CAMBRIAN

USING CAMBRIAN'S HOME AUGMENTATION SDK

Cambrian Home Augmentation SDK

Implementation is made to be as simple as possible, handling computer vision and rendering tasks under the hood without interaction from the application. The entire core SDK is encapsulated within a single UIView that can be placed within a storyboard.

Primary components

- `View`
  - Holds everything, manages state and lifecycle
- `Scene`
  - Holds information about current image/video being edited
  - Assets, lighting, masks, etc
- `Asset`
  - Generic parent type for remodeling types. Paint, floor, furniture, etc

CBRemodelingView

- Parent object of the Cambrian Home Augmentation SDK
- Responsibilities:
  - Lifecycle
  - Maintaining state
  - Setting tool modes for interaction

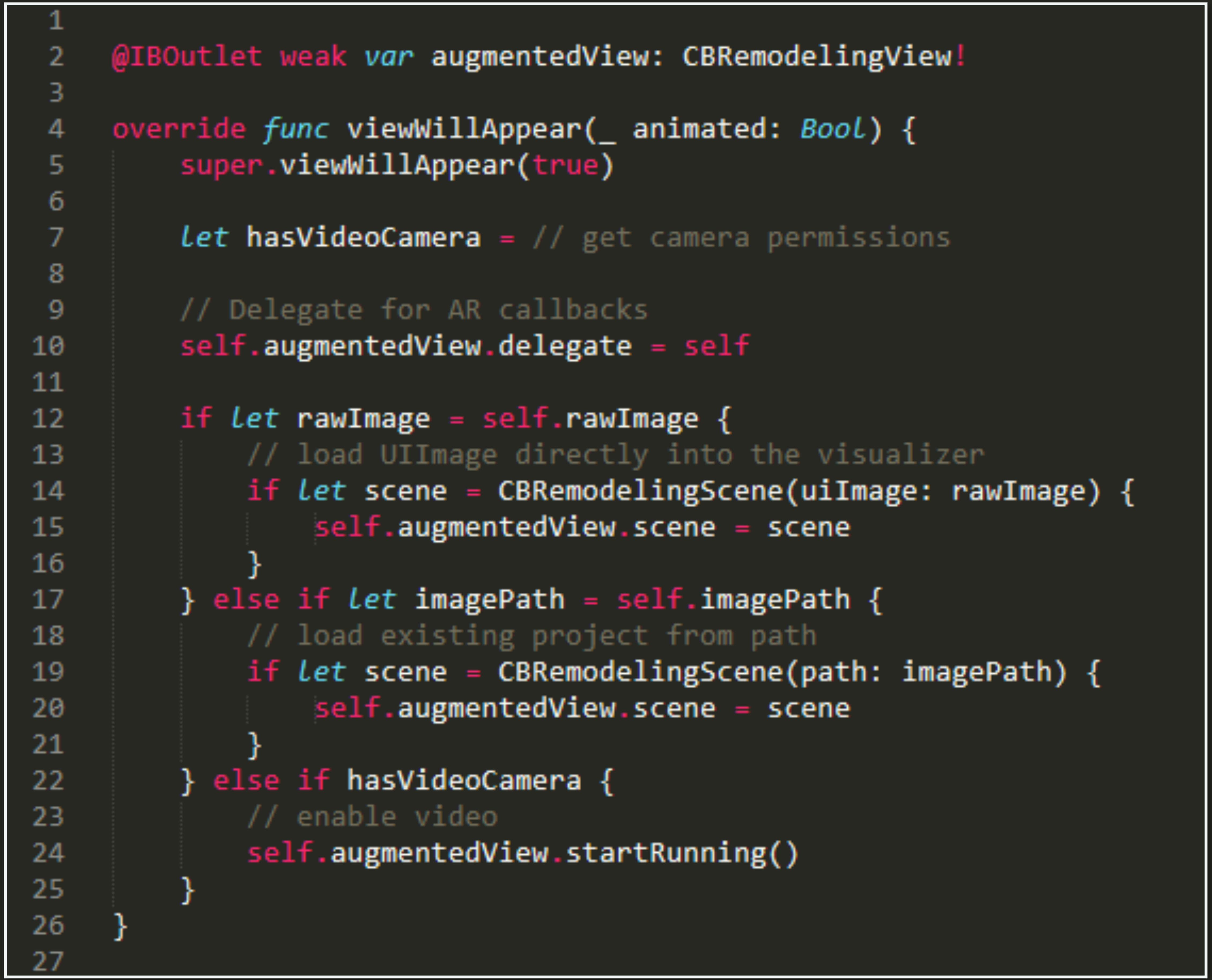
CBRemodelingScene

- Container for all information about current image or video being edited
- Responsibilities:
  - Initialization of image/video to be edited
  - Storing and modifying assets (paints, floors, furniture, etc)
  - Saving a fully-editable image that can be loaded later
  - Storing and modifying lighting and color temperature

CBRemodelingPaint

- Paint-specific asset object
- A mutable container for paint data, allowing hot-swapping of any paint information while maintaining existing masks, in still or video
- Values:
  - Color
  - Transparency (For stains)
  - Sheen (Gloss level)
  - UserData - for storing any value within the asset, typically ID’s or history

Initializing CBRemodelingView



```swift

