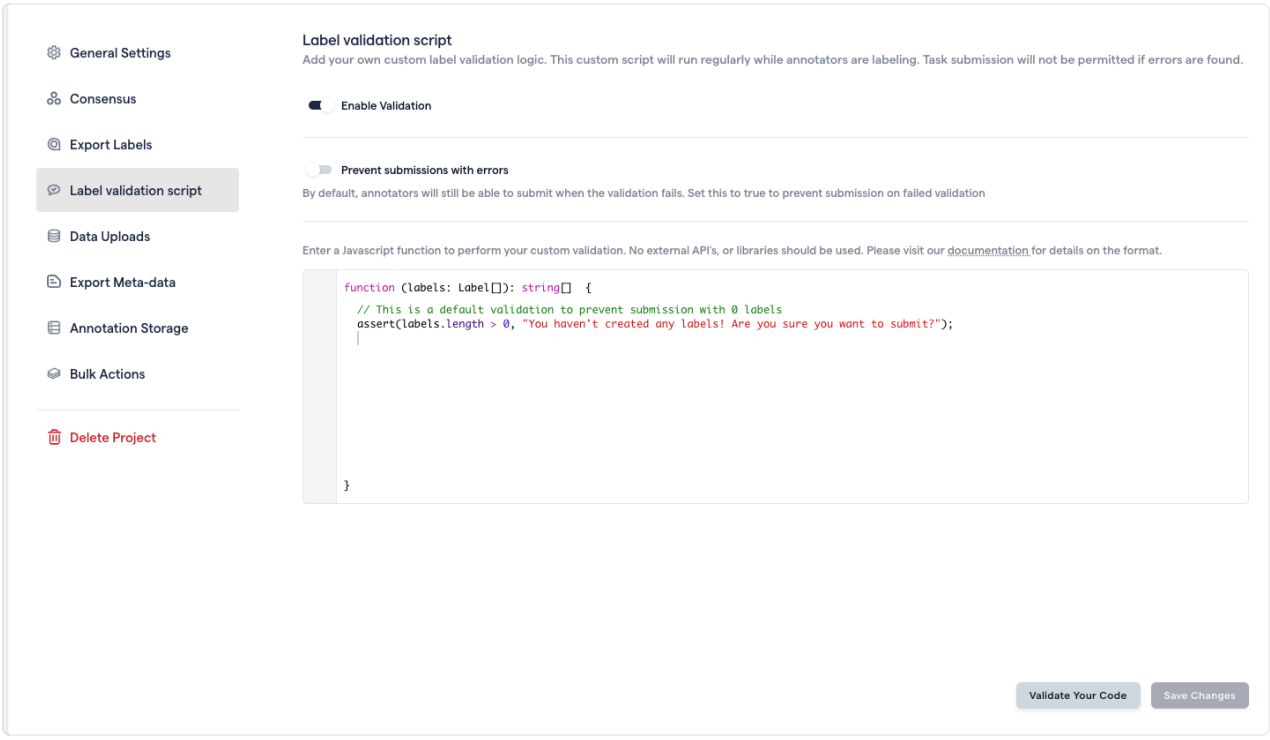
Prevent simple, recurrent errors from occurring by writing a set of tests that will be run regularly, informing annotators of any mistakes they're making.



Overview of custom label validation

Overview

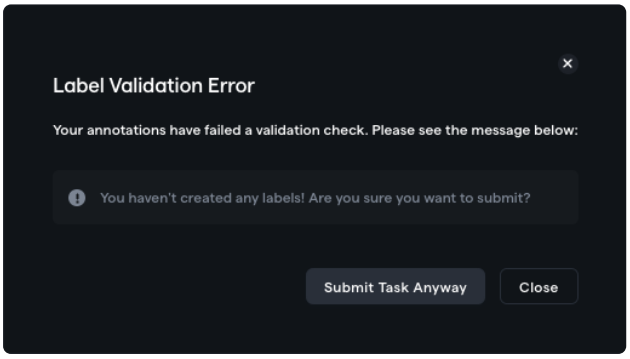
By default, all projects have a custom check to warn annotators when they submit tasks without any annotations. You can enable/disable custom validation under Project Settings -> Label Validation.



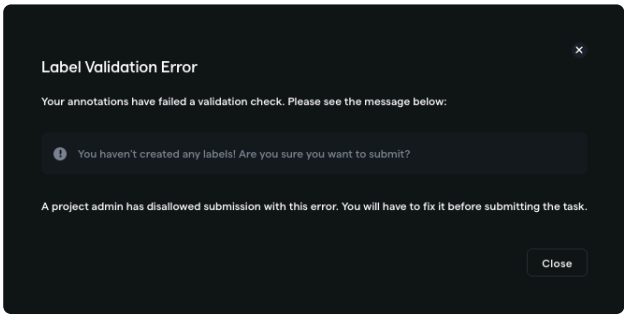
Custom label validation in settings

Prevent Submissions with Errors

By default, annotators will just receive the error messages as a warning, and they will still be able to submit the task anyway. To prevent the annotators from submitting with any errors present, toggle the *Prevent submission with errors* switch.



Submission allowed



Submission with errors prevented

Custom Javascript function

You will write the custom validation as a Javascript function. This Javascript function will run on each annotator's browser while they are annotating data.

The Javascript function has the following definition:

```
function(task: Task, labels: Label[]): string[]  {
  // Your custom validation logic
  assert(false, "This assertion was false");
}
```

`label: Label[]`

The validation function has a single input - a list of labels containing minimal meta-data about the labels. Please see the definition of the `Label` object below:

```ts
interface Label {
  category: string[];
  attributes: LabelAttribute[];
  labelType: TaskType;
  numFramesLabeled?: number;
  instanceTracks?: { [name: string]: FrameState[] };
  seriesIndex?: number;
}

// Label attribute
interface LabelAttribute {
  name: string;
  value: boolean | number | string;
}

// Task Type
enum TaskType {
  ITEMS = 'ITEMS',
  CLASSIFY = 'CLASSIFY',
  BBOX = 'BBOX',
  POLYGON = 'POLYGON',
  POLYLINE = 'POLYLINE',
  POINT = 'POINT',
  ELLIPSE = 'ELLIPSE',
  SEGMENTATION = 'SEGMENTATION',
  MULTI = 'MULTI',
  MULTICLASSIFY = 'MULTICLASSIFY',
  LENGTH = 'LENGTH',
  ANGLE = 'ANGLE',
}


interface Task {
    orgId: string;
    projectId: string;
    stageName: string; // i.e. "Label" or "Review_1"
    taskId: string;
    name: string; // Name given for the task at upload
    metaData: Record <string, any>;
    classification?: Classification;
    series: Series[];
}

interface Series {
  name: string;
  metaData: Record <string, any>;
  dimensions: [number, number, number];
```

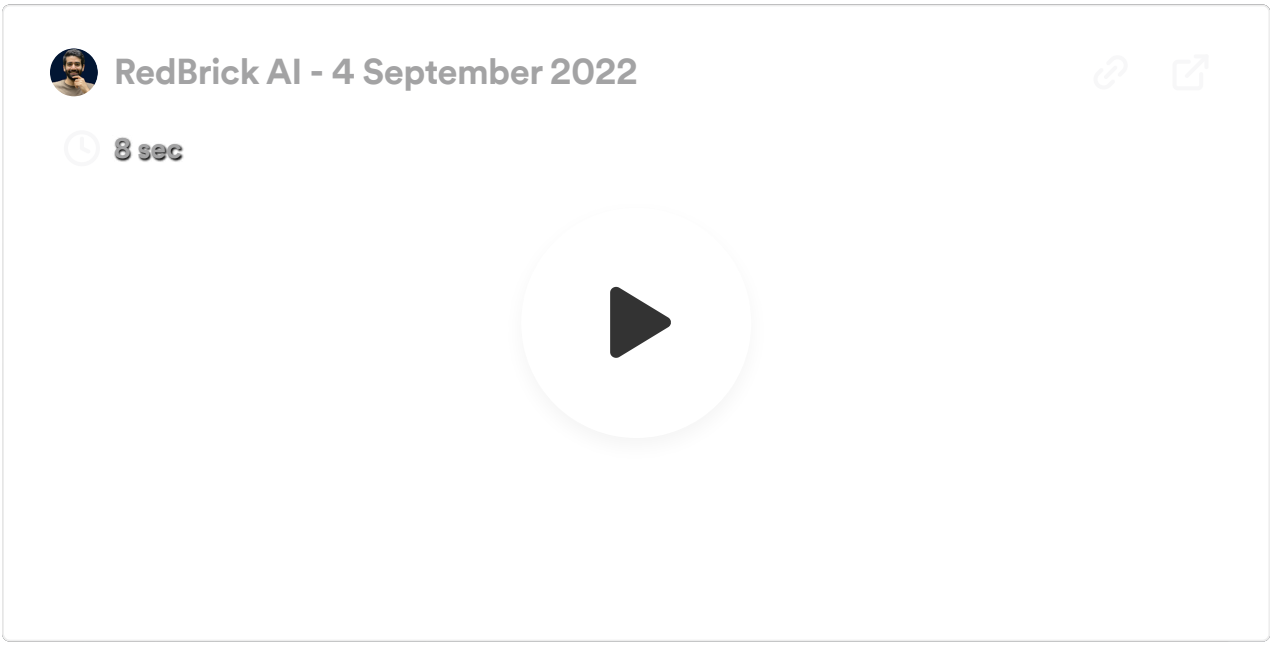
```
  classifications?: Classification[];
  instanceClassifications?: InstanceClassification[];

  landmarks?: Landmark[];
  landmarks3d?: Landmark3D[];
  measurements?: (MeasureLength | MeasureAngle)[];
  boundingBoxes?: BoundingBox[];
  cuboids?: Cuboid[];
  ellipses?: Ellipse[];
  polygons?: Polygon[];
  polylines?: Polyline[];

  segmentMap?: {
    [instanceId: string]: {
      category: Category;
      attributes?: Attributes;
      overlappingGroups?: number[];
    };
  };
}


interface VideoMetaData {
  frameIndex: number;
  trackId: string;
  keyFrame: boolean;
  endTrack: boolean;
}

interface Classification {
  attributes: Attributes;
  video?: VideoMetaData;
}

interface InstanceClassification {
  fileIndex: number;
  values: Attributes;
}

interface MeasurementStats {
  average?: number;
  area?: number;
  volume?: number;
  minimum?: number;
  maximum?: number;
}

interface Landmark {
```

```
  point: Point2D;
  category: Category;
  attributes?: Attributes;
  video?: VideoMetaData;
}

interface Landmark3D {
  point: VoxelPoint;
  category: Category;
  attributes?: Attributes;
}

interface MeasureLength {
  type: 'length';
  point1: VoxelPoint;
  point2: VoxelPoint;
  absolutePoint1: WorldPoint;
  absolutePoint2: WorldPoint;
  normal: [number, number, number];
  length: number;
  category: Category;
  attributes?: Attributes;
}

interface MeasureAngle {
  type: 'angle';
  point1: VoxelPoint;
  point2: VoxelPoint;
  vertex: VoxelPoint;
  absolutePoint1: WorldPoint;
  absolutePoint2: WorldPoint;
  absoluteVertex: WorldPoint;
  normal: [number, number, number];
  angle: number;
  category: Category;
  attributes?: Attributes;
}

interface BoundingBox {
  pointTopLeft: Point2D;
  wNorm: number;
  hNorm: number;
  category: Category;
  attributes?: Attributes;
  stats?: MeasurementStats;
  video?: VideoMetaData;
}
```

```ts
interface Cuboid {
  point1: VoxelPoint;
  point2: VoxelPoint;
  absolutePoint1: WorldPoint;
  absolutePoint2: WorldPoint;
  category: Category;
  attributes?: Attributes;
  stats?: MeasurementStats;
}

interface Ellipse {
  pointCenter: Point2D;
  xRadiusNorm: number;
  yRadiusNorm: number;
  rotationRad: number;
  category: Category;
  attributes?: Attributes;
  stats?: MeasurementStats;
  video?: VideoMetaData;
}

interface Polygon {
  points: Point2D[];
  category: Category;
  attributes?: Attributes;
  stats?: MeasurementStats;
  video?: VideoMetaData;
}

interface Polyline {
  points: Point2D[];
  category: Category;
  attributes?: Attributes;
  video?: VideoMetaData;
}

// i is rows, j is columns, k is slice
interface VoxelPoint {
  i: number;
  j: number;
  k: number;
}

// The position of VoxelPoint in physical space (world coordinate) comput
interface WorldPoint {
  x: number;
  y: number;
  z: number;
}
```

```
}

interface Point2D {
  xNorm: number;
  yNorm: number;
}

type Category = string;
type Attributes = { [attributeName: string]: string | boolean | string[]
```

You can generate a sample `Label[]` object by going to the labeling tool -> opening command bar `cmd/ctrl + k` -> *Copy current label state to clipboard.*



Returns `string[]`

Your custom validation script must return a list of warning/error messages describing the issues found. Return an empty array `[]` if no errors are found. These error message strings will be displayed to the annotators on the labeling tool.

To help you write a validation function with several checks, RedBrick AI has a custom-defined function `assert` that accepts a boolean statement and a corresponding error message. The two scripts below will produce the same result:

With assert()

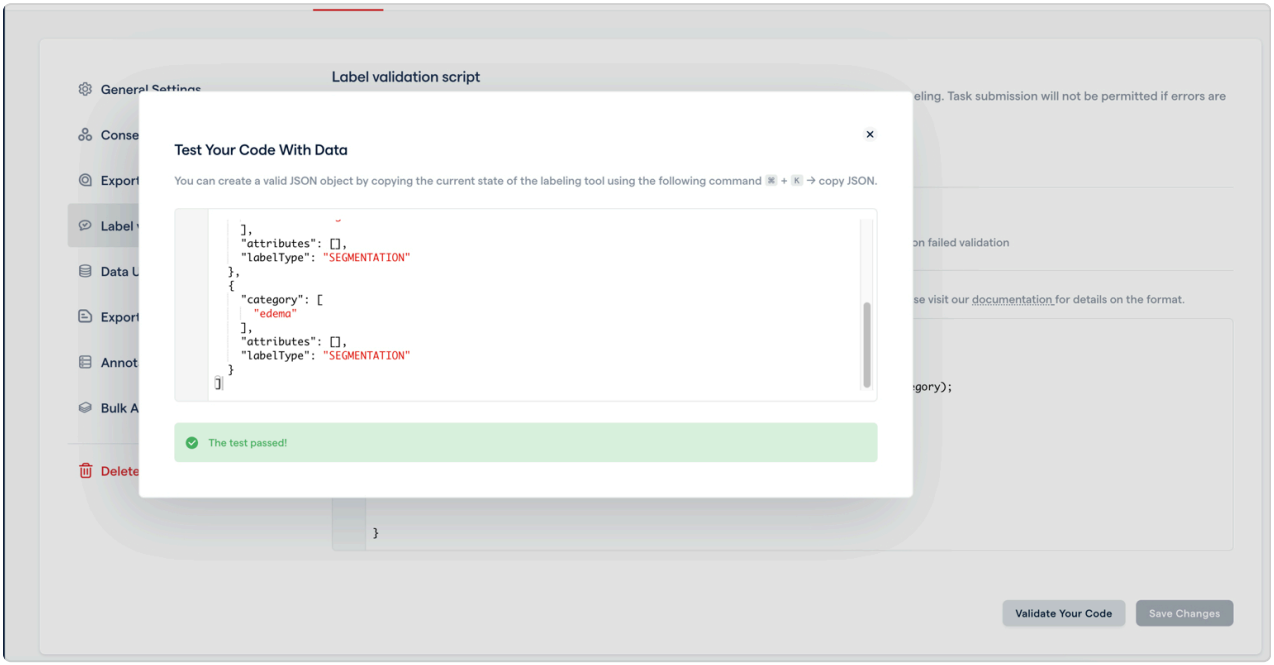
```
function(task: Task, labels: Label[]): string[] {
    assert(labels.length >= 1, "You have not created any labels!");
    assert(labels.length <= 5, "You have created too many labels!");
}
```

Without assert()

```
function(task: Task, labels: Label[]): string[] {
    const errors = [];

    if (labels.length < 1) {
        errors.push("You have not created any labels!");
    }
    if (labels.length > 5) {
        errors.push("You have created too many labels!");
    }

    return errors;
}
```

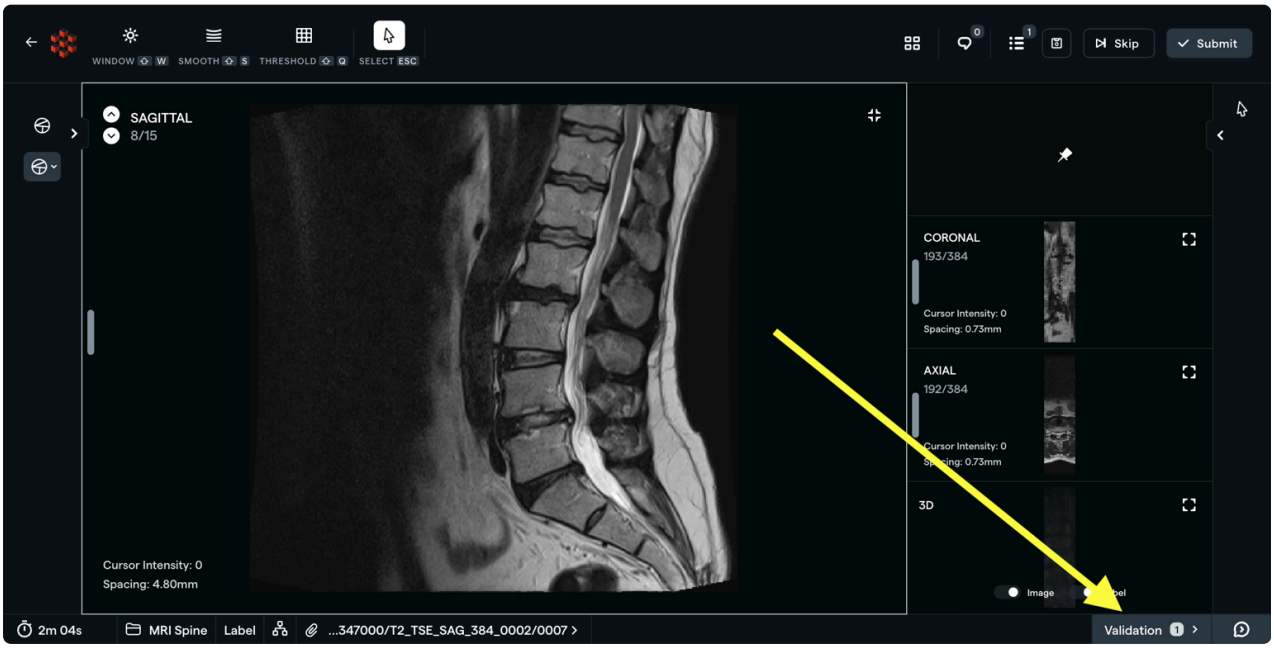
Validate Your Code

Before saving your script, you should validate that your code executes as expected. Click on the validate button on the bottom right of the Settings page, and paste the JSON object copied from the labeling tool to see if your code executes as expected:

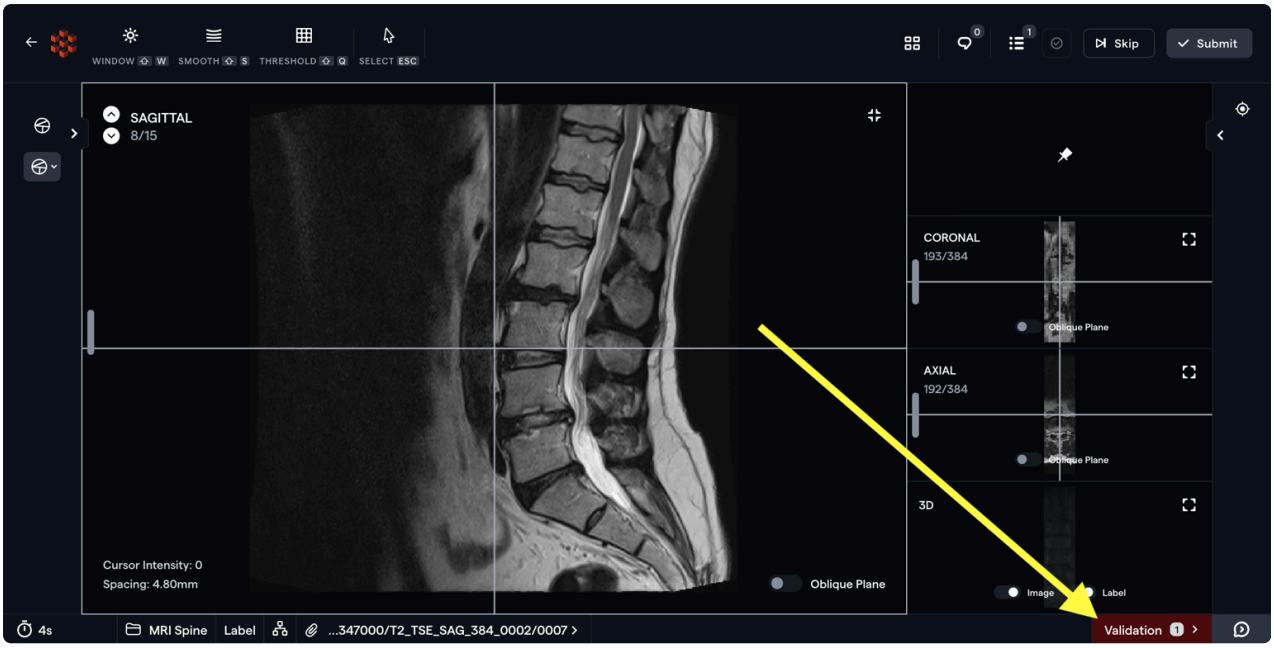


Displaying the Validation on the Labeling Tool

Your custom validation script will be regularly run. If any warnings are found, an indicator will appear on the right side of the bottom bar. If you have enabled **Prevent submissions with errors**, the indicator will be red.



Submission with errors is allowed



Submission with errors is prevented

Example Scripts

Check if Exact Categories are Present

For this example, let's say we are expecting each task to contain the following segmentations - *necrosis, enhancing tumor, non-enhacing tumor and edema.*

```
function(task: Task, labels: Label[]): string[] {
  const expectedCategories = [
    'necrosis',
    'enhancing tumor',
    'non-enhancing tumor',
    'edema',
  ];

  // Iterate over all expected categories
  for (const category of expectedCategories) {

    // Check if the category is present in any label
    const categoryPresent = labels.some(
      (label) => label.category[0] === category
    );

    // assert with message
    assert(categoryPresent, `The ${category} category is missing!`);
  }
}
```

Validate Only Single Instance of a Category has been Created

This script validates only a single instance of a particular category has been created. If you're expecting semantic segmentation labels, this check can ensure annotators don't accidentally create multiple instance segmentations.

```
function(task: Task, labels: Label[]): string[] {
  const semanticCategory = 'edema';

  const labelsFiltered = labels.filter((label) => label.category[0] === s

  assert(
    labelsFiltered.length === 1,
    `Expected exactly 1 ${semanticCategory} annotation`
  );
}
```

Verify that Specific Segmentation Type is Visible on Specific Series

The following script examines the Series Identifier and verifies whether a specific Segmentation type is present on it. In this example, you could use this script to be sure that labelers cannot finalize a Series that ends in "DWI" (a common naming convention for DWI images) without including an "Infarct" segmentation on the Series.

More broadly speaking, this script is an example of the extensive functionality available when combining the `label`, `task`, and `series` objects.

```
function (task: Task, labels: Label[]): string[]  {
  assert(labels.length > 0, "You haven't created any labels! Are you sure
  for (let label of labels) {
    if (label.category[0] === "Infarct") {
      assert(
        task.series[label.seriesIndex].name.startsWith("DWI_"),
        "Segmentation 'Infarct' is allowed only on 'DWI' images"
      );
    }
  }
}
```

Labeler Evaluation

Calculating Labeler Quality Scores

RedBrick AI allows you to upload a Ground Truth annotation file alongside any image or volume file for the purposes of evaluating labeler quality.

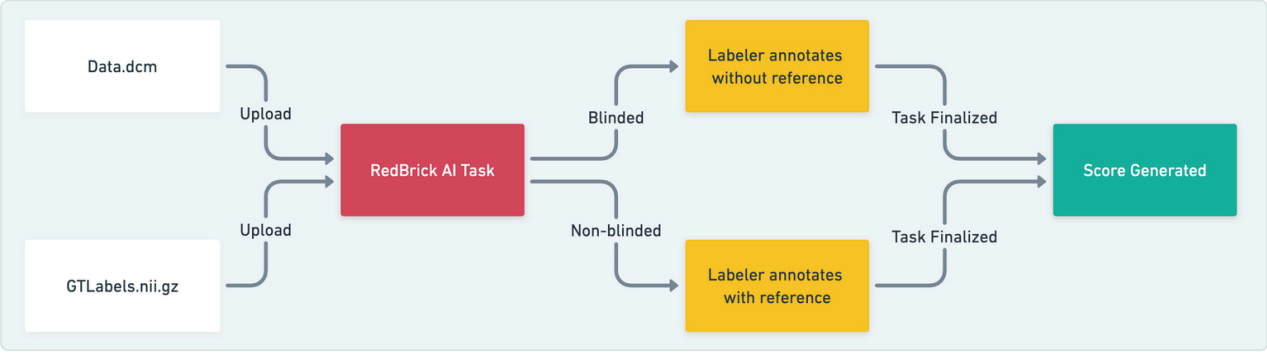
This can be useful when you'd like to have RedBrick AI calculate a score that you can use to compare a specific labeler's performance against a known Ground Truth label set.

Blinded vs. Non-blinded Annotations

When evaluating labeler quality using these Evaluation Tasks, you have the option of allowing your labelers to visually reference the Ground Truth annotations that you are using as a baseline or keeping them invisible to the labeler.

This feature is referred to as either **Non-blinded Annotations** or **Blinded Annotations,** respectively.

To create an Evaluation Task in RedBrick AI, you can take the following steps:



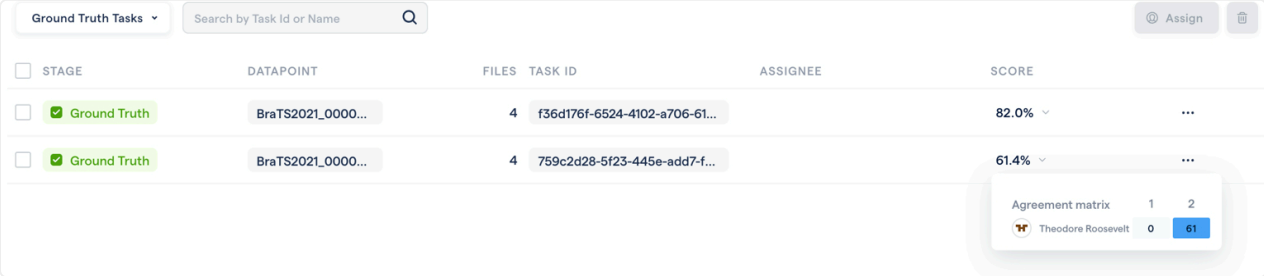
Sample Flow

1. Upload your image/volume alongside your Ground Truth annotation file ("Baseline Annotations"). A walkthrough of how to do so can be found in our documentation for importing annotations.

2. After your Task has been created, determine whether you would like your labelers to see the Baseline while working. Navigate to your **Project Settings** and enable or disable the **Show reference annotations** toggle.
    1. With the toggle enabled, labelers will be able to see the Baseline Annotations while working. With the toggle disabled, the Baseline Annotations will be invisible to the labeler.
    2. Please note that we also generally recommend disabling **Automatic Task Assignment** when testing labeler quality.

Relevant toggles in Project Settings

3. Assign the Task to your labeler for completion.
4. After your labeler finalizes the Task, an agreement score will be displayed on the Data Page.



A completed Evaluation Task and score

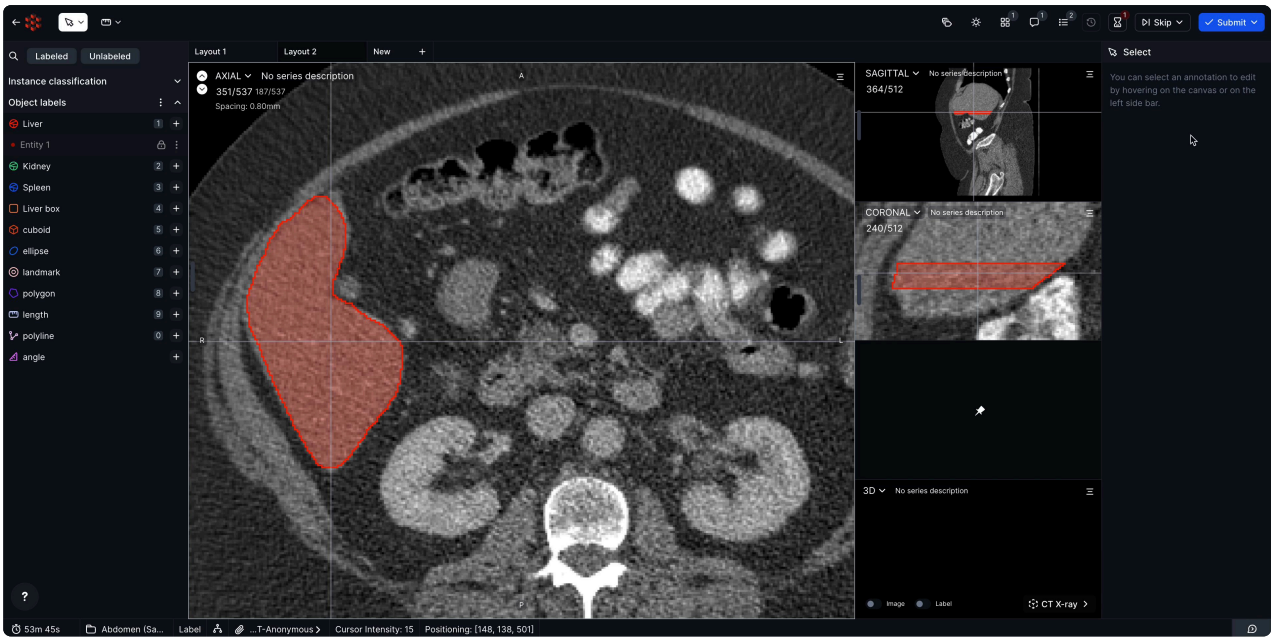
Comments & Raise Issue

Comments

Comments allow you to communicate with your team right alongside your annotation task. Comments are usually used to ask questions or provide feedback on annotations.

Create a comment

To create a comment, head over to the **Comments** tab & leave a comment.

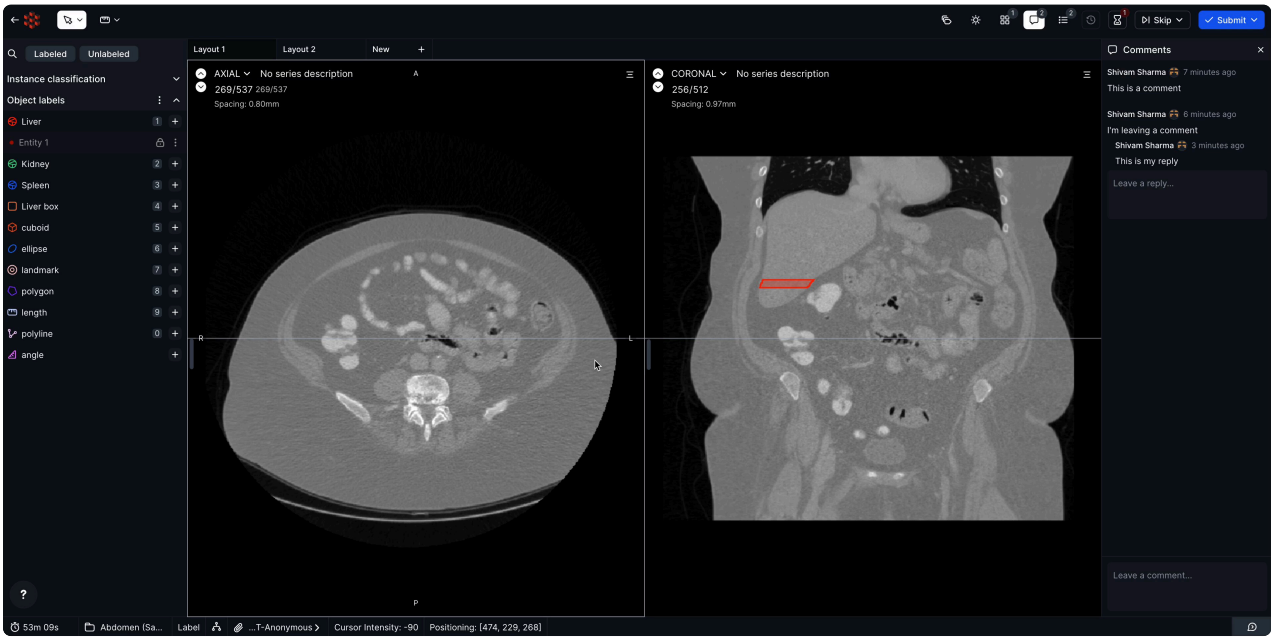


Reply to a comment

You can organize threads of conversations on comments by replying to a specific comment. This is especially useful in organizing comments across stages in a project.

Pin a comment

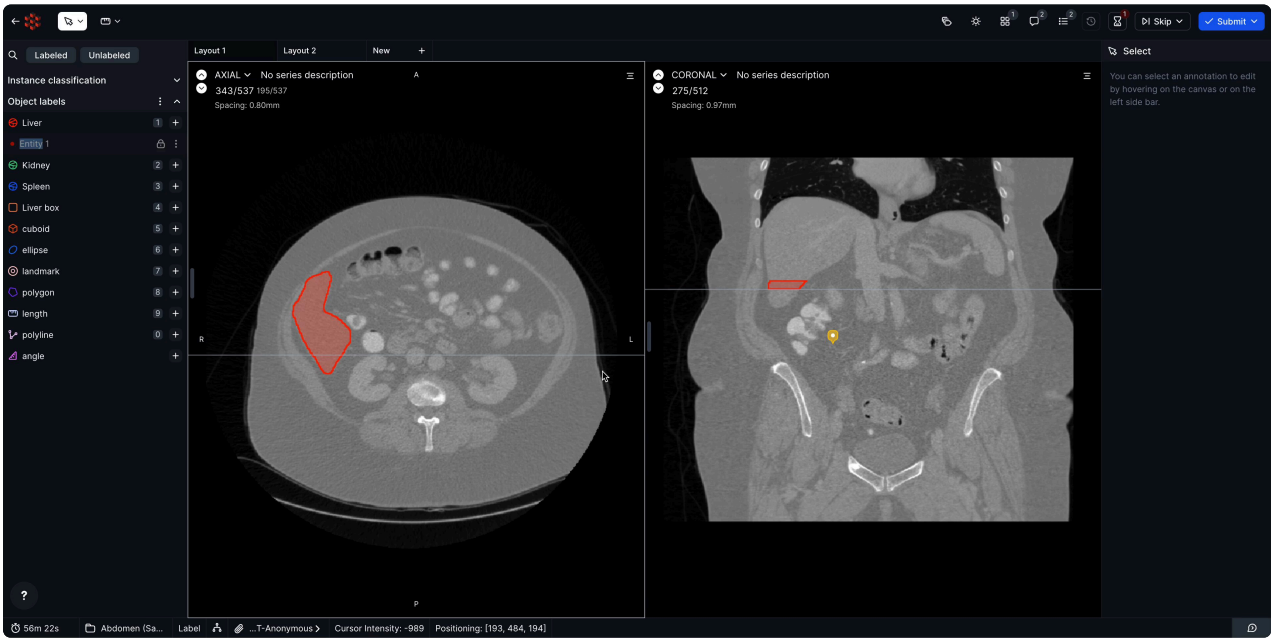
Often you may want to visually pin comments to some part of the image or annotation canvas. You can do so by **selecting** a comment and clicking the **pin** icon. After you drop a pin on the canvas, you can select the comment on the right sidebar to **jump to the pin location**.



