Balabanoff Website documentation

Document written by Emmanuel Echeverri in hopes the website can be properly maintained.

Contents

[Introduction 2](#_Toc188022588)

[Summary (TL:DR) 3](#_Toc188022589)

[File structure 4](#_Toc188022590)

[HTML files 5](#_Toc188022591)

[Index.html 5](#_Toc188022592)

[landing\_gallery.html 6](#_Toc188022593)

[contact.html 6](#_Toc188022594)

[group-members.html 8](#_Toc188022595)

[publications.html 10](#_Toc188022596)

[research.html 12](#_Toc188022597)

[news.html 13](#_Toc188022598)

[news\_2022.html 14](#_Toc188022599)

[CSS files 16](#_Toc188022600)

[JS files 17](#_Toc188022601)

[nav.js 17](#_Toc188022602)

[slideshow.js 17](#_Toc188022603)

[current\_members.js 18](#_Toc188022604)

[past\_members.js 20](#_Toc188022605)

[news\_generator.js 21](#_Toc188022606)

[JSON files 23](#_Toc188022607)

[past\_members.json 23](#_Toc188022608)

[current\_members.json 23](#_Toc188022609)

[news\_log.json 23](#_Toc188022610)

## Introduction

## Summary (TL:DR)

### GUI

The GaussianToBlender.py application provides a GUI for converting Gaussian input files into 3D object files using Blender's API. The main script, utilizing tkinter, manages different regions and functionalities including input, output, console, and ion management. Key features involve configuring the main window, placing GUI regions, managing conversion processes, handling ionic parameters, and managing animations.

The application consists of several modules:

* **Information.py**: Manages the help and about sections, providing instructions and application details.
* **BlenderPath.py**: Allows users to set the path to the Blender executable.
* **InputRegion.py**: Handles input file selection and management.
* **OutputRegion.py**: Manages output paths and file types.
* **ConsoleRegion.py**: Displays console output and error messages.
* **IonRegion.py**: Manages ion information for the molecules.
* **IonConventions.py**: Displays ion coordination codes and their meanings.
* **ActionsRegion.py**: Provides buttons for resetting values and initiating conversions.
* **Utility.py**: Contains utility functions for file management.
* **Coordinates.py**: Extracts Cartesian coordinates from molecular structure files.
* **TextRedirector.py**: Redirects print statements to the console region within the GUI.

### API

The Blender Python API facilitates the conversion of molecular data into 3D models using Blender’s Python API, enabling representation and animation of molecules.

* **Main\_Body.py** Manages the entire molecular data conversion process into 3D models using Blender’s API. It imports modules, sets parameters, extracts data, builds the model, rigs/animates it, and exports the result.
* **AtomData**.**py** Stores atomic and ionic data like radii and colors for use in molecule construction.
* **ImportData.py** Handles the extraction and filtering of molecular data from input **files. RefineData.py** Refines molecular coordinates and connectivity data for 3D modeling.
* **RefineElements.py** Identifies unique elements and manages their positions and connectivity in the molecule.
* **Create\_Materials.py** Creates and manages materials for molecular elements in Blender.
* **Primitives.py** Instantiates and manages 3D primitives to represent atoms and bonds in Blender.
* **Export\_Data.py** Exports Blender scenes and animations to various file formats, supporting animation export for elements and bonds.
* **Ions.py** Handles ionic data, including ion creation, refinement, and management within molecular structures.
* **Instantiate\_Molecules.py** Instantiates 3D geometries (elements and ions) from Cartesian coordinates and sets object origins.
* **Raw\_Parameters.py** Extracts and processes raw coordinates and connectivity data from input files, including animation data.
* **Animate.py** Manages and animates molecular structures, including keyframe updates for elements and bonds based on animation data.
* **Clear\_Transforms.py** Applies location and rotation transformations to bonds and elements in a molecular structure.
* **Receive\_Parameters.py** Extracts and processes parameter data from a file into a dictionary for configuring the molecular conversion process.

### Batch script

This batch script automates running Blender with a specified Python script to convert molecular data. It is currently for Windows OS and will be complemented by a future ReadMolecules.sh script.

Key Components:

* **Environment Setup**: Sets the working directory and Blender path from provided arguments.
* **Blender Execution**: Executes Blender in background mode with the given Python script.

Main Steps:

* Sets the working directory and Blender path.
* Changes directory to Blender’s installation path.
* Executes Blender with the specified Python script and Blender file.

## File structure

## Graphical User Interface

### GaussianToBlender.py

This is the main script for the Gaussian-to-Blender application, which provides a graphical user interface (GUI) for converting Gaussian input files into 3D object files using Blender's API. The script utilizes the tkinter library to create the GUI and integrates various utility and GUI modules to manage different regions and functionalities of the application.