@IBOutlet weak var augmentedView: CBRemodelingView!

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(true)

    let hasVideoCamera = // get camera permissions

    // Delegate for AR callbacks
    self.augmentedView.delegate = self

    if let rawImage = self.rawImage {
        // load UIImage directly into the visualizer
        if let scene = CBRemodelingScene(uiImage: rawImage) {
            self.augmentedView.scene = scene
        }
    } else if let imagePath = self.imagePath {
        // load existing project from path
        if let scene = CBRemodelingScene(path: imagePath) {
            self.augmentedView.scene = scene
        }
    } else if hasVideoCamera {
        // enable video
        self.augmentedView.startRunning()
    }
}

```

- Get CBRemodelingView instance from storyboard
- Set delegate for callbacks
- Determine state view should initialize into based on variables set by segues or other methods
    - New image
    - Previously saved CBScene
    - Live video
- Ensure camera permissions are available before starting video
- Parent class must implement CBRemodelingViewDelegate

Add a paint (asset) to the scene

```swift

func appendPaint(_ color: UIColor) {
    let paint = CBRemodelingPaint(assetID: UUID().uuidString)
    paint.color = color
    self.augmentedView.scene.appendAsset(paint)
}

```

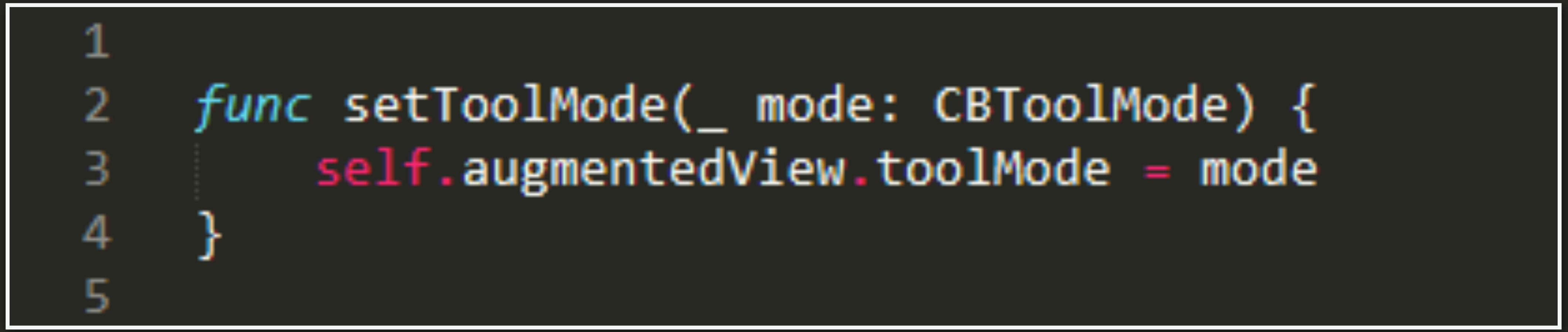
Set selected asset

```swift

func selectedPaint(_ id: String) {
    if let selected = self.augmentedView.scene.assets[id] {
        self.augmentedView.scene.selectedAsset = selected
    }
}

