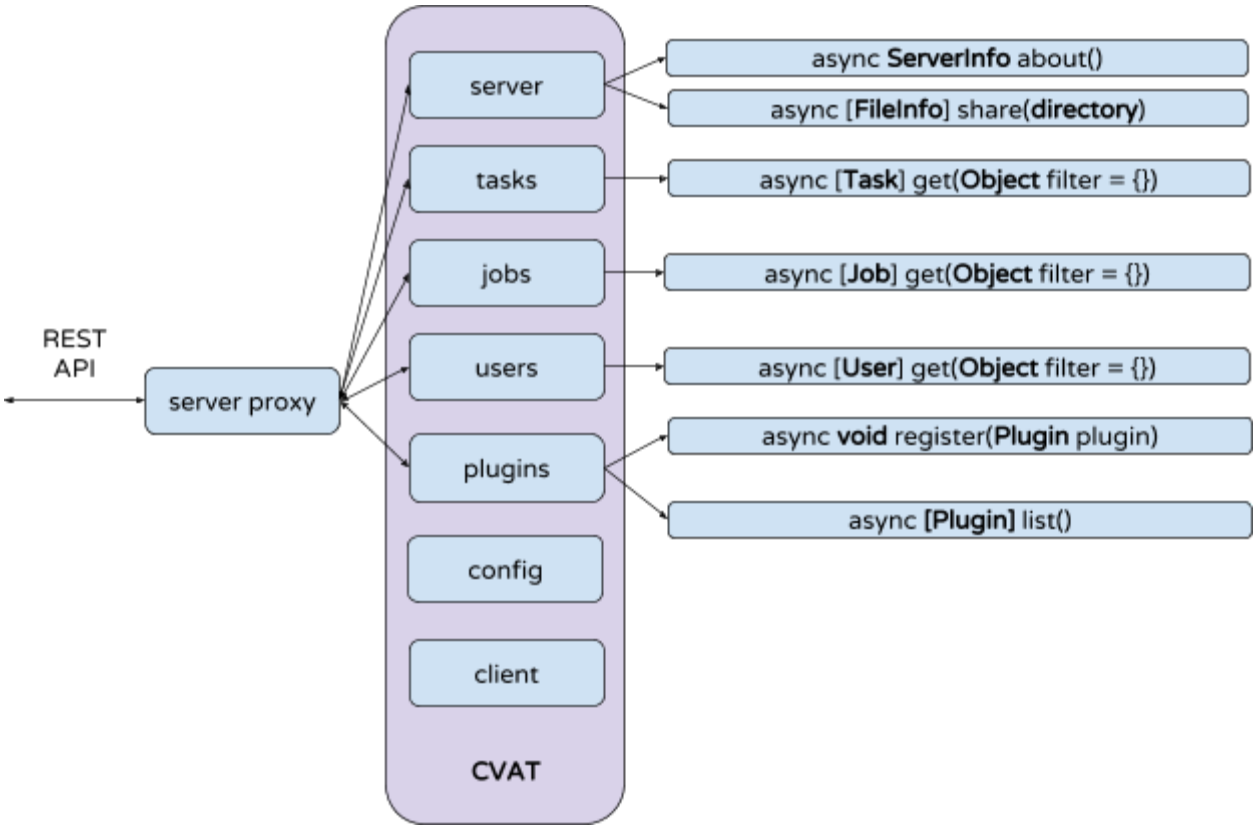
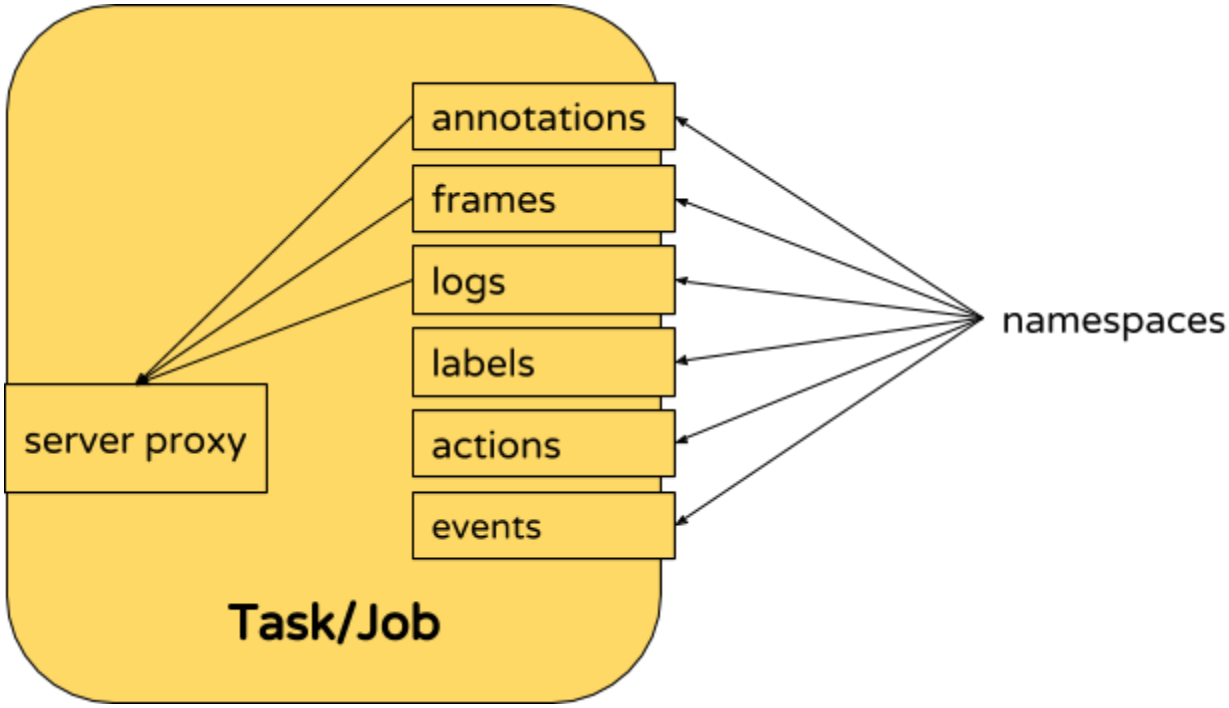
CVAT JS API