Leave a comment on a label

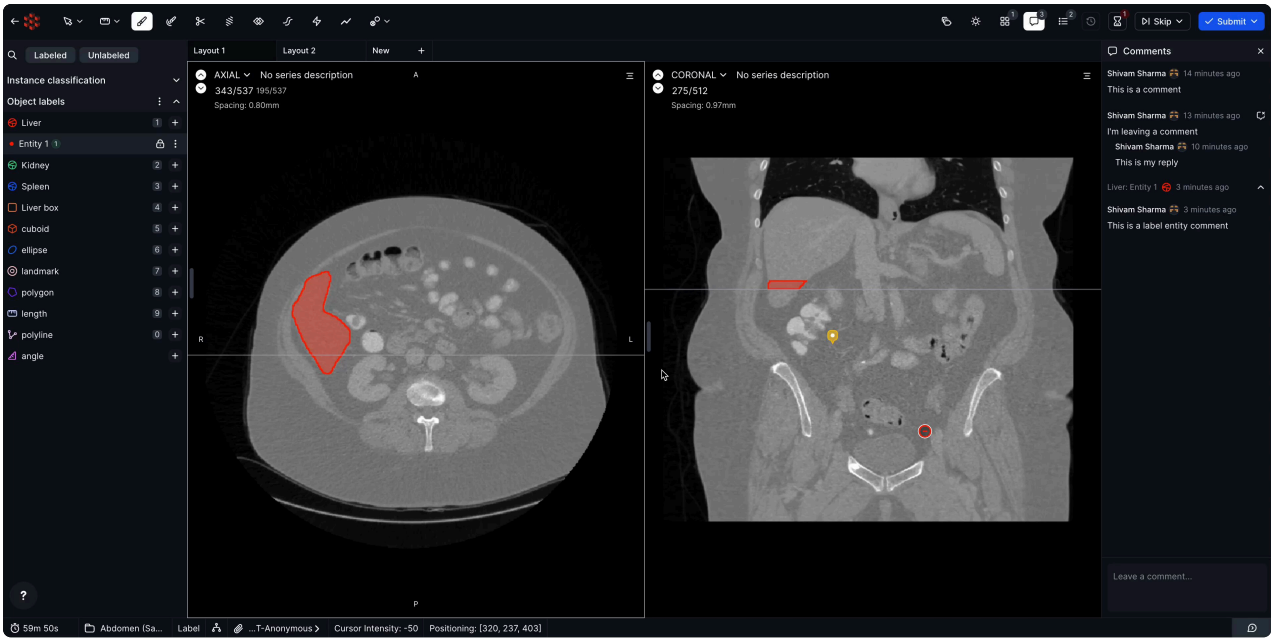
You can reference a specific label entity in your comment by leaving a comment on that entity. To do so, select an entity, then go to the **Entity comments** section on the right sidebar and leave a comment there.

All comments will appear in the main comments section (accessed from the top bar), and label-specific comments will also appear by selecting an entity within the Entity comments section.



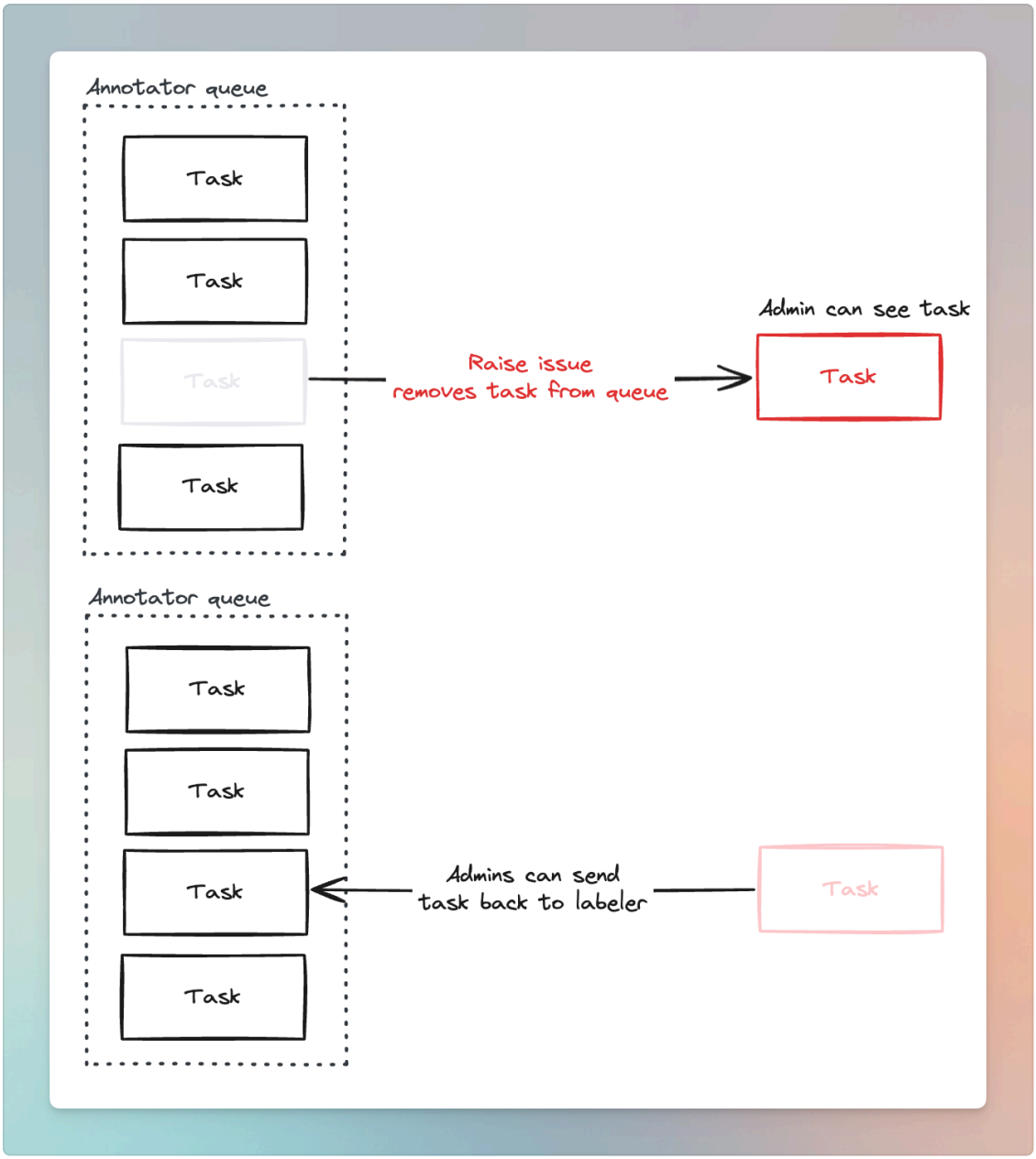
Resolving comments

Once you have completed a conversation thread or resolved the question/issues outlined in a comment, you can "resolve" the comment to visually differentiate the comment from others.



Raise Issue

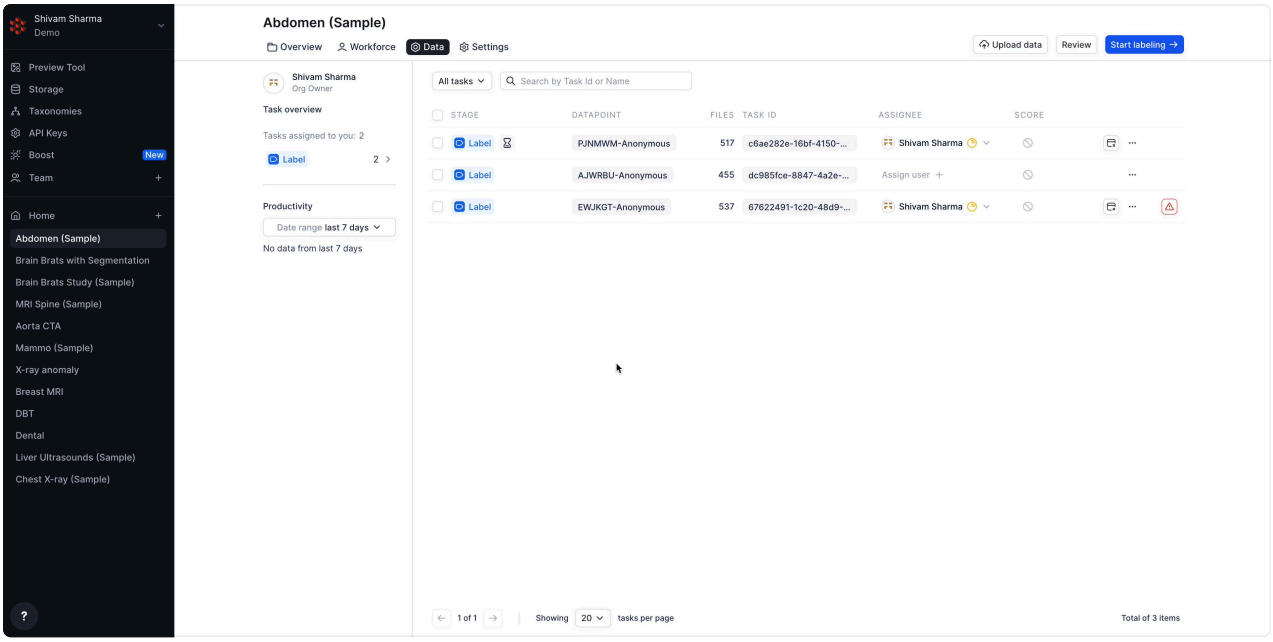
Raise issue is a tool that allows annotators to *remove a task from their* queue and ask Project Admins questions or help resolve the problem.



Once an annotator raises an issue, they will not be able to access the task until the admin resolves it and sends it back to the annotator.

Annotators can raise an issue from the labeling tool. Click on **skip task,** then **raise issue.**

When an Admin opens an issue task, they will be able to see it in "view mode," i.e., the annotations will not be editable. Admins can respond to comments left by the annotators and send the task back to the labeler.



Reference Standards

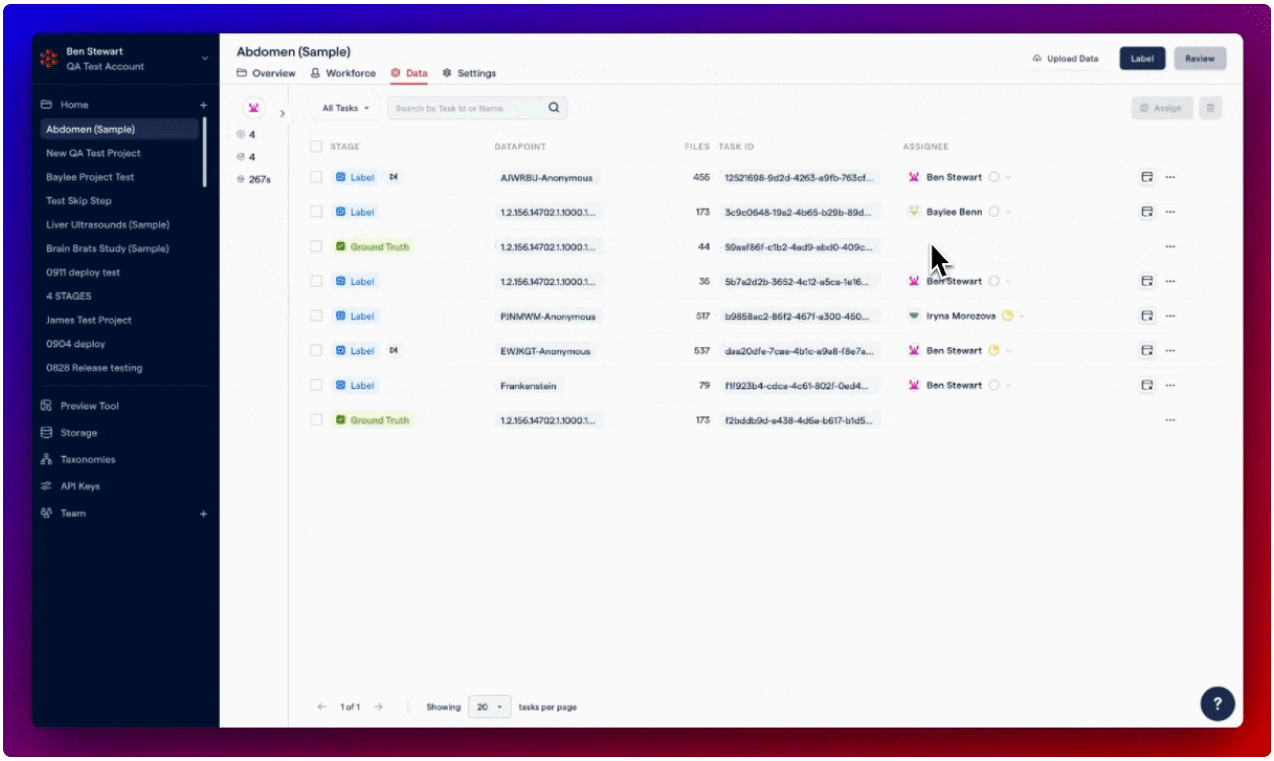
RedBrick AI allows you to designate Tasks in the Ground Truth Stage as Reference Standards, making them visible to all members of a Project's Workforce.

Reference Standards make it easier than ever to provide your team with an accessible, interactive version of the annotation work to be done, as well as communicate instructions and expectations to labelers.

Setting a Task as a Reference Standard

Project Admins (and above) have the ability to set a Ground Truth Task as a Reference Standard.

To set a Ground Truth Task as a Reference Standard, simply click on the hamburger menu on the right hand side of the data page and select **Set as reference standard**.



Setting a Ground Truth Task as a Reference Standard (Admin)

The Task will then become visible to all members of the Project.

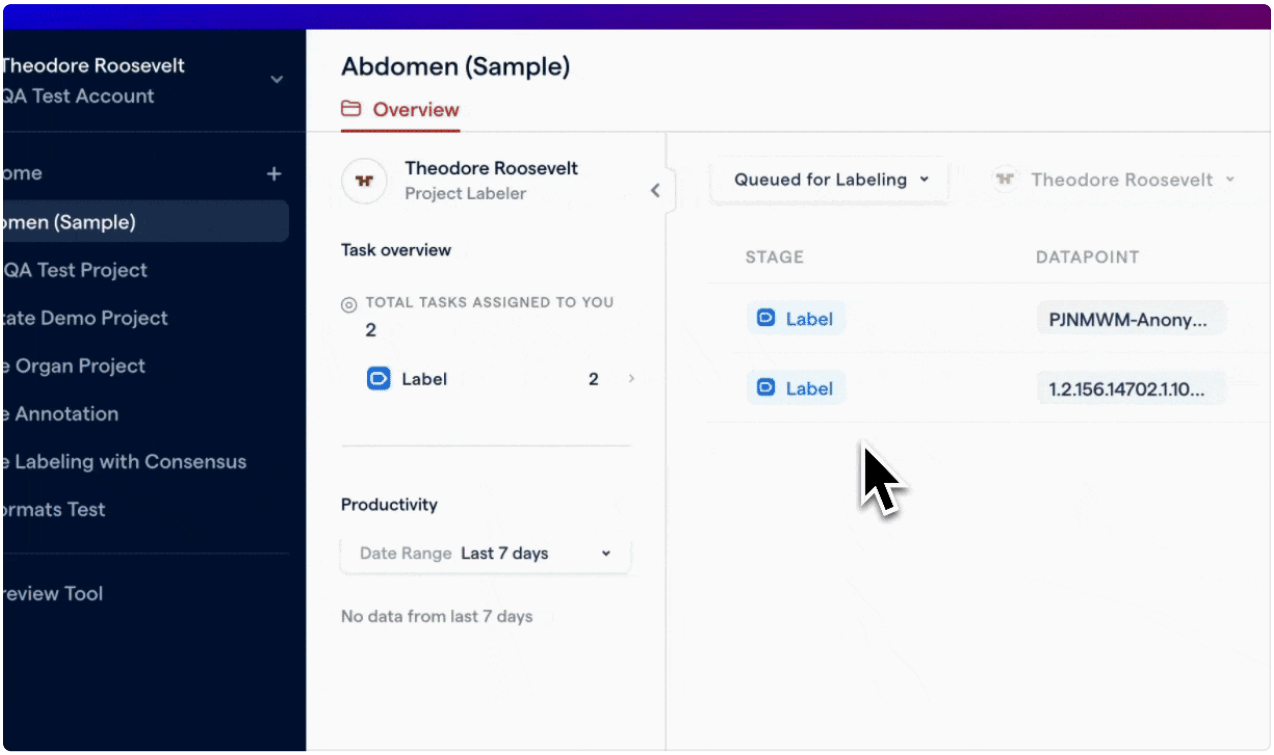
Admins may manipulate a Reference Standard just as they would any other Ground Truth Task (i.e. remove "Reference Standard" designation, edit, move back to Label Stage, etc.).

Project Members can view a Reference Standard, but they cannot modify it.

> There are currently no SDK methods available to create, upload, or modify a Reference Standard. Be on the lookout for improvements coming soon!

Viewing a Reference Standard as a Project Member

To view a Reference Standard as a Project Member, simply filter your Task view by **Reference Standards** and click on **Open in Editor**.



Viewing a Project's Reference Standards as a Labeler

Webhooks

Webhooks allow you to receive an HTTP push notification triggered by certain events within a project. Currently, webhooks are triggered by the following events:

1. **Task created**: When a data point is uploaded, and a task is created from that data point.
2. **Task entered stage**: When a task enters a new stage in the labeling workflow.
3. **Task deleted**: When a task is deleted.

Task created

```
{
  "version": "v1.0",
  "events": 1,
  "payload": [
    {
      "event": "TASK_CREATED",
      "id": "...",
      "timestamp": ...,
      "data": {
        "orgId": "...",
        "projectId": "...",
        "taskId": "...",
        "taskName": "...",
        "updatedBy": "..."
      }
    }
  ]
}
```

Task entered stage

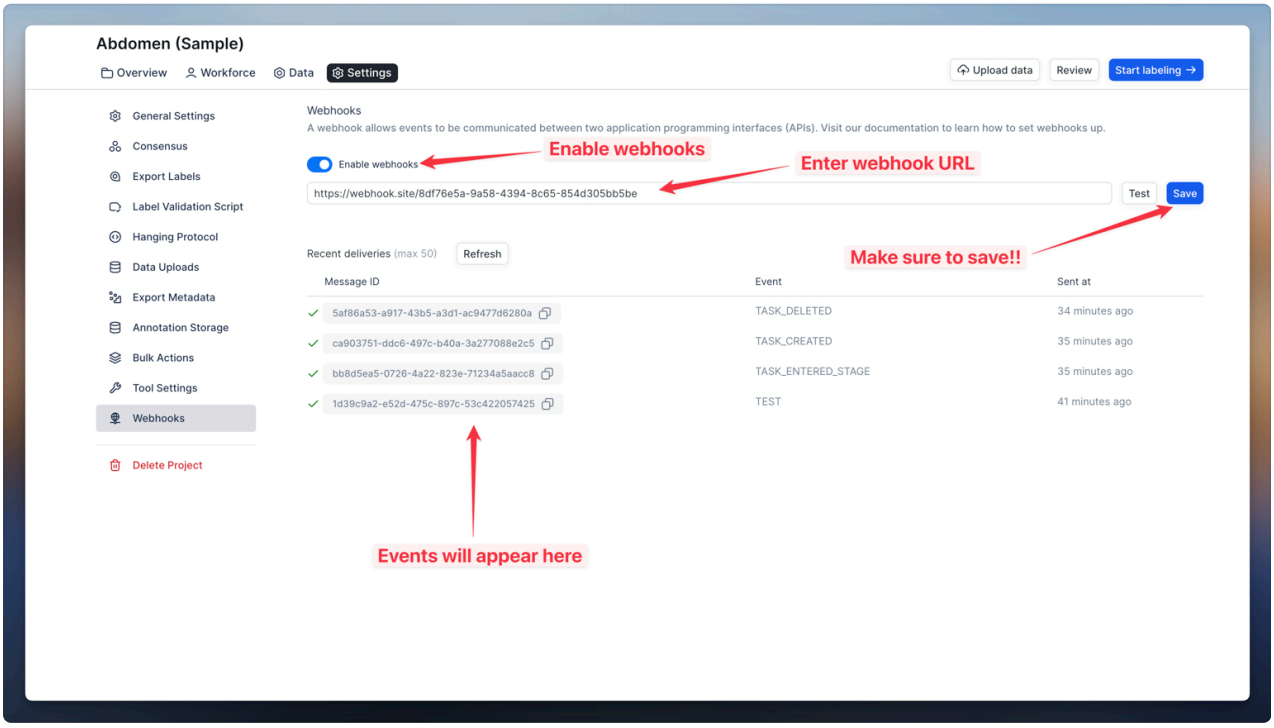
```
{
  "version": "v1.0",
  "events": 1,
  "payload": [
    {
      "event": "TASK_ENTERED_STAGE",
      "id": "...",
      "timestamp": ...,
      "data": {
        "orgId": "...",
        "projectId": "...",
        "taskId": "...",
        "stageName": "...",
        "updatedBy": "..."
      }
    }
  ]
}
```

Task deleted

```
{
  "version": "v1.0",
  "events": 1,
  "payload": [
    {
      "event": "TASK_DELETED",
      "id": "...",
      "timestamp": ...,
      "data": {
        "orgId": "...",
        "projectId": "...",
        "taskId": "...",
        "taskName": "...",
        "updatedBy": "..."
      }
    }
  ]
}
```

Using webhooks

Configure webhook from project settings, as shown in the image below.

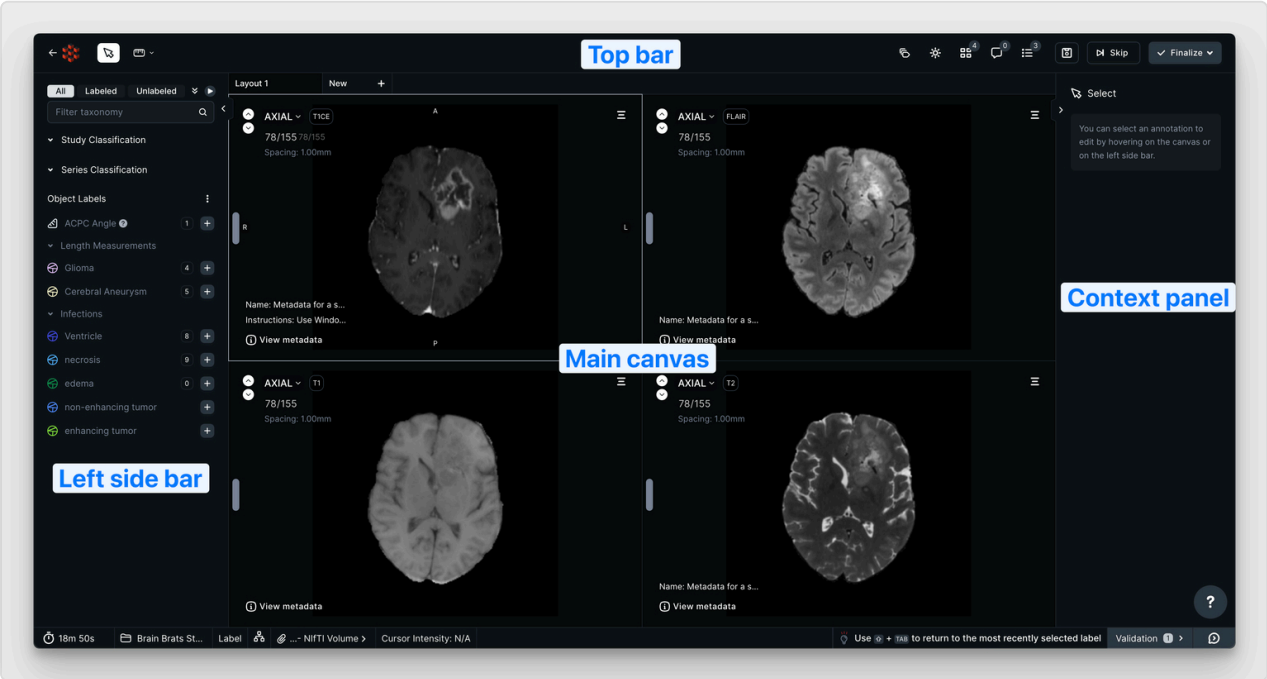


You can use tools like https://webhook.site/ to test the webhook and inspect the response format.

Annotation & viewer

Viewer Basics

RedBrick AI is designed for native medical image viewing and annotation, supporting all radiology modalities. The platform supports X-ray, CT, MRI, Ultrasound, and other 2D, 3D, and video modalities.



There are 4 main components to the annotation interface. We will refer to each of these components throughout the documentation.

1. **Left sidebar** is where you create, edit, and interact with annotations and attributes.
2. **Top bar** contains Task-level actions such as task submission and saving and the environment settings (Windowing, Layout), Version Explorer, Segmentation Tools, and Quick Measurement Tools.
3. **Context panel** shows additional information and settings for any currently selected tool(s).
4. **Main canvas** is where you interact with your images and apply your annotations to your images/volumes.

Viewing and navigating through your volume

Interacting with your images in RedBrick AI is similar to other medical imaging & PACS viewers. This section covers the basic shortcuts and functions for navigating through a volume.

> You can find key shortcuts by clicking on the (?) Help button on the bottom right.

Changing slices

1. **Scroll:** Scroll over any of the viewports.
2. **UI:** Use the viewport's slider or up/down slice buttons.
3. **Shortcut:** Use the `up arrow` and `down arrow` keyboard shortcuts on the selected viewport.
4. **Quick slice change:** Hold `alt` / `option` and `left click` `drag` to quickly change slices.

Zoom and pan

1. **Zoom:** Hold `ctrl` and `scroll` to zoom.
2. **Quick zoom:** Hold `ctrl` and `right click` `drag` for a quick zoom.
3. **Pan:** Hold `shift` and `left click` `drag` to pan.

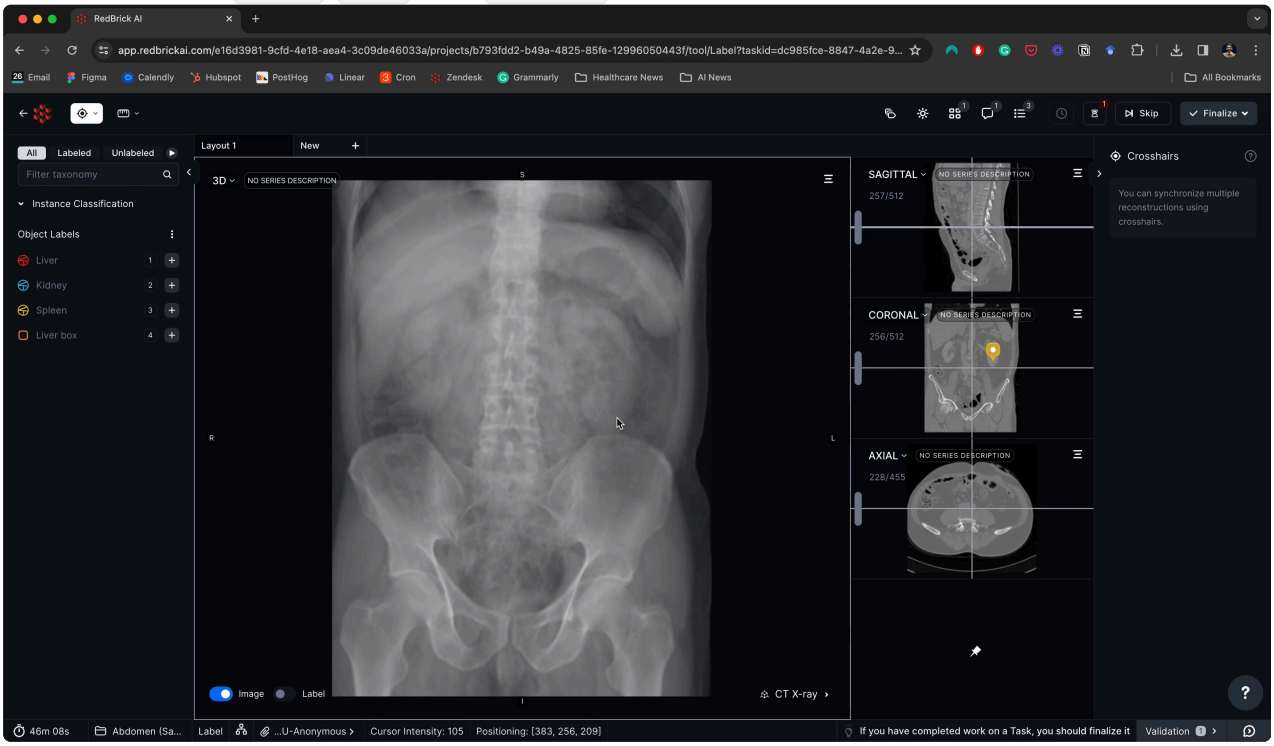
Windowing

1. **UI:** Adjust windowing on the context panel by activating it from the top bar.
2. **Shortcut:** Hold `ctrl` and `left click` `drag` vertically to adjust the level and horizontally to adjust the width.

3D model viewing

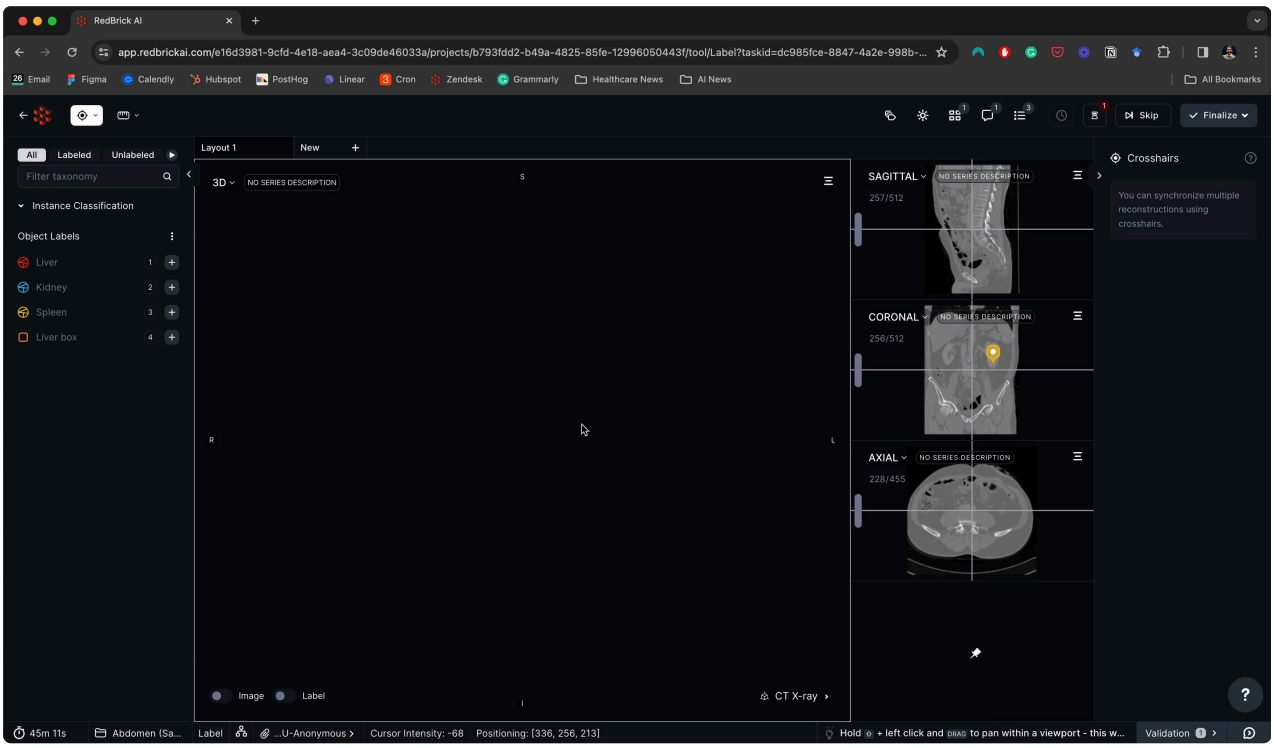
1. **Rotate**: `Left click` `drag` to rotate the volume in 3D.

2. **Rotate fixed plane**: Hold `ctrl` and `left click` `drag` to rotate the model fixed in the plane.
3. **Pan**: Hold `shift` and `left click` `drag` to pan.
4. **Zoom**: Hold `ctrl` / `cmd` and `scroll` to zoom.



Basic viewing interactions.

1. **Rendering preset**: adjusts the photorealistic rendering style.
2. **Shift**: adjusts the volume rendering transfer function to modify the rendering.
3. **Maximum opacity:** sets the maximum voxel opacity of the rendered image.



Basic windowing functions.

Crosshairs and oblique plane

Crosshairs will synchronize multiple projections of a single volume. Oblique planes allow you to view a non-orthogonal view of a volume.

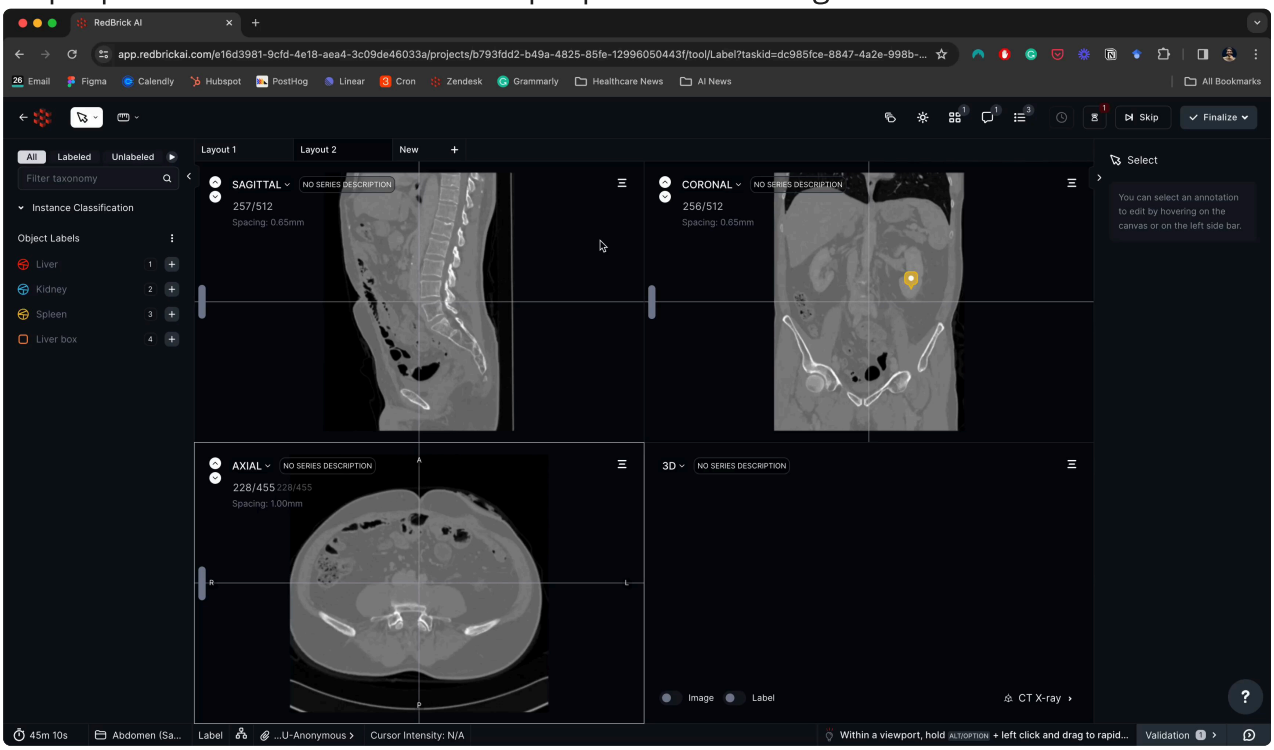
> Crosshairs will only be available on 3D modalities. Also, you must have multiple orthogonal projections in your viewport (for example, Axial and Sagittal) for cross-hairs to appear.

