FLAXBLOCKS

Blockly-files/blocks.js

Properties:

Name	Type	Description
type	string	The type of the block, set to "python_function".
message0	string	The message template for the block's UI.
args0	Array	The arguments for the block.
args0[].type	Object	The type of the field/input (e.g., field_input, input_dummy, input_statement).
args0[].name	string	The name identifier for the field/input.
args0[].text	string	The default text for the input field (only for field_input).
colour	string	The color of the block in the Blockly workspace.
previousStatement	null	Indicates that this block can be followed by another block.
nextStatement	null	Indicates that this block can be preceded by another block.
tooltip	string	The tooltip text that appears on hover.
helpUrl	string	A URL to more help related to this block.

Source: Blockly-files/blocks.js, line 1

Blockly-files/javascript.js

How blockly generates code for a given block is stored here

Source: Blockly-files/javascript.js, line 2

Example

```
pythonGenerator.forBlock['self'] = function(block, generator) {
    var model = generator.valueToCode(block, 'func', Order.NONE);
    return '@nn.compact\\n' +
           'def __call__(self, x):\\n';
  };
```

```
Notes:
- The `provideFunction_` utility ensures that the function is only defined once and can be reused across multiple blocks.
- Adjust the block logic and code output based on the specific behavior you want to implement in Python or JavaScript.
```

Blockly-files/serialization.js

Provides function to save and load a Blockly workspace

License: Copyright 2023 Google LLC SPDX-License-Identifier: Apache-2.0

Source:

Blockly-files/serialization.js, line 1

Blockly-files/toolbox.js

This object defines a Blockly toolbox with different categories and blocks. To create blocks for the toolbox: - Each category is an object with properties: - 'kind': Specifies it's a 'category'. - 'name': The display name of the category. - 'categorystyle': Defines the style of the category, which is pre-configured in Blockly (e.g., 'logic_category'). - 'contents': An array that lists the blocks within that category. - Each block is defined by: - 'kind': Specifies it's a 'block'. - 'type': The type of block to include. The block type must correspond to the block definition created in Blockly. Example: - To add a new block, add an object inside the 'contents' array of a category: { 'kind': 'block', 'type': 'my_custom_block' // This must match the defined block type in your Blockly code. } - Categories can also be nested within other categories by adding a new 'category' object inside 'contents'. This configuration is essential for setting up the visual toolbox in Blockly, allowing users to drag and drop blocks for coding.

Source: Blockly-files/toolbox.js, line 1

main.js

This file is the main process of the Electron application. It initializes the app, manages inter-process communication (IPC) between the renderer and the main process, and handles file operations, including saving, reading, and modifying files. It also provides functionality to run external scripts, such as the 'blockify' script. The file handles: - Initializes the Electron BrowserWindow and loads the renderer (index.html). - Handles file I/O operations like reading, writing, and modifying Python files. - Interacts with the renderer process via IPC (save, read, start script commands). - Clears cache, cookies, and storage data before launching the main window. The file is automatically executed when the Electron application starts. It sets up the main process and connects the back-end functionality to the front-end interface.

Source: main.js, line 1

workspaceInit.js

This is a utility file for creating and connecting blocks The `workspace.init` function sets up the Blockly workspace with custom blocks, toolbox configuration, event listeners, and other necessary configurations. The file is crucial for initializing the environment where users can drag and drop blocks to build visual code. It interacts with various Blockly APIs to generate code and maintain state. Key features include: - Initializing the Blockly workspace with a predefined toolbox and custom blocks. - Setting event listeners for block changes and code generation. - Handling workspace resizing and layout adjustments.

Source: workspaceInit.js, line 3

Documentation generated by JSDoc 4.0.3