Common



Notes:

- Any async function returns promise, but specified return value will be passed to **resolve**.
- **Exception** will be passed to **reject** if any exception occurs.



Space <annotations>:

Description: Provides information about shapes, tracks, tags etc.

API:

async **void** upload(**File** annoFile)
async **void** save()
async **void** clear()
async **Stream** dump()
async **Statistics** statistics()
async **void** put (**[ObjectState** data**]**)
async [**ObjectState**] get(**integer** frame, **Filter** filter)
async **integer** search(**Filter** filter, **integer** from, **integer** to)
async **integer** or **null** select(**integer** frame, **float** x, **float** y)

Space <frames>:

Description: Provides frames from the server and information about it.

API:

async **FrameData** get(**integer** frame)

Space <logs>:
Description: Provides some API for Logging.
API:

async **Log** put(**LogType** type, **Object** details)
async **void** save()

Space <labels>:
Description: Provides information about labels.
API:

async [**Labels**] get(**Object** filter = {})

Space <actions>:
Description: Provides undo/redo API.
API:

async **[Object]** undo(**integer** count = 1) // Object = [id:action]
async **[Object]** redo(**integer** count = 1) // reversed action
async **void** clear()

Space <events>:
Description: Provides API in order to subscribe to several events.
API:

async **void** subscribe(**EventType** type, **Object** listener, **string** method)
async **void** unsubscribe(**EventType** type, **Object** listener**, string** method)

Structures

ServerInfo:

.name:**string** **[ReadOnly]**
.description:**string** **[ReadOnly]**
.version:**string** **[ReadOnly]**

Exception:

.message:**string** (human-readable)
.stack:**string**

async .save()
constructor(Object = {})

FileInfo:

.name:**string** **[ReadOnly]**
.type:**string** **[ReadOnly]**

Label:

.id:**integer** **[ReadOnly]**
.name:**string** **[ReadOnly]**
.attributes:[**Attribute**] **[ReadOnly]**

constructor(Object) // object contains name and attributes

Attribute:

.id:**integer** **[ReadOnly]**
.name:**string** **[ReadOnly]**
.mutable:**boolean** **[ReadOnly]**
.type:**string** **[ReadOnly]**
.default:**string** **[ReadOnly]**
.values:[**string**] **[ReadOnly]**

constructor(Object)

Task:

.id:**integer** **[ReadOnly]**
.name:**string**
.status:**string** **[ReadOnly]**
.size:**integer** **[ReadOnly]**
.mode:**string** **[ReadOnly]**
.owner:**integer** **[ReadOnly]**

.assignee:**integer**
.createdDate:**string** **[ReadOnly]**
.updatedDate:**string** **[ReadOnly]**
.data:**Object** {server_files: [**string**], client_files: [**string**], remote_files: [**string**]}
.bugTracker:**string**
.overlap:**integer**
.segmentSize:**integer**
.zOrder:**boolean**
.labels:**[Label]** **[AppendOnly]**
.jobs**[Job]** **[ReadOnly]**

constructor(Object)
async addLabel(Label label**)**
async save(function onChangeStatus(**string** status)**)**
async delete()