Crosshairs

1. **Activate**: From the top bar or by pressing `c`.
2. **Deactivate:** From the top bar or by pressing `esc`. By default, deactivated crosshairs will be shown on the canvas and can be reactivated by selecting them. To hide deactivated cross-hairs press `cmd/ctrl` `shift` `c`.

Oblique plane

1. **Activate**: `Right click` on any viewport, then select `activate oblique`. This will enable an oblique plane for just the selected projection.
2. **Usage**: The oblique plane crosshair for each viewport will be color-coded. For example, the Sagittal oblique plane is purple in the video below. Rotating the purple crosshair creates an oblique plane on the Sagittal view.



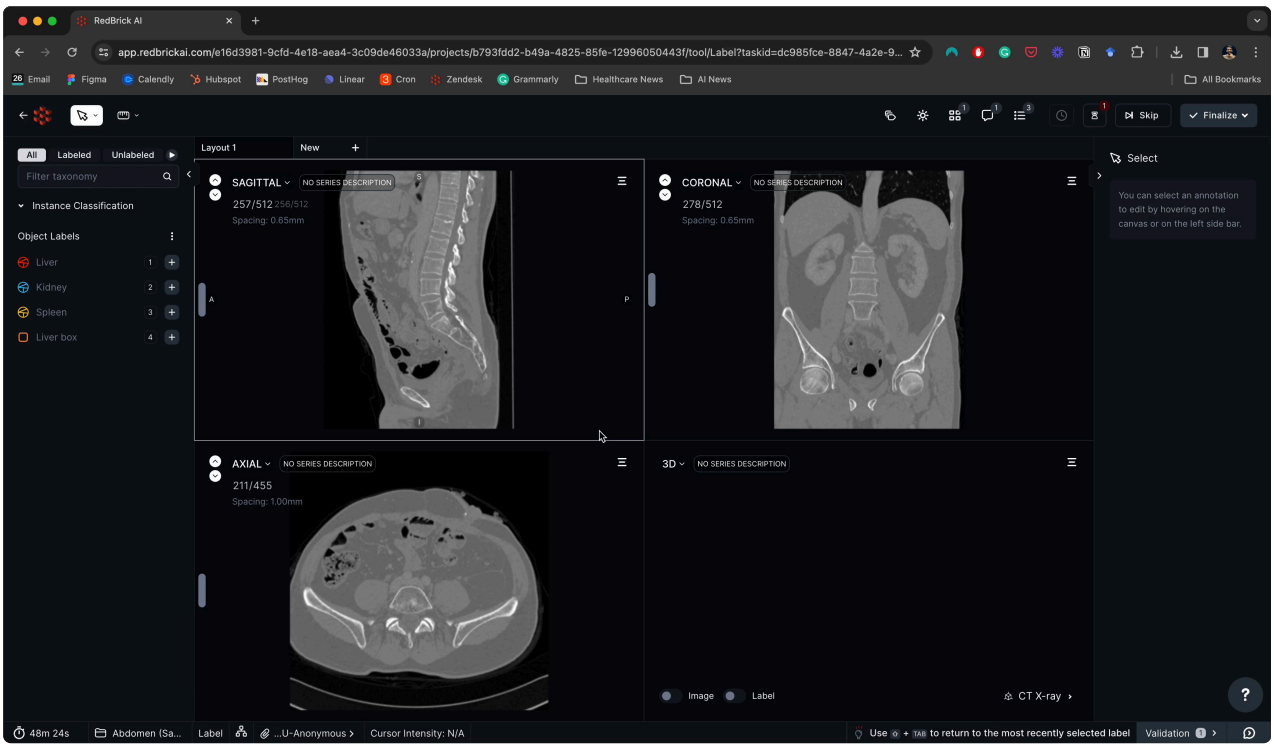
Using crosshairs and oblique planes.

Maximum and minimum intensity projection (MIP)

Maximum Intensity Projection (MIP) displays the highest intensity values in a 3D image along a viewing axis, useful for highlighting bright structures like blood vessels. Minimum Intensity Projection shows the lowest values, useful for revealing dark structures like airways.

> You can only view MIP along the imaging axis for any volume.

To **activate,** change the displayed view to MIP in any viewport using the view dropdown on the top left.



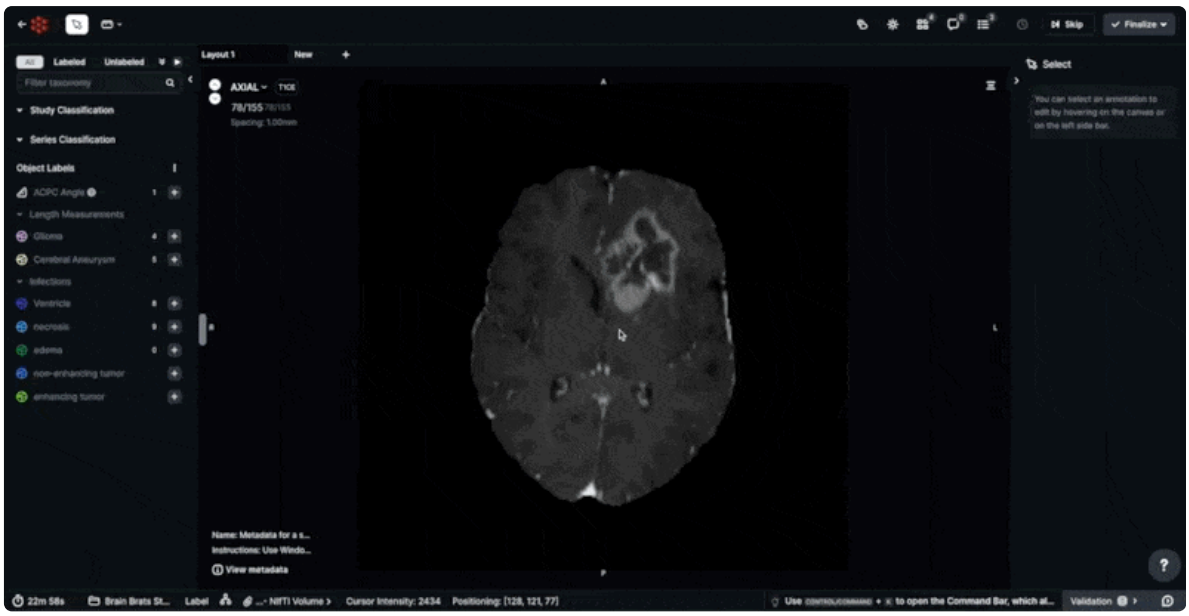
Displaying MIP.

Managing your layout

RedBrick AI's viewer is flexible in how it displays series. You can customize the layout manually or use hanging protocols to display single or multiple series.

Changing layout and displaying series

1. **Change layout grid:** Each modality has a default layout grid which can be modified from the layout grid on the top bar. The viewer supports everything between 1x1 and 3x3, showing a maximum of 9 views.
2. **Displaying series:** You can customize what is shown in each viewport; this can be any 2D or 3D view.
    - **Drag and drop:** You can drag and drop any series or projection from the layout context panel to a target viewport.



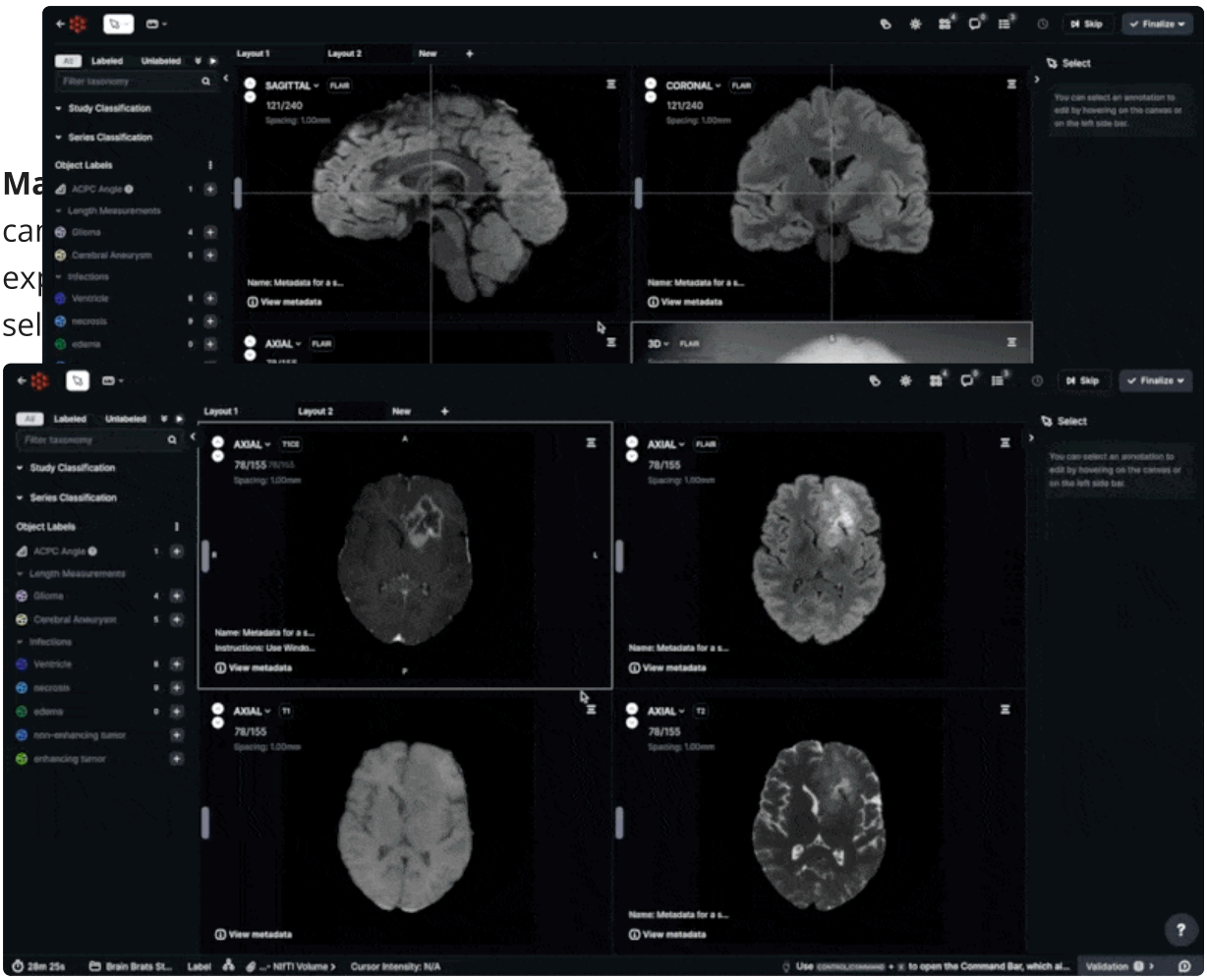
Drag and drop.

- **Viewport selector:** You can use the dropdown on any viewport to cycle between projections of that series.

Viewport selector.

- **Quick change:** You can hover over the thumbnails on the layout context panel to directly place the view in the corresponding layout position.

3. **Ma**
ca
ex
sel



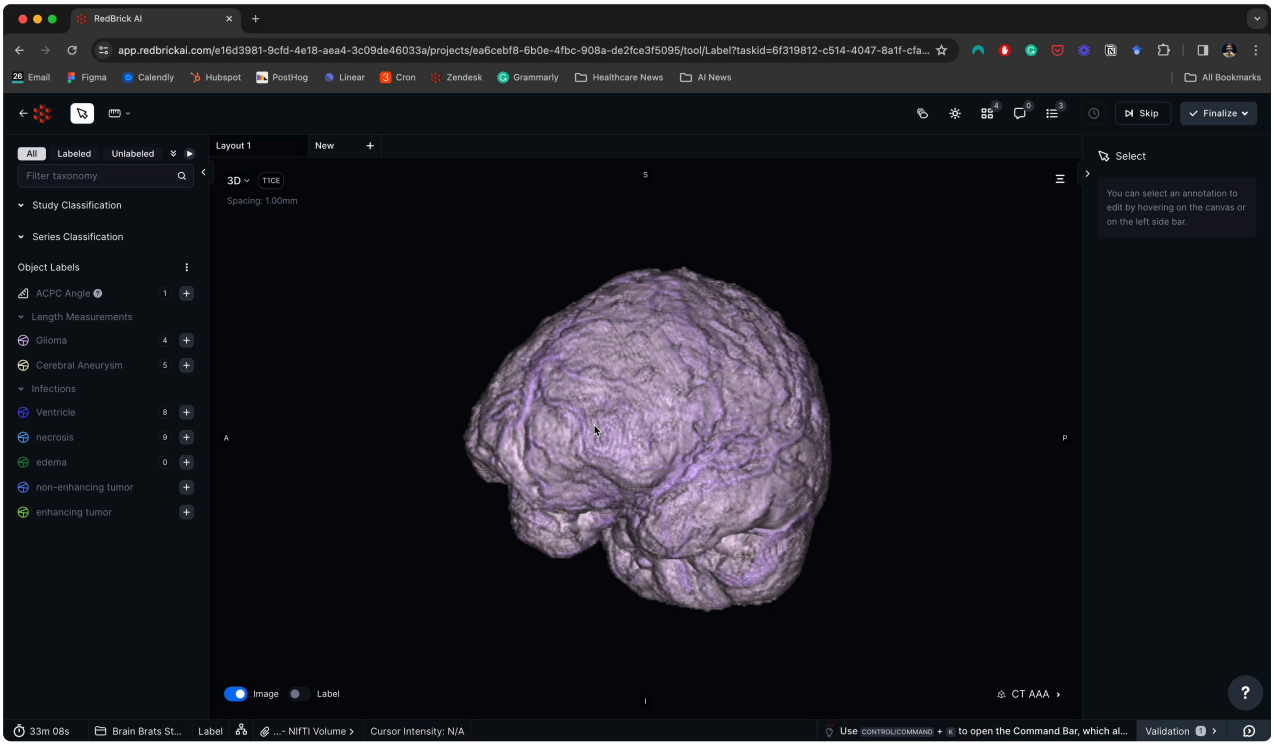
4. **Full-screen mode:** Press `f` to enter full-screen mode for distraction-free annotation.

Multiplanar reconstruction (MPR)

1. **Manually:** You can create an MPR view by manually configuring the viewport and selecting the projections of your series following the instructions above.
2. **Right-click menu:** You can also display an MPR view by `right click` on the viewport and selecting `MPR layout`. This will create a new layout tab with the MPR view.

Creating multiple layout tabs

Often you may want to move between two pre-set viewing configurations. For example, between a large 3D view and a view of all projections - Sagittal, Coronal, and Axial. This can be accomplished by creating multiple layout tabs.



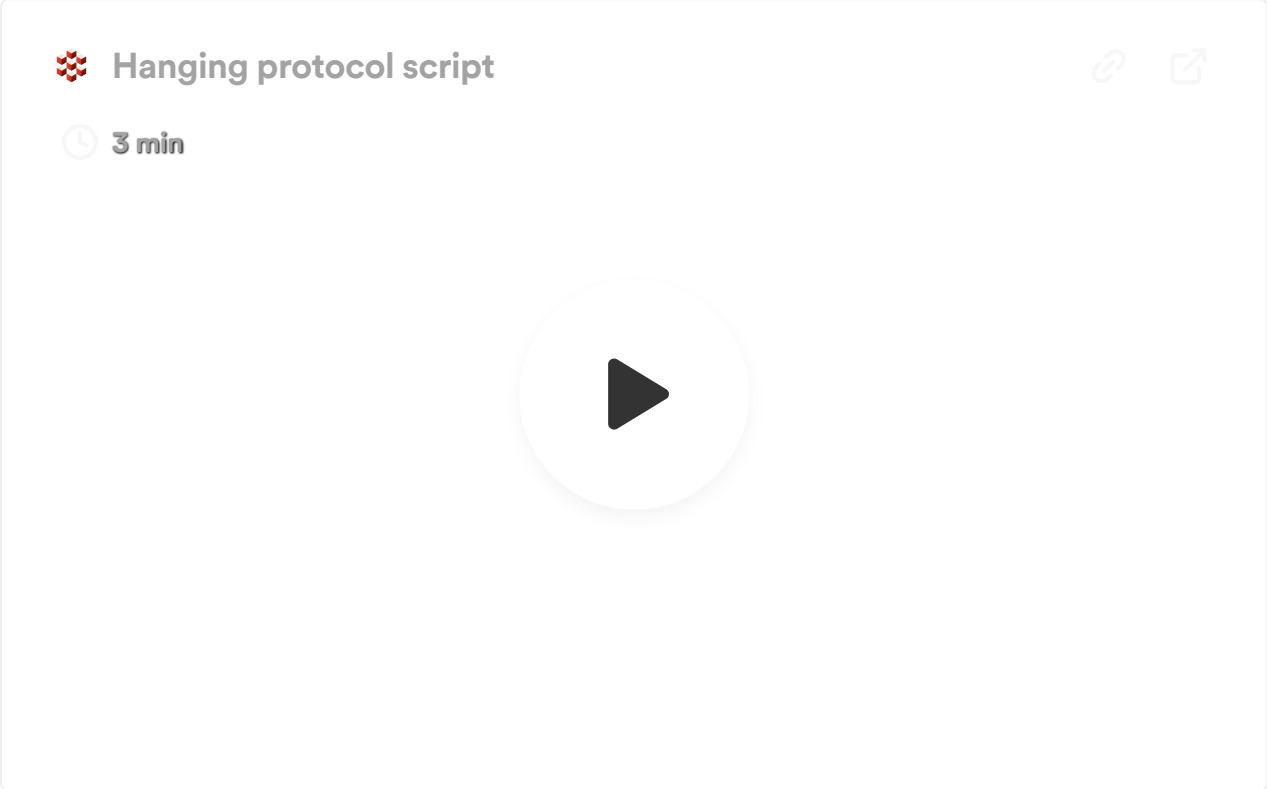
Creating multiple layout tabs

Custom Hanging Protocol

Write a script to dynamically arrange the viewports for your project

The Custom Hanging Protocol feature allows you to write a script that will programmatically define the visual layout of your Annotation Tool **at the Project level**.

Pre-configuring parameters such as Windowing settings, Thresholding settings, the number of viewports in a Layout Tab, which views display by default, etc., is both an easy way to save time for your annotators and makes for a much smoother overall annotation experience.



Script Usage Guide

This guide provides an overview of the available functions and types to help you effectively manage these settings. At present, you can control the following:

- The dimensions of a Layout Tab (`setDimensions`):
  -

    - **REQUIRED:** the number of columns in a Layout Tab (`numColumns`)
    - **REQUIRED:** the number of rows in a Layout Tab (`numRows`)

- The contents of each viewport in a Layout Tab (`setViews`):
    - **REQUIRED:** an array describing each viewport's content (`views`)
    - **REQUIRED:** Which series to show (`seriesIndex`)
    - **REQUIRED:** Which way to view the series (`plane`)
    - Flip the view horizontally (`flippedHorizontally`)
    - Flip the view vertically (`flippedVertically`)
    - Activate Intellisync (`synchronized`)
    - Maximize a single viewport in a Layout Tab (`expanded`)

- The default Windowing setting for each Series (`setWindowing`):
    - **REQUIRED:** the number of the Series (`seriesIndex`)
    - **REQUIRED:** the desired Windowing Level (`level`)
    - **REQUIRED:** the desired Windowing Width (`width`)

- The default Thresholding setting for each Series (`setThresholding`):
    - **REQUIRED:** the number of the Series (`seriesIndex`)
    - **REQUIRED:** the lower limit of the Thresholding range (`min`)
    - **REQUIRED:** the upper limit of the Thresholding range (`max`)

- Create and configure a new Layout Tab in your Task (`nextTab`)
- Configuration settings for the Annotation Tool (`setSegmentationSettings`)*
    - **(Note: this function has been replaced by the Tool Settings page)**

> The Custom Hanging Protocol script takes the available Series for a particular Task as input and returns the layout dimensions and list of views to display.

Custom Hanging Protocol Format Reference

```
function setViews(views: View[]) {
 //...
}
function setDimensions(numColumns: number, numRows: number) {
 // ...
}
function setWindowing(seriesIndex: number, level: number, width: number){
 // ...
}
function setThresholding(seriesIndex: number, min: number, max: number) {
 // ...
}

function nextTab()


// this function has been replaced by the Tool Settings page of Project S
function setSegmentationSettings(
  [
    {
      toolName: ToolOptions;
      enabled: boolean;
      modes?: ToolModes[];
      defaultMode?: ToolModes;
      defaultTool?: boolean;
    }
  ]
);

// When a user uploads a Task and enables Hanging Protocols,
// the hangingProtocol() function takes Series[] and the following parame
interface Series {
  seriesIndex: number;
  is2DImage: boolean;
  isVideo: boolean;
  numFrames: number;
  name: string; // User defined name if available, else "A", "B", ...
  imagingAxis: 'AXIAL' | 'SAGITTAL' | 'CORONAL';
}

interface View {
  plane: 'AXIAL' | 'SAGITTAL' | 'CORONAL' | '3D' | 'MIP';
  seriesIndex: number;
  flippedHorizontally?: boolean;
  flippedVertically?: boolean;
  synchronized?: boolean;
  expanded?: boolean; // Only applicable to a single view in a given Layo
}
```

Examples

Default Script

This default script uses some defined macros to make setting the view easier.

```
function hangingProtocol(allSeries: Series[]) {
  // This is the default layout script
  if (allSeries.length > 1) {
    setMultiSeries();
  } else if (allSeries[0].is2DImage) {
    setSingleView();
  } else {
    setMPR();
  }
}
```

Set Single View

```
function setSingleView(seriesIndex=0) {
  setDimensions(1,1);
  setViews([
    {
      plane: allSeries[seriesIndex].imagingAxis,
      seriesIndex: seriesIndex,
    }
  ]);
}
```

Set Multi-Series Layout

This script sets each Series as a single viewport that is viewed on the imaging axis.

```
function setMultiSeries() {
  function singleSeries(series_, index) {
    return {
      plane: series_.imagingAxis,
      seriesIndex: index,
    };
  }
  let views = allSeries.map(singleSeries);

  setViews(views.slice(0,9));
}
```

Set Multi-Planar Reconstruction

```
function setMPR(seriesIndex=0) {
  let targetSeries = allSeries[seriesIndex];
  setDimensions(2,2);
  setViews([
    {
      plane: 'SAGITTAL',
      seriesIndex: seriesIndex,
      expanded: targetSeries.imagingAxis === 'SAGITTAL',
    },
    {
      plane: 'CORONAL',
      seriesIndex: seriesIndex,
      expanded: targetSeries.imagingAxis === 'CORONAL',
    },
    {
      plane: 'AXIAL',
      seriesIndex: seriesIndex,
      expanded: targetSeries.imagingAxis === 'AXIAL',
    },
    {
      plane: '3D',
      seriesIndex: seriesIndex,
    }
  ]);
}
```

Set and Configure Multiple Layout Tabs

The following script creates 2 Layout Tabs, each containing 2 Series.

```
if (allSeries.length === 4) { // executes when there are 4 total Series i
   setDimensions(2, 1); // set 2x1 layout for Layout Tab 1
   setViews( // adding the first and second image/volume to Layout Tab 1
   [
      {
         seriesIndex: 0,
         plane: 'SAGITTAL'
      },
      {
         seriesIndex: 1,
         plane: 'SAGITTAL'
      }
   ]
);
   nextTab(); // create and configure Layout Tab 2
   setDimensions(2, 1); // set 2x1 layout for Layout Tab 2
   setViews( // add third and fourth image/volume to Layout Tab 2
   [
      {
         seriesIndex: 2,
         plane: 'SAGITTAL'
      },
      {
         seriesIndex: 3,
         plane: 'SAGITTAL'
      }
   ]
 )
```

Synchronize Views

Hanging protocols can be used along side Intellisyncfor ease of use when annotating scans in a study.

For example, let's assume that we have uploaded a single Task containing 4 Series from an MRI study: T1, T1CE, T2, and Flair weighted MR scans.

After we enable Hanging Protocols, the `hangingProtocol()` function will take the 4 Series as an input and parse them in the following way:

```
[
    {
        seriesIndex: 0,
        is2DImage: false,
        isVideo: false,
        numFrames: 1,
        name: 'T1',
        imagingAxis: 'SAGITTAL',
    },
    {
        seriesIndex: 1,
        is2DImage: false,
        isVideo: false,
        numFrames: 1,
        name: 'T2',
        imagingAxis: 'SAGITTAL',
    },
    {
        seriesIndex: 2,
        is2DImage: false,
        isVideo: false,
        numFrames: 1,
        name: 'T1CE',
        imagingAxis: 'SAGITTAL',
    },
    {
        seriesIndex: 3,
        is2DImage: false,
        isVideo: false,
        numFrames: 1,
        name: 'Flair',
        imagingAxis: 'SAGITTAL',
    },
]
```

We can then use the information that has been parsed by the `hangingProtocol()` function to generate a script that sorts our views, displays the imaging axis and activates Intellisync.

```
// sort series by Name
let priorities = ['t1', 't1ce', 't2', 'flair'];
allSeries.sort((a, b)=>priorities.indexOf(a.name.toLowerCase()) - priorit

// display Series along the imaging axis
let imagingAxis = allSeries[0].imagingAxis;

// filter out views that were imaged in a different axis
let eligibleSeries = allSeries.filter((series) => series.imagingAxis ===

// Configure viewports
setViews(eligibleSeries.map((series) => {
  return {
    seriesIndex: series.seriesIndex,
    plane: series.imagingAxis,
    synchronized: true,
  };
}));
```

Tool Configuration with Hanging Protocols

> Configuring your Project's toolkit is now done on the Tool Settings page of your Project Settings.

Multiple Modalities

You can now combine multiple modalities such as 2D image, videos, and 3D scans and annotate them all together as a single study.

To create a study using multiple modalities, just upload the data together as a single task. You can upload as many DICOM or NIfTI series as you want. You can also include standard web image formats such as .jpg and .png however these will be treated as a single entity, either a single video clip or a single image, depending on the number of files.

Depending on which viewport you have selected and the type of data in that viewport, different label types will be available to create.

Available tools for 2D X-ray

Available tools for 3D CT

Intellisync

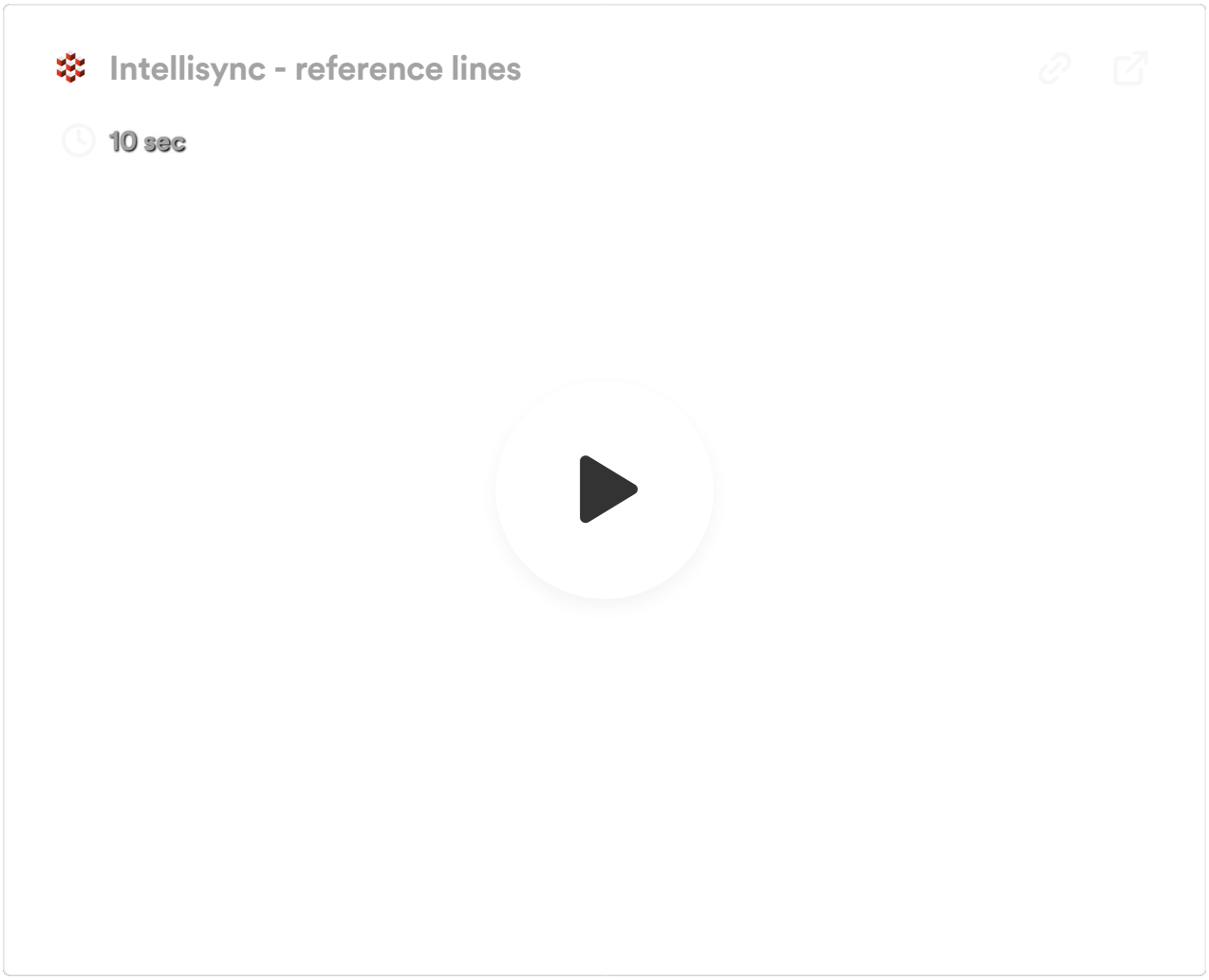
Smart synchronization between multiple volumes in a single study.

It is common to have multiple scans that are taken with different parameters in the same axis (i.e. T1 vs T2 MRI). Intellisync is a feature that synchronizes multiple series in different viewports to make annotation and diagnosis faster. For viewports that are aligned, scroll position will be synchronized. For viewports that are out of alignment, the intersection between the current instance and the other viewport is shown as a reference line.

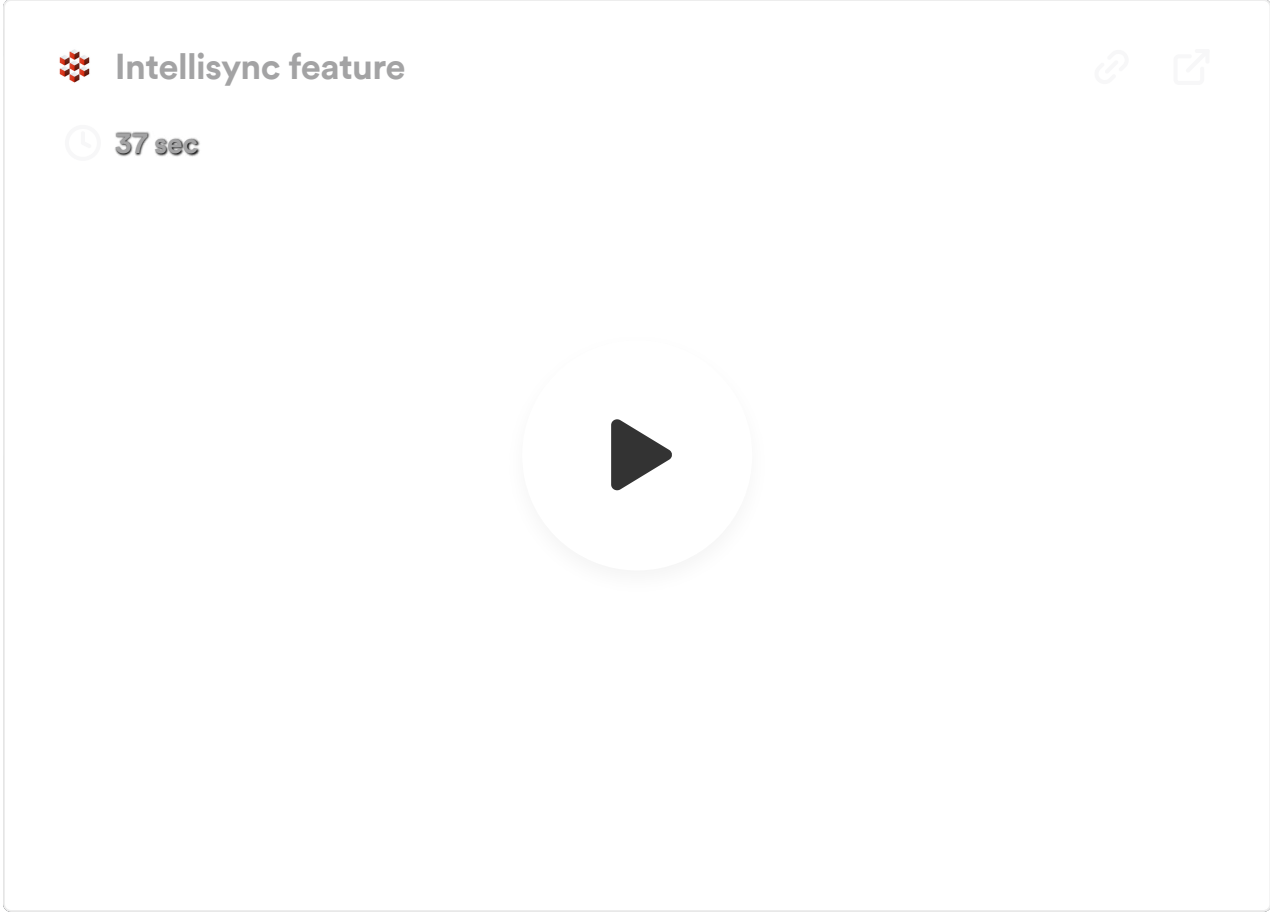
Activating Intellisync

When activated, intellisync applies to all 2D viewports of the chosen series. It can also be activated for all series at once.