**Key Components:**

* **Initialization:** Sets up the main application window and initializes various regions (e.g., input, output, console, ion regions).
* **Configuration:** Configures the main window properties such as title, icon, size, and layout.
* **Region Placement:** Places different GUI regions within the main window using a grid layout.
* **Conversion Management:** Manages the process of converting Gaussian input files to 3D object files, including validation of input parameters and execution of conversion scripts.
* **Ionic Parameters:** Handles the assignment and management of ionic parameters for the conversion process.
* **Animation Handling:** Manages the creation of animation frames if the input files are to be treated as an animation.

**Main Functions:**

* \_\_init\_\_: Initializes the main application and sets up the GUI regions.
* \_configure\_root: Configures the main window properties.
* place: Places a specified region in the parent widget.
* \_initialize\_regions: Initializes and places all GUI regions.
* reset\_to\_defaults: Resets the application to its default state.
* exceptions\_test\_passed: Validates input parameters for the conversion process.
* overwrite\_parameters\_script: Overwrites the parameters script for the conversion process.
* overwrite\_animation\_frames: Manages the creation of animation frames.
* individual\_convert: Executes the conversion process for individual files.
* assign\_ionic\_params: Assigns ionic parameters for the conversion process.
* convert\_manager: Manages the overall conversion process.
* run: Starts the main application loop.

### Information.py

This module defines the Information class, which is responsible for displaying instructions and information about the Gaussian2Blender application within the GUI.

**Key Components:**

* **Initialization:** Sets up the Information region as a labeled frame within the parent widget, containing buttons for "Help" and "About" functionalities.
* **Help Button:** Opens a popup window with detailed instructions on how to use the Gaussian2Blender application.
* **About Button:** Opens a popup window with version information and details about the application.

**Main Functions:**

* \_\_init\_\_: Initializes the Information region and sets up the help and about buttons with tooltips.
* \_initialize\_popup: Configures the layout and content of the popup windows for help and about information.
* help: Displays a popup window with step-by-step instructions for using the application.
* open\_url: Opens the URL to the referenced paper in a web browser.
* about: Displays a popup window with version and author information about the application.

### BlenderPath.py

This module defines the BlenderPath class, which is responsible for managing the section of the GUI where users can search for and set the path to the Blender executable.

**Key Components:**

* **Initialization:** Sets up the BlenderPath region as a labeled frame within the parent widget, containing a label, a path display, and a button to search for the Blender executable.
* **Path Search:** Provides functionality to automatically search for the Blender executable in the default installation directory or allow the user to manually select the path.

**Main Functions:**

* \_\_init\_\_: Initializes the BlenderPath region and sets up the label, path display, and search button with tooltips.
* searchBlenderPath: Searches for the Blender executable in the default installation directory and returns the path if found.
* lookForBlenderPath: Opens a file dialog for the user to manually select the Blender installation directory and validates the presence of the Blender executable.
* setBlenderPath: Sets the Blender path to the specified directory.

### InputRegion.py

This module defines the InputRegion class, which is responsible for managing the section of the GUI where users can specify the input files to be converted.

**Key Components:**

* **Initialization:** Sets up the InputRegion as a labeled frame within the parent widget, containing labels, buttons, and dropdowns for selecting input files and model types.
* **Input Management:** Provides functionality to select multiple input files, validate their extensions, and manage input-related variables.

**Main Functions:**

* \_\_init\_\_: Initializes the InputRegion and sets up the frame, canvas, and widgets.
* initialize\_variables: Initializes instance variables related to input paths, names, and types.
* clear\_variables: Clears input-related variables.
* setup\_frame: Sets up the main frame for the input region.
* setup\_canvas: Sets up the canvas and scrollable frame for displaying input files.
* add\_widgets: Adds and configures widgets such as labels, buttons, and dropdowns.
* position\_widgets: Positions the widgets within the frame.
* updateAnimationState: Updates the state of the animation checkbox and validates the number of input files.
* allFilesHaveSameValidExtension: Checks if all selected files have the same valid extension.
* setInputName: Opens a file dialog to select input files and updates the input names list.
* isValidExtension: Checks if a file has an acceptable extension.
* updateInputNameList: Updates the list of input names and paths.
* canvasConfig: Configures the canvas scroll region.
* dropdouwn\_callout: Handles the event when the model type dropdown is updated.

### OutputRegion.py

This module defines the OutputRegion class, which is responsible for managing the section of the GUI where users can specify the output path and type for the converted files.

**Key Components:**

* **Initialization:** Sets up the OutputRegion as a labeled frame within the parent widget, containing labels, entry fields, and dropdowns for selecting the output path and type.
* **Output Path Management:** Provides functionality to select the output directory and specify the output file type.

**Main Functions:**

* \_\_init\_\_: Initializes the OutputRegion and sets up the frame, labels, entry fields, and dropdowns.
* setOutputPath: Opens a file dialog to select the output directory and updates the output path variable.
* dropdown\_callout: Handles the event when the output type dropdown is updated.