```

Remove asset

```swift

func removePaint(_ id: String) {
    if self.augmentedView.removeAsset(id) {
        // successfully deleted
    } else {
        // failed to delete (no matching id)
    }
}

```

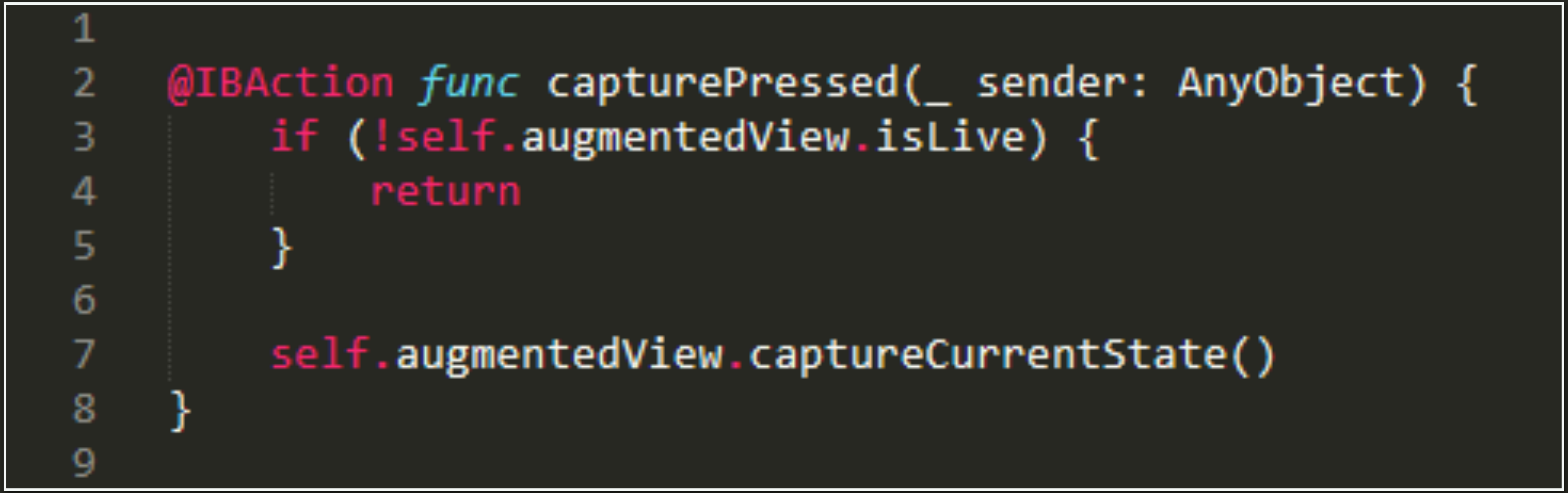
Set Tool Mode



```swift

func setToolMode(_ mode: CBToolMode) {
    self.augmentedView.toolMode = mode
}