1. Command bar: search for `Intellisync`
2. Keyboard (selected viewport): `Command + a` (Mac) or `Control + a` (Windows)
3. Keyboard (All series): `Shift + Command + a` (Mac) or `Shift + Control + a` (Windows)
4. Mouse: Through the action dropdown in the top right of the given viewport



Intellisync with orthogonal imaging axis



Intellisync with label mirroring and weighted MRIs

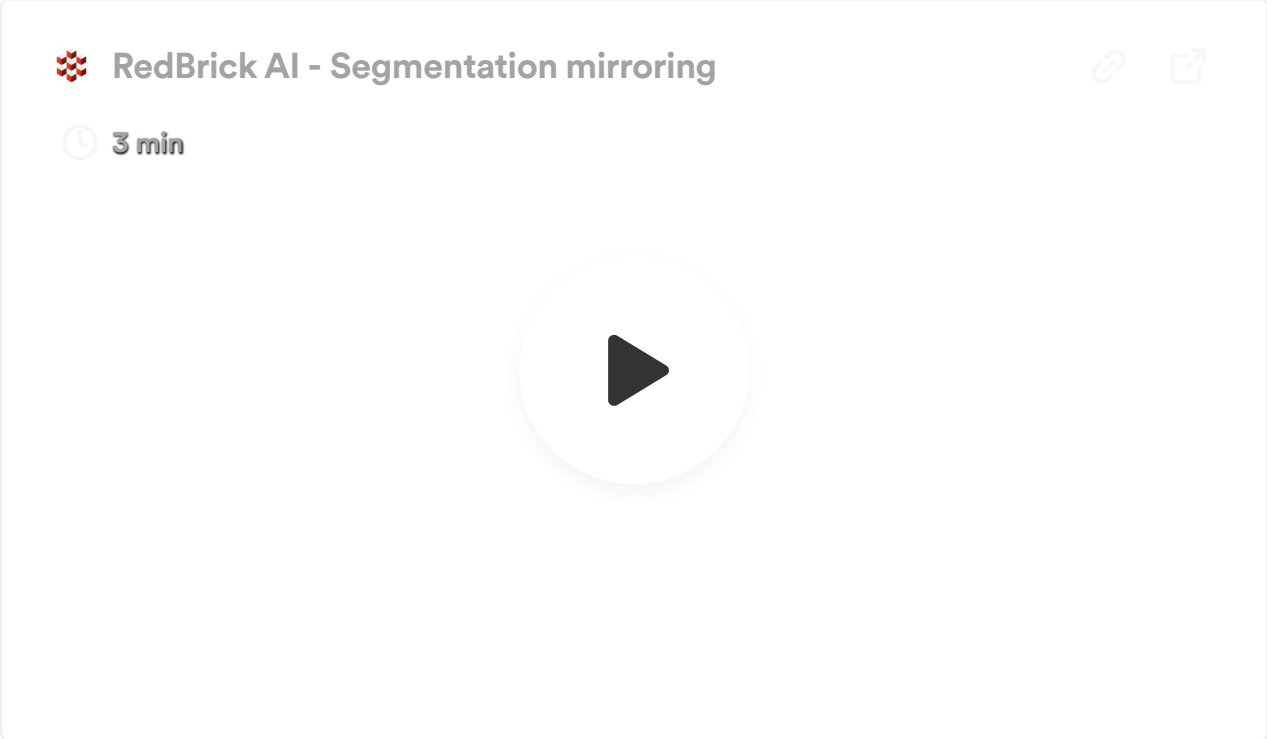
Troubleshooting and limitations

- Reference lines are computed based on image position patient and image orientation patient headers in each DICOM instance. Therefore, reference lines are only available for data sourced from DICOM files.
- For scroll syncing, the absolute world position is used. This means that data must be registered correctly. This feature will work best on data with identical headers (position, direction, dimensions, and orientation).
- For scroll syncing, eligibility for syncing is computed based on the viewing angle of the viewports. Therefore, two views that seem like they should sync because they both say "AXIAL" may not if their coordinate systems don't align well.

Annotation Mirroring

It is common to have multiple series that show the same anatomy with different scanner directions or scanner settings.

The segmentation mirroring feature allows you to "mirror" the segmentations from a single volume onto another volume. This allows you to compare and contrast information from different scans to modify the same segmentation file.



Segmentation mirroring demonstration

Activating and using

To activate, load up a task with more than one 3D series, and select "Mirror labels to all volumes." This option will not be available if your task only has a single series.

Activating label mirroring

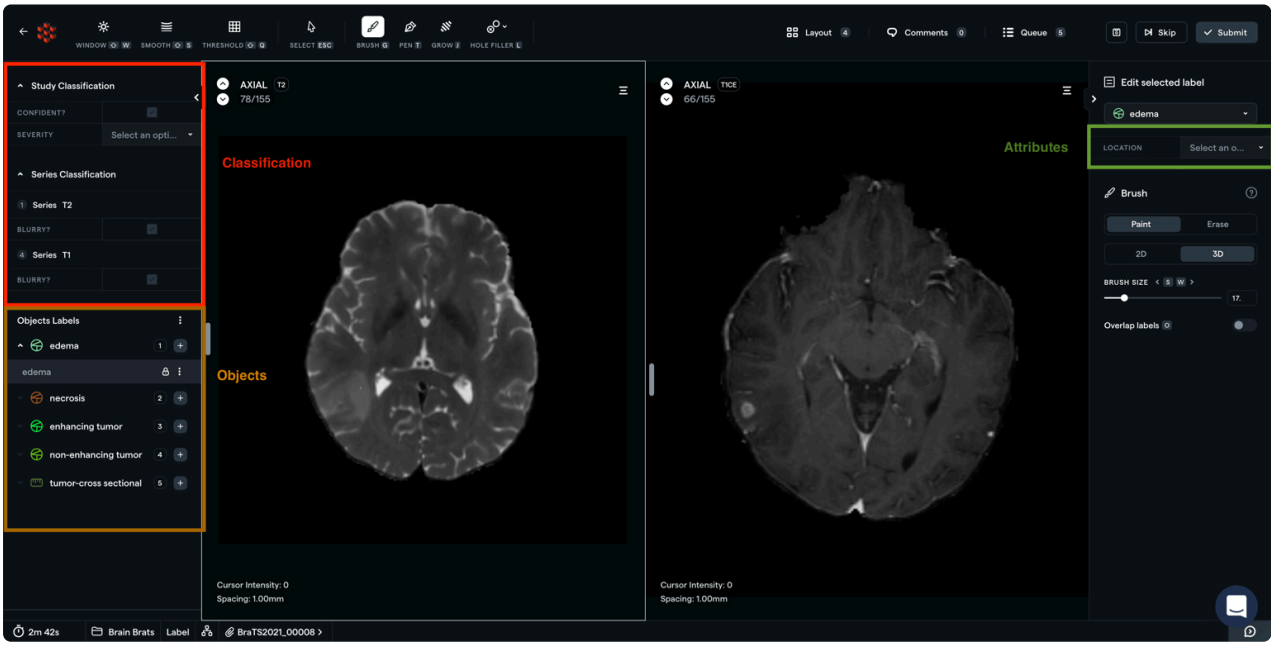
Once activated, you can now edit any segmentation labels from the series that you are mirroring in the same way as any other segmentation label.

Notes, warnings, and limitations

- This feature will only be effective if your data is properly registered. We use the header information of your DICOM and NIfTI files to align the data in three dimensions. If your series are not properly aligned, the mirroring will also be incorrectly aligned.
- While using label mirroring, 2D tools will not be available, only 3D variants.

Creating, Editing and Deleting Annotations

All of the Object Labels that you created inside of your Taxonomy will display in the the left side bar of the Annotation Tool. Depending on the elements you included in your Taxonomy, the left side bar will contain up to 4 sections: **Study Classification**, **Series Classification, Instance Classification** and **Object Labels.**

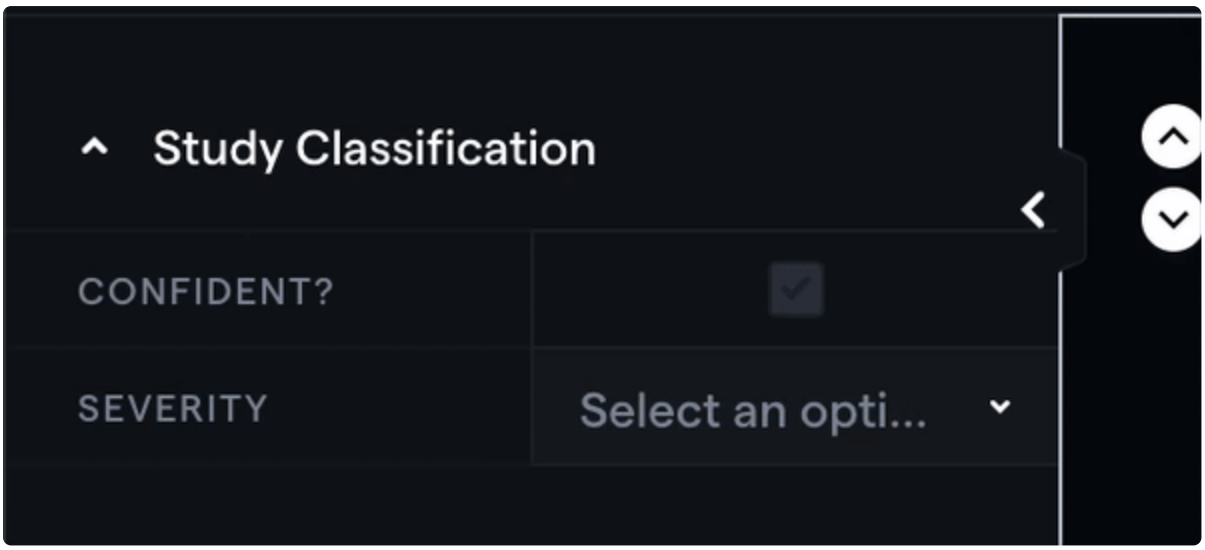


Study, Series and Instance Classification

If present, your Study, Series and Instance Classifications will be present under their own expansion panels in the lefthand toolbar.

Depending on the type (**Boolean**, **Text**, **Select**, or **Multiselect**), you can directly fill in the corresponding checkbox, select value, or textfield in the grid.

> Click and drag to adjust the size of the grid for easy interaction and viewing!



Object Labels

An Object Label has three components:

1. a **Name** (e.g. "Aortic Calcification")
2. a **Type** (e.g. "Segmentation", "Polygon", "Bounding Box", etc.)
3. **Attributes** ("Boolean", "Text", "Select" and "Multiselect", all of which are **optional**)

Creating Object Labels

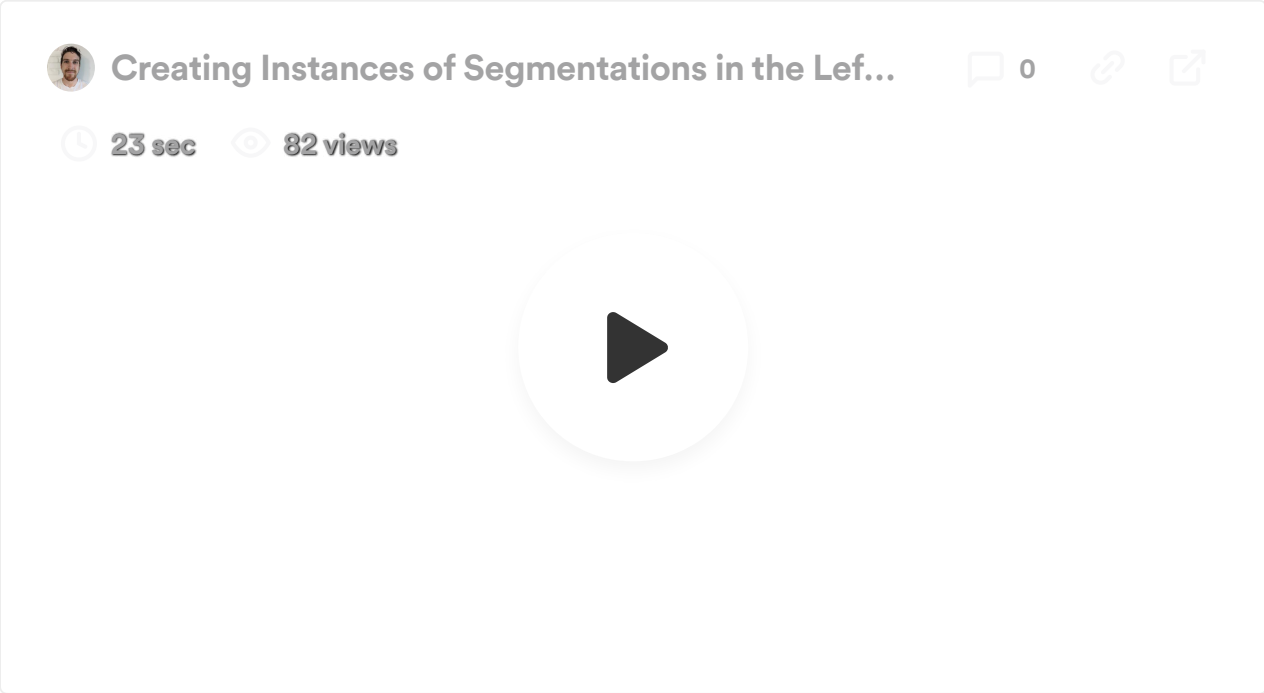
The left side bar will show all the Object Labels with an option to create *Entities of that Object Label*. There are two ways to create an Entity of an Object Label:

1. Click on the **"+"** button next to the corresponding Object Label;
2. Use the numeric hotkeys (e.g. 1, 2, 3, etc.) displayed next to the corresponding Object Label

When you create an Entity, the default tool for that Label type will be automatically selected (e.g. the **Brush Tool** will be selected by default when creating an Entity of a **Segmentation**).

- [x] Your default Segmentation Tool can be configured on the Tool Settings page within your Project Settings.

All Entities are organized within each Object Label's expansion panel.



A Quick Note on Annotation Mapping and Exports

When an annotation is created inside of the Annotation Tool, a corresponding `segmentMap` value is also generated to reflect the order in which the annotation was created.

In other words, when exporting a Task's annotations, the first annotation created by a labeler will have a `segmentMap` value of "1" in the accompanying JSON file; the second annotation will have a `segmentMap` value of "2", and so on. For more detailed information about how segmentations are mapped, please see our Format Reference for Exported Annotations.

The RedBrick AI SDK also supports both semantic export and exports of binary masks using the `export_tasks()` SDK method.

Selecting and Editing Object Labels

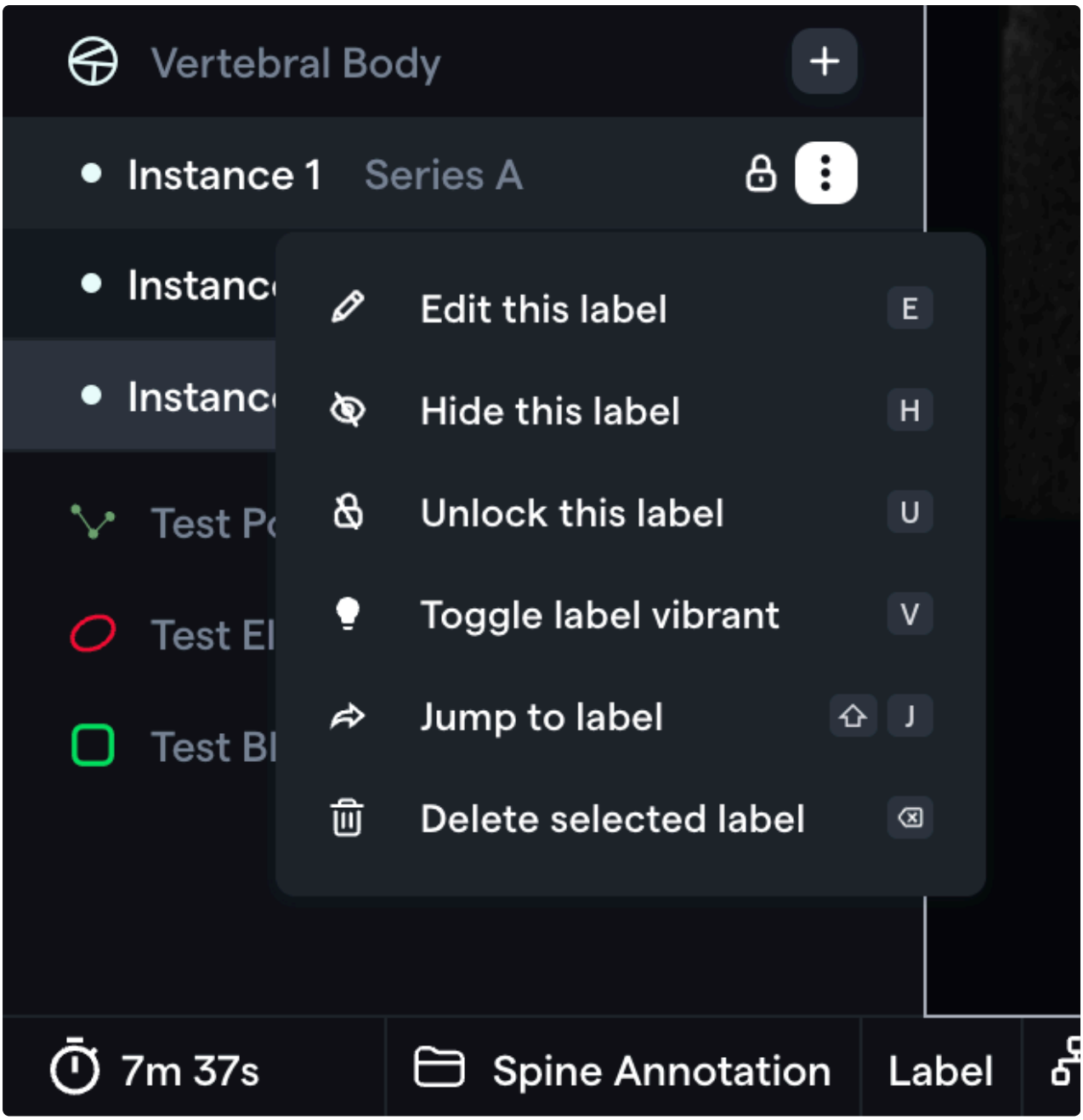
To edit an Object Label, you must first select it from the left sidebar. Once you select the Entity from the left side bar, the default tool for that label type will be selected automatically, and you can interact with the canvas to apply edits to that Entity.

> When a Object Label Entity is selected, all interactions with the canvas will only modify that particular Entity.

Other Object Label Actions

There are several actions available that are designed to make selecting, viewing, and editing Object Labels as easy as possible for labelers and reviewers.

All actions can be accessed by clicking the three-dot menu button on a Label Entity.

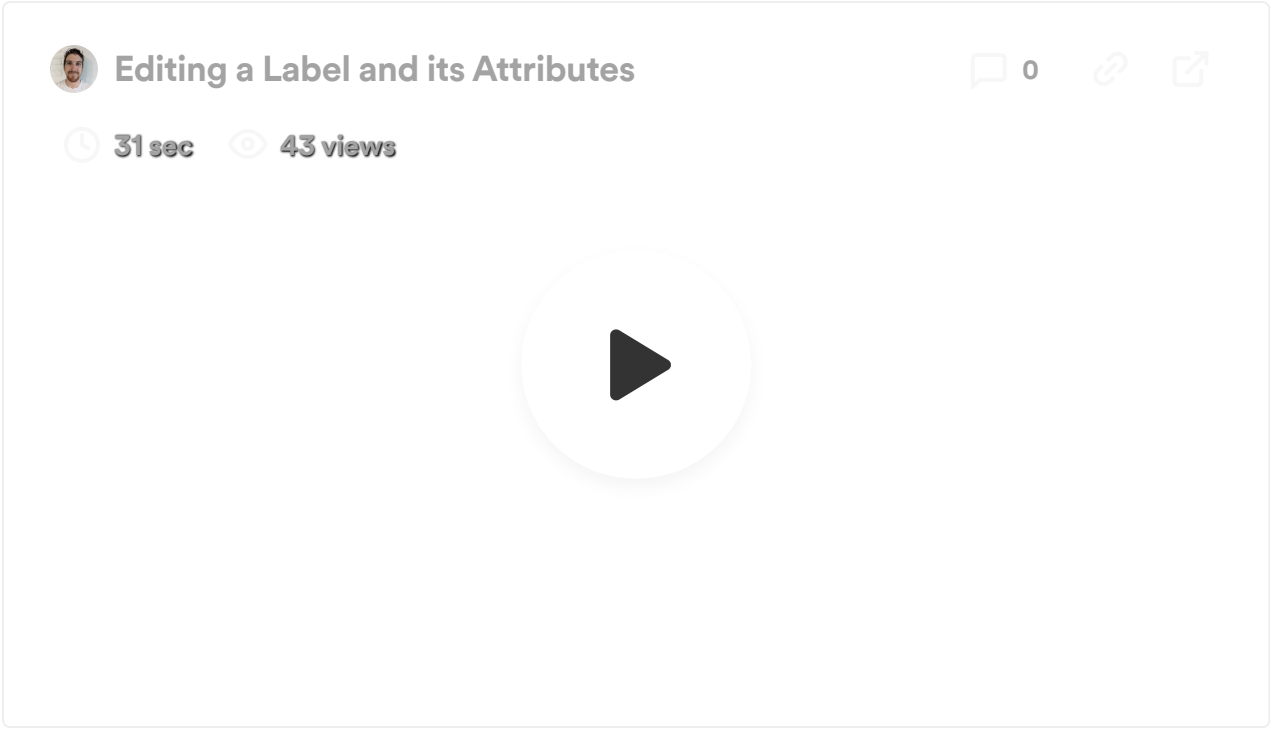


Editing a Selected Label and Attributes

You can quickly swap between existing Labels of the same Type (i.e. all of your Segmentation Object Labels, all of your Polygon Object Labels, all of your Bounding Boxes, etc.) in the right side context panel.

To reveal this menu in the right hand Context Panel, either click on the Object Label in the viewport or use the "Edit" label action.

With the Context Panel open, you can also add, modify, and delete Attributes for an Object Label.



Deleting Object Labels

You can delete the selected Object Label by using the `delete` / `backspace` hotkey. Alternatively, you can use the Delete Action in the actions dropdown.

To delete all Labels, use the shortcut `shift + delete` / `shift + backspace`.

> The Delete All Labels action **cannot be undone**. Please ensure that you wish to irrecoverably delete the most recent version of all of your labels before using the Delete All Labels action.

Alternatively, you can use the Delete All action in the Object Labels three-dot menu dropdown.

Hide and Show Labels

You can hide the label you are currently hovering over by using the `h` hotkey. If you are not hovering over a label (either on the canvas or on the left side bar), the `h` hotkey will hide/show the currently selected label.

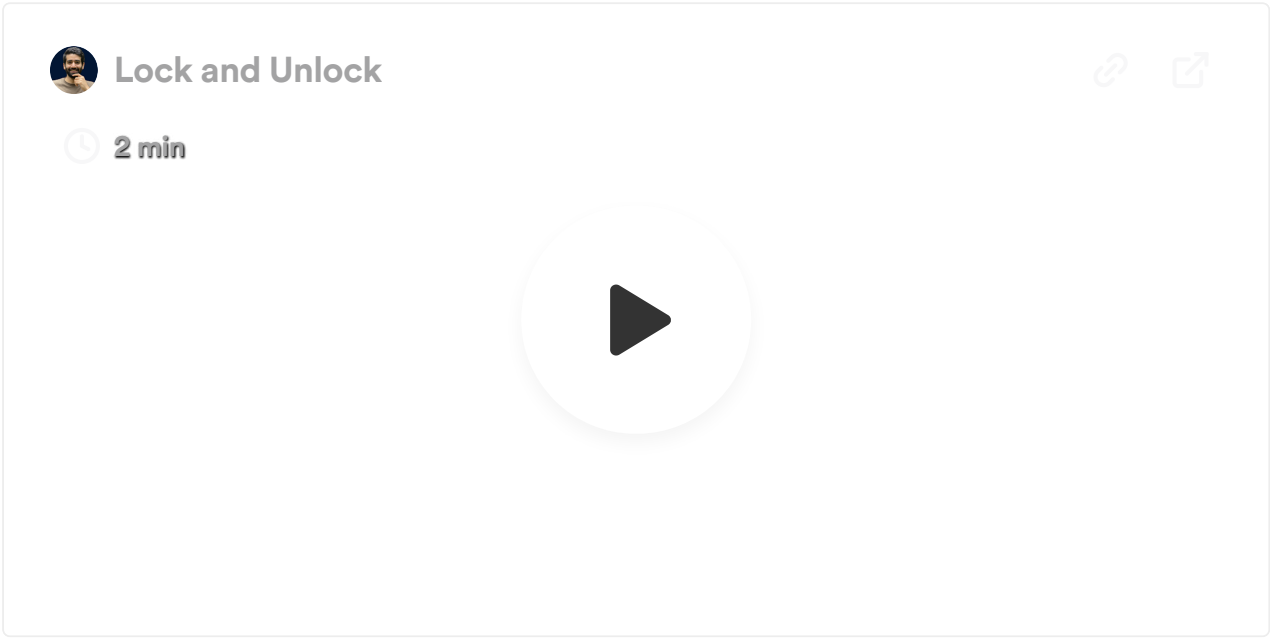
To hide/show all labels use the shortcut `shift + h` or the hide all action in the *Object Labels* three dot menu dropdown.

Lock and Unlock Labels

You can lock/unlock the label you are currently hovering over by using the `u` hotkey. If you are not hovering over a label (either on the canvas or in the left side bar), the `u` hotkey will lock/unlock the currently selected label.

To lock/unlock all labels, use the shortcut `shift + u` or the Lock All Action in the Object Labels three-dot menu dropdown.

Watch the video below to understand how to prevent/allow the overwriting of annotations.



Toggle Vibrant Mode

Vibrant Mode allows you to temporarily highlight a particular Entity. For example, if you have several small Entities of nodules in a chest CT, you can hover over any particular Entity on the left side bar or on the canvas and use the `v` shortcut to activate Vibrant Mode to highlight that Entity.

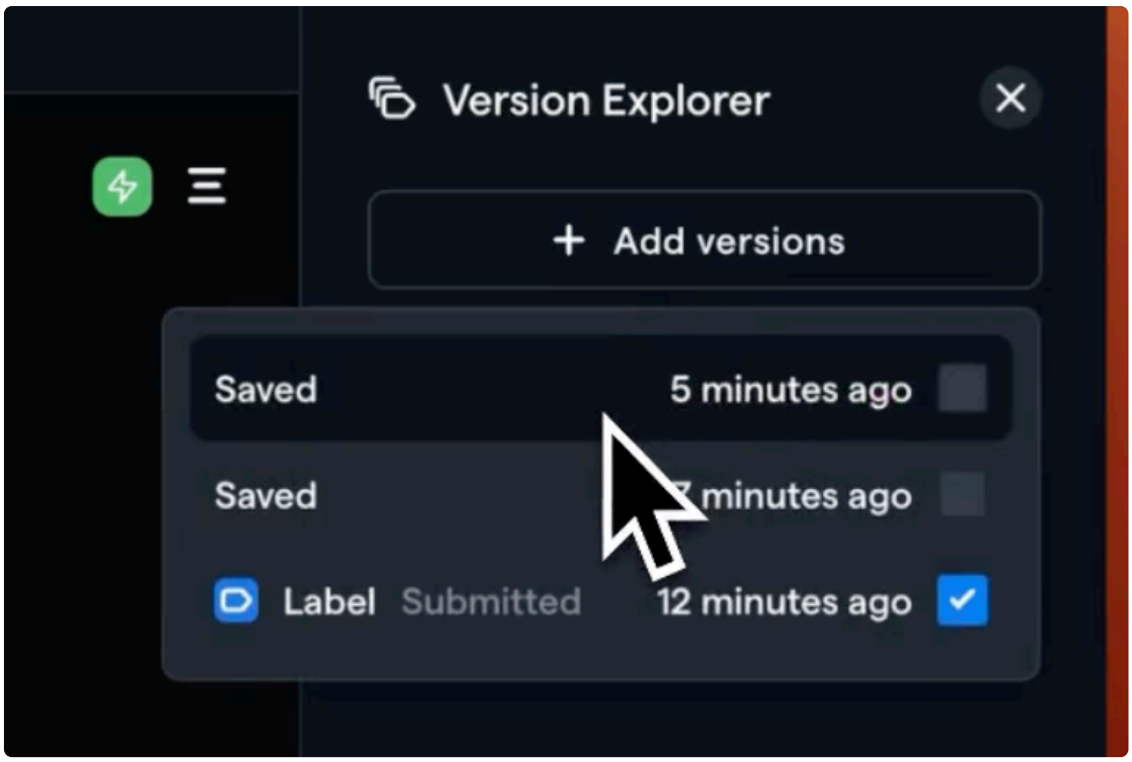
Jump to Label

The Jump to Label Action will change the current slice position to the closest slice position that contains a particular annotation. This is useful for revealing annotations on the canvas.

Annotation Version Explorer

RedBrick AI's Version Explorer allows users to reference (and, if necessary, restore) previous versions of their annotations within the Annotation Tool.

To reference a previously saved set of annotations, click on the Version Explorer button in the top right corner of the screen. The Version Explorer will then display in the right hand Context Panel., allowing you to access previously saved versions of your annotations.

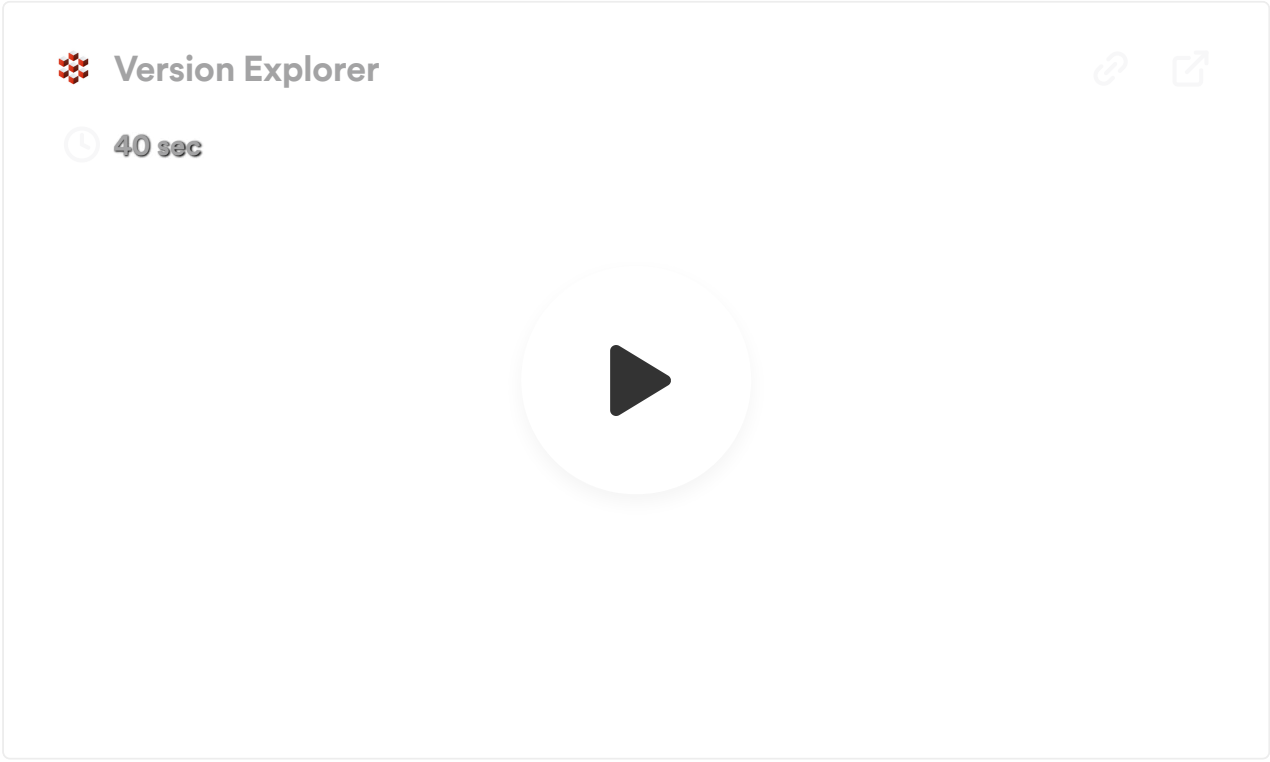


Restoring an older set of annotations will both:

1. Force a save of the most current set of annotations;
2. Duplicate the older version of annotations and create a new version based on that duplicate.

For example, let's say you (a reviewer) open a Task and see that the latest version of a labeler's annotations is **Version 5**, but you'd like to restore **Version 3**. Choosing to restore **Version 3** will immediately create a duplicate of that version, designate it as the most current version (in this case, **Version 6**), and display the labels in the Editor.

Please see the short video tutorial below for a full overview:



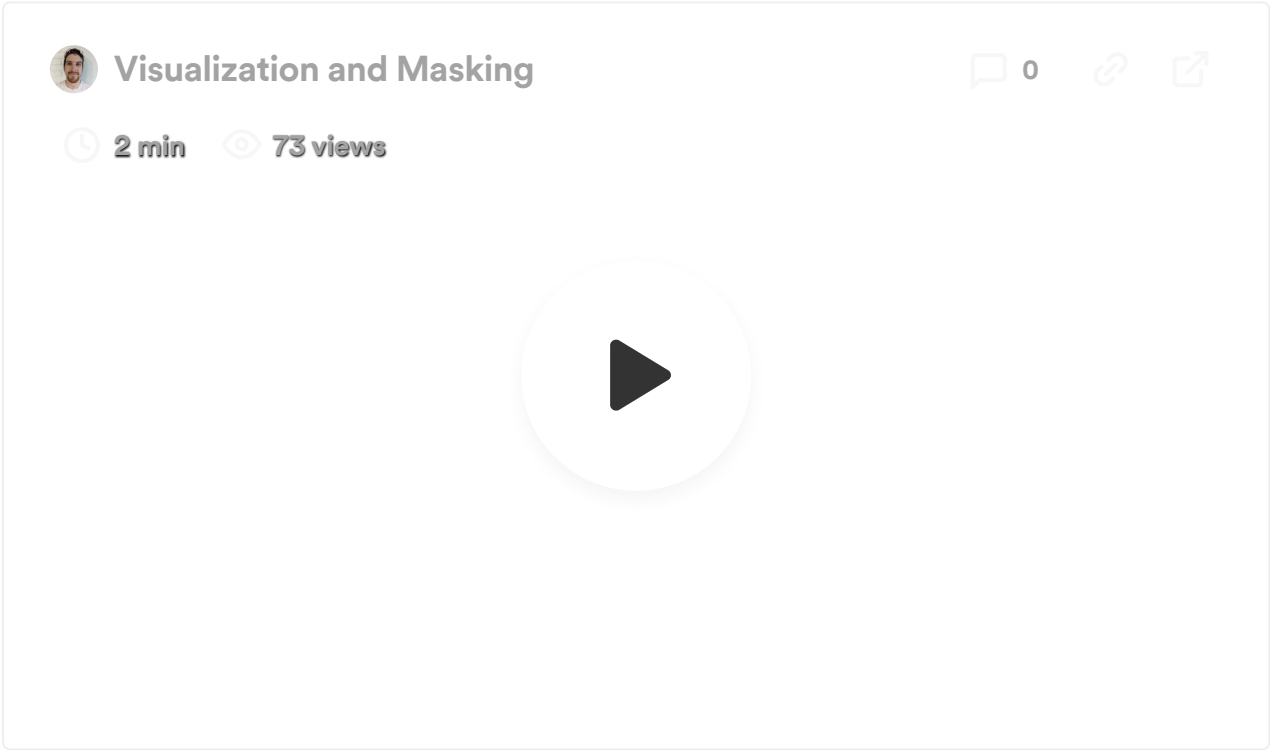
Version Explorer Tutorial

Visualization and Masking

> As of v2023.10.31.1:
> - **Thresholding** is now referred to as **Masking;**
> - the **Windowing Panel** is now referred to as the **Visualization Panel;**

RedBrick AI allows you to apply filters to your images and volumes to help with visualization and segmentation.