### ConsoleRegion.py

This module defines the ConsoleRegion class, which is responsible for managing the console section of the GUI where users can observe print statements and error messages.

**Key Components:**

* **Initialization:** Sets up the ConsoleRegion as a labeled frame within the parent widget, containing a text widget for displaying console output and a scrollbar for navigation.
* **Text Redirection:** Redirects standard output (stdout) and standard error (stderr) to the text widget for real-time display of messages.

**Main Functions:**

* \_\_init\_\_: Initializes the ConsoleRegion and sets up the frame, text widget, and scrollbar.
* clear\_content: Clears the content of the console text widget.

### IonRegion.py

This module defines the IonRegion class, which is responsible for managing the section of the GUI where users can input information about possible ions present in the molecules.

**Key Components:**

* **Initialization:** Sets up the IonRegion as a labeled frame within the parent widget, containing checkboxes, buttons, and a scrollable canvas for managing ion information.
* **Ion Management:** Provides functionality to add and remove ions, and manage ion-related variables.

**Main Functions:**

* \_\_init\_\_: Initializes the IonRegion and sets up the frame, canvas, and widgets.
* initialize\_variables: Initializes instance variables related to ion information.
* clear\_variables: Clears ion-related variables and resets the ion input section.
* create\_widgets: Creates all widgets and frames for the ion information section.
* setup\_layout: Arranges the widgets and frames in the grid layout.
* addIon: Adds a new ion entry to the ion list.
* removeIon: Removes the last ion entry from the ion list.
* removeAllIons: Removes all ion entries from the ion list.
* activator: Activates or deactivates the ion input section based on the state of the ion checkbox.
* canvasConfig: Configures the canvas scroll region.

### IonConventions.py

This module defines the IonConventions class, which is responsible for displaying the meanings of various ion coordination values within the GUI.

**Key Components:**

* **Initialization:** Sets up the IonConventions region as a labeled frame within the parent widget, containing labels that specify the codes and their corresponding meanings for different ion coordination values.

**Main Functions:**

* \_\_init\_\_: Initializes the IonConventions region and sets up the frame and labels with tooltips.
* **Labels and Tooltips:** Provides labels for different ion coordination codes (e.g., "hs", "ls", "PY", "SQ") and their meanings (e.g., "high spin", "low spin", "pyramidal", "square-planar").

### ActionsRegion.py

This module defines the ActionsRegion class, which is responsible for managing the section of the GUI that contains action buttons for resetting the interface and initiating the conversion process.

**Key Components:**

* **Initialization:** Sets up the ActionsRegion as a labeled frame within the parent widget, containing buttons for resetting input values and converting the molecules.
* **Action Buttons:** Provides functionality to reset the interface to default values and to start the conversion process.

**Main Functions:**

* \_\_init\_\_: Initializes the ActionsRegion and sets up the frame and action buttons.
* \_create\_widgets: Creates and positions the reset and convert buttons within the frame.

### Utility.py

This module defines the Utility class, which contains various utility functions used by several other classes within the application.

**Key Components:**

* **File Management:** Provides functions to find files, manage resource paths, and handle file contents.

**Main Functions:**

* findFile: Checks if a file with the specified name exists in the given path.
* resource\_path: Returns the absolute path to a resource, accommodating for different execution environments.
* clear\_file\_contents: Clears the contents of a specified file.
* append\_lines\_to\_file: Appends a list of lines to a specified file, converting boolean values to strings if necessary.

### Coordinates.py

This module defines the Coordinates class, which provides methods for extracting and managing Cartesian coordinates from molecular structure files.

**Key Components:**

* **Initialization:** Sets up a mapping of newline characters for different operating systems.
* **Coordinate Extraction:** Provides functionality to extract Cartesian coordinates from molecular structure files and manage animation frames.

**Main Functions:**

* \_\_init\_\_: Initializes the Coordinates class and sets up the newline character map.
* get\_coordinates\_line\_numbers: Identifies the start and end lines of the coordinates section in a file.
* extract\_cartesian\_coordinates: Extracts Cartesian coordinates from a molecular structure file.
* check\_newline\_characters: Determines the type of newline characters used in a file.
* check\_animationframes: Checks if all files in a list have the same number and identity of elements for animation.
* combine\_animation\_frames: Combines Cartesian coordinates from multiple files into a single list for animation.

### TextRedirector.py

This module defines the TextRedirector class, which is used to redirect print statements and error messages to a text widget within the GUI.

**Key Components:**

* **Initialization:** Sets up the TextRedirector with a specified text widget and tag.
* **Text Redirection:** Provides functionality to write text to the text widget, allowing print statements and errors to be displayed in the GUI.

**Main Functions:**

* \_\_init\_\_: Initializes the TextRedirector with the given widget and tag.
* write: Writes the provided string to the text widget, inserting it at the top and applying the specified tag.

