GUI Technical Documentation

## gui Folder

This folder contains all of the classes that make up the graphical user interface for the user to interact with and modify the loaded board design files in visual format.

### **MainApplication.java**

This class serves as the primary view window for the auto-router application’s graphical user interface. It creates new frames on the window whenever new board design files are created or loaded. It is also responsible for exporting board designs created in the application as individual files.

**Key Methods**

* private static boolean InitializeGUI()
* This method intializes the graphical user interface whenever it is called.
* private static BoardFrame create\_board\_frame(DesignFile p\_design\_file, JTextField p\_message\_field, boolean p\_is\_test\_version, Locale p\_locale, String p\_design\_rules\_file, boolean p\_save\_intermediate\_stages, float p\_optimization\_improvement\_threshold, String[] p\_ignore\_net\_classes\_by\_autorouter)
* This method creates a new board frame based on the input design file when it is called.
* This method is called within this class’s **InitializeGUI()** method.
* private void open\_board\_design\_action(ActionEvent evt)
* This method opens up a file explorer to allow the user to choose which design file to load.
* This method can only load either a SPECCTRA Design file (*dsn*) or a Freerouter Binary file (*frb*).

### **DesignFile.java**

This class serves a file handler to import and export specific design file formats. These are the following formats the application can accept or export through this class:

* **dsn** - SPECCTRA Design file
* **frb** - Freerouting Binary file
* **rules** - Rules file
* **ses** - SPECCTRA Session file
* **scr** - Eagle Script file

The DesignFile class restricts the application to only import and load *dsn* and *frb* files while allowing it to save and export *ses* and *rules* files. By default, the files are saved in *ses* format.

### **BoardFrame.java**

This class serves as the main view of the application. It consists of the dropdown menu, sidebar, canvas, and status bar. Whenever a new design file is loaded, the **MainApplication.java** class creates new BoardFrame objects through the **InitializeGUI()** method by calling the **create\_board\_frame()** method. This class is also responsible for writing and exporting the files in the formats specified in the **DesignFile.java** class.

**Key Methods**

private boolean saveAsBinary(File outputFile)

* This method writes the current board design file as a Freerouting Binary file (*frb*).
* The method will return **false** if the the writing process fails at any point.
* public boolean saveAsSpecctraSessionSes(File outputFile, String designName)
* This method writes the current board design file as a SPECCTRA Session file (*dsn*)
* The method will return **false** if the the writing process fails at any point.
* By default, this method is called when the user exists out of the save window.

private boolean saveRulesAs(File rulesFile, String designName, BoardHandling p\_board\_handling)

* This method writes the current board design file as a Rules file.
* The method will return **false** if the the writing process fails at any point.

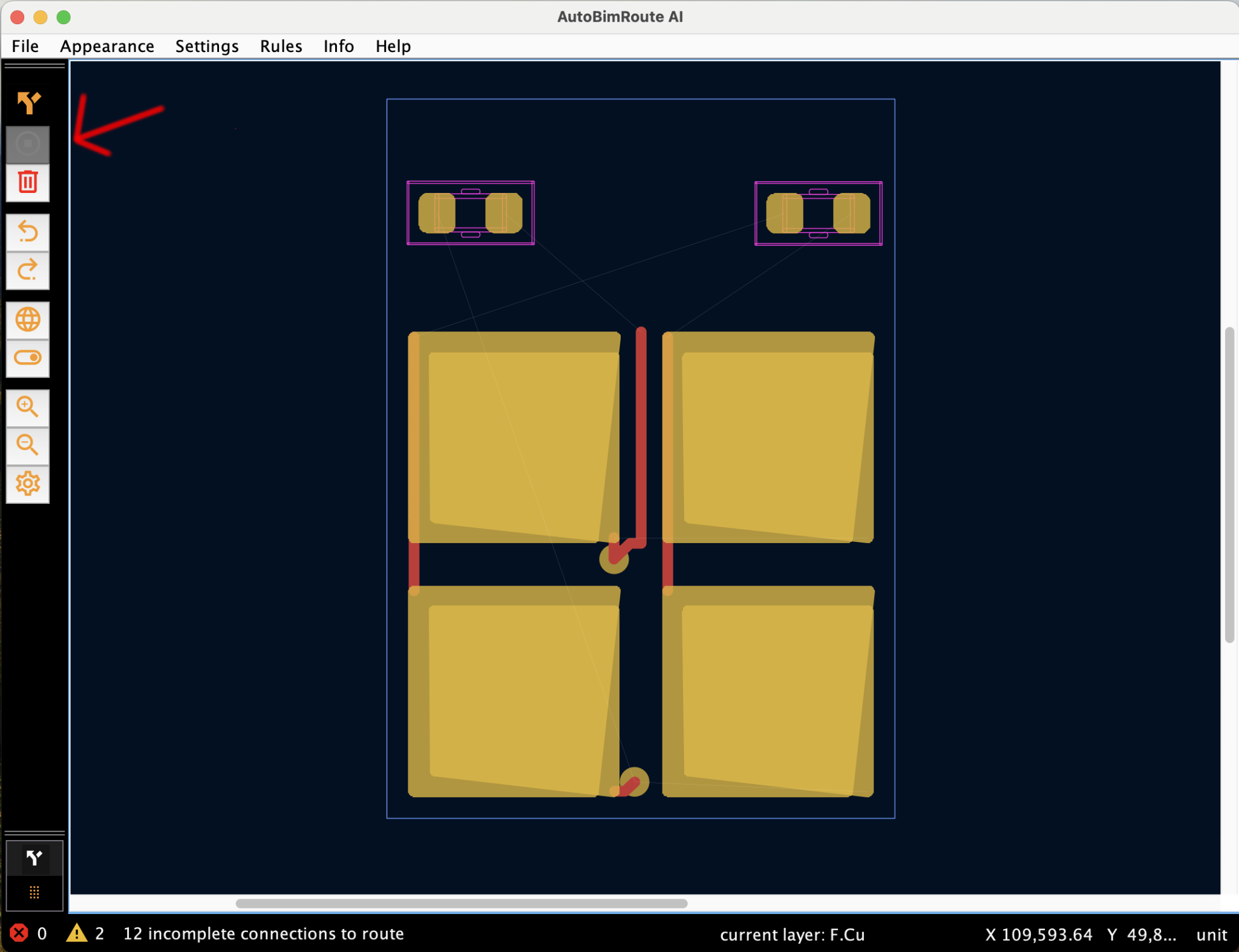
public void saveAsEagleScriptScr(File outputFile, String design\_name)