For a brief visual overview, please see the following video tutorial:



Visualization

You can adjust the Window Width and Level of your volumes by pressing `CTRL` and `LEFT CLICK` dragging on any viewport - `up/down` to adjust Window Level, `left/right` to adjust Window Width.

You see and/or modify the current Visualization settings by selecting the Visualization Panel on the top right of the tool bar.

When the Visualization Panel is open, its settings will display in the right hand Context Panel:

- Window Width
- Window Level
- Optional Presets
- Inverted View
- Pixel Interpolation
- Label Opacity
- Label Outlines

Masking

The Masking Panel consists of the following elements:

- **Editable Area** dropdown
- **Modify Other Segments** dropdown
- **Restrict by pixel intensity** toggle
  - Masking Range slider
  - Threshold Range Selector

Editable Area

The Editable Area dropdown has two selections - **Everywhere** and **Inside all segments**.

Selecting **Everywhere** allows you to draw on any part of the canvas.

Selecting **Inside all segments** makes it impossible for the user to annotate on an unannotated area of the canvas. In other words, the user must annotate within the bounds of an existing annotation.

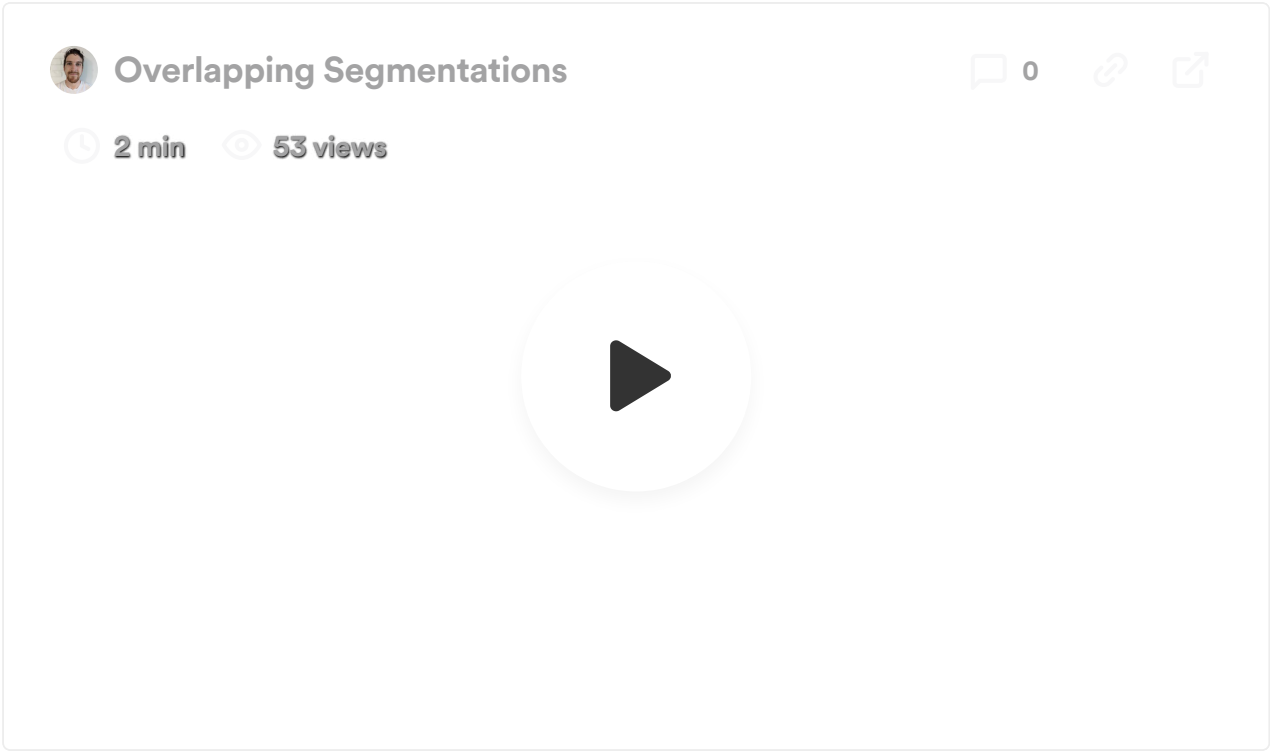
Modify Other Segments

The Modify Other Segments dropdown helps you control how your painting affects other existing annotations.

Selecting **Overlap** will allow you to paint on top of other annotations. This process is additive, which means annotating with **Overlap** does not alter or delete other annotations.

Selecting **Overwrite unlocked segments** will also allow you to paint on top of other annotations, but you will overwrite (and therefore delete) anything else that you paint on top of.

For a visual walkthrough of how to configure **Editable Area** and **Modify Other Segments** settings, as well as a demonstration of the differences between them, please see the following video tutorial:



Restrict by Pixel Intensity

To speed up segmentation, you can create a masking range by applying an upper and lower boundary for the HU values/intensities of your image or volume.

To enable Masking, select any Object Label of type **Segmentation** in the left hand toolbar and click on **Restrict by pixel intensity** in the right hand Context Panel.

With Masking enabled, you will only be able to create annotations within the range that you set, making it easy to avoid painting "outside the lines" of your target structure.

You can also use the **Threshold Range Selector**, which allows you to interactively define the bounds of your masking range. With the Range selector activated, `left click` any part of a viewport to include it in the range or `right click` to exclude it from the range.

> The pixel restriction will be applied as long as the masking filter is toggled ON. These settings are visible in the right hand Context Panel.

Segmentation

Segmentation Tools

An overview of our Segmentation Toolkit, features that are often used alongside it, and how to configure your toolkit for a Project.

Segmentation Tools >

Instance vs. Semantic Segmentation

An overview of the difference between instance and semantic segmentation, and a video walkthrough of how to do each on RedBrick AI.

Instance vs. Semantic >

Overlapping Segmentations

A video overview of how to overlap segmentations on RedBrick AI.

Overlapping Segmentations >

Reference Standards

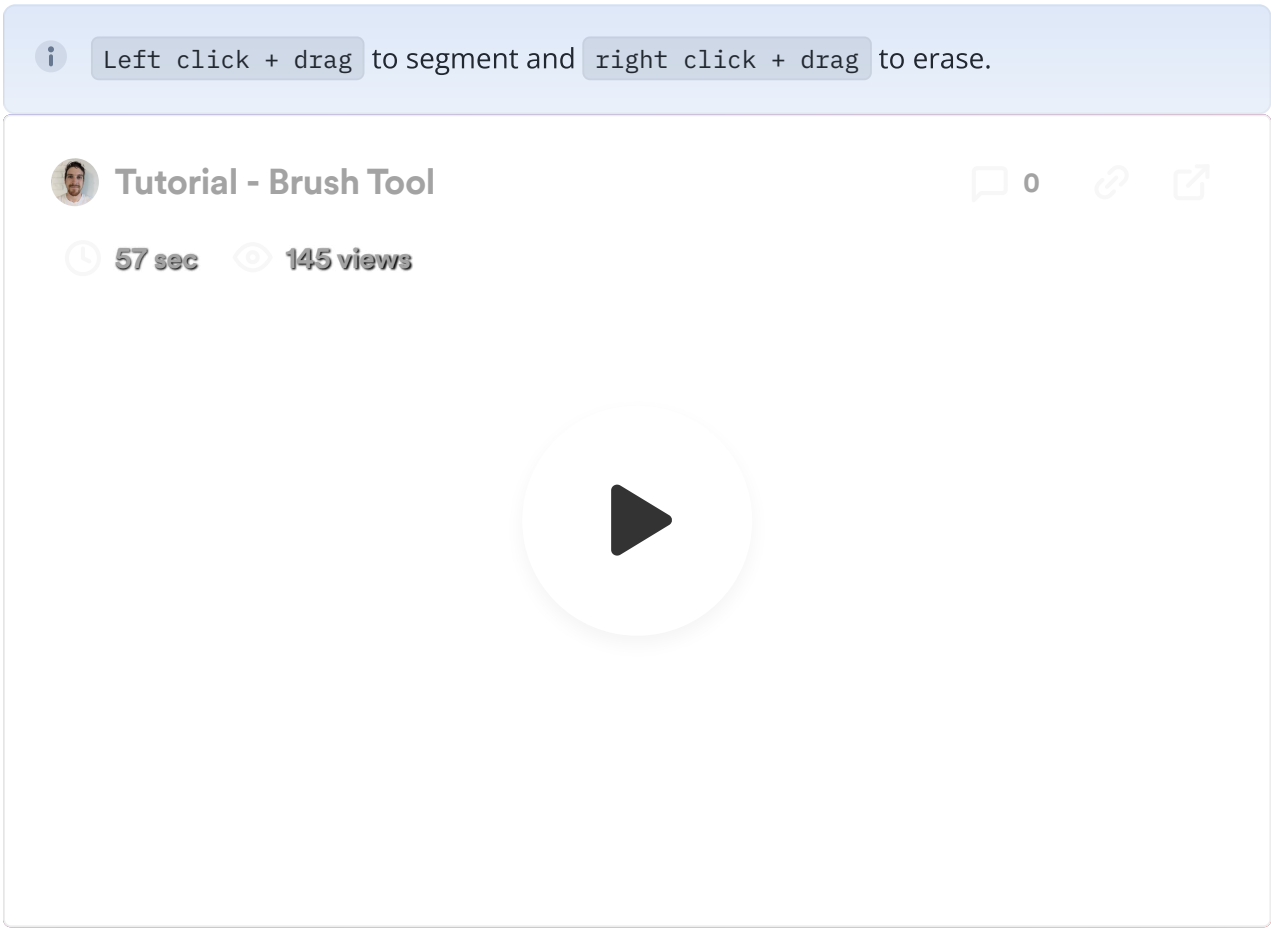
An overview of Reference Standards, i.e. a set of Ground Truth annotations that you can add to your Project as a visual reference for your labelers.

Reference Standards >

Segmentation Tools

Brush Tool

The Brush Tool has two modes - 2D and 3D (toggled on the right-side panel). The 2D brush is a circle, and the 3D brush is a sphere that segments across slices. You can adjust the size of the brush using the slider in the right hand Context Panel or the `W` & `S` hotkeys.

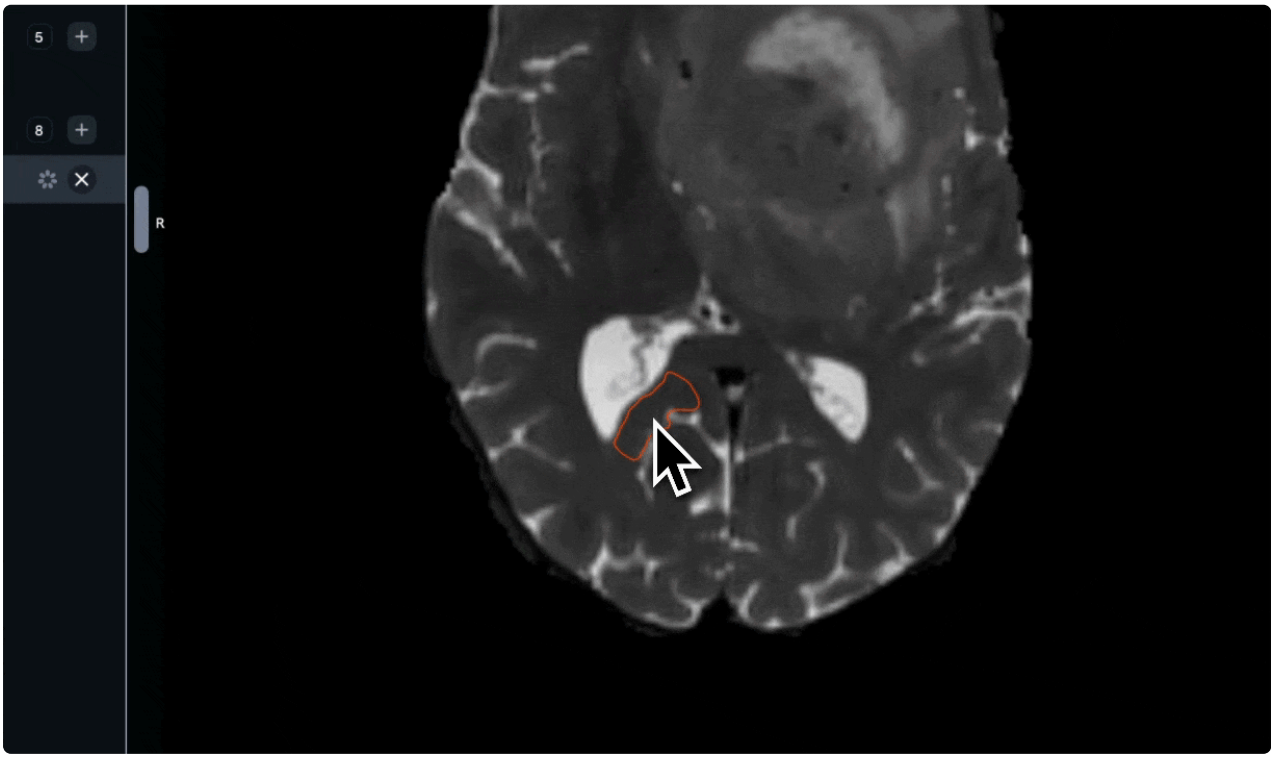


Brush Tool Overview

Adaptive Brush

The Adaptive Brush automatically defines the edges of the structure you are annotating, resulting in a smarter and more accurate workflow.

The Adhesion parameter allows you to determine how "strict" the Adaptive Brush is with similar intensity values. A lower Adhesion value, for example, would be recommended for annotating regions or structures with smooth gradients, whereas a higher Adhesion value would be best for annotating with sharper gradients.



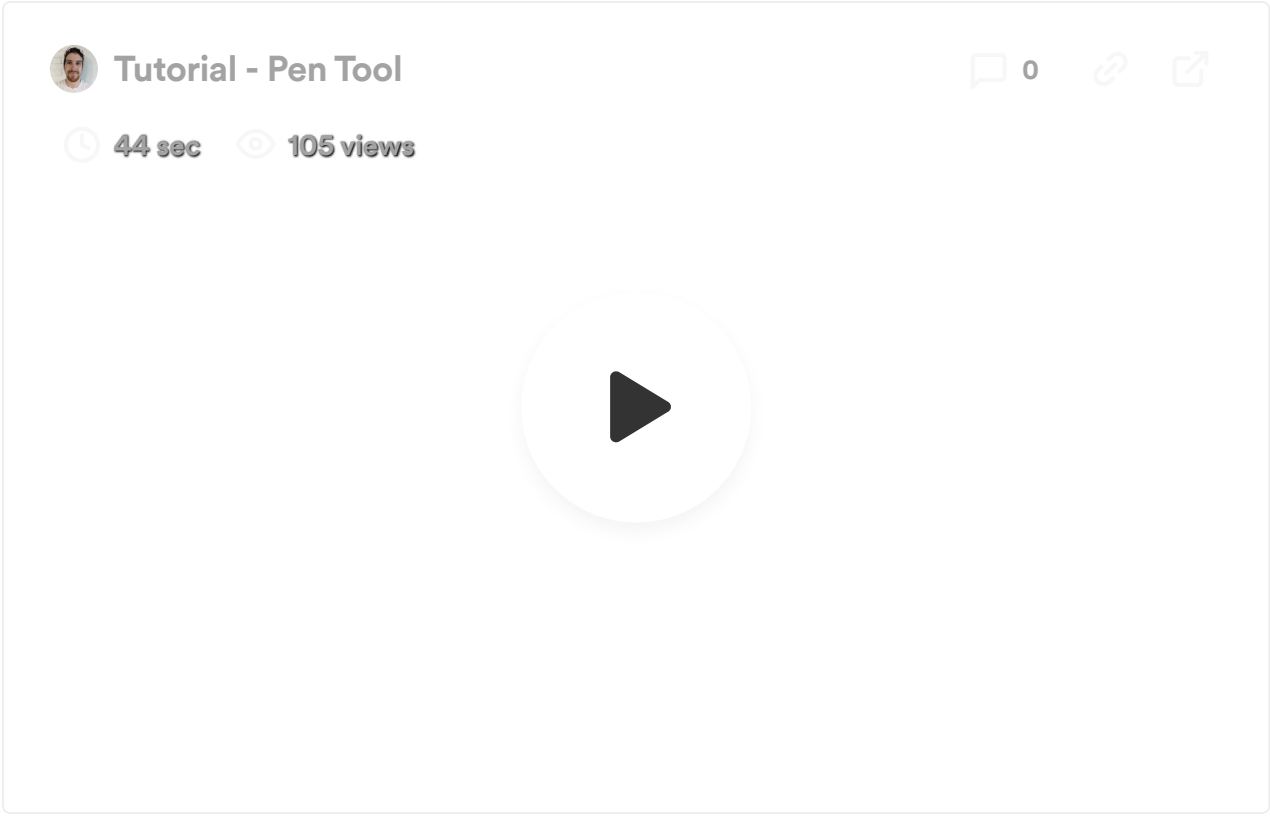
The Adaptive Brush in action

Adaptive Brush Mastery: we recommend adjusting your Windowing (and other Visualization) settings along with your Adhesion levels to ensure optimal brush behavior.

Pen Tool

The Pen Tool has two modes - 2D and 3D (toggled on the right-side panel). The Pen Tool allows you to annotate using a free-form contour. In 3D mode, the free-form contour is extruded above and below the current slice.

`Left click + drag` to add a segment region and `Right click + drag` to remove a region.

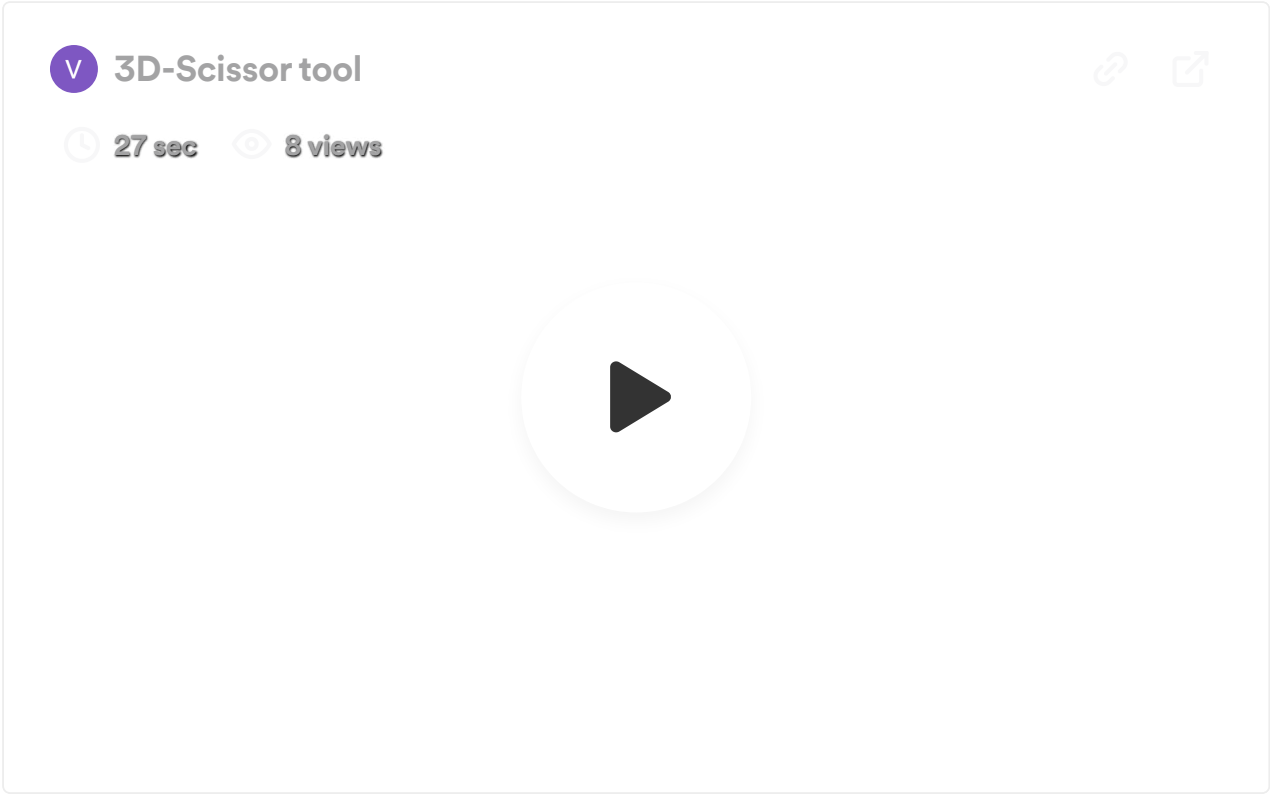


Pen Tool Overview

3D Scissor Tool

The Scissor tool allows you to erase the unwanted section of your segmentation in the 3D view plane.

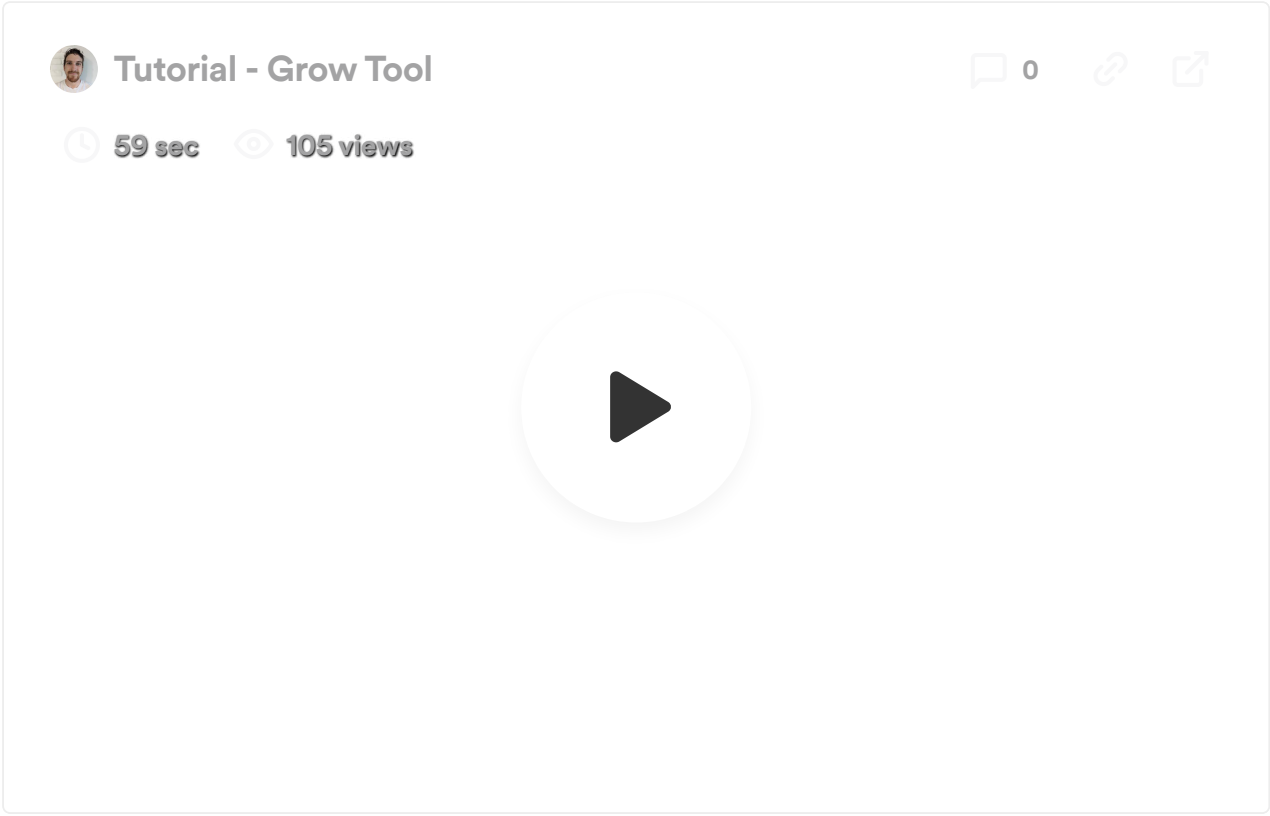
Left click + Drag to erase the unwanted section



Region Growing (Grow Tool)

The Grow Tool is a semi-automated tool that uses image intensity information to segment regions. By clicking and holding in a region, the segmentation will grow outward from the point that you clicked on. The longer you hold, the longer the region will grow.

`Left click + hold (+ drag)` to segment, `right click + hold (+ drag)` to erase.



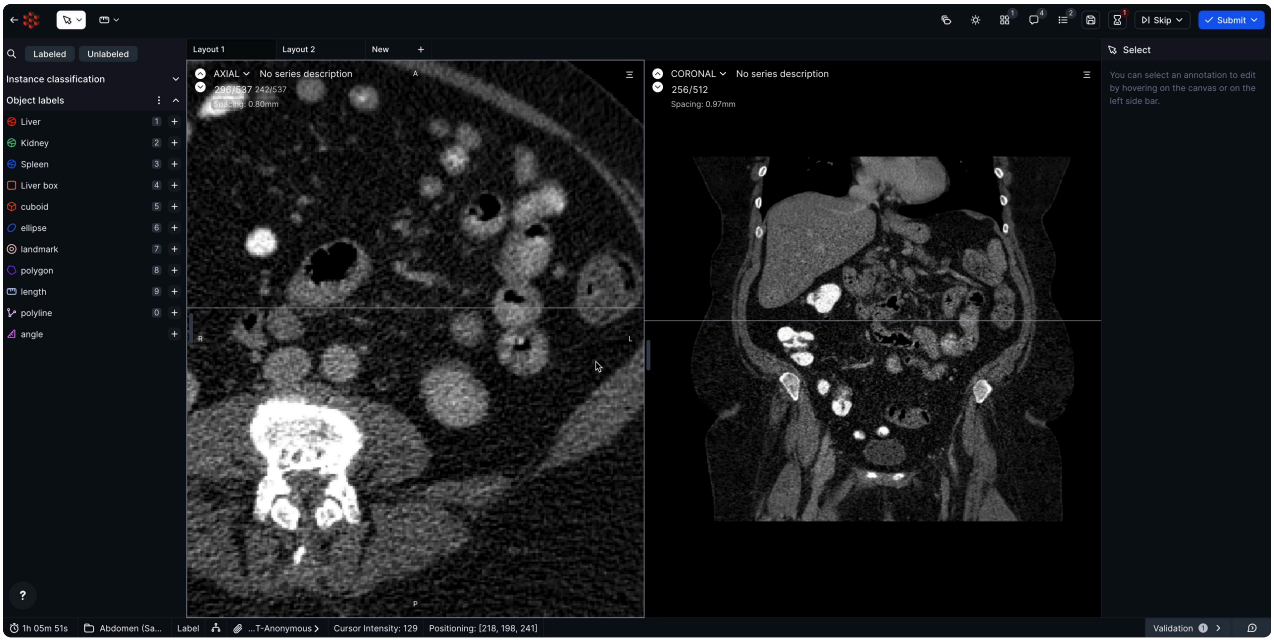
Grow Tool Overview

Contour Tool

The contour tool lets you interact with masks as *contours*. Interacting with masks as contours gives you special tools to create and edit annotations.

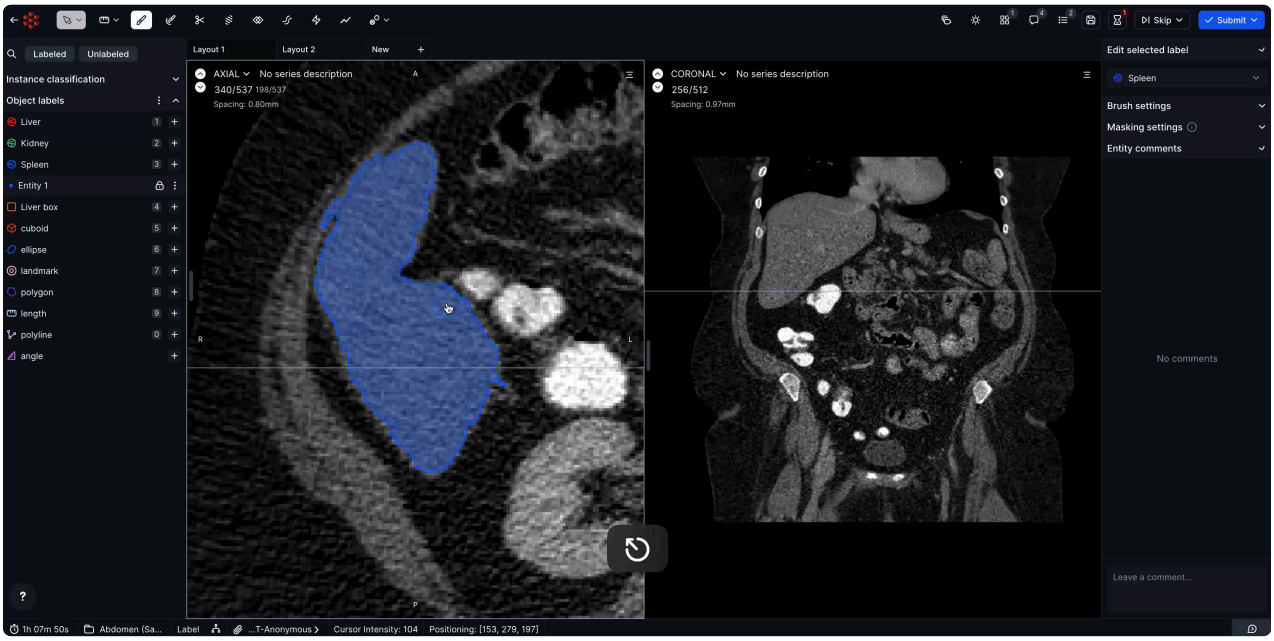
Creating a contour from scratch.

Select the contour tool, and click and drag on the canvas to draw a contour. To complete the contour, simply intersect the initial node or any other edge of the contour.

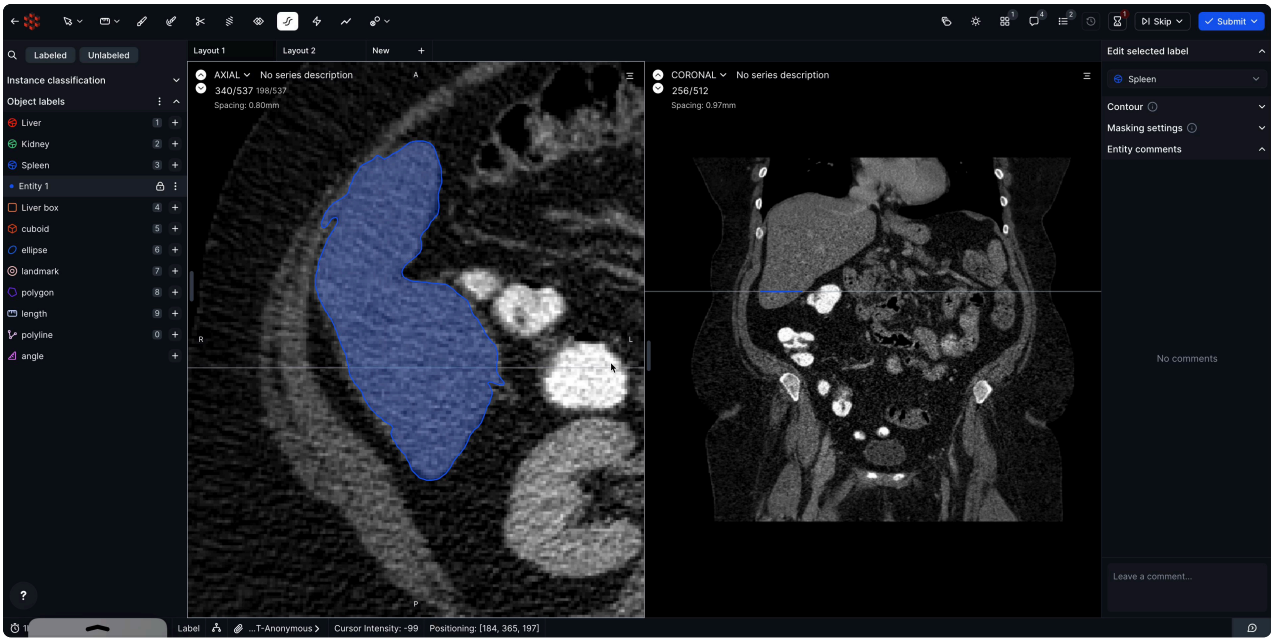


Editing existing masks as contours

- **Node edit.** Select an existing mask and select the contour tool. When you hover over an edge, you will see a node with a "target spline" appear. You can drag this node to edit the "target spline" area interactively.



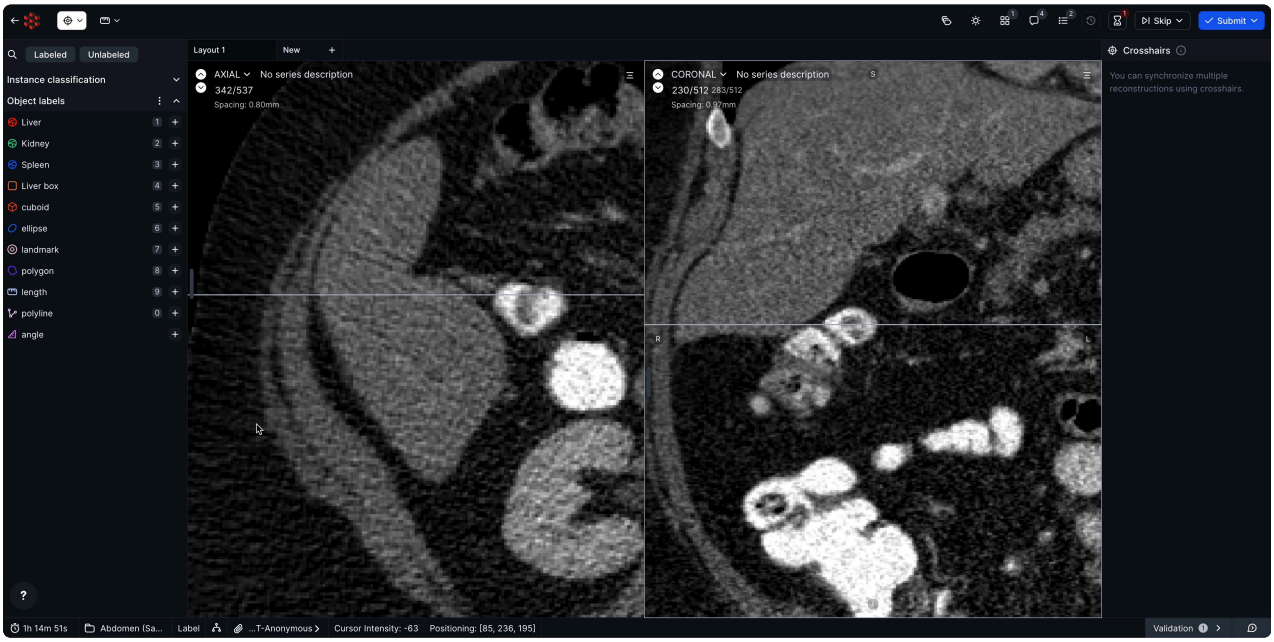
- **Edge redraw.** You can redraw any edge by simply "overwriting" the edge by intersecting it two times.



