The IPOL Demo Web interface

Compiled on Tuesday 13th June, 2017 at 15:35

This document contains the technical documentation for the Demo Web interface 3.0 that executes **[Miguel] The web interface does not execute the demos** demos and shows it's **[Miguel] avoid contractions. Also, is 'its', not 'it's'** results.

[Miguel] Say something about the generic controls which allow to draw lines, points, etc.

Contents

List of Figures

1 IPOL web interface

current version is 3.0.

1.1 Introduction

The IPOL demo Web interface has been developed with HTML5, CSS3 **[Miguel] Add a comma before 'and' in enumerations** and Jquery. This web page is responsible for letting the users execute demos using either blobs from the demo or blobs uploaded by the user. **[Miguel] Rewrite: It allows the users to execute the IPOL algorithm's with an ergonomic interface. The users can use their own data or the examples offered by each of the demos.**

[Miguel] We shall describe the modules of which the web interface is made, its flow diagram, how the asynchronous calls work, and the data types accepted.

2 Modules

This section will try to explain the different modules existing in the demo app. All of these modules are divided across the javascript files of the application. **[Miguel] Rewrite: The Javascript application is made of several modules:**

- Inputs.
- Upload.
- Editor.
- Parameters.
- Run.
- Results.
- Helpers. **[Miguel] Remove the final dot at the end of each item**

Each of them is described in the following sections.

2.1 Inputs

This module will do everything [Miguel] 'Everything' is too vague. related to the demo blobs. As it is the starting point of the application [Miguel] I don't get what you mean with the 'starting point of the application'. it will also make the Async calls needed to obtaing data from the demo using id [Miguel] its ID from the URL, like [Miguel] the same way as demoinfo and blobs back-end modules. These two calls will obtaing the information necesary [Miguel] necessary. Use a spellchecker. to execute the demo, letting [Miguel] allowing the user choose along the way [Miguel] along the way? what's that? the input blobs, edit these blobs [Miguel] and comma and tweak demo execution parameters. This input module will let [Miguel] Do not write in future tense. Change it to 'allows the user...'. You describe what already exist now, not in the future. the user choose among blobs that the demo provides. These blobs will be either images, video [Miguel] videos [Miguel] add commma or audio blobs primarily. It will [Miguel] Write in present tense. Describe only what has been done, not in the future display a line of blobs or sets of blobs to choose from.

2.2 Upload

If the user wants to use their [Miguel] If the users want to use their own blobs for the demo this module will let you [Miguel] Not future. Do not use 'you'. Write more formal upload them. Every demo has predefined upload slots each with their own characteristics [Miguel] Add some examples of these characteristics. The user will [Miguel] Not future be able to upload a [Miguel] the minimum required number of uploads. This module will [Miguel] not future listen for events en [Miguel] in every upload slot and get the file information in order to upload them when the run button is pressed.

2.3 Editor

This module will [Miguel] Change ALL future tenses and describe only what is done load after the user selects a set of blobs to run the demo with or when the users upload their blobs. It will load a view of the chosen blobs where there will be a zoom and crop functions when the sets have one blob per set. When the sets have more than one blob per set, the user will also be able a [Miguel] will have the possibility to compare them with the corresponding 'Compare button', but not to crop

an area. compare button but not a crop option.

2.4 Parameters

This module will print itself after **[Miguel] renders the parameters of the demo described in the DDL after** the user chooses the blobs to use with the demo. It will print itself after the editor **[Miguel] The 'editor' hasn't been yet define** and will contain as many parameters as the DDL specifies. Parameters have their own specification as well as different types.