## Batch script

### ReadMolecules.bat

This batch script is used to automate the process of running Blender with a specified Python script for converting molecular data. **Note: This script is currently designed to work on Windows OS only. A ReadMolecules.sh script will be implemented in the future.**

**Key Components:**

* **Environment Setup:** Sets the working directory and Blender path based on the provided arguments.
* **Blender Execution:** Changes the directory to the Blender installation path and executes Blender in background mode with the specified Python script.

**Main Steps:**

* set WORK\_DIR=%~dp0: Sets the working directory to the directory of the batch script.
* set BLENDER\_PATH=%1: Sets the Blender path to the first argument passed to the script.
* cd %BLENDER\_PATH%: Changes the directory to the Blender installation path.
* blender %WORK\_DIR%\ReadMolecules00.blend --background --python %WORK\_DIR%\Main\_Body.py: Executes Blender in background mode with the specified Blender file and Python script.

## Blender Python API

### Main\_Body.py

This module defines the Main\_Body class, which orchestrates the process of converting molecular data into 3D models using Blender's Python API. It imports and reloads necessary modules, sets parameters, extracts data, builds the molecule, manages parent-child relationships, resets transforms, rigs the molecule, animates it if needed, and exports the final result.

**Key Components:**

* **Module Imports:** Dynamically imports and reloads necessary modules for the conversion process.
* **Main\_Body Class:** Manages the entire workflow from setting parameters to exporting the final 3D model.

**Main Functions:**

* \_\_init\_\_: Initializes the Main\_Body instance with input and output parameters.
* Set\_Raw\_Parameters: Sets and confirms raw parameters for the conversion process.
* Extract\_Data: Refines and extracts necessary data from the input files.
* Build\_Molecule: Builds the 3D molecule model by creating and assigning materials, and instantiating primitives.
* Manage\_Parent\_Relations: Manages parent-child relationships between elements in the molecule.
* Reset\_Transforms: Applies transforms to bonds and elements.
* Rig\_Molecule: Rigs the molecule for animation.
* Animate: Animates the molecule if the input is treated as animation frames.
* Export: Exports the final 3D model to the specified output format.
* ExportForAnimation: Exports the 3D model for animation.

**Execution:**

* The script reads parameters from a parameters.txt file and initializes a Main\_Body instance.
* It then sets raw parameters, extracts data, builds the molecule, and either exports the model or handles animation based on the input parameters.

### AtomData.py

**AtomData.py**

This module defines classes and data structures for storing atomic and ionic data, including covalent radii, van der Waals radii, colors, and ionic radii for various elements.

**Key Components:**

* **Ionic Class:** Stores information about ionic radius, charge, coordination, and spin.
* **Atom\_Data Class:** Stores information about atomic data, including covalent radius, van der Waals radius, color, and ionic data.

**Main Classes:**

* Ionic: Represents ionic radius data with attributes for charge, coordination, and radius.
  + \_\_init\_\_: Initializes the ionic data with default values.
  + set\_radius: Sets the ionic radius value.
* Atom\_Data: Represents atomic data with attributes for covalent radius, van der Waals radius, color, and ionic data.
  + \_\_init\_\_: Initializes the atomic data with default values.
  + get\_radius: Returns the covalent radius.
  + get\_color: Returns the color as an RGBA tuple.
  + get\_vanDerWaals: Returns the van der Waals radius.
  + get\_ionicData: Returns the ionic data.

**Data Structures:**

* IonicRadii: A dictionary containing ionic radii data for various elements, organized by element symbols.
* Elements: A dictionary containing atomic data for various elements, organized by element symbols.

**References:**

* Atomic radii data is sourced from the CRC Handbook of Chemistry and Physics, 101st Edition.

### ImportData.py

This module provides functions for reading and filtering data from molecular structure files, specifically designed to handle Gaussian input files (.com).

**Key Components:**

* **Data Extraction:** Reads data from a specified file and stores each line as a list of values.
* **Data Filtering:** Filters out extra information from the data based on specified criteria.

**Main Functions:**

* ExtractDataFromFile: Reads the content of a file and returns a list of data, where each entry corresponds to a line in the file.
  + path: The path to the file to read.
  + Returns: A list of lists, where each inner list contains the values from a line in the file.
* FilterOutExtraInformation: Filters out extra information from the data based on the specified criteria.
  + spec: A string specifying whether to remove lines 'above' or 'below' a certain number of line breaks.
  + line\_break\_nmbr: The number of line breaks to use as a reference for filtering.
  + extra\_nmbr: An extra number of lines to remove according to the specified criteria.
  + raw\_data: The data matrix to filter.
  + Returns: A filtered data matrix without the information before or after the specified number of line breaks.

### RefineData.py

This module provides functions for refining and processing molecular data, including coordinates and connectivity information, to prepare it for use in Blender.