Interpolate

You can interpolate between two masks using the *Interpolate* tool.

1. First, draw two masks on two different slices. These masks must be part of the same entity.
2. With the Interpolate tool, select the edges of both masks one by one.
3. You will see an interpolated annotation in between the masks.

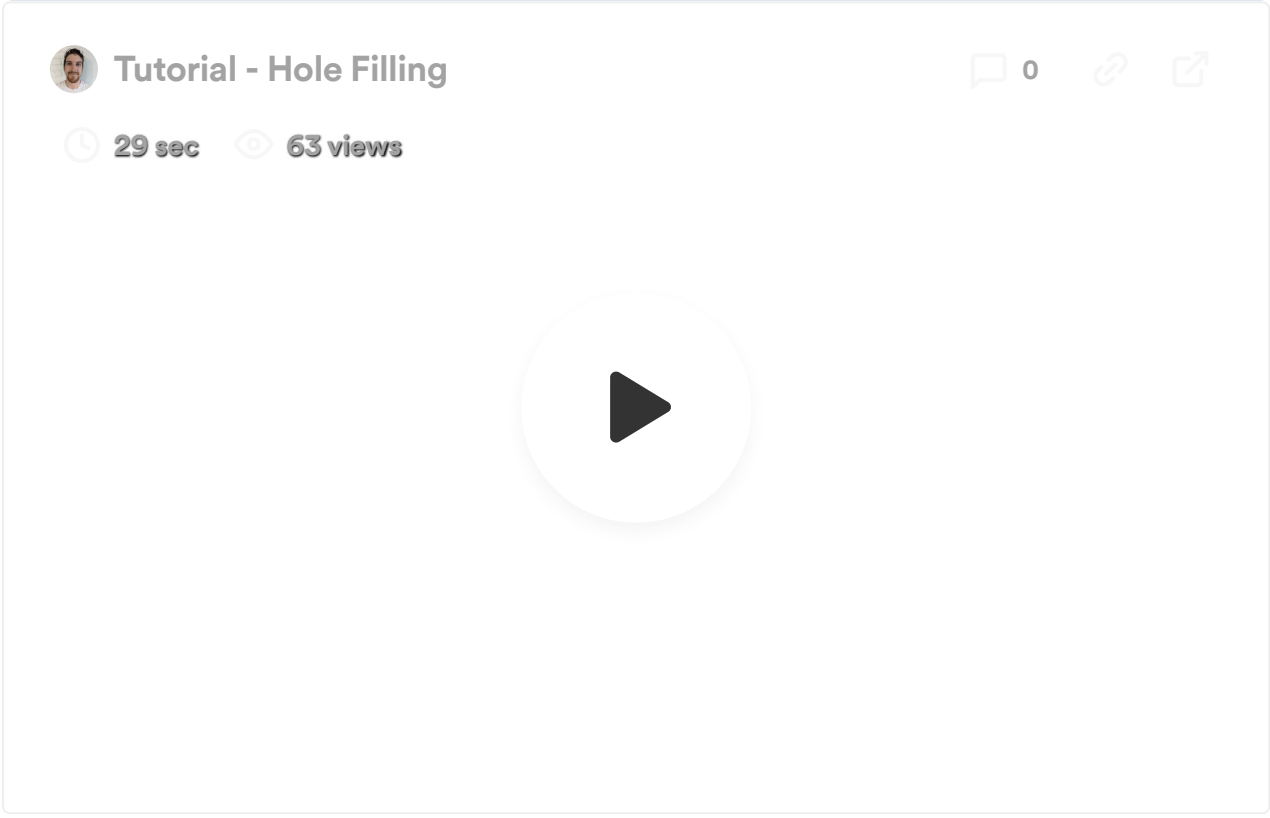


Toggle Cineloop to see the slices and where the interpolated masks are.

Hole Filling Tool

The Hole Filling Tool iteratively fills small holes in your segmentation. Click anywhere on the canvas to start filling in small holes.

The Hole Filling Tool is designed to fill small holes. For larger holes, you may need to run the Hole Filling Tool more than once (i.e. click several times).



Hole Filling Overview

For large volumes, 3D hole filling can be very computationally expensive. If your data has more than 800 slices, we recommend only using 2D hole filling.

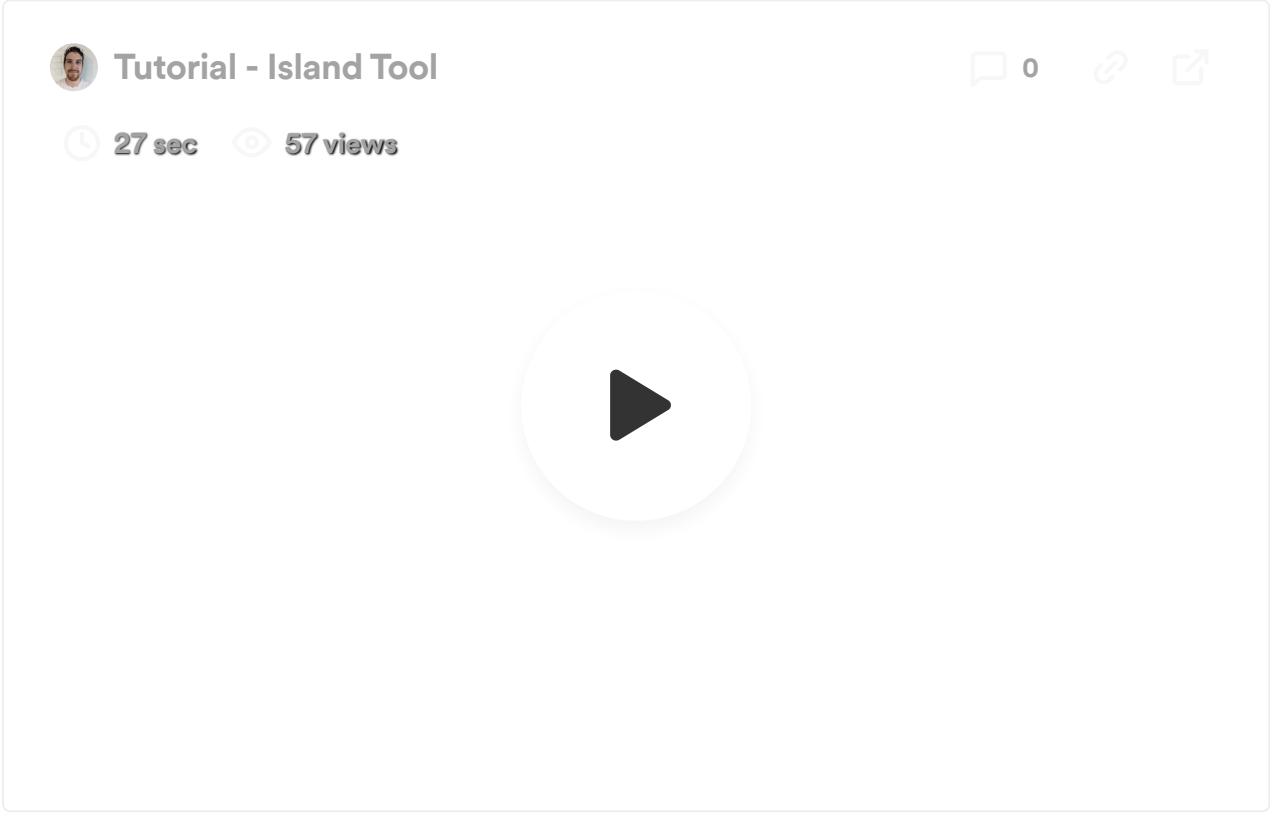
Paint Bucket Tool

The Paint Bucket is helpful for closing single large holes. With the paint bucket tool selected, click in any large hole to fill it automatically.

Island Removal Tool

Island Removal deletes "islands" of segmentations. Simply click on any island segmentation to remove it.

Conversely, you can enable **Keep Currently Selected** in the right hand Context Panel to remove all of the islands **except** the one you clicked on.



Island Tool Overview

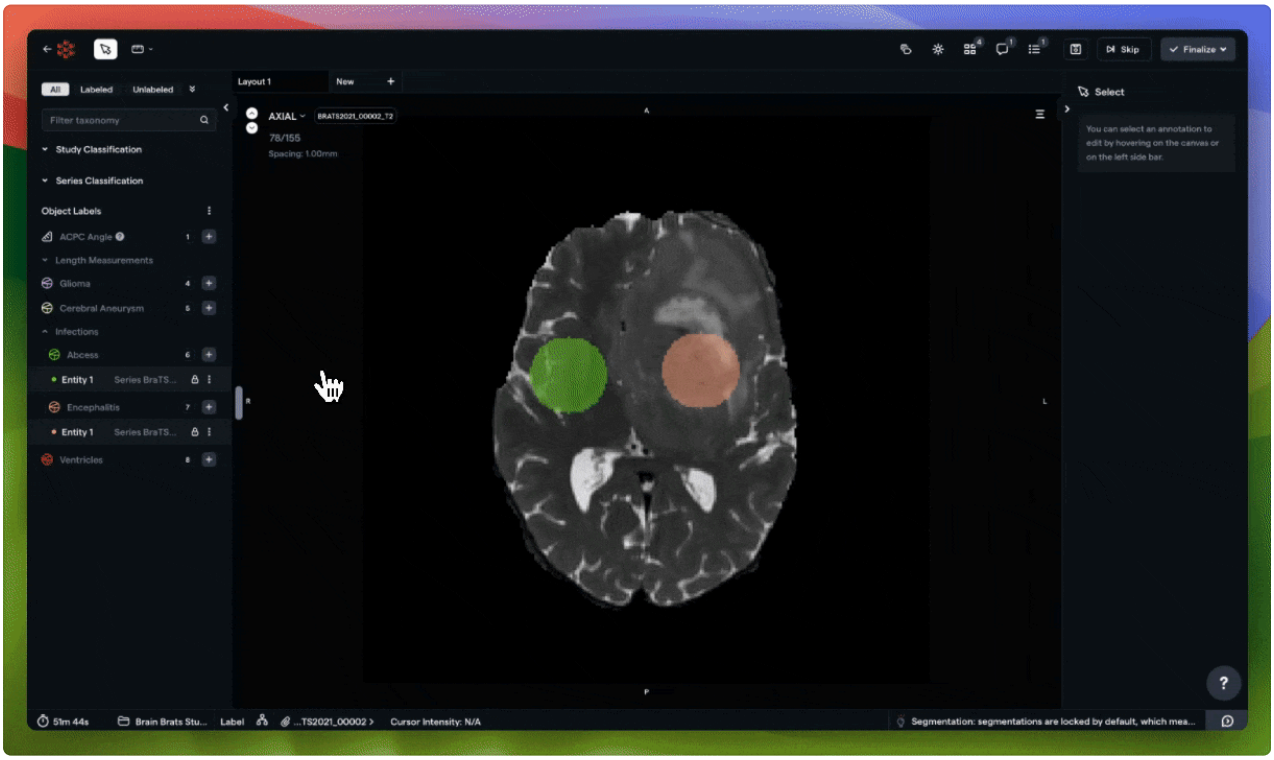
Boolean Operator Tool

The Boolean Operator Tool allows you to perform four operations on segmentations: Copy, Add, Subtract, and Merge.

Copy Tool

The Copy Tool allows you to transform a segmentation into a perfect copy of another.

With Segmentation X selected, enable the Copy Tool and click on Segmentation Y to create a perfect copy of Segmentation X that occupies the same space.



The Copy Tool in action

Add Tool

The Add Tool allows you to add the pixel values of a segmentation to another.

With Segmentation X selected, enable the Add Tool and click on Segmentation Y to add the pixel values of Segmentation Y to Segmentation X.

Subtract Tool

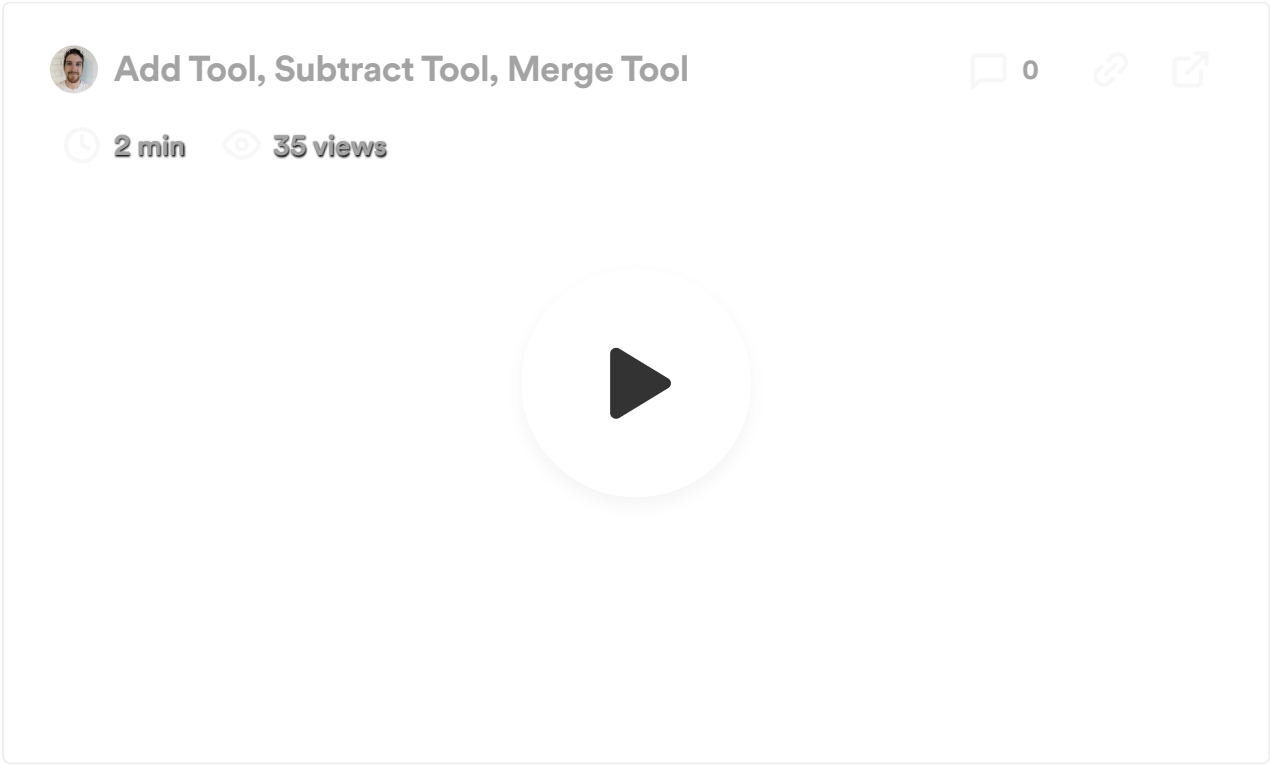
The Subtract Tool allows you to remove the pixel values of a segmentation from another segmentation.

With Segmentation X selected, enable the Subtract Tool and click on Segmentation Y to remove the pixel values of Segmentation Y from Segmentation X.

Merge Tool

The Merge Tool allows you to transform one type of segmentation island into another.

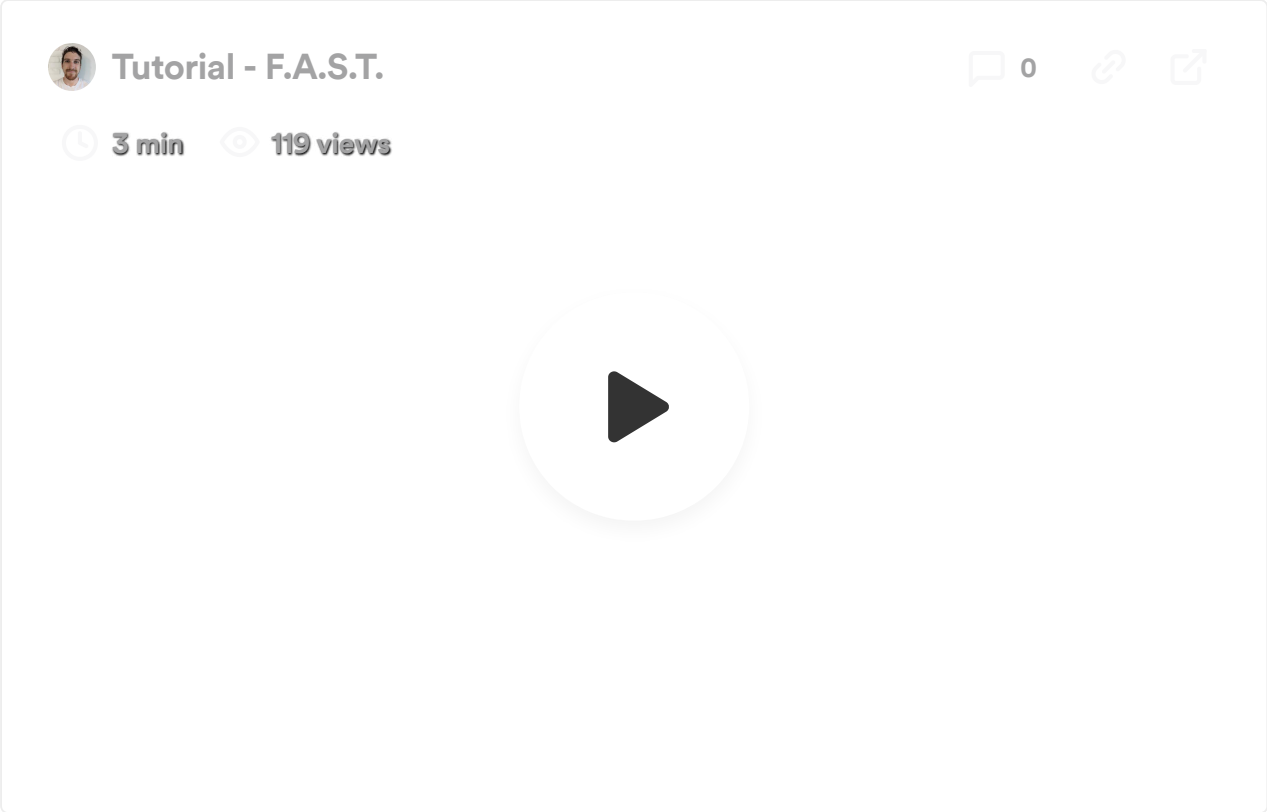
With Segmentation X selected, enable the Merge Tool and click on an island of Segmentation Y to merge the island to Segmentation X.



Add, Subtract, and Merge Tool Tutorial

Fast Automated Segmentation Tool (F.A.S.T. ⚡)

The Fast Automated Segmentation Tool (F.A.S.T.) is an automatic segmentation tool powered by Meta AI's Segment Anything Model that allows users to rapidly generate 2D and 3D segmentations.



F.A.S.T. Overview

F.A.S.T. is powerful because of the way users can prompt the tool to generate an accurate segmentation in real time. Under the hood, there are two components to the system:

1. **Image encoding**, which takes place on the server side. The result of image encoding is *embeddings*, which are special vector representations of the images that are useful for machine learning.
2. **Segmentation decoding**, which takes place in the browser in real time.

> **To enable F.A.S.T. for your team**, please do the following:
>
> 1. Request access at https://redbrickai.com/fast.
> 2. Within a Project, navigate to the Tool Settings page and enable F.A.S.T.

Generating 2D Segmentations with F.A.S.T.

To start segmenting, create a segmentation instance, select the F.A.S.T. tool from the top bar or using `cmd/ctrl + b`. Once the tool is selected, hover over a viewport to

start embedding computation for a single slice (you will see a loader spinner on the top right of the viewport). You can prompt F.A.S.T. in a few different ways after the embedding computation is complete:

1. **Bounding Box prompts.** `Click + move mouse + click` to draw a bounding box. You will see the segmentation prediction compute in real time while you draw the bounding box.
    1. **Key point refinement.** After you draw the bounding box, you can optionally refine the segmentation by prompting the system with key points.
    `Left click` to add regions you want to *add to the segmentation.*
    `Right click` to remove regions from the segmentation prediction.
    2. Once you are happy with the segmentation preview, confirm it by using the button on the right panel or `shift + enter`.
2. **Instant click.** `alt/option + hover` over objects to view a prediction preview. If you are satisfied with any preview, click while pressing `alt/option` to confirm the segmentation.

Generating 3D Segmentations with F.A.S.T.

3D F.A.S.T. allows users to draw Bounding Boxes to define an interpolation range for a 3D structure that is to be annotated.

The process for creating annotations with 3D F.A.S.T. is extremely similar to that of 2D F.A.S.T. You can find a full step-by-step breakdown of how to use 3D F.A.S.T. below.

1. Create a new Instance of your desired Object Label by clicking on the “+” in the left hand toolbar;

2. Select F.A.S.T. in the top of the screen and wait for the embedding computation to complete on Slice X;

3. Create a Bounding Box around the structure you wish to annotate;

4. (Optional) Provide F.A.S.T. with additional input by using `LMB/RMB`;

5. Navigate to the end of the range that you want to interpolate across (i.e., Slice Y);

6. Repeat Step 3 (and optionally, Step 4) for the structure you wish to annotate on Slice Y;

7. Once you are satisfied with the annotations, press Enter or click on “Finalize” in the right hand toolbar to generate the pixel masks on every slice between Slice X and Slice Y.

> Processing times may increase when interpolating across large ranges. However, please note that all subsequent work across the same range should be much faster, as the embedding computations only have to be computed once per slice.

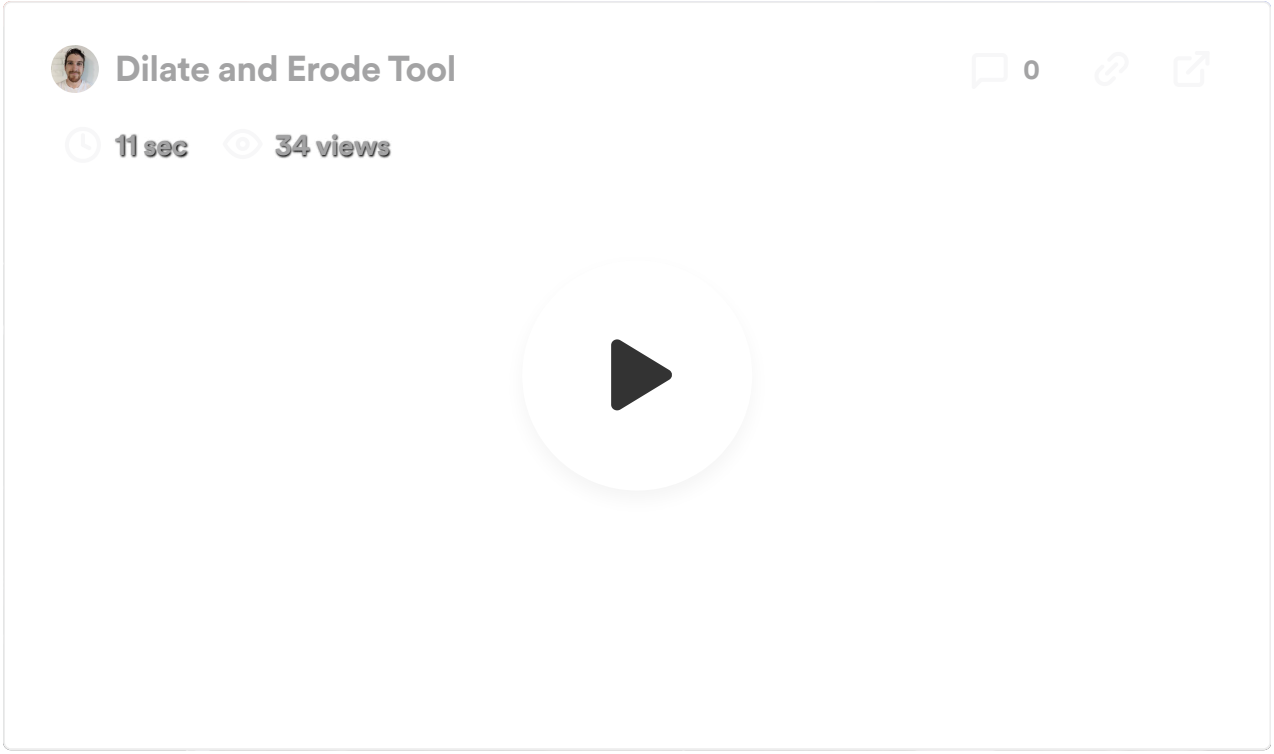
> Firewalls, ad blockers, privacy extensions, and any other browser extensions that block HTTP traffic are known to interfere with FAST.

Dilate & Erode Tool

The Dilate & Erode Tool allows you to expand or shrink the area of a segmentation.

With a segmentation selected, use `left click` to Dilate (grow) the area of the segmentation and `right click` to Erode (shrink).

Use the **Pixels to Change** slider in the right hand Context Panel to control the severity of the dilation or erosion.

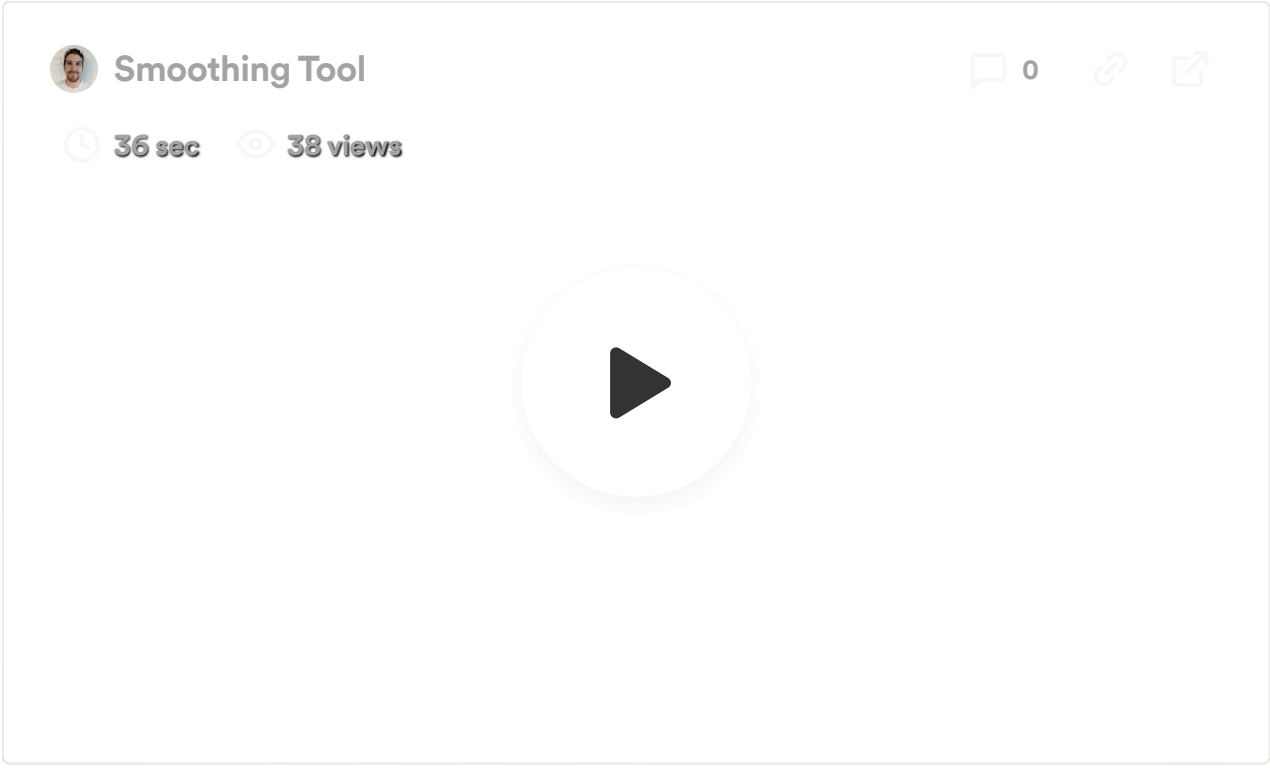


The Dilate and Erode Tool in action

Smoothing Tool

The Smoothing Tool helps you remove peaks and valleys in noisy annotations.

With the Smoothing Tool selected, use `left click` to remove valleys and `right click` to remove peaks.

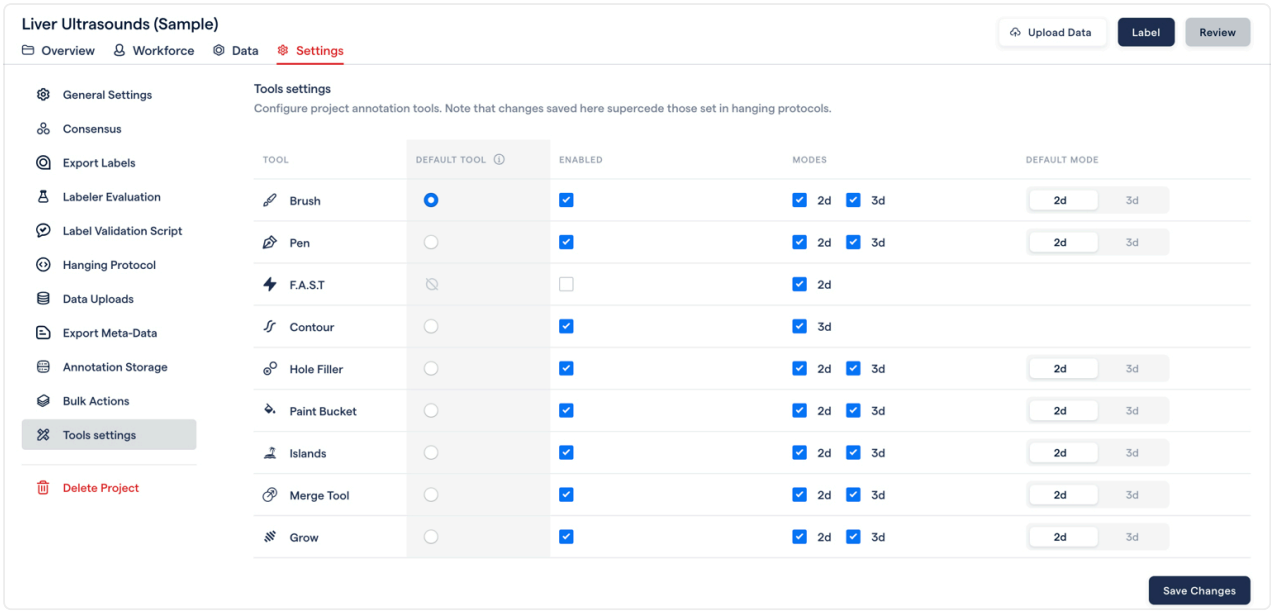


Tool Configuration

Your labeler toolkit can be customized at the Project level by navigating to the **Tool Settings** page within your Project Settings.

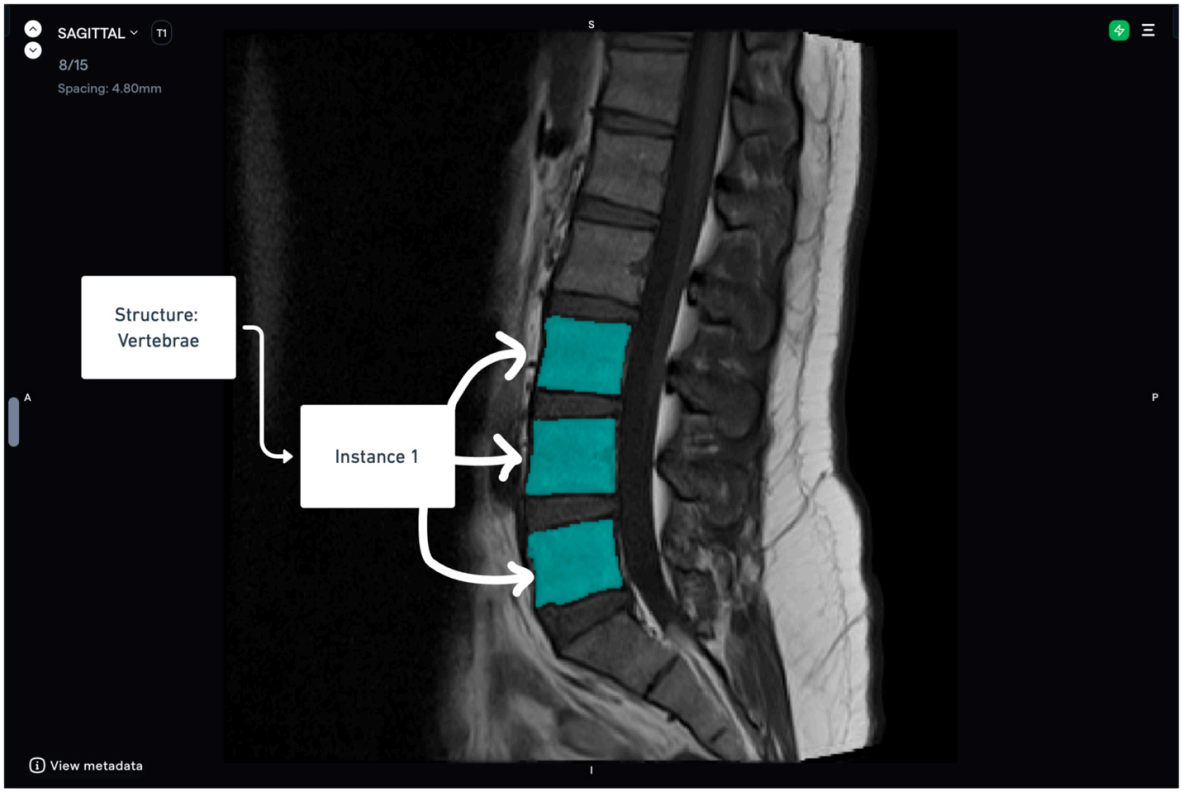
Simply utilize the checkboxes for each tool to:

- set a custom default tool;
- restrict/enable 2D and/or 3D annotation;
- set a default mode (i.e. 2D or 3D);
- enable/disable a Tool entirely;


Liver Ultrasounds (Sample)
Upload Data
Label
Review
Overview
Workforce
Data
Settings
General Settings
Consensus
Export Labels
Labeler Evaluation
Label Validation Script
Hanging Protocol
Data Uploads
Export Meta-Data
Annotation Storage
Bulk Actions
Tools settings
Delete Project
Tools settings
Configure project annotation tools. Note that changes saved here supercede those set in hanging protocols.
TOOL
DEFAULT TOOL
ENABLED
MODES
DEFAULT MODE
Brush
2d
3d
Pen
2d
3d
F.A.S.T
2d
Contour
3d
Hole Filler
2d
3d
Paint Bucket
2d
3d
Islands
2d
3d
Merge Tool
2d
3d
Grow
2d
3d
Save Changes

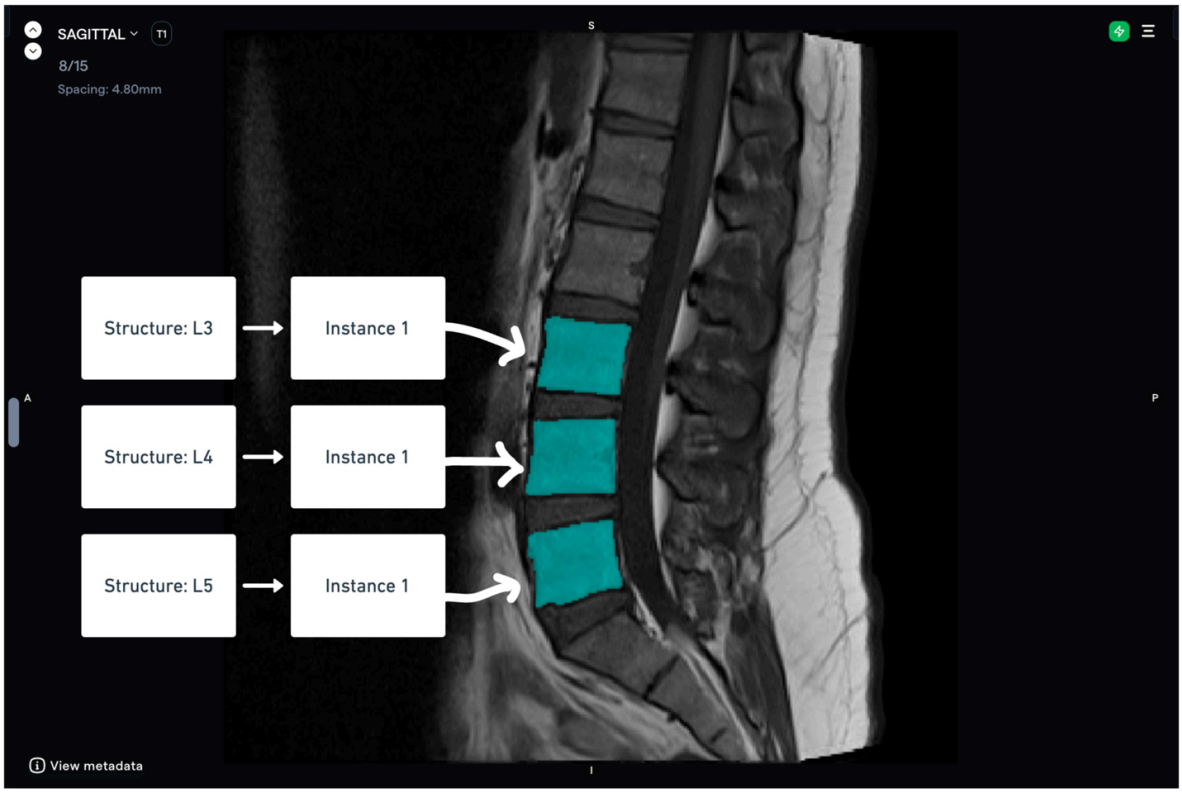
Instance vs. Semantic

Semantic segmentation treats multiple objects of the same class as a single entity. In the example below, you can see that the labeler is responsible for annotating a single structure ("Vertebrae"), regardless of the vertebrae's classification (e.g. L1, S1, etc.).