```

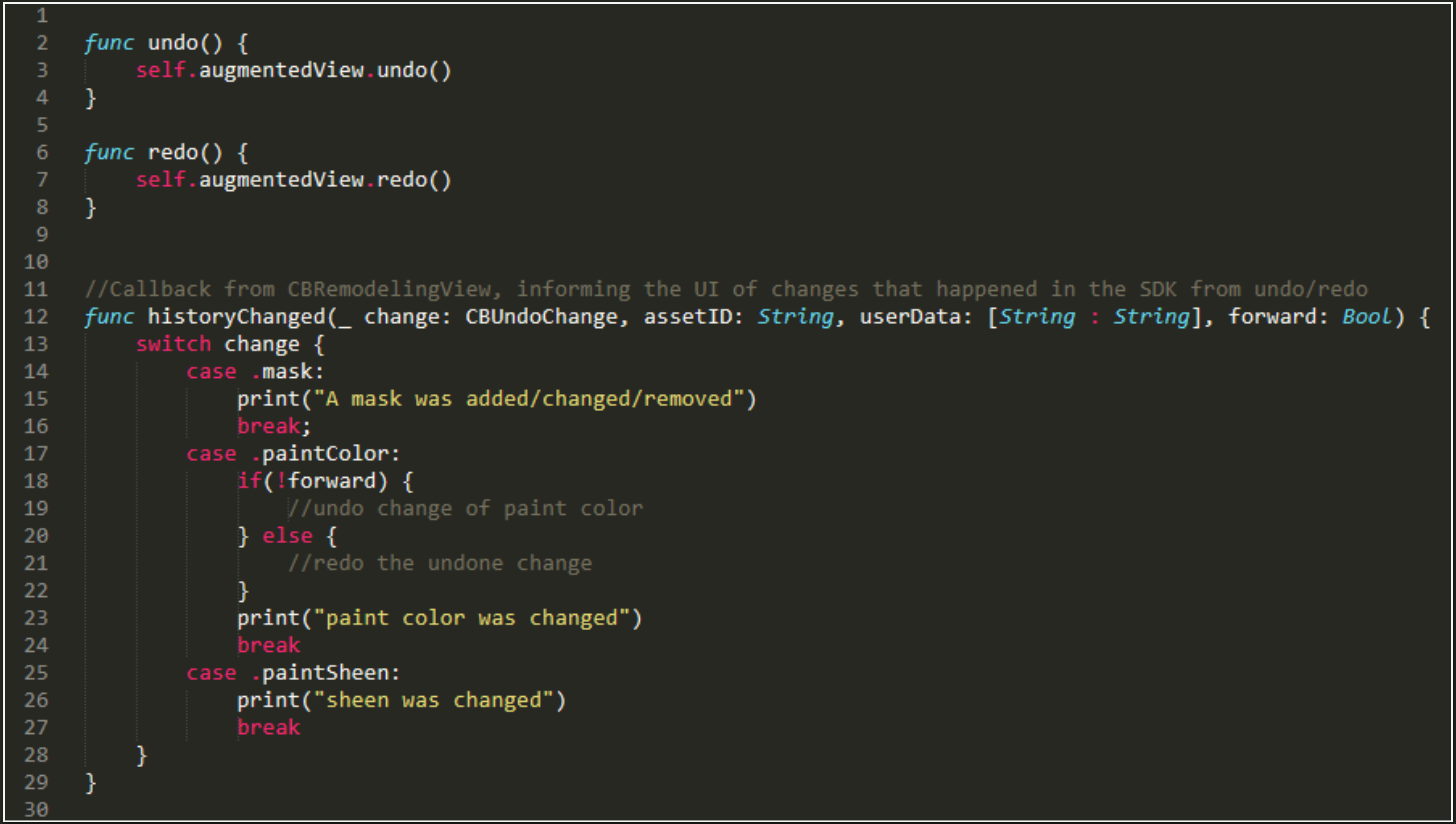
- Available Modes
  - None
  - Fill
  - Unfill
  - Paintbrush
  - Eraser
  - FindColor

Capture video into image



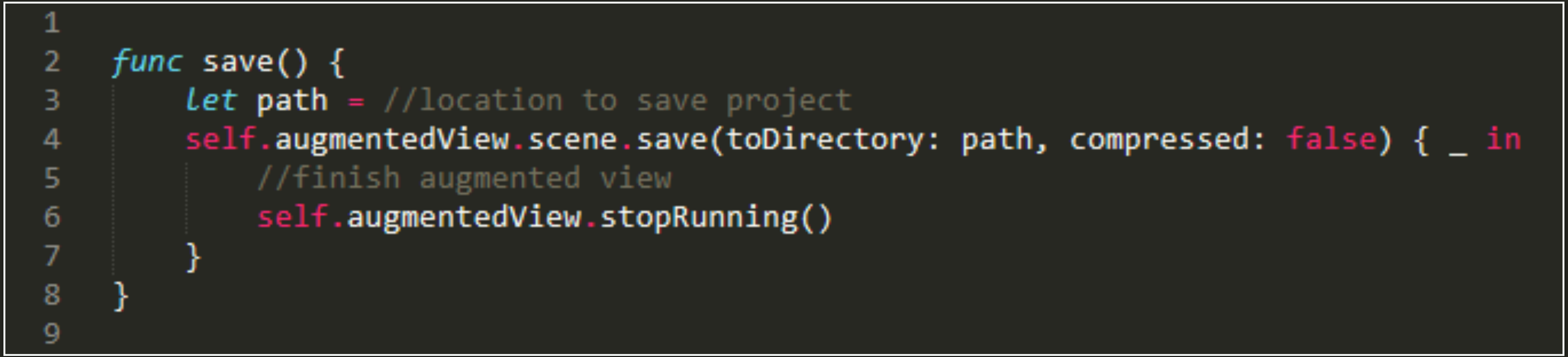
```swift

@IBAction func capturePressed(_ sender: AnyObject) {
    if (!self.augmentedView.isLive) {
        return
    }

    self.augmentedView.captureCurrentState()
}

```

Handling undo and redo



```swift

func undo() {
    self.augmentedView.undo()
}

func redo() {
    self.augmentedView.redo()
}


//Callback from CBRemodelingView, informing the UI of changes that happened in the SDK from undo/redo
func historyChanged(_ change: CBUndoChange, assetID: String, userData: [String : String], forward: Bool) {
    switch change {
        case .mask:
            print("A mask was added/changed/removed")
            break;
        case .paintColor:
            if(!forward) {
                //undo change of paint color
            } else {
                //redo the undone change
            }
            print("paint color was changed")
            break
        case .paintSheen:
            print("sheen was changed")
            break
    }
}

```

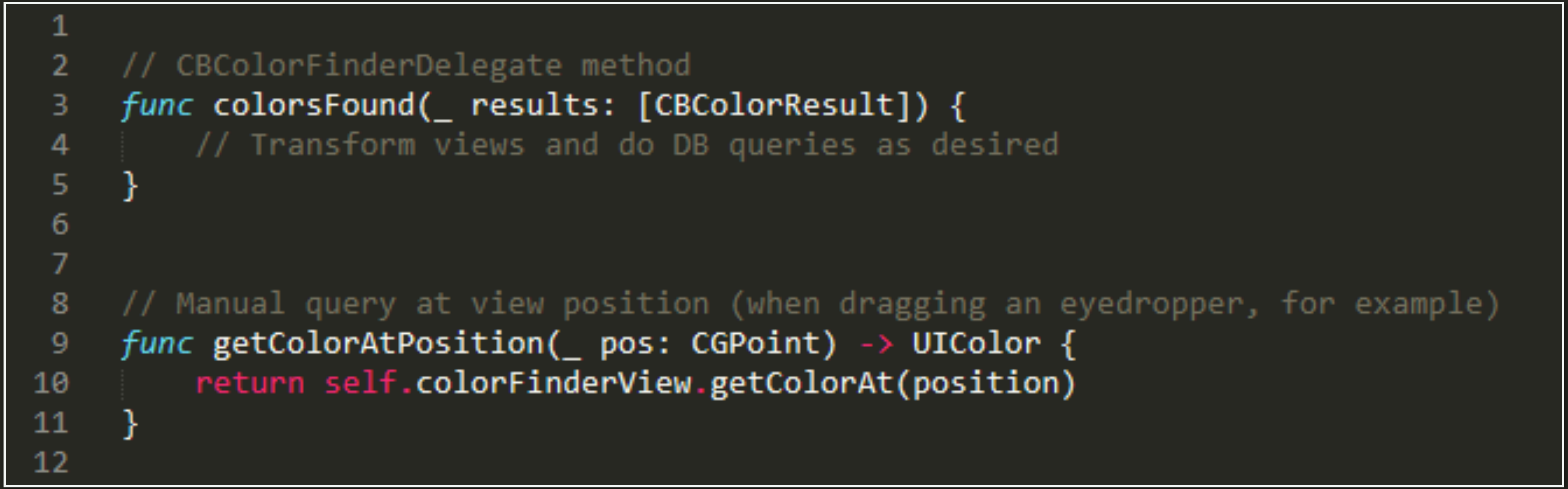
Saving to a CBImage



```swift

func save() {
    let path = //location to save project
    self.augmentedView.scene.save(toDirectory: path, compressed: false) { _ in
        //finish augmented view
        self.augmentedView.stopRunning()
    }
}

```

CBColorFinderView