**Key Components:**

* **Coordinate Refinement:** Converts string coordinates to float and refines the list of coordinates.
* **Connectivity Refinement:** Processes connectivity data, converting bond types to text labels and refining the list of connections.
* **Data Conversion:** Converts strings to integers where applicable and processes lists of tuples.

**Main Functions:**

* RefineCoordList: Converts string coordinates to float for each entry in the list.
  + list: A list where each entry is a list of four strings (symbol and coordinates).
  + Returns: A refined list where coordinates are converted to float.
* RefineConnectivity: Processes connectivity data, converting bond types to text labels and refining the list of connections.
  + list: A list where each entry contains connectivity information for an atom.
  + Returns: A refined list of connections with text labels for bond types.
* ConvertStringToIndexInList: Converts strings that represent integers to actual integers in the list.
  + list: A list containing strings and integers.
  + Returns: A list with strings converted to integers where applicable.
* AddElementSymbolsToConnecrivityList: Adds element symbols to the connectivity list based on the coordinates and number of elements.
  + connect: The connectivity list.
  + coords: The list of coordinates.
  + number\_of\_elements: The number of elements in the molecule.
  + Returns: A list with element symbols added to the connectivity information.
* rebuild\_list: Converts a string representation of a list into an actual list of strings.
  + str\_list: A string representation of a list.
  + Returns: A list of strings.
* make\_tuple: Converts a string representation of a tuple into an actual tuple.
  + str\_in: A string representation of a tuple.
  + Returns: A tuple.
* make\_tuple\_in\_list: Converts a list of string representations of tuples into a list of actual tuples.
  + a\_list: A list of string representations of tuples.
  + Returns: A list of tuples.
* refine\_key\_frames: Refines raw key frame data into a list of vectors.
  + raw\_key\_frames: The raw key frame data.
  + Returns: A refined list of vectors.
* create\_frames\_dict: Creates a dictionary of frames from key frame data.
  + key\_frames: The key frame data.
  + Returns: A dictionary where keys are element names and values are lists of vectors.

### RefineElements.py

This module provides functions for refining and processing element data within molecular structures, including identifying present elements, creating dictionaries of element positions, and updating connectivity lists.

**Key Components:**

* **Element Identification:** Identifies and lists all unique elements present in a molecular structure.
* **Dictionary Creation:** Creates dictionaries mapping element names to their positions.
* **Connectivity Update:** Updates connectivity lists with element labels and indices.

**Main Functions:**

* GetElementsPresentInMolecule: Identifies all unique elements present in the molecule.
  + list: A list of elements and their coordinates.
  + Returns: A list of unique elements present in the molecule.
* CreateDictionaryWithNamesAndPositions: Creates a dictionary mapping element names to their positions.
  + list: A list where each row contains an element name and its coordinates.
  + number\_of\_elements: The number of elements in the molecule.
  + Returns: A dictionary where keys are element names with indices (e.g., "C01") and values are their positions as vectors.
* AddAtomLabelsToConnectList: Updates the connectivity list with element labels and indices.
  + atom\_dict: A dictionary containing atom labels with symbols and indices.
  + connect\_list: The connectivity list to update.
  + Returns: The updated connectivity list with element labels and indices.
* GetDataForExistingElements: Retrieves data for elements present in the molecule from a reference dictionary.
  + list: A list of elements present in the molecule.
  + ref\_dict: A reference dictionary containing element data.
  + Returns: A dictionary containing data for the elements present in the molecule.

### Create\_Materials.py

This module provides functions for creating and assigning materials to elements in a molecular structure using Blender's Python API.

**Key Components:**

* **Material Creation:** Creates materials for elements present in the molecule and assigns colors based on atomic data.
* **Material Management:** Provides functionality to remove all existing materials in the Blender scene.

**Main Functions:**

* CreateAndAssignMaterials: Creates and assigns materials for elements present in the molecule.
  + ref\_dict: A dictionary containing elements and their data.
  + Returns: A dictionary mapping element symbols to their corresponding material objects.
* AssignMaterial: Creates a new material with the specified name and color if it does not already exist.
  + material\_name: The name of the material.
  + material\_color: The RGBA color values for the material.
  + Returns: The created or existing material object.
* RemoveAllMaterials: Removes all materials from the Blender scene.

### Primitives.py

This module provides functions for instantiating and managing 3D primitives (spheres and cylinders) in Blender to represent elements and bonds in a molecular structure.

**Key Components:**

* **Element Instantiation:** Creates spheres to represent elements at specified positions.
* **Bond Instantiation:** Creates cylinders to represent bonds between elements based on connectivity data.
* **Material Assignment:** Assigns materials to the instantiated elements and bonds.