2.5 Run

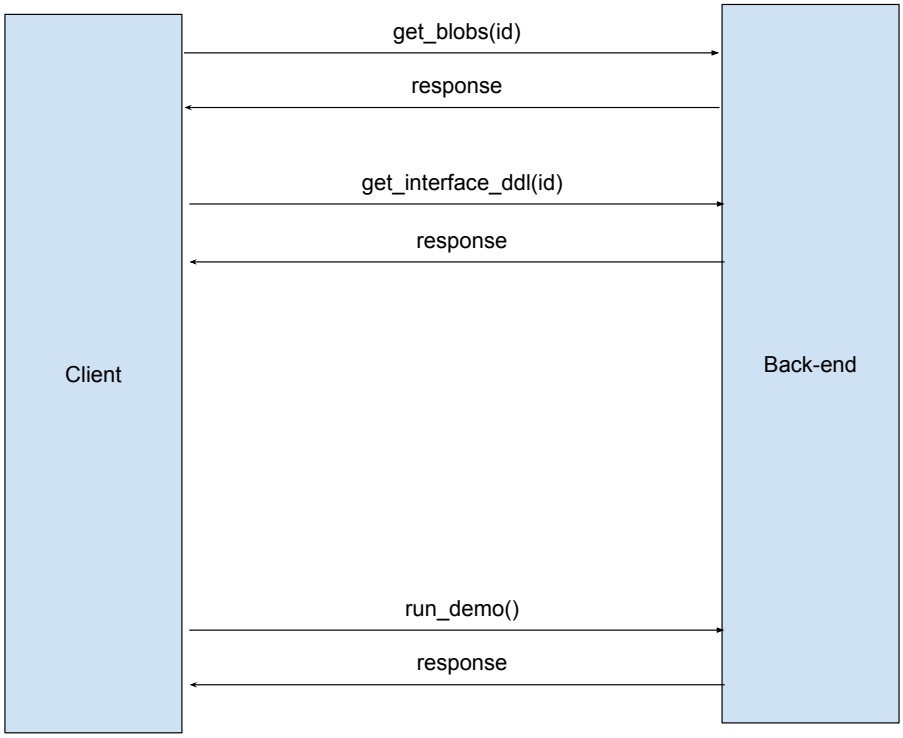
This module will compile **[Miguel] Do not use 'compile', because this is used to compile the code** all the blobs and requirements needed to run the demo and will send this information to the demorunner **[Miguel] DemoRunner** module. Then it will wait **[Miguel] block** until there is a response and print the results **[Miguel] Do not use 'print'. Be more specific.**.

2.6 Results

This module will print the results the demorunner gives back **[Miguel] returns** letting the user compare the input and results.

2.7 Helpers

This module act as an interface for common utility methods like read and write to sessionStorage, make http **[Miguel] HTTP** requests and read the origin **[Miguel] You don't say which is an origin and what origins are possible** of the chosen blobs for the demo.