Semantic Segmentation

On the other hand, **instance segmentation** treats multiple objects of the same class as distinct individual objects (or instances). In the example below, the annotator must create a unique annotation (as represented by an Instance in Redbrick AI) for each specific vertebrae.



Instance Segmentation

On RedBrick AI, you can **perform both semantic and instance segmentation** by controlling how many instances of a particular category you create on the left side bar. If you create more than one instance of a single category, on export, you will be able to correlate the instance ID's in your segmentation mask to your category names.

Check out this video for an overview:

Semantic vs. Instance Segmentation

0

2 min 57 views

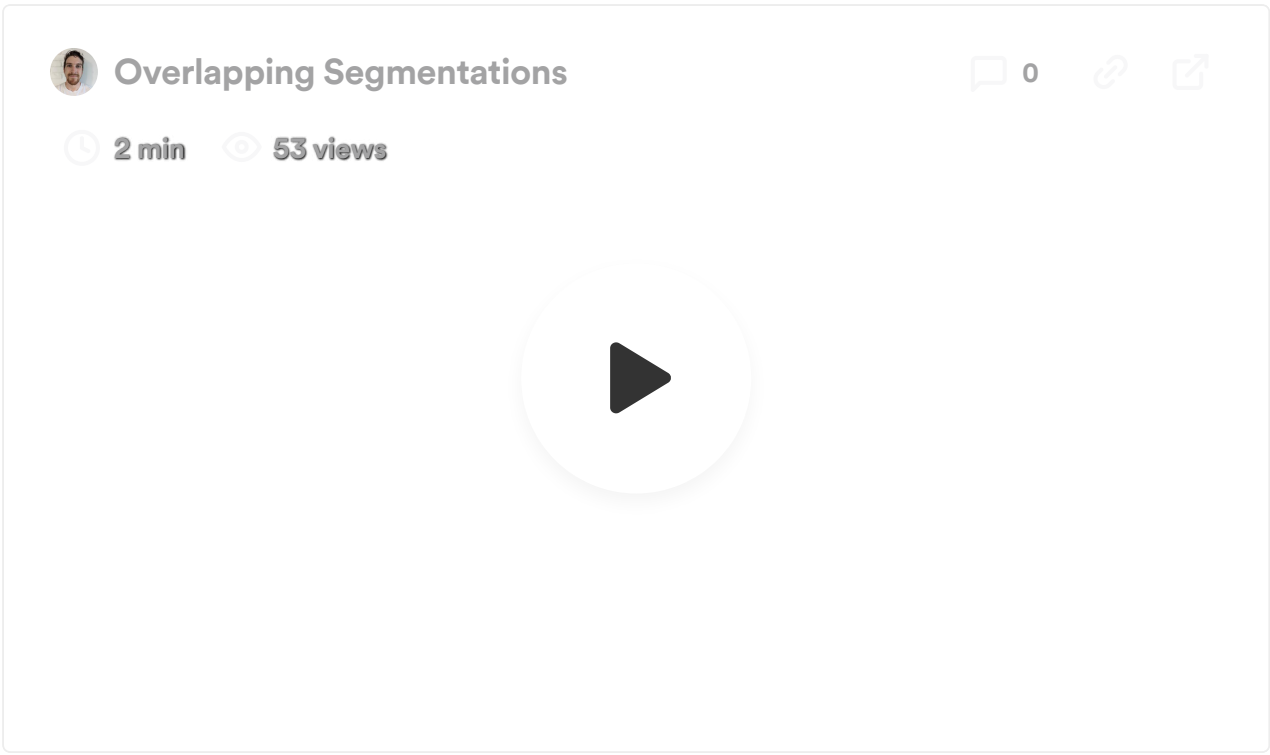
Overlapping Segmentations

If you are annotating multiple structures in the same pixel/voxel space, you can utilize overlapping segmentations in RedBrick AI.

Any time you are using a Segmentation Tool (e.g. Brush Tool, Pen Tool, Contour Tool, etc.), you can use the options in the Masking Panel to configure your overlapping (or overwriting) behavior.

Specifically, you can utilize various combinations of the **Editable Area** and **Modify Other Segments** menus to determine how your annotations will be overlapped or overwritten. For a more comprehensive description of these menus and their functions, please reference the relevant documentation.

Alternatively, please see the following short tutorial for a visual walkthrough of the differences:



An overview of how to overlap segmentations

Exporting Overlapping Segmentations

When exporting a Task with overlapping segmentations, RedBrick AI generates a unique annotation file for each segmentation.

The following JSON block is an example of how a single Task with two overlapping annotations will be exported.

```
projectId/
|-- TaskName
      |-- SeriesName.nii.gz // NIfTI file containing all annotations
      |-- SeriesName
        |-- instance-1.nii.gz // NIfTI file for first overlapping segmenta
        |-- instance-2.nii.gz // NIfTI file for second overlapping segmen
```

Heat maps

Beta release on https://preview.redbrickai.com

Heatmaps provide a way to overlay scalar volumetric data over the base image for reference purposes. You will have the ability to adjust the multiple color gradients and thresholding options to visualize the data.

> Heatmaps must be uploaded via the SDK.

Format:

```
type Series {
  ...
  heatMaps: [HeatMap]
}

type HeatMap {
  name: string;
  item: string; // file path
  preset?: string;
  dataRange?: [number, number];
}
```

For the complete task format, please refer to
https://sdk.redbrickai.com/formats/index.html

Upload

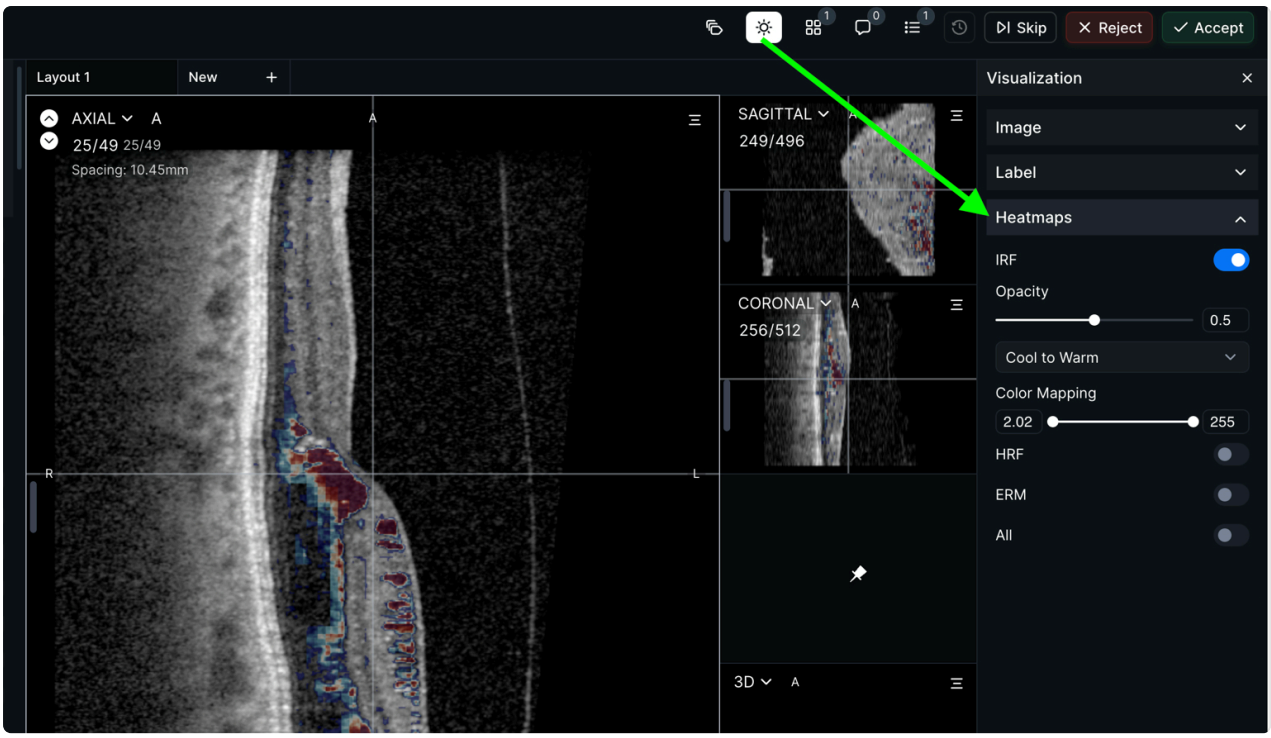
Please install the pre-release version of the RedBrick SDK (`pip install redbrick-sdk==2.17.7b1`)

Here is a sample script to upload heat-maps in task.

```python
from typing import List

import redbrick
from redbrick.types.task import InputTask


ORG_ID = "ORG_ID"
PROJECT_ID = "PROJECT_ID"
API_KEY = "API_KEY"
URL = "https://preview.redbrickai.com"

project = redbrick.get_project(ORG_ID, PROJECT_ID, API_KEY, URL)

points: List[InputTask] = [
    {
        "name": "sdk-public",
        "series": [
            {
                "items": [
                    "/path/to/image/inst1.dcm",
                    "/path/to/image/inst2.dcm",
                    "/path/to/image/inst3.dcm",
                ],
                "heatMaps": [
                    {"name": "heatmap 1", "item": "/path/to/heatmap1.nii.
                    {"name": "heatmap 2", "item": "/path/to/heatmap2.nii.
                ],
            }
        ],
    }
]

project.upload.create_datapoints(redbrick.StorageMethod.REDBRICK, points)
```

Visualization

Visualization settings can be toggled under the Visualization panel on the right sidebar.


Skip
Reject
Accept
Layout 1
New
AXIAL A
25/49 25/49
Spacing: 10.45mm
SAGITTAL A
249/496
CORONAL A
256/512
3D A
Visualization
Image
Label
Heatmaps
IRF
Opacity
0.5
Cool to Warm
Color Mapping
2.02
255
HRF
ERM
All

Python SDK & CLI

Installation & API Keys

Installation

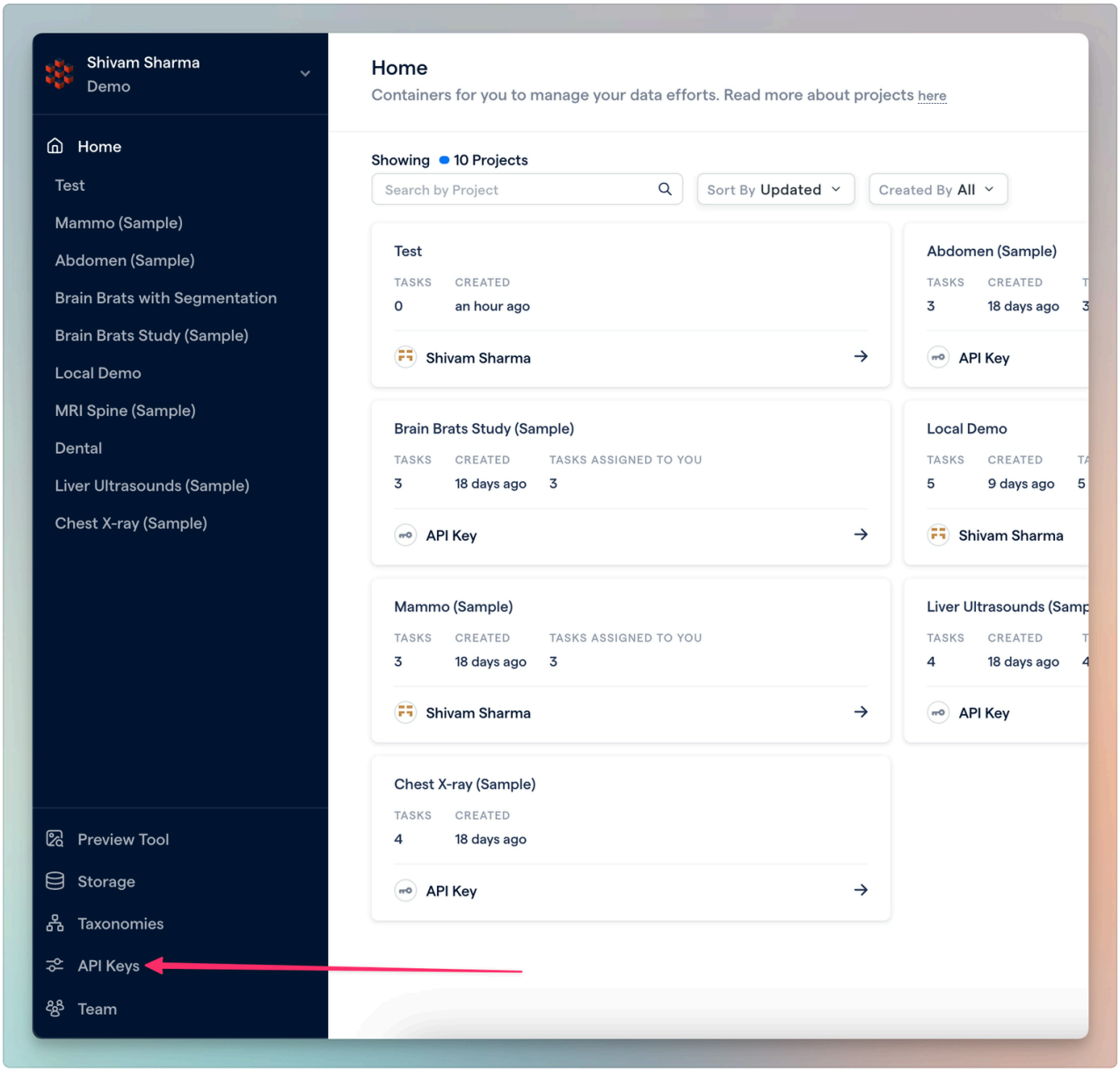
The RedBrick AI SDK and CLI are available on PyPI and can be installed using `pip`. The SDK and CLI are packaged together.

```
$ pip install -U redbrick-sdk
```

> The SDK and CLI work on Mac, Windows, and Linux.
>
> They are compatible with **python version 3.8+**

API Keys

An API Key is needed to use either the Python SDK or Command Line Interface. Create an API key in the API Keys section on the left sidebar.

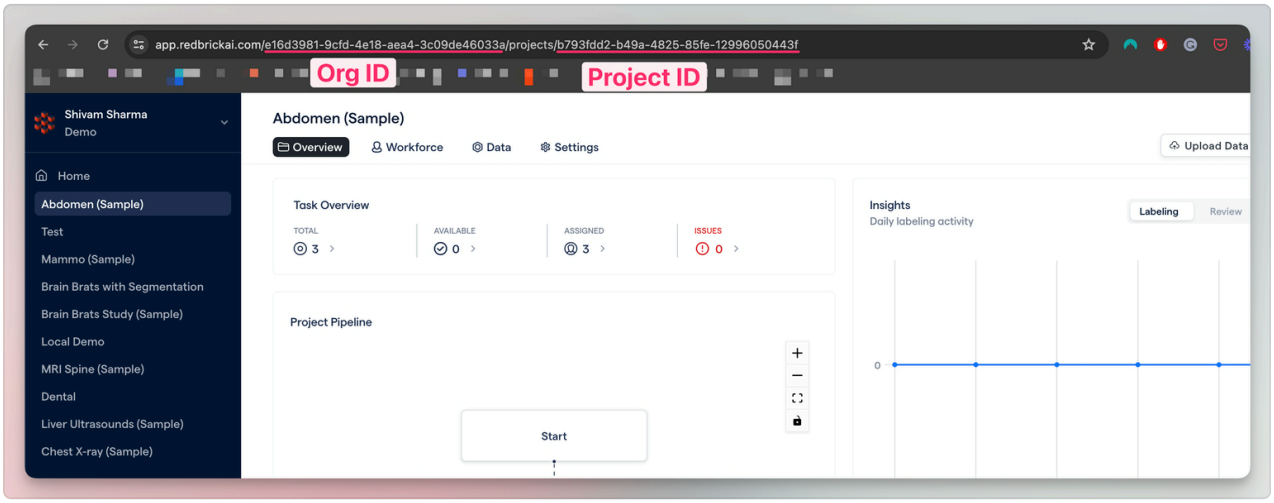


Only Organization Admins can create API Keys. The API key will have all the same permissions that Organization Admins have.

Organization and Project IDs

For most SDK / CLI operations, you will need your organization and/or project ids. These are unique ids for each entity. You can find both the Organization and Project ID inside the **Settings Page of any Project**.

You can also find the Organization and Project IDs within the browser URL -> head over to any project - https://app.redbrickai.com/<org_id>/projects/<project_id>.



SDK Overview

The RedBrick AI Python SDK is a Python package that allows developers to interact with the RedBrick AI application programmatically.

We recommend you use the Python SDK if you want to:

- Build data pipelines with Python and want to integrate your RedBrick AI annotation;
- Write advanced scripts beyond simple import & export;
- Take advantage of certain features such as HTML Tooltips, Series and/or Task Level Metadata, or Taxonomy Nesting;

For simple data import and annotation export, we recommend you use the CLI, which has a simple interface with optimizations for basic use cases.

CLI Overview ›

This SDK documentation is intended to cover high-level guides and use cases. If you are interested in more detailed documentation of the SDK interface, please visit the full SDK reference documentation.

Initializing the RedBrick SDK in Python

Nearly all operations with the SDK are performed on either the Project or Organization objects. You can instantiate these objects using your API Key, Org ID, and Project ID.

```python
import redbrick

project = redbrick.get_project(
    org_id="...",
    project_id="...",
    api_key="...",
)
organization = redbrick.get_org(
    org_id="...",
    api_key="...",
)
```

Both redbrick.get_project and redbrick.get_org take an optional `url` argument that defaults to https://api.redbrickai.com.

If you are using a private/single-tenant deployment of RedBrick AI, this will need to be changed for your deployment - reach out to us for confirmation of what your case-specific URL needs to be.

Importing Data & Annotations

The RedBrick AI SDK allows you to programmatically import data and/or annotations with a Python script. You can import either locally or externally stored data via the SDK using the `create_datapoints` method.

> Please see the full reference documentation for `create_datapoints` here.

Perform the standard RedBrick AI SDK setup to create a Project object.

```
project = redbrick.get_project(org_id, project_id, api_key)
```

Import locally stored images

To import locally stored data, create a list of `points` with relative file paths to your locally stored data, and use the `redbrick.StorageMethod.REDBRICK` storage ID.

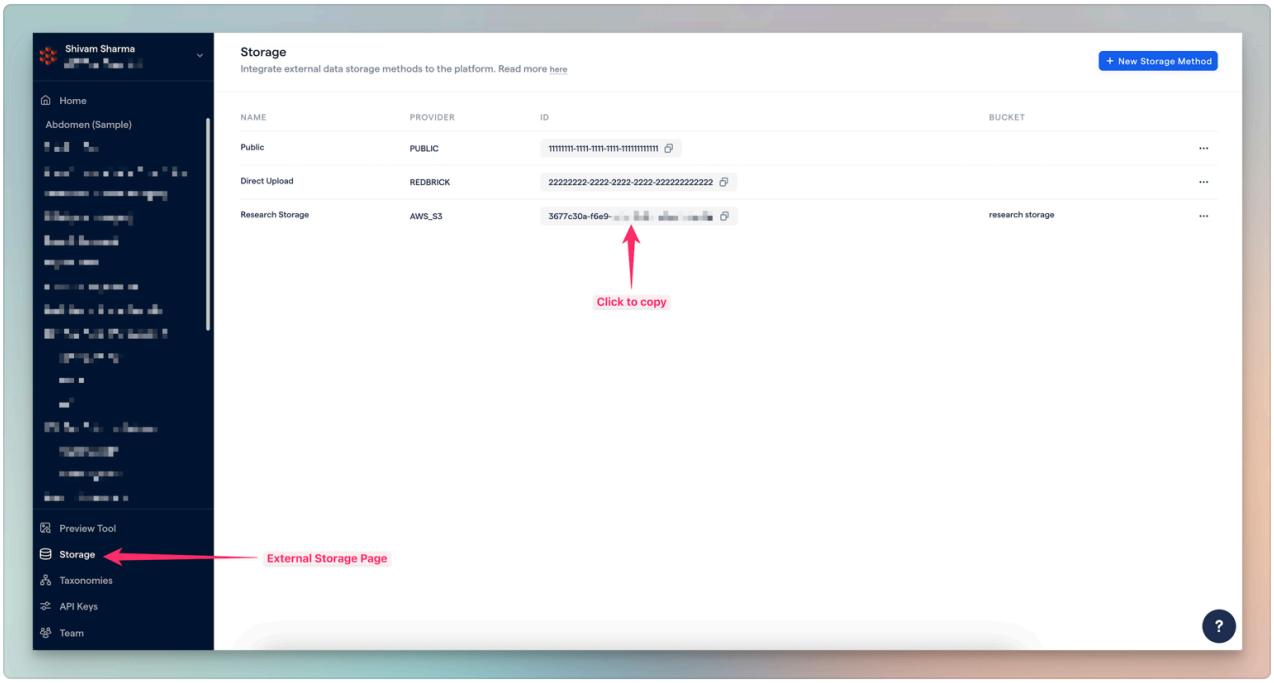
```
# create a list of file paths to your locally stored data
points = [{"items": ["path/to/data.nii"], "name": "..."}]

# perform the upload operation
project.upload.create_datapoints(
    storage_id=redbrick.StorageMethod.REDBRICK,
    points=points
)
```

> The points array follows the format of the items list.

Import externally stored images

To import data stored in an external storage method such as AWS s3, be sure to use the Storage ID found on the Storage tab of your RedBrick AI account instead of

`redbrick.StorageMethod.REDBRICK`.



Click on the field to copy the Storage ID to your clipboard!

Visit our documentation on external storage to learn how to integrate your own external storage like AWS s3, GCS, or Azure blob.

Import annotations

First, please follow our guide on preparing your items list for annotation import to prepare your points object correctly.

Importing Annotations Guide ›

Importing locally stored annotations & images

```
project.upload.create_datapoints(
    storage_id=redbrick.StorageMethod.REDBRICK,
    points=points
)
```

Importing locally stored annotations with externally stored images

```
project.upload.create_datapoints(
    storage_id="your_storage_id",
    label_storage_id=redbrick.StorageMethod.REDBRICK
    points=points
)
```

Importing externally stored annotations and images

```
project.upload.create_datapoints(
    storage_id="your_storage_id",
    label_storage_id="your_storage_id"
    points=points
)
```

Programmatic Label & Review

It may be useful to programmatically add labels to your uploaded data or perform a review on queued tasks. This scenario may arise if you have an automated way of reviewing data or if you want to bulk-process tasks.

> Please see the detailed reference documentation for `put_tasks` here.

> You can only use `put_tasks` on Tasks **assigned to your API key.**
>
> Please consult our documentation to learn more about how to assign Tasks to your API key.

First, perform the standard RedBrick AI SDK set-up to create a project object.

```
project = redbrick.get_project(org_id, project_id, api_key)
```

Next, you need to get a list of Tasks you want to label/review. You can do this by:

1. Searching for the `task_id` through the RedBrick AI UI.
2. Retrieving the `task_id` from your filename/custom `name` from the Items List using search_tasks.
3. Retrieving tasks assigned to your API key using `list_tasks`.

Programmatically Label Tasks

Add your annotations within the `series` field, along with the `task_id`. Please refer to the reference documentation for the format of the annotations in Series.

The corresponding Task must be queued in the Label Stage and assigned to your API key.

```
tasks = [
    {
        "taskId": "...",
        "series": [{...}]
    },
]

# Submit tasks with new labels
project.labeling.put_tasks("Label", tasks)

# Save tasks as draft with new labels
project.labeling.put_tasks("Label", tasks, finalize=False)
```

Programmatically Review Tasks

Add your review decision in the `review_result` argument, along with the `task_id`. The corresponding Task must be queued in the Review stage that you specify in `stage_name` and must be assigned to your API key.

```
# Set review_result to True if you want to accept the tasks
project.review.put_tasks("Review_1", [{taskId: "..."}], review_result=Tru

# Set review_result to False if you want to reject the tasks
project.review.put_tasks("Review_1", [{taskId: "..."}], review_result=Fal

# Add labels if you want to accept the tasks with correction
project.review.put_tasks("Review_1", [{taskId: "...", series: [{...}]}])
```

Re-annotate Ground Truth Tasks

Once your Task goes through all of the stages in your workflow, it will be stored in the Ground Truth Stage. If you notice issues with one or more of your Ground Truth Tasks, you can either modify them manually within the UI while the Tasks are still in the Ground Truth Stage or **send them back to the Label Stage** for correction.

First, get a list of the `task_id`s you want to send back to Label. You can do this by exporting only Ground Truth Tasks and filtering them. Then, use `move_tasks_to_start` to send them back to Label.

```
task_ids = ["...", "..."]
project.labeling.move_tasks_to_start(task_ids=task_ids)
```

All corresponding Tasks need to be in the Ground Truth Stage. This function will not work for Tasks queued in Review.

Assigning & Querying Tasks

The SDK offers multiple ways to query/search through your project tasks and programmatically assign them to various users.

Search by Task Name

Use `list_tasks` to search for tasks by `name` and get their corresponding `task_id`. Often, users will have Task `name`s readily accessible, and can use `list_tasks` to get the corresponding `task_id,` which may be needed in other SDK functions.

> ✔ Please see a detailed reference for `list_tasks` here.

```
project = redbrick.get_project(org_id, project_id, api_key)

tasks = project.export.list_tasks() # fetches all tasks
specific_task = project.list_tasks(task_name="...") # fetches specific ta
```

Assign Tasks to a User

Use `assign_task` when you already have the `task_id` you want to assign to a particular user. If you don't have the `task_id`, you can query all the Tasks using `list_tasks` or query tasks assigned to a particular user/unassigned tasks using `list_tasks(user_id="...")`.

Assign to a Specific User

```
project = redbrick.get_project(org_id, project_id, api_key)

# Assign tasks in Label stage to a specific user
project.labeling.assign_tasks(task_ids=["..."], email="...")

# Assign tasks in Review stage to specific user
project.review.assign_tasks(task_ids=["..."], email="...")
```

Retrieve Queued Tasks

Use `list_tasks` in conjunction with a specific `user_id` when you want to retrieve the Tasks assigned to a particular user. This can be useful in preparation for using `assign_tasks` to programmatically assigning unassigned tasks, or `put_tasks` to programmatically label/review tasks assigned to you.

Retrieve Tasks Assigned to Specific User

```
project = redbrick.get_project(org_id, project_id, api_key)

# Get Tasks assigned to email@email.com in Label Stage
project.export.list_tasks(labeling.(stage_name="Label", user_id="email@ema

# Get Tasks assigned to email@email.com in Review_1 Stage
project.export.list_tasks(stage_name="Review_1", user_id="email@email.com
```

Retrieve Unassigned Tasks

You can also fetch all unassigned Tasks in a particular stage. This information may be useful when choosing which Tasks to assign to users.

```
project = redbrick.get_project(org_id, project_id, api_key)

# Get unassigned tasks in Label labeling stage
project.export.list_tasks(redbrick.TaskFilters.UNASSIGNED, stage_name="Lab

# Get unassigned tasks in Review_1 review stage
project.export.list_tasks(redbrick.TaskFilters.UNASSIGNED, stage_name="Rev
```

Retrieve Tasks Assigned to You

With the correct configuration of list_tasks(), you can perform functions as specific as retrieving a list of Tasks from a specific Stage to your specific API key. Please see the code snippet below for an example:

```
project = redbrick.get_project(org_id, project_id, api_key)

# Get tasks assigned to your API key in Label stage
project.export.list_tasks(
    redbrick.TaskFilters.QUEUED,
    stage_name="Label",
    user_id=project.context.key_id
    )
```

Exporting Annotations

You can make use of RedBrick AI's Python SDK to export your annotations using a Python script.

Within the Python SDK, annotations are exported in two ways:

1. The `export_tasks` function **returns a Python object** containing meta-data information and any vector annotations (measurements, landmarks, etc.). Please see the format of the object here.
2. By default, segmentation data is written to your disk **in NIfTI format**. Segmentation data can also be exported in PNG or RT Struct by manipulating the parameters of the `export_tasks` function. Please view the detailed `export_tasks` reference here.

> If you're attempting a one-time export or don't have intensive requirements for your export, the **CLI** also provides a simple and optimized workflow for exporting a Project's annotations.

Export Folder Structure

RedBrick AI exports annotations in a JSON structure, accompanied by NIfTI-1 masks for segmentations. All data will be exported within a folder named after your `project_id`, with the following structure:

```
project_id/
├── segmentations
│   ├── study01
│   │   └── series1.nii
│   └── study02
│       ├── series1.nii
│       └── series2.nii
└── tasks.json
```

- [x] The above structure is for a standard export (i.e. not semantic, not binary mask, etc.) and assumes no overlapping segmentations.

Segmentations Subdirectory

The segmentation directory will contain a single sub-directory for each task in your export. The sub-directories will be named after the task `name`. A single task (depending on whether it was single series or multi-series) can have one or more segmentations.

The individual segmentation files will be in NIfTI-1 format and be named after the user-defined series name. If no series name is provided on upload, RedBrick will assign a unique name. Corresponding meta-data ex. category names will be provided in tasks.json.

Code Examples

As always, you should first perform the standard RedBrick AI SDK setup to create a Project object.

```
project = redbrick.get_project(org_id, project_id, api_key)
```

With a new Project object created, you can export your Project's Tasks in various ways. Please see some common examples below.

Export All Tasks

The `export_tasks()` function exports segmentation files for all Ground Truth Tasks by default. To export All Tasks, set the `only_ground_truth` parameter to `False`.

```
annotations = project.export.export_tasks(only_ground_truth=False)
```

Export Only Ground Truth

You can export only the Tasks in Ground Truth, i.e., Tasks that have successfully made it through all Label and Review Stages.

```
gt_annotations = project.export.export_tasks(only_ground_truth=True)
```

Export Specific Tasks

Export selected Tasks by specifying Task IDs.

```
specific_annotations = project.export.export_tasks(task_id="...")
```

Generate an Audit Trail

An audit trail can be useful for regulators interested in your quality control processes, as well as for managing your internal QA processes.

> ☑ Please see a detailed reference for `get_task_events` here.

First, perform the standard RedBrick AI SDK setup to create a Project object.

Audit Trail - All Tasks

If you'd like to generate an audit trail for **all Tasks** (not only those in the Ground Truth Stage), be sure to include the `only_ground_truth=False` parameter.

```
# Return an audit trail for all Tasks in all Stages
audit_trail = project.export.get_task_events(only_ground_truth=False)
```

Audit Trail - Ground Truth Tasks Only

Retrieve an audit trail for all Ground Truth Tasks**.** Please note that by default, `get_task_events` only returns audit information for Tasks in the Ground Truth Stage.

```
project = redbrick.get_project(org_id, project_id, api_key)

# Return an audit trail for all Tasks currently in the Ground Truth Stage
audit_trail = project.export.get_task_events()
```

The returned object will contain data similar to the code snippet below, where each entry will represent a single Task (uniquely identified by `taskId`). The `events` array contains all key events/actions performed on the Task, with `events[0]` being the first event.

```
[
  {
    "taskId": "...",
    "currentStageName": "Label",
    "events": [
      {
        "eventType": "TASK_CREATED",
        "createdAt": "...",
        "isGroundTruth": false,
        "createdBy": "..."
      },
      {
        "eventType": "TASK_ASSIGNED",
        "createdAt": "...",
        "assignee": "...",
        "stage": "Label"
      }
    ]
  }
]
```

Additional Capabilities

The following is a non-exhaustive list of other available functionalities when using the `Export` class. A full list of the capabilities of our `Export` class can be found here.

- Track labeler or reviewer time spent on a Task with `get_active_time()`;

- Fetch Task events from a specific timestamp to the present day using `get_task_events()` and the `from_timestamp` parameter;
- Easily search for Tasks based on a wide variety of criteria using `list_tasks()`;
- Perform a semantic export (that exports a single file per category name) using `export_tasks()`;
- Configure Hanging Protocols;
- Upload a script for Custom Label Validation;

CLI Overview

The RedBrick AI Command Line Interface is a package that allows developers to interact with the RedBrick AI application programmatically.

We recommend you use the CLI for most regular import & export actions. If you want to write Python scripts to perform these actions, or you are interested in more advanced scripting, please use our Python SDK.

This documentation covers high-level guides and usecases. If you are interested in more detailed documentation of the CLI interface, please visit the full CLI reference documentation.

Configure your CLI credentials

Once you have installed the CLI, you need to configure your credentials.

```
$ redbrick config
> Org ID: ...
> API KEY: ...
> URL: https://api.redbrickai.com
> Profile name: ...
✔ RedBrick AI Organization
```

> The URL should default to https://api.redbrickai.com. If you are using a private/single-tenant deployment of RedBrick AI, this will need to be changed for your deployment - reach out to us for confirmation on what the URL needs to be.

For most scenarios, you will only need a single credentials profile. However, if you want to create multiple profiles (perhaps for different organizations), you can do it in the following way:

```
$ redbrick config add
```

To change your profile:

```
$ redbrick config set
```

Creating & Cloning Projects

Almost all CLI operations need to be performed within a *local project* directory. A local project directory can be created by cloning a RedBrick AI project, or by creating a new project using the CLI.