**Main Functions:**

* InstantiateElementsFromDictionary: Instantiates spheres for elements at specified positions and assigns materials.
  + pos\_dict: A dictionary mapping element symbols and labels to their positions.
  + element\_data: A dictionary containing data for the present elements.
  + materials\_dict: A dictionary mapping element symbols to their corresponding materials.
  + van\_der\_waals: A boolean indicating whether to use van der Waals radii.
* InstantiateIonsFromDictionary: Instantiates spheres for ions at specified positions and assigns materials.
  + pos\_dict: A dictionary mapping ion symbols and labels to their positions.
  + ion\_data: A dictionary containing data for the present ions.
  + materials\_dict: A dictionary mapping ion symbols to their corresponding materials.
* ModifyNamesAndMaterials: Changes the names of the active object and assigns the required material.
  + obj\_name: The name of the sphere to be instantiated.
  + e\_symbol: The atom symbol used to access materials.
  + materials\_dict: A dictionary mapping element symbols to their corresponding materials.
* InstantiateBondsFromConnectivity: Instantiates bonds between elements based on connectivity data.
  + pos\_dict: A dictionary mapping element symbols and labels to their positions.
  + mat\_dict: A dictionary mapping element symbols to their corresponding materials.
  + connect\_list: A list of connectivity data.
  + unit\_cell: A string indicating whether to use unit cell boundaries.
* CreateAndJoinTrantientBond: Creates and joins transient bonds between elements.
  + pos\_dict: A dictionary mapping element symbols and labels to their positions.
  + mat\_dict: A dictionary mapping element symbols to their corresponding materials.
  + key1: The first element in the bond.
  + key2: The second element in the bond.
  + bond\_type: The type of bond.
  + dash\_len: The length of each dash in the bond.
  + bond\_radius: The radius of the bond.
  + h\_bonding: A boolean indicating whether to use hydrogen bonding.
* CreateFragmentedBonds: Instantiates bonds from atoms to the midpoint and joins them.
  + pos\_dict: A dictionary mapping element symbols and labels to their positions.
  + mat\_dict: A dictionary mapping element symbols to their corresponding materials.
  + atom1: The first element in the bond.
  + atom2: The second element in the bond.
  + bond\_type: The type of bond.
  + unit\_cell: A string indicating whether to use unit cell boundaries.
* MoveObjectOnLocalAxis: Moves an object along its local axis.
  + obj\_name: The name of the object to move.
  + value: The vector by which to move the object.
* InstantiateBondBetweenTwoPoints: Instantiates a bond between two points.
  + p1: The origin point.
  + p2: The end point.
  + r: The radius of the bond.
* SelectTwoMeshesAndJoin: Selects two meshes and joins them.
  + name1: The name of the first mesh.
  + name2: The name of the second mesh.
* JoinMeshesFromObjectList: Joins a list of meshes into a single object.
  + obj\_list: A list of objects to join.

### Export\_Data.py

This module provides functions for exporting the Blender scene to various file formats, including support for exporting animations.

**Key Components:**

* **Scene Export:** Exports the entire Blender scene to specified file formats.
* **Animation Export:** Exports the scene for animation, including elements and bonds.

**Main Functions:**

* ExportSceneAs: Exports the Blender scene to the specified file format.
  + folder\_path: The directory where the file will be saved.
  + file\_name: The name of the output file.
  + file\_type: The file format for export (e.g., .fbx, .dae, .obj, .x3d, .stl).
  + Exports the scene using Blender's export operators based on the specified file type.
* ExportForAnimation: Exports the scene for animation, including elements and bonds.
  + names\_and\_pos: A dictionary mapping element names to their positions.
  + bond\_list: A list of bond objects.
  + folder\_path: The directory where the file will be saved.
  + file\_name: The name of the output file.
  + file\_type: The file format for export (e.g., .fbx).
  + Selects the elements, bonds, and armature, and exports the scene using Blender's FBX export operator with animation settings.

### Ions.py

This module provides functions for handling ionic data within molecular structures, including creating ion data from input, refining ion positions, and managing ion specifications.

**Key Components:**

* **Ion Data Creation:** Creates ion data from input lists and refines ion positions.
* **Ion Management:** Provides functionality to remove non-specified ions and retrieve correct ionic radii data.

