Interactive Technical Documentation

## interactive Folder

This folder contains all the classes that handle the graphical user interface interactivity. It registers all the user input sent from the interface to perform the necessary calculations and send the results back to the interface.

### **InteractiveState.java**

This class serves as the base class of all interactive states within the graphical user interface. It is the controller behind the application’s interface, which handles all of the user inputs sent from the front end to the back end.

**Key Methods**

* public void draw(Graphics p\_graphics)
* This is the default draw method to be overwritten in derived classes.
* It will be called by other classes that require the board to be drawn on the graphical user interface.
* public InteractiveState left\_button\_clicked(FloatPoint p\_location)
* This method handles the cases when the left mouse button is clicked. It serves to be overwritten by InteractiveState’s derived classes. Otherwise, it return **this** state.
* public InteractiveState button\_released()
* This method handles the cases when a mouse button is released. It serves to be overwritten by InteractiveState’s derived classes. Otherwise, it returns **this** state.
* public InteractiveState mouse\_moved()
* This method handles cases when the mouse cursor is moved. The default function defined in this method is to be overwritten in derived classes. Otherwise, it will return **this** state if it ends at any point.

### **BoardHandling.java**

This class serves as the controller behind the graphical user interface. It serves as a connection between the interface and the board database by handling all of the user inputs with the board design rendered in the interface. Since this class inherits from the **BoardHandlingHeadless.java** class, it has access to all its variables and methods.

**Key Methods**

* public void draw(Graphics p\_graphics)
* {
* if (board == null)
* {
* return;
* }
* board.draw(p\_graphics, graphics\_context);
* if (ratsnest != null)
* {
* ratsnest.draw(p\_graphics, graphics\_context);
* }
* if (clearance\_violations != null)
* {
* clearance\_violations.draw(p\_graphics, graphics\_context);
* }
* if (interactive\_state != null)
* {
* interactive\_state.draw(p\_graphics);
* }
* if (interactive\_action\_thread != null)
* {
* interactive\_action\_thread.draw(p\_graphics);
* }
* }
* This method is draws the board design with all the configurations from the **WindowsSelectParameter.java** class.
* public void set\_layer\_visibility(int p\_layer, double p\_value)
* This method modifies the board layer visibility displayed in the graphical user interface.
* The p\_value is expected to be between 0 and 1.
* public void generate\_snapshot()
* This method calls the routing board within **BoardHandlingHeadless.java** class to generate a snapshot for undoing and redoing.
  + The snapshot will contain the board design’s saved state to allow the user to revisit should they need to correct the design.
* This method calls **generate\_snapshot()** within **BoardHandlingHeadless.java**’s **RoutingBoard** board variable, which inherits from the **BasicBoard.java** class.
  + **BasicBoard.java**’s **generate\_snapshot()** method will be called via **BoardHandlingHeadless.java**’s **RoutingBoard** board.
* public void repaint()
* This method repaints the board panel in the graphical user interface.
* public void repaint(Rectangle p\_rect)
* By using a Rectangle object, this method repaints the rectangular boarder of the board panel in the graphical user interface.
* public void generate\_snapshot()
* {
* if (board\_is\_read\_only)
* {
* return;
* }
* board.generate\_snapshot();
* activityReplayFile.start\_scope(ActivityReplayFileScope.GENERATE\_SNAPSHOT);
* }
* This method generates snapshots for the user to restore previous or most recent actions performed on the board.
* The snapshot won’t be generated if the board is restricted to read-only.
* public void undo()
* This method restores the situation before the previous snapshot generated.
* This method is called when the user clicks on the **toolbar\_undo\_button** in the **BoardToolbar.java** class.
* public void redo()
* This method restores the situation before the last undone action.
* This method is called when the user clicks on the **toolbar\_redo\_button** in the **BoardToolbar.java** class.
* public boolean loadFromBinary(ObjectInputStream p\_design)
* This method loads a *frb* file from the file explorer upon selecting a specific *frb* file.
* public DsnFile.ReadResult loadFromSpecctraDsn(InputStream p\_design, BoardObservers p\_observers, IdNoGenerator p\_item\_id\_no\_generator)
* This method loads a *dsn* file from the file explorer upon selecting a specific *dsn* file.
* The method will return false if the file is corrupted and unreadable.

### **BoardHandlingHeadless.java**

This class is similar to the **BoardHandling.java** class, except that it is primarily used to handle the board interactions in case the graphical user interface is not available to use. It serves as a parent class to the **BoardHandling.java** class.

**Key Methods**

* @Override
* public RoutingBoard get\_routing\_board()
* {
* return this.board;
* }
* This method retrieves the current routing board loaded when it is called.
* @Override
* public void create\_board(IntBox p\_bounding\_box, LayerStructure p\_layer\_structure, PolylineShape[] p\_outline\_shapes, String p\_outline\_clearance\_class\_name, BoardRules p\_rules, Communication p\_board\_communication)
* This method creates a new circuit board design when it is called.
* This method will be overridden by the create\_board() method defined in the **BoardHandling.java** class.

## 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.