* This method writes the current board design file as an Eagle Script file (*scr*).
* public boolean saveAsSpecctraDesignDsn(File outputFile, String designName, boolean compatibilityMode)
* This method writes the current board design as a SPECCTRA Design file (*dsn*).
* boolean load(InputStream p\_input\_stream, boolean isSpecctraDsn, JTextField p\_message\_field)
* This method reads an existing board design from a file. The method is configured to only read *dsn* files, so the method will return **false** if it attempts to read any other format.
* Upon successful *dsn* reading, the method will call **BoardHandling**’s **loadFromSpecctraDsn** with the input file, board observers, and a newly generated ID number.
  + If unsuccessful, the method will return **false**.

### **BoardPanel.java**

This class serves as the panel where the board design is rendered on the graphical user interface. The user can view and interact with the board design rendered on this panel. All of the user’s mouse actions (e.g. moving, clicking, dragging, holding, etc.) are handled in this class.

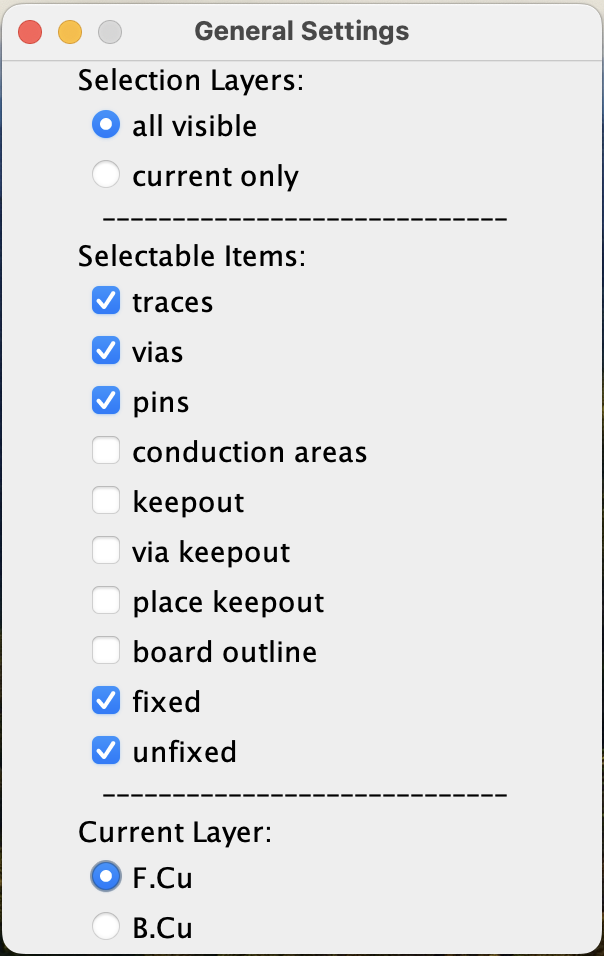
### **BoardToolbar.java**



This class serves as the application’s sidebar, containing all of the buttons used for auto-routing, undoing and redoing, zooming in and out, and much more. These are the component variables associated with the toolbar’s buttons:

* **settings\_button**: opens up the auto-router settings on click
* **toolbar\_autoroute\_button**: initiates potential route generation and optimization on the board design on click
* **cancel\_button**: cancels route generation on click
* **toolbar\_undo\_button**: renders the board design back to the previous action on click
* **toolbar\_redo\_button**: renders the board design forward to the most recent action on click
* **toolbar\_incompletes\_button**: toggles the display the rats nests (incomplete) routes on or off with every click
* **toolbar\_violation\_button**: toggles the display of clearance violations on or off with every click
* **toolbar\_display\_region\_button**: enables the user to zoom into their desired region on click
* **toolbar\_display\_all\_button**: zooms out to the whole board on click
* **delete\_all\_tracks**: deletes all existing tracks on the board on click
* **route\_button:** enables manual routing mode on click
* **drag\_button**: enables component dragging mode on click

### **WindowSelectParameter.java**



This class serves as the general settings window for the application, where the user can toggle the selectable layers, selectable components, and the current visible layer.

It is composed of several ActionListeners, all of which are assigned to the respective categories in the window.

**Key Methods**

* private class CurrentLayerListener implements ActionListener
* {
* public final int signal\_layer\_no;
* public final int layer\_no;
* public CurrentLayerListener(int p\_signal\_layer\_no, int p\_layer\_no)
* {
* signal\_layer\_no = p\_signal\_layer\_no;
* layer\_no = p\_layer\_no;
* }
* @Override
* public void actionPerformed(ActionEvent p\_evt)
* {
* board\_handling.set\_current\_layer(layer\_no);
* }
* }
* The **CurrentLayerListener.java** class is configured to set the currently visible layer to the layer that the user clicked on.
* Once clicked on, the board components that do not belong on the selected layer will be grayed out.
* private class AllVisibleListener implements ActionListener
* {
* @Override
* public void actionPerformed(ActionEvent p\_evt)
* {
* board\_handling.settings.set\_select\_on\_all\_visible\_layers(true);
* }
* }
* The **AllVisibleListener** class is configured to enable all components within the visible layers to be selectable.
* private class CurrentOnlyListener implements ActionListener
* {
* @Override
* public void actionPerformed(ActionEvent p\_evt)
* {
* board\_handling.settings.set\_select\_on\_all\_visible\_layers(false);
* }
* }
* The **CurrentOnlyListener** is configured to enable only the components within the currently selected layer to be selectable.
* private class ItemSelectionListener implements ActionListener
* The **ItemSelectionListener** is configured to enable specific components to be selectable within the selected layers.
* Note that due to the length of the code block, it was only shortened to the class definition for this documentation.

## DSN/Application Flow

**DSN File Opening (Open)**

1. User starts up the application
2. The **MainApplication** object is created
3. **MainApplication** calls **InitializeGUI()**
4. User clicks on “File” dropdown menu
5. User clicks on “Open”
6. MainApplication calls **create\_board\_frame(...)** to open up the file explorer
7. User selects desired dsn file
8. User clicks “Open”
9. **MainApplication** calls **load()** from the **BoardFrame** class
10. **BoardFrame** checks if the file is in dsn format
11. If the file is in dsn format:
    1. **BoardFrame** calls **loadFromSpecctraDsn()** via **BoardHandling** from within **BoardPanel**.
    2. If the file was read successfully, **BoardFrame** calls **intialize\_windows()** to initialize the windows. **BoardFrame** then notifies all its observers that a new board has been loaded via **listener.accept(boardpanel\_.board\_handling.get\_routing\_board()).**
12. If not:
    1. **load()** returns false

**File Writing (Save As)**

1. User clicks on “File” dropdown menu
2. User clicks on “Save as”
3. User selects dsn file format to save as
4. User clicks “Save”
5. **MainApplication** calls **ExportBoardToFile()**
6. If the no file exists or the file name extension does not end with dsn, the method returns null.
7. Otherwise, MainApplication calls **saveAsSpecctraDesignDsn()** through the **BoardHandling** class from within **new\_frame**’s **board\_panel**.
8. **BoardHandling** will check to see if the the board is read-only or if there is no output stream. If either one satisfies the condition, the **saveAsSpecctraDesignDsn()** method will immediately return **false**.
9. Otherwise, it will call **DsnFile.write()** to write the file and save it as **true** if the writing process was successful.
10. If the file writing was successful (**true**), the CRC32 (Cyclic Redundancy Check 32-bit) will be calculated via the **calculateCrc32()** method to check for any alterations in the writing process.
11. The boolean wasSaveSuccessful will be returned.