**Main Functions:**

* CreateIonDataFromInput: Creates a dictionary of ion data from an input list.
  + ionInputList: A list of ions and their properties (symbol, charge, coordination).
  + Returns: A dictionary mapping ion symbols to Ionic class instances with specified properties.
* GetIonPositions: Refines the positions of ions based on the input list.
  + names\_and\_pos: A dictionary containing the positions of each labeled element.
  + ion\_input: A dictionary of Ionic class instances with specified properties.
  + Returns: A refined dictionary with only the ions from the input list.
* RemoveNonSpecifiedIons: Removes elements from the ion dictionary that were not specified in the input list.
  + ion\_dict: A dictionary containing symbols and possible ion radii for all present elements.
  + ion\_input: A dictionary of Ionic class instances with specified properties.
  + Returns: A smaller dictionary with only the specified ions.
* RemoveSpecifiedIonsFromElementDict: Removes specified ions from the dictionary of all elements present.
  + ion\_dict: A dictionary containing symbols and possible radii for specified elements.
  + element\_dict: A dictionary of all elements present.
  + Returns: A smaller dictionary with non-specified ions removed.
* GetIonDataFromInput: Retrieves the correct ionic radii data from the input list for each element in the ion dictionary.
  + ion\_data\_dict: A dictionary containing symbols and possible ionic radii for elements of interest.
  + ion\_input: A dictionary of Ionic class instances with specified properties.
  + Returns: A dictionary mapping element symbols to their ionic radii.

### Instantiate\_Molecules.py

This module provides functions for managing the instantiation of 3D geometries (elements and ions) from Cartesian coordinates using Blender's Python API.

**Key Components:**

* **Geometry Instantiation:** Instantiates elements and ions based on their Cartesian coordinates and specified representation types.
* **Origin Setting:** Sets the origin of all instantiated objects to their geometric center.

**Main Functions:**

* Instantiate: Manages the instantiation of all elements and/or ions from Cartesian coordinates.
  + is\_ionic: A string indicating whether the molecule contains ions.
  + represent\_type: The type of representation (e.g., "Ball-and-Stick", "Stick-only", "Van-der-Waals").
  + names\_and\_pos: A dictionary mapping element symbols and labels to their positions.
  + materials\_dict: A dictionary mapping element symbols to their corresponding materials.
  + connect\_with\_symbols: A list of connectivity data with element symbols.
  + element\_data: A dictionary containing data for the present elements.
  + ion\_data: A dictionary containing data for the present ions.
  + ion\_input: A dictionary of Ionic class instances with specified properties.
  + unit\_cell: A string indicating whether to use unit cell boundaries.
  + Instantiates elements and bonds based on the specified representation type and ionic data.
* set\_every\_object\_origin: Sets the origin of all instantiated objects to their geometric center.
  + Iterates through all mesh objects in the scene and sets their origin to the median of their geometry.

### Raw\_Parameters.py

This module provides functions for setting raw parameters from input files, specifically for converting and animating molecular data using Blender's Python API.

**Key Components:**

* **Parameter Extraction:** Extracts raw coordinates and connectivity data from input files.
* **Animation Handling:** Processes raw coordinate frames for animation.

**Main Functions:**

* Set\_Raw\_Parameters\_Convert: Extracts raw coordinates and connectivity data from a specified input file.
  + i\_folder\_path: The input folder path.
  + i\_file\_name: The input file name.
  + Returns: A tuple containing raw coordinates and connectivity data.
* Set\_Raw\_Parameters\_Animate: Extracts raw coordinate frames and connectivity data for animation.
  + Returns: A tuple containing raw coordinates, connectivity data, and key frames.
* split\_coord\_frames: Splits raw coordinate frames into individual coordinates and frames.
  + raw\_coord\_frames: The raw coordinate frames.
  + Returns: A tuple containing lists of coordinates and frames.
* count\_animation\_frames: Counts the number of animation frames in the raw coordinate frames.
  + raw\_coord\_frames: The raw coordinate frames.
  + Returns: The number of animation frames.

### Animate.py

This module provides functions for animating molecular structures in Blender, including clearing existing animations, calculating frames, and updating keyframes for elements and bonds.

**Key Components:**

* **Animation Management:** Clears existing animations, calculates the number of frames, and separates elements from bonds.
* **Keyframe Updates:** Updates keyframe locations and rotations for elements and bonds.
* **Data Extraction and Refinement:** Extracts and refines animation data from files.