Creating a new project

To create a new project, first navigate to an empty directory. We recommend *creating a new* directory, and naming it after your new project.

```
$ mkdir my-new-project
$ cd my-new-project
```

Now to create a new project, simply run:

```
$ redbrick init
> Name: my-new-project
> Taxonomy: my-taxonomy-name
> Reviews: 1
```

You can now verify your current directory is a *local project directory* by doing:

```
$ redbrick info
                Organization
┌──────┬──────────────────────────┐
│ ID   │ ...                      │
│ Name │ My Organization          │
└──────┴──────────────────────────┘

                    Project
┌──────────┬──────────────────────────────────────────────┐
│ ID       │ ...                                          │
│ Name     │ my-new-project                               │
│ Taxonomy │ my-taxonomy-name                             │
│ URL      │ https://app.redbrickai.com/.../projects/.../ │
└──────────┴──────────────────────────────────────────────┘
```

Clone an existing project

You can clone an existing project that you (or someone else) created.

```
$ redbrick clone
> Project:
❯ 3/3
❯ my-new-project (...)
  my-first-project (...)
  my-old-project (...)
```

You can directly clone a project using it's project ID.

```
$ redbrick clone PROJECTID # replace PROJECTID with your project's ID
```

The project will now be cloned **in a directory named after your project** (within your current working directory).

Import Data & Annotations

You can easily import large amounts of data from the command line interface. Before following this guide, make sure to set up credentials for the CLI.

Importing locally stored images

Upload using an items file

To upload images that are stored in a non-conventional folder structure, you can define the structure using an items JSON file and upload it like this.

```
$ redbrick upload path/to/items.json
```

If you don't want to use an items file for upload, ensure your data is stored within the correct folder structure defined in our documentation. You can only upload a single data type in one upload operation. See the supported file types here.

```
$ redbrick upload path/to/data/ --type DICOM3D
```

You can see all available types in the CLI upload reference documentation.

Group images by study

To group your images by study, see here for examples, input the following:

```
$ redbrick upload path/to/data/ --as-study
```

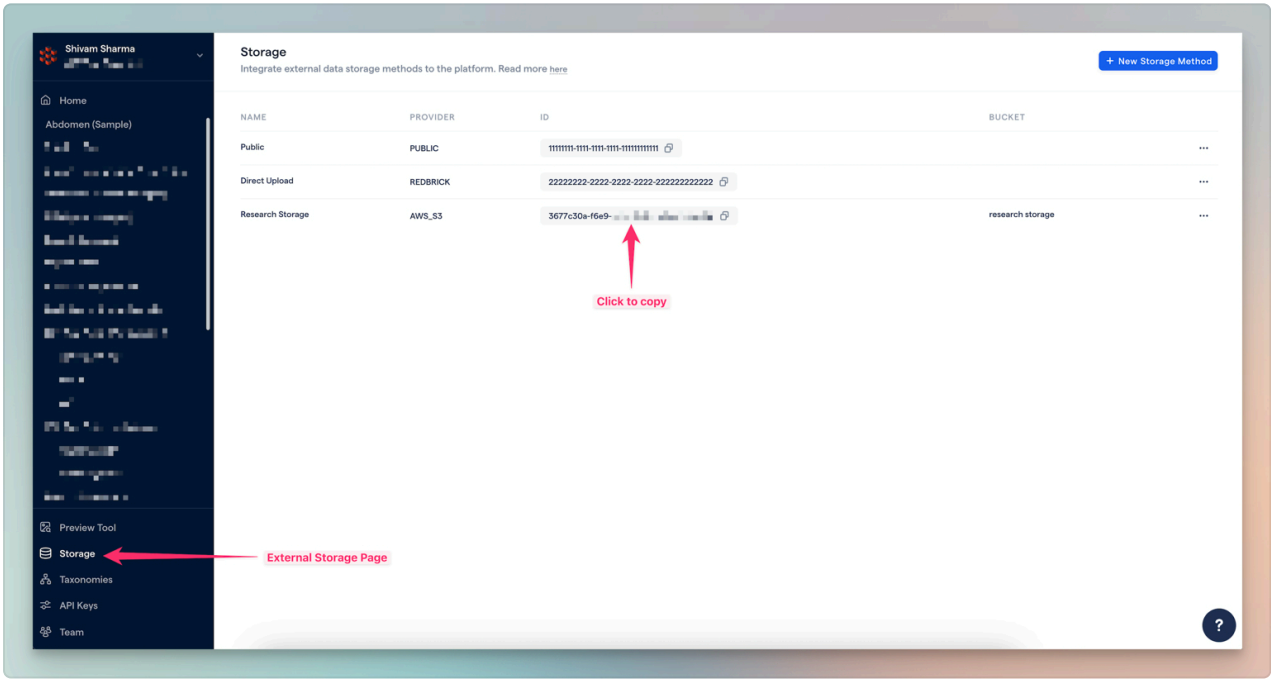
Upload video frames

To upload a video by uploading individual frames, input the following:

```
$ redbrick upload path/to/videoframes/ --as-frames --type VIDEOFRAMES
```

Importing externally stored images

To import data that is stored externally (e.g., in an AWS s3 bucket), you must specify the storage ID. You can find your storage solution's unique Storage ID in the *Storage tab* of the RedBrick AI platform.



Click on the field in order to copy the value to your clipboard.

Prepare an Items List containing references to your externally stored files.

```
$ redbrick upload items.json --storage STORAGEID # replace STORAGEID with
```

Import annotations

To import annotations with your data, you must first create an items JSON file following the import annotations guide.

Importing Annotations Guide >

Import locally stored annotations & images

```
$ redbrick upload path/to/items.json # items.json must have local file pa
```

Import locally stored annotations with externally stored images

The following command will upload your (locally stored) annotation files and your image files (stored in `STORAGEID`):

```
$ redbrick upload path/to/items.json --storage STORAGEID
```

Import externally stored annotations & images

If your annotation files are also stored externally, you can run the following command:

```
$ redbrick upload path/to/items.json --storage STORAGEID --label-storage
```

Exporting Annotations

Overview

The RedBrick CLI allows you to easily export your Project's annotations within a local project directory.

Please note that the `export` function only fetches **newly created annotations** when run. It will not generate an annotation file for a Task if no annotation work has been completed and saved on said Task.

For example, if you upload 100 images to your Project, annotate 80 of them and initiate an export using the CLI, the CLI will export **annotations for 80 Tasks**.

If your team annotates 5 additional Tasks the next day and initiates an export, the CLI will only export annotations for the **5 newly annotated Tasks**, bringing the total number of annotation files in your local directory to **85**.

- [x] By default, all segmentation files are exported in **NIfTI-1 format**. Please see our Format Reference for more information on exported annotations and alternative formats (such as PNG or RT Struct).

Export Folder Structure

RedBrick AI exports annotations in a JSON structure, accompanied by NIfTI-1 masks for segmentations. All data will be exported within a folder named after your `project_id`, with the following structure:

```
project_id/
├── segmentations
│   ├── study01
│   │   └── series1.nii
│   └── study02
│       ├── series1.nii
│       └── series2.nii
└── tasks.json
```

Segmentations Subdirectory

The segmentation directory will contain a single sub-directory for each task in your export. The sub-directories will be named after the task `name`. A single task (depending on whether it was single series or multi-series) can have one or more segmentations.

The individual segmentation files will be in NIfTI-1 format and be named after the user-defined series name. If no series name is provided on upload, RedBrick will assign a unique name. Corresponding meta-data ex. category names will be provided in tasks.json.

Export Annotations to a Local Directory using the CLI

> You can also find all of these steps, as well as pre-configured CLI commands, inside the "Export Labels" section of your Project Settings

To export your data, first ensure that your credentials file has been properly configured and you have created a local project directory.

Next, navigate to the newly created Project directory.

```
$ cd my-project
```

Once inside your local project directory, you can initiate several types of exports. Please see some common examples below or use `redbrick export -h` to see a full list of export-related commands inside of the Terminal.

Export Annotations for All Tasks

To export the latest state of all annotations for all Tasks (including those in Label and Review stages) run the following command.

```
$ redbrick export
```

Export Ground Truth Tasks Only

For exporting only those annotations associated with Tasks in the Ground Truth Stage.

```
$ redbrick export groundtruth
```

Export Tasks and Clear Cache

For clearing your local Redbrick cache and forcing a fresh download of all annotation files within a Project.

```
$ redbrick export --clear-cache
```

Export Tasks with Images

For downloading your Project's image and/or volume files along with any created annotations.

```
$ redbrick export --with-images
```

DICOM to NIfTI Conversion

If you initially uploaded DICOM images/volumes to RedBrick and would like to convert them to NIfTI upon export (ensuring that both your annotation files and images/volumes are in the same format), use the following command.

```
$ redbrick export --with-images --dicom-to-nifti
```

Export Tasks from a Specific Stage

If you want to export tasks that are queued in a specific stage, for example, exporting all tasks queued in Review_2, you can do so in the following way:

```
$ redbrick export --stage Review_2
```

Export an Audit Trail

Generating an audit trail can be useful material for regulators interested in your quality control processes and for managing your internal QA processes.

You can create such a report by running the following command within your local project directory.

```
$ redbrick report
```

The exported JSON object will contain data similar to what is shown below. Each entry will represent a single task (uniquely identified by `taskId`). The `events` array contains all key events/actions performed on the task, with `events[0]` being the first event.

```
[
  {
    "taskId": "...",
    "currentStageName": "Label",
    "events": [
      {
        "eventType": "TASK_CREATED",
        "createdAt": "...",
        "isGroundTruth": false,
        "createdBy": "..."
      },
      {
        "eventType": "TASK_ASSIGNED",
        "createdAt": "...",
        "assignee": "...",
        "stage": "Label"
      }
    ]
  }
]
```

Importing Annotations Guide

You can import all annotation types that are supported in RedBrick AI, including segmentations, classifications, bounding boxes, and more. Imported annotations will appear automatically on your annotator's interface.

> Annotations and images must be imported together at the start.
>
> If you want to add annotations programmatically to images that have already been uploaded, please use the programmatic label & review.

> Annotation import is only supported through the SDK and CLI.
>
> That is, you cannot use the direct upload UI to import annotations, and you must use the items list with the SDK/CLI to provide the required metadata along with annotations.

To import images along with segmentations, you must provide us with:

1. Images in any supported format and NIfTI segmentation files.
2. An items list that provides a mapping of:
   - Segmentation files to volumes so that segmentations are applied to correct images.
   - Values within segmentation file to taxonomy categories.

> Once you've prepared the items list in the format defined below, you can import the images and annotations using the `create_datapoints` SDK method or CLI `upload` method.

Items list for importing segmentations

You can find the full format reference here. In this section, we will focus on importing segmentations. In the examples below, pay attention to the following fields:

1. `segmentations`: The segmentation files to be applied to the task.
2. `segmentMap`: Map the values present in the segmentation files to their corresponding taxonomy categories.

I: One segmentation file per task

```
{
    "name": "...",
    "series": [
        {
            "items": ["instance-01.dcm", "instance-02.dcm", ...],
            "segmentations": "segmentation.nii.gz",

            // Read more about "segmentMap"
            "segmentMap": {
                "1": "category-a",
                "2": "category-b"
            }
        }
    ]
}
```

II: Multiple segmentation files per task

Sometimes, segmentations for a single volume are stored in multiple segmentation files, but these segmentation files are not binary masks. In this case, follow the format below.

```
{
    "name": "...",
    "series": [
        {
            "items": ["instance-01.dcm", "instance-02.dcm", ...],

            // Read more about "segmentations"
            "segmentations": ["segmentation-1.nii.gz", "segmentation-2.ni

            // Read more about "segmentMap"
            "segmentMap": {
                "1": "category-a",
                "2": "category-b"
            }
        }
    ]
}
```

Common mistakes for I and II.

- The values 1 and 2 must be present in either `segmentation-1.nii.gz` or `segmentation-2.nii.gz.`
- Values in `segmentation-1.nii.gz` & `segmentation-2.nii.gz` that are not in `segmentMap` will not map to any taxonomy category. This will result in uneditable, view-only annotations.
- All values in `segmentation.nii.gz` that are not in `segmentMap` will not be mapped to any taxonomy category in the editor.

III: Multiple binary segmentation files per task

A common pattern is to store each segmentation instance in a separate NIfTI file as a binary mask. In the example below, all non-zero values in `segmentation-1.nii.gz` are meant to correspond to the taxonomy category `category-a`.

```
{
    "name": "...",
    "series": [
        {
            "items": ["instance-01.dcm", "instance-02.dcm", ...],

            // Read more about "segmentations"
            "segmentations": ["path/segmentation-1.nii.gz", "path/segment
            "segmentMap": {

                // Read more about "1"
                "1": {

                    // Read more about "category"
                    "category": "category-a",

                    // Read more about "mask"
                    "mask": "path/segmentation-1.nii.gz",
                },
                "2": {
                    "category": "category-b",
                    "mask": "path/segmentation-2.nii.gz"
                },

                // Read more about "binaryMask"
                "binaryMask": true,
            }
        }
    ]
}
```

Formats

Full Format Reference

Items List and `tasks.json`

Most RedBrick flows incorporate two key JSON files:

1. Items List - a file which points RedBrick AI to visual assets within a third-party storage solution;
2. `tasks.json` - a file generated upon export that contains a record of the annotation work completed within a Project. Upon export, the `tasks.json` file will contain a **single entry for each Task**;

If you'd like to upload annotations along with your data using either the CLI or the SDK, please see the corresponding documentation.

Object Reference

Please see the definition (in TypeScript) of RedBrick's various objects below:

```
type Tasks = Task[];

// Single task on RedBrick can be single/multi-series
type Task = {
  // Required on upload and export
  name: string;
  series: Series[];

  // Task level annotation information
  classification?: Classification;

  // Not required on upload, present in tasks.json
  taskId?: string;
  currentStageName?: string;
  createdBy?: string;
  createdAt?: string;
  updatedBy?: string;
  updatedAt?: string;

  // assign metadata to a Task on upload, present in tasks.json
  metaData?: { [key: string]: string }

  // assign priorities to a Task on upload
  priority?: number;

  // Prescribe task assignent upon upload
  preAssign?: {
    [stageName: string]: string
  }
};

// How to correctly format a Series object for upload
// A single series can be 2D, 3D, video etc.
// Also present in tasks.json
type Series = {
  items: string | string[];
  name?: string;

  segmentations?: string | string[];
  segmentMap?: {
    [instanceId: string]: number | string | string[] | {
      category: number | string | string[];
      attributes?: Attributes;
      mask?: string;
    };
  };
  binaryMask?: boolean;
  semanticMask?: boolean;
```

```
  pngMask?: boolean;

  landmarks?: Landmark[];
  landmarks3d?: Landmark3D[];
  measurements?: (MeasureLength | MeasureAngle)[];
  boundingBoxes?: BoundingBox[];
  cuboids?: Cuboid[];
  ellipses?: Ellipse[];
  polygons?: Polygon[];
  polylines?: Polyline[];

  classifications?: Classification[];
  instanceClassifications?: InstanceClassification[];
   metaData?: { [ key: string ]: string };
};

// Label Types

type Landmark = {
  point: Point2D;
  category: number | string | string[];
  attributes?: Attributes;

  // video meta-data
  video?: VideoMetaData;
};

type Landmark3D = {
  point: VoxelPoint;
  category: number | string | string[];
  attributes?: Attributes;
};

type MeasureLength = {
  type: 'length';
  point1: VoxelPoint;
  point2: VoxelPoint;
  absolutePoint1: WorldPoint;
  absolutePoint2: WorldPoint;
  normal: [number, number, number];
  length: number;
  category: number | string | string[];
  attributes?: Attributes;
};

type MeasureAngle = {
  type: 'angle';
  point1: VoxelPoint;
```

```
  point2: VoxelPoint;
  vertex: VoxelPoint;
  absolutePoint1: WorldPoint;
  absolutePoint2: WorldPoint;
  absoluteVertex: WorldPoint;
  normal: [number, number, number];
  angle: number;
  category: number | string | string[];
  attributes?: Attributes;
};

type BoundingBox = {
  pointTopLeft: Point2D;
  wNorm: number;
  hNorm: number;
  category: number | string | string[];
  attributes?: Attributes;
  stats?: MeasurementStats;

  // video meta-data
  video?: VideoMetaData;
};

type Cuboid = {
  point1: VoxelPoint;
  point2: VoxelPoint;
  absolutePoint1: WorldPoint;
  absolutePoint2: WorldPoint;
  category: number | string | string[];
  attributes?: Attributes;
  stats?: MeasurementStats;
}

type Ellipse = {
  pointCenter: Point2D;
  xRadiusNorm: number;
  yRadiusNorm: number;
  rotationRad: number;
  category: number | string | string[];
  attributes?: Attributes;
  stats?: MeasurementStats;

  // video meta-data
  video?: VideoMetaData;
}

type Polygon = {
  points: Point2D[];
```

```
  category: number | string | string[];
  attributes?: Attributes;
  stats?: MeasurementStats;

  // video meta-data
  video?: VideoMetaData;
};

type Polyline = {
  points: Point2D[];
  category: number | string | string[];
  attributes?: Attributes;

  // video meta-data
  video?: VideoMetaData;
};

type Classification = {
  category: number | string | string[];

  // video meta-data
  video?: VideoMetaData;
};

type InstanceClassification = {
  fileIndex: number;
  fileName?: string;
  values: { [attributeName: string] : boolean};
}

type Attributes =
  | { // Taxonomy V2 attributes
      attrId?: number;
      name?: string;
      optionId?: number | number[];
      value?: string | boolean | string[];
    }[]
  | { // Taxonomy V1 attributes
      [attributeName: string]: string | boolean | string[];
    };

type VideoMetaData = {
  frameIndex: number;
  trackId?: string;
  keyFrame?: number;
  endTrack?: Boolean;
};
```

```
// i is rows, j is columns, k is slice
type VoxelPoint = {
  i: number;
  j: number;
  k: number;
};
// The position of VoxelPoint in physical space (world coordinate) comput
type WorldPoint = {
  x: number;
  y: number;
  z: number;
};
type Point2D = {
  xNorm: number;
  yNorm: number;
};

type MeasurementStats = {
  average: number;
  area?: number;
  volume?: number;
  minimum: number;
  maximum: number;
};
```

Object Glossary

Task

The `Task` object represents a single task on RedBrick AI. It contains task-level meta-data information about all the series within the task. A task can contain a single series or multiple series (ex. a full MRI study).

`name: string`

A user-defined string is defined on upload, it is *required to be unique* across all tasks in your project. The `name` is meant to be a human-readable string that can help identify tasks ex. you can set `name` of a task to `patient/study01`.

`taskId?: string`

A unique identifier generated for each Task by RedBrick AI. This value is provided on export.

`currentStageName?: string`

The Stage a Task is in is when it is exported.

`createdBy?: string`

The email address of the user who uploaded the Task.

`createdAt?: string`

The datetime this Task was created (i.e. uploaded).

`updatedBy?: string`

The email of the last user to make edits to this Task.

`updatedAt?: string`

The datetime that this Task was last edited.

`preAssign?: { [ stageName : string ] : string }`

When uploading a Task, prescribe which users will have the Task assigned to them by Stage.

You can define the assignment for each Stage of the workflow, for example, Label and Review, `{"Label": "name1@redbrickai.com", "Review": "name2@redbrickai.com"}`.

`classification: { attributes : [ string : boolean ] }`

A list of attributes assigned to an entire Task (or study, if the Task encapsulates an entire study).

`metaData?: { [ key: string ]: string }`

A list of key value pairs that can be affixed to a Task. This information is visible in the Annotation Tool.

`priority?: number`

Assign a priority value to a specific Task, which will influence the order in which the Task displays on the Data Page. The Automatic Assignment protocol will also auto-assign Tasks with a priority value to a user's Labeling / Review Queue before moving on to Tasks without priority values.

Series

The `Series` object has metadata and annotations for a single Series within a Task. A Series can represent anything from a single MRI/CT series, a video, or a single 2D image.

If a Series contains annotations, you can expect one or more of the label entries to be present (e.g. `segmentations`, `polygons` etc.).

`items: string | string[]`

The items entry is a list of file paths that point to your data. Please have a look at the #items-listdocumentation for a fuller explanation of how to format for various modalities and series/study uploads.

`name: string`

An optional user-defined string, *needs to be unique* across all series. Individual series will be named after this value on the labeling tool. Exported segmentation files will also be named using this value. Using the Series Instance UID here is good practice.

`classifications: { attributes: [ string : boolean ] }`

A list of attributes assigned to a specific Series.

```
instanceClassifications: fileIndex | fileName | [ values: { string : boolean } ]
```

The `instanceClassifications` object defines a series of boolean values that can be assigned to individual instances (e.g. frames in a video).

`metaData?: { [ key: string ]: string }`

A list of key value pairs that can be affixed to a Series. This information is visible in the Annotation Tool.

`binaryMask?: boolean`

Reflects the user's choice of optionally exporting annotations as a binary mask.

`semanticMask?: boolean`

Reflects the user's choice of optionally using semantic export.

`pngMask?: boolean`

Reflects the user's choice of optionally exporting annotations as a PNG mask.

Common Label Keys

Here are the definition for some common entries present in some/all label entries.

`category: string | string[]`

The class of your annotations. This value is part of your Project Taxonomy. If the class is nested, `category` will be `string[]`.

`attributes: { [ attributeName: string ]: string | boolean }`

Each annotation can have accompanying attributes, that are also defined in your Project Taxonomy. `attributeName` is defined when creating your Taxonomy.

`voxelPoint: { i: number, j: number, k: number }`

`VoxelPoint` represents a three-dimensional point in image space, where i and j are columns and rows, and k is the slice number.

`worldPoint: { x: number, y: number, j: number }`

`WorldPoint` represents a three-dimensional point in physical space/world coordinates. The world coordinates are calculated using `VoxelPoint` and the Image Plane Module.

`point2D: { xnorm: number, ynorm: number }`

`Point2D` represents a two dimensional point. This is used to define annotation types on 2D data. `xnorm` has been normalized by image width, `hnorm` has been normalized by image height.

`fileIndex: number`

`fileIndex` is an integer that corresponds to a specific frame in a video series.

`fileName: string`

`fileName` represents the name given to an image or specific frame in a video series.

`measurementStats: Dict`

A dictionary containing a variety of geometric information about certain Object Labels.

Measurement Stats

A dictionary (`measurementStats`) containing geometric information about certain Object Labels.

`average: number`

The average pixel intensity value inside of a structure.

`area?: number`

The area of a 2D Object Label (e.g. Bounding Box, Ellipse), measured in square millimeters.

`volume?: number`

The volume of a 3D structure (e.g. Cuboid), measured in cubic millimeters.

`minimum: number`

The lowest pixel intensity value present in the structure.

`maximum: number`

The highest pixel intensity value present in the structure.

Video Meta Data

`videoMetaData: Dict`

A dictionary containing `frameIndex` , `trackId` , `keyFrame` , and `endFrame` .

`frameIndex: number` (video)

This specifies which frame in a video sequence the annotation was created.

`trackId: string` (video)

A unique string that identifies distinct object tracks in a video sequence.

`keyFrame: boolean` (video)

If true, this annotation was manually added on a particular video sequence. If false, this annotation is a result of linear interpolation.

`endFrame: boolean` (video)

If true, the annotation is the last annotation for a particular video track segment.

Segmentations and `segmentMap`

`segmentations?: string | string[]`

A list of file paths of segmentation files for this series. Either a single `.nii` file, or multiple `.nii` files containing different instances.

```
segmentMap?: { [ instanceId: number ]: { category: string | string[];
attributes?: Attributes } };
```

A mapping between a segmentation's instance ID, your Taxonomy category name, and any accompanying attributes. The mapping will apply only to the current series, and instance IDs must be unique across all series in a task (this is useful for instance segmentation).

Please note that the `segmentMap`'s instanceId is generated **incrementally based on the order in which annotations were created by the labeler**. You can find an example JSON output below.

```
"items": ["image-file.ima",],
"segmentations": "./path/to/segmentation/file.nii.gz",
"segmentMap": {
  "1": {
    // This is the first annotation the labeler created
    "category": "Vertebral Body"
  },
  "2": {
    // This is the second annotation the labeler created
    "category": "Vertebral Body"
  },
  "3": {
    // This is the third annotation the labeler created
    "category": "Vertebral Body"
  },
  "4": {
    // This is the fourth annotation the labeler created
    "category": "Spinal Canal Mass"
  },
  "5": {
    // This is the fifth annotation the labeler created
    "category": "Disc Pathology"
  }
},
```

`mask?: string`

The path for the annotation file associated with a specific instanceId.

BoundingBox

Contains information about the Bounding Box Object Label.

`pointTopLeft:` `Point2D`

The location of the top-left point of the bounding box.

`wNorm, hNorm: number`

The width and height of the bounding box, normalized by the width and height of the image.

Polygon

Contains information about the Polygon Object Label.

`points:` `Point2D` `[]`

A list of 2D points that are connected to form a polygon. This list is ordered such that, $point_i$ is connected to $point_{i+1}$. The last point is also connected to the first point to close the polygon.

MeasureLength

Contains information about the Length Measurement Object Label.

`point1, point2 :` `VoxelPoint`

A length measurement is defined by two points, and the length measurement is the distance between the two points.

`absolutePoint1, absolutePoint2 :` `WorldPoint`

Corresponding to `point1`, `point2` these are points in physical space.

`normal: [number, number, number]`

Measurements can be made on oblique planes. `normal` defines the normal unit vector to the slice on which this annotation was made. For annotations made on non-oblique planes, the normal will be `[0,0,1]`.

`length: number`

The value of the measurement in mm.

MeasureAngle

Contains information about the Angle Object Label.

`point1, point2, vertex :` `VoxelPoint`

Angle measurement is defined by three points, where the vertex is the middle point. The angle between the two vectors (vertex -> point1 and vertex -> point2) defines the angle measurement. These points are all represented in IJK image coordinate space.

`absolutePoint1, absolutePoint2 :` `WorldPoint`

Corresponding to `point1`, `point2`, `vertex`, these values are coordinates in the DICOM world coordinate system i.e. physical space.

`normal: [number, number, number]`

Measurements can be made on oblique planes. `normal` defines the normal unit vector to the slice on which this annotation was made. For annotations made on non-oblique planes, the normal will be `[0,0,1]`.

`angle: number`

The value of the angle in degrees.

Ellipse

Contains information about the Ellipse Object Label.

`pointCenter: point2D`

Information regarding the exact center of the Ellipse Object Label.

`xRadiusNorm: number`

A numeric value equivalent to half the length of the Ellipse Object Label's major axis.

`yRadiusNorm: number`

A numeric value equivalent to half the length of the Ellipse Object Label's minor axis.

`rotationRad: number`

The rotation angle of the Ellipse Object Label, expressed in radians.

Landmark

Contains information about the Landmark Object Label on 2D images.

`point: point2D`

The point in physical space on a 2D image where the Landmark is located.

Landmark 3D

Contains information about the Landmark Object Label on 3D volumes.

`point: voxelPoint`

The point in physical space on a 3D volume where the Landmark is located.

Cuboid

Contains information about the Cuboid Object Label.

`point1, point2: voxelPoint`

Information about the initial point of the Cuboid (`point1`) and the final point (`point2` , opposite diagonal corner).

`absolutePoint1, absolutePoint2: worldPoint`

The position of `VoxelPoints` `point1` and `point2` in physical space (world coordinate) computed using the Image Plane Module.

Consensus Formats

Consensus `tasks.json`

```
// Single task on RedBrick can be single/multi-series
type Task = {
  // Required on upload and export
  name: string;
  consensus: true;
  consensusScore: number; // overall consensus score
  consensusTasks: ConsensusTask[]

  // Only required on export
  taskId?: string;
  currentStageName?: string;
  createdBy?: string;
  createdAt?: string;
  updatedBy?: string;
  updatedAt?: string;
};

type ConsensusTask = {
  updatedBy: string;
  updatedAt: string;
  scores: {secondaryUserEmail: string, score: number}[]
  series: Series[]
}
```

The `ConsensusTask` Object

The consensus task object contains information about the consensus annotations for this task. There will be a single entry for every annotator who annotated this task. For example, if 3 users annotated each task in your project, the length of the `consensusTasks` array will be 3.

`updatedBy: string`

The e-mail of the user who annotated the task.

`updatedAt: string`

The datetime when the user last updated the task.

`scores: {secondaryUserEmail: string, score: number}[]`

The `scores` entry compares the current users' annotations with every other user. The scores array will be of length n-1, where n is the number of users who annotated this task. `score` is the similarity score between the current user, and `secondaryUserEmail`.

`series: Series[]`

The series entry for the current user only.

Taxonomy Object

```
type Taxonomy = {
    orgId: string;
    name: string;
    createdAt: datetime;
    archived: boolean;
    isNew: true;
    taxId: string;
    studyClassify: Attribute[];
    seriesClassify: Attribute[];
    instanceClassify: Attribute[];
    objectTypes: ObjectType[];
}

type ObjectType = {
    category: string;
    classId: number; // [0, n)
    labelType: BBOX | POINT | POLYLINE | POLYGON | ELLIPSE | SEGMENTATION
    attributes?: Attribute[];
    color?: string;
    archived?: boolean;
    parents?: string[];
    hint?: string;
}

type Attribute = {
    name: string;
    attrType: BOOL | TEXT | SELECT | MULTISELECT;
    attrId: number;
    options?: AttributeOption[];
    archived?: boolean;
    parents?: string[];
    hint?: string;
    defaultValue?: number | number[]; // pre-populated optionId(s) for SE
}

type AttributeOption = {
    name: string;
    optionId: number;
    color?: string;
    archived?: boolean;
}
```

Export Structure

RedBrick AI offers several ways to structure exports based on the needs of your annotation workflow.

These variations in export directly impact the contents of the subdirectory that is generated when exporting your annotation files (or, optionally, images) from RedBrick AI, and occasionally the `tasks.json` file as well.

Please see the following walkthrough for example structures, explanations, and other clarifications regarding export structures.

Standard Export

For the purposes of this documentation, a standard export is any RedBrick AI export that **does not include overlapping segmentations or variant export parameters** such as `semantic_mask` or `binary_mask`.

Single Series

For Tasks with a single Series or object (e.g. a single 2D DICOM X-ray), RedBrick AI generates a **single NIfTI file** upon export.

Each file contains all of the individual annotations associated with a particular Series as defined by the `segmentMap`, which can be found in the `tasks.json` file.

```
projectId/
|-- segmentations
      |-- Task_001 // Folder with the name of the Task (i.e. "Datapoint" 
            |-- SeriesName.nii.gz
```

Multi-Series

For Tasks with multiple Series, RedBrick AI generates a **single NIfTI file per Series** upon export.

Each file contains all of the individual annotations associated with a particular Series as defined by the `segmentMap`, which can be found in the `tasks.json` file.

```
projectId/
|-- segmentations
      |-- Task_001 // Folder with the name of the Task (i.e. "Datapoint" (
            |-- Series_001.nii.gz
            |-- Series_002.nii.gz
            |-- Series_003.nii.gz
```

Overlapping Segmentations

For Tasks with overlapping segmentations, RedBrick AI generates the following for each Series:

- an **aggregated annotation file** containing all of the annotations in the Series;
- a **subdirectory** that has the same name as the Series;
- within the subdirectory, an **individual annotation file** for each segmentation;

Please note that the annotation files in the subdirectory are generated **incrementally based on the order in which annotations were created by the labeler**.

```
projectId/
|-- segmentations
      |-- TaskName
            |-- SeriesName.nii.gz // NIfTI file containing all annotation
            |-- SeriesName
              |-- instance-1.nii.gz // NIfTI file for first overlapping s
              |-- instance-2.nii.gz // NIfTI file for second overlapping
```

> **SDK Mastery:** RedBrick AI automatically enables `binary_mask` for Tasks that have overlapping segmentations.

You can read more about the binary_mask parameter and how it affects exports here.

Semantic Export

If you would prefer to map your annotations directly to the corresponding Object Label of your Taxonomy, you can use the `semantic_mask` parameter of the Python SDK's `export_tasks()` function.

The Semantic Export enforces a direct mapping between an Object Label and the `segmentMap` value. While this does not change the structure of the export subdirectory itself, Semantic Export does bring significant changes to the `tasks.json` file generated upon export.

Please compare the two sample `segmentMap` outputs below.

Standard Export

```
tasks.json/
  "items": ["image-file.ima",],
  "segmentations": "./path/to/segmentation/file.nii.gz",
  "segmentMap": {
    "1": {
      // This is the first annotation the labeler created
      "category": "Glioma"
    },
    "2": {
      // This is the second annotation the labeler created
      "category": "Glioma"
    },
     "3": {
      // This is the third annotation the labeler created
      "category": "Encephalitis"
    },
    "4": {
      // This is the fourth annotation the labeler created
      "category": "Abscess"
    },
    ...
  }
```

Semantic Export

```
tasks.json/
  "items": ["image-file.ima",],
  "segmentations": "./path/to/segmentation/file.nii.gz",
  "segmentMap": {
    "3": {
      "category": "Glioma"
    },
    "6": {
      "category": "Abscess"
    },
     "8": {
      "category": "Encephalitis"
    },
    ...
  }
```

Common Questions

Why are the integer values different for the Semantic Export's `segmentMap`?

When a Taxonomy is created, each Object Label is assigned an immutable and unique integer value - a **Category Number**.

Unlike Standard Export, which generates the `segmentMap` integer incrementally based on the order the labeler created the annotations, the Semantic Export directly correlates this **Category Number** to your annotation within the NIfTI file, regardless of how the annotator did their work.

In other words (and using the example above), any **Glioma** annotation will always have a `segmentMap` value of 3 upon semantic export, any **Abscess** will always have a value of 6, and so on.

What if a labeler creates 2 Entities for a single Object Label and then I try to use Semantic Export?

Semantic Export will strictly enforce the principles of semantic annotation even in the presence of human error.

If you attempt to use Semantic Export on a Task that has multiple Entities associated with a single Object Label, RedBrick AI will aggregate all of the Entities into a single annotation file.

In this case, only the Object Label Attributes of the very first Entity will be preserved.

What if I need to use Semantic Export on Tasks with overlapping segmentations?

The output format will be the same, but the output annotation files will be named according to their Category Number.

```
projectId/
|-- segmentations
      |-- TaskName
            |-- SeriesName.nii.gz // Annotation file containing all segme
            |-- SeriesName // subdirectory
              |-- category-x.nii.gz // where x = the structure's Category
              |-- category-y.nii.gz // where y = the structure's Category
```

What if I re-ordered the Object Labels in my Taxonomy? Does this affect the `segmentMap` for Semantic Export?

No. The Category Number is immutable for all Object Labels, regardless of how you re-order or otherwise manipulate the contents of your Taxonomy in the UI.

Consensus Export

When exporting annotations from a Consensus Project, the `tasks.json` file and the `segmentations/` directory will have a unique structure.

Your export subdirectory will contain the annotation files for all users who generated and saved annotations on RedBrick AI. Each individual annotation file is marked with a numeric index (e.g. "_1" at the end of the file name, and you can map this file to the corresponding user by referencing your `tasks.json` file.

```
project_id/
├── segmentations
│   ├── Task_001
│   │   ├── seriesA_1.nii
│   │   └── seriesA_2.nii
│   └── Task_002
│       ├── seriesA_1.nii
│       └── seriesA_2.nii
└── tasks.json
```

Useful Links