.annotations:**object** **[ReadOnly]**
.frames:**object** **[ReadOnly]**
.logs:**object** **[ReadOnly]**
.labels:**object** **[ReadOnly]**
.actions:**object** **[ReadOnly]**
.events:**object** **[ReadOnly]**

Job:

.id:**integer** **[ReadOnly]**
.assignee:**integer**
.status:**string**
.startFrame:**integer** **[ReadOnly]**
.stopFrame:**integer** **[ReadOnly]**
.task:**Task** **[ReadOnly]**

async save()

.annotations:**object** **[ReadOnly]**
.frames:**object** **[ReadOnly]**
.logs:**object** **[ReadOnly]**
.labels:**object** **[ReadOnly]**
.actions:**object** **[ReadOnly]**
.events:**object** **[ReadOnly]**

User:

.id:**integer** **[ReadOnly]**

.username:**string** **[ReadOnly]**
.email:**string** **[ReadOnly]**
.firstName:**string** **[ReadOnly]**
.lastName:**string** **[ReadOnly]**
.groups:[**string**] **[ReadOnly]**
.lastLogin:**string** **[ReadOnly]**
.dateJoined:**string** **[ReadOnly]**
.isStaff:**boolean** **[ReadOnly]**
.isSuperuser:**boolean** **[ReadOnly]**
.isActive:**boolean** **[ReadOnly]**

ObjectState:
.type:**string**
.position:**points**
.group:**integer**
.zOrder:**integer**
.outside:**boolean**
.occluded:**boolean**
.attributes:**Object** **// id:value pairs**
.label:**integer**
.lock:**boolean**

constructor(**Object** = {})
async **save()**
async **delete()**

FrameData
.height:**integer** **[ReadOnly]**
.width:**integer** **[ReadOnly]**

async Image image() **[ReadOnly]**

Config
.frameLoadPolicy:**string** <full, incremental(count)>

Client
.version:**string**

Examples:

```
// Dump of annotations
dummyTaskID = 55;

async function dump() {
   try {
       const task = await cvat.tasks.get({id: dummyTaskID});
       const stream = await task.annotations.dump();
       // load file through a stream
   } catch (exception) {
       if (exception instanceof cvat.Exception) {
           showMessage(exception.message);
           await exception.save();
       } else {
           // handling of user exceptions
       }
   }
}


// Get objects and changes it
dummyTask = Task() // some created task
dummyFrame = 555;

async function updateObjects() {
   let objects = null;
   try {
       objects = await dummyTask.get(dummyFrame);

       for (let object of objects) {
           if (object.type === 'rectangle') {
               const [xtl, ytl, xbr, ybr] = object.position;
               // change position in any way
               object.position = [xtl, ytl, xbr, ybr];
               await object.save();
           } else {
               await object.delete();
           }
       }
   } catch (exception) {
       if (exception instanceof cvat.Exception) {
           showMessage(exception.message);
```

```
            exception.save();
        } else {
            // handling of user exceptions
        }
    }
}


// Semi automatic segmentation
dextrPlugin = {
    cvat: {
        Job: {
            annotations: {
                put: {
                    enter([objects], self) {
                        for (const object of objects) {
                            if (object.type === 'dextr' && self.parameters.enabled) {
                                response = serverRequest(object.position);
                                object.type = 'polygon';
                                object.position = response.position;
                            }
                        }
                    },
                },
            },
        },
    },
    name: 'Deep Extreme Cut',
    parameters: {
        enabled: true,
    },
}

cvat.plugins.register(dextrPlugin).catch((exception) => {
    showMessage(`Could not register the plugin "${dextrPlugin.name}":
${exception.message}`);
    exception.save();
});


dummyJob = Job()
async function addDextr() {
    const obj = new cvat.ObjectState({
```

```
        points: [], //some points array
        type: 'dextr',
        frame: 10,
        // etc
    });

    try {
        await dummyJob.annotations.put([obj]);
    } catch (exception) {
        showMessage(exception.message);
        exception.save();
    }
}



// create task
async function createTask() {
    const dummyTask = new cvat.Task()
    dummyTask.overlap = 5;
    dummyTask.zOrder = false;
    dummyTask.bugTracker = 'some url';
    dummyTask.data = {
        'server_files': [file1, file2],
        'client_files': [],
        'remote_files': [],
    }

    labels = Array.from(jsonLabels, (jsonLabel) => new Label(jsonLabel));

    try {
        for (const label of labels) {
            await dummyTask.addLabel(label);
        }

        await dummyTask.save((message) => {
            // do something with a message
        });
    } catch (exception)  {
        // handle exception
    }
```