**Main Functions:**

* clear\_all\_animations: Removes all keyframes for all objects in the scene.
* calculate\_number\_of\_frames: Calculates the number of frames from the animation data file.
  + anim\_frames\_path: The file path to the animation frames data.
  + Returns: The number of frames.
* separate\_elements\_from\_bonds: Separates objects in the scene into elements and bonds based on their names.
  + Returns: A tuple containing a list of elements and a list of bonds.
* filter\_bond\_list\_by\_type: Filters bonds into categories based on their names.
  + bond\_list: List of bond objects.
  + Returns: A tuple containing lists of different types of bonds.
* insert\_keyframes\_to\_all: Inserts keyframes for all objects in the scene at specified intervals.
  + number\_of\_frames: The total number of keyframes to insert.
  + step\_size: The spacing between each frame.
* update\_keyframe\_locations: Updates keyframe locations for a target object.
  + target: The object to update.
  + step\_size: The interval between frames.
  + locations: The locations for each frame (list of vectors).
* update\_keyframe\_rotations: Updates keyframe rotations for a target object based on normals.
  + target: The object to update.
  + step\_size: The interval between frames.
  + normals: The normal vectors for each frame.
* ExtractDataFromFile: Reads the content of a file and returns a list of data.
  + path: The path to the file to read.
  + Returns: A list of lists, where each inner list contains the values from a line in the file.
* refine\_anim\_data: Refines raw animation data into vectors.
  + raw\_anim\_data: The raw animation data.
  + Returns: A refined list of data points with vectors.
* get\_bond\_locations: Calculates the center of mass for each bond location.
  + bond\_name: The name of the bond.
  + anim\_data: The animation data.
  + type: The bond type.
  + Returns: The list of center locations for each bond.
* get\_bond\_normals: Calculates the normal vector for each bond location.
  + bond\_name: The name of the bond.
  + anim\_data: The animation data.
  + type: The bond type.
  + Returns: The list of normal vectors for the bond.
* animate\_elements\_from\_anim\_data: Animates elements based on provided animation data.
  + anim\_data: The animation data.
  + step\_size: The interval between frames.
* animate\_bonds\_by\_type\_list: Animates bonds based on their type and provided animation data.
  + bond\_type\_list: The list of bonds to animate.
  + anim\_data: The animation data.
  + bond\_type: The bond type.
  + step\_size: The interval between frames.
* detect\_bond\_types: Extracts the unique bond types from the bond list.
  + bond\_list: List of bond objects.
  + Returns: A set of unique bonds in the bond list.
* build\_animations: Animates elements and bonds based on the provided data.
  + anim\_data: The animation data for elements.
  + bond\_list: The list of bonds to animate.
  + bond\_types: A dictionary with all the unique bond types in the molecule.
  + step\_size: The interval between frames.
  + extra\_frames: The number of additional frames.
  + end\_frame: The end frame for the animation.
* bake\_all\_animations: Bakes all animations for every object in the scene.
  + end\_frame: The end frame for the animation.
* animate: Animates elements and bonds in the scene based on the provided animation data.
  + anim\_frames\_path: The file path to the animation frames data.
  + step\_size: The number of frames between keyframes.
* export\_animation: Exports the animation to the given filepath.
  + filepath: Path to save the exported file.

### Clear\_Transforms.py

This module provides functions for applying transformations to bonds and elements in a molecular structure using Blender's Python API.

**Key Components:**

* **Transform Application:** Applies location and rotation transformations to bonds and elements.
* **Bond List Retrieval:** Retrieves a list of bond objects from the Blender scene.

**Main Functions:**

* Apply\_Bond\_Transforms: Applies location and rotation transformations to a list of bond objects.
  + bond\_list: A list of bond objects to transform.
  + Selects each bond, applies the transformations, and then deselects all objects.
* Apply\_Element\_Transforms: Applies location and rotation transformations to elements based on their positions.
  + names\_and\_pos: A dictionary mapping element names to their positions.
  + Selects each element, applies the transformations, and then deselects all objects.
* get\_bond\_obj\_list: Retrieves a list of bond objects from the Blender scene.
  + Returns: A list of bond objects whose names contain "-", "=", or "#".

### Receive\_Parameters.py

This module provides functions for extracting and processing parameter data from a file, which is used to configure the molecular conversion process in Blender.

**Key Components:**

* **Parameter Extraction:** Reads parameter data from a specified file and processes it into a usable format.
* **Parameter Dictionary:** Converts the extracted data into a dictionary for easy access.

**Main Functions:**

* extract\_parameters\_data: Reads the content of a parameters file and returns a list of stripped lines.
  + params\_path: The path to the parameters file.
  + Returns: A list of stripped lines from the file.
* get\_parameters\_data: Processes the extracted parameter data into a dictionary.
  + params\_path: The path to the parameters file.
  + Returns: A dictionary containing the parameter data with keys for input folder path, input file name, output folder path, output file name, representation type, output file type, ionic information, unit cell information, ion input list, and animation flag.

### BondOrderCalculator.py

This module, BondOrderCalculator, is designed to compute bond orders between atoms using their positions in space and their covalent radii. Here's a brief overview of each method:

1. **\_\_init\_\_**: Initializes the BondOrderCalculator by loading covalent radii data from a JSON file.
2. **initialize\_covalent\_radii**: Loads covalent radii data from a JSON file located in the external\_data directory relative to the script's location.
3. **get\_covalent\_lengths\_for\_atoms**: Retrieves and sums the covalent radii for two given atoms. It returns a list containing the summed covalent radii for single, double, and triple bonds, with any '-' values represented as None.
4. **get\_bond\_length\_from\_coordinates**: Calculates the distance between two points in Cartesian space, given their coordinates.
5. **get\_bond\_order\_from\_coordinates**: Determines the bond order based on the distance between two atoms and their covalent radii. It returns the bond order (1 for single, 2 for double, 3 for triple) or None if no bond order is found within the given threshold.