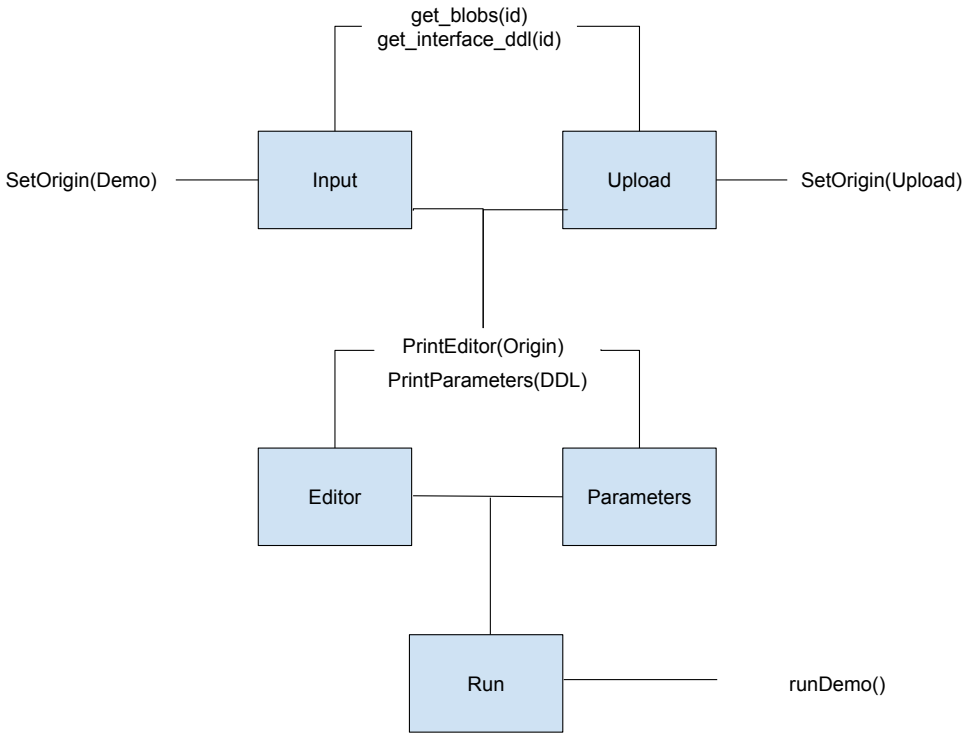


[Miguel] Convert the graphics into a figure with a caption. Also, refer to it in the text using `ref`

3 Flow diagram

The first thing the application will do when loaded is look for an id which **[Miguel] When the application is loaded, the XXX module will locate the demo according to its ID** represents the demo id to be represented **[Miguel] Reread and rewrite**. This ID is a get parameter of the URL. At the same time **[Miguel] As the same time it reads the ID?** the main file, demo.js will load the different html files into the DOM. They have been divided into various **[Miguel] several** files to improve mantainability and coupling.

After all this **[Miguel] Do not write 'all this'** is loaded, the app will continue adding the main section to the DOM, containg in this case the blobs viewer and the blobs upload dialog. At the same time **[Miguel] At the same time? Events? Be more specific**, the the information regarding the current demo, responses from **[Miguel] the** blobs **[Miguel] Blobs** and demoInfo modules will be saved at **[Miguel] in** sessionStorage.



As the diagram shows **[Miguel] Put a footer**, either input or upload modules will pass information to the following modules. Each will set a variable in sessionStorage that will indicate if the user has chosen to upload blobs for the demo or a default blob from the list.

After the user chooses a blob, the editor and parameter modules will load independently and wait for changes. Editor **[Miguel] The Editor control** will allow to either zoom, crop **[Miguel] add comma** and compare blobs when **[Miguel] remove well, add 'if'** possible, as well as do inpainting operations when the demo requires it. The parameters will allow tweaking **[Miguel] The configure the parameter controls, but that's not tweaking** according to ddl **[Miguel] to the DDL** specification. Parameter values will be stored in a variable for refreshing feature purpose as the ddl states. **[Miguel] This last sentence is hard to understand.**

After the user hits the run button an http post will be executed to run the demo and send the necessary information and wait for a response. When the response is obtained it will either print the results interface or the error message.

3.1 External modules

The IPOL demo Web interface uses external libraries for extra functionalities. Currently it uses:

- Cropper.js: Cropper.js it is a simple image cropping JQuery plugin. It is used in the editor panel with the image blobs.

3.2 Async calls

The IPOL demo Web interface uses Async calls to get the necessary information from the IPOL server and use it **[Miguel] 'and use it' is redundant**.

The current version uses Async calls for:

- Get the demo ddl: Used to show the inputs description and the upload modal in the Inputs panel, also uses this information to show the parameters.
- Get blobs: Used to show the blobs in the inputs panel.
- Run demo: It will send all parameters needed to run the demo and will responde with either the results of the demo or an error response.

3.3 Data types

The IPOL platform supports mainly images, but recent updates include audio and video files to use in demos **[Miguel] You're describing the new interface. Thus, it doesn't make sense to say that it supports mainly images**. The new web interface allows to choose a set with any combination of images, audio and video. Depending on the data types and sets length options will vary. If a set contains multiple images, the user will be able to compare and make zoom using any image on the set. If the user chooses a set with only one image blob, options will depend on DDL limitations and will include zoom and crop features, as well as inprinting editing.

In case of uploading an audio or video file, it wont have a visual representation at the moment **[Miguel] But they will. It's a work in progress. Do not document that we don't support it.**, only videos and audio files in the default set from the demo will have a visual representation.

[Miguel] In general, please improve the writing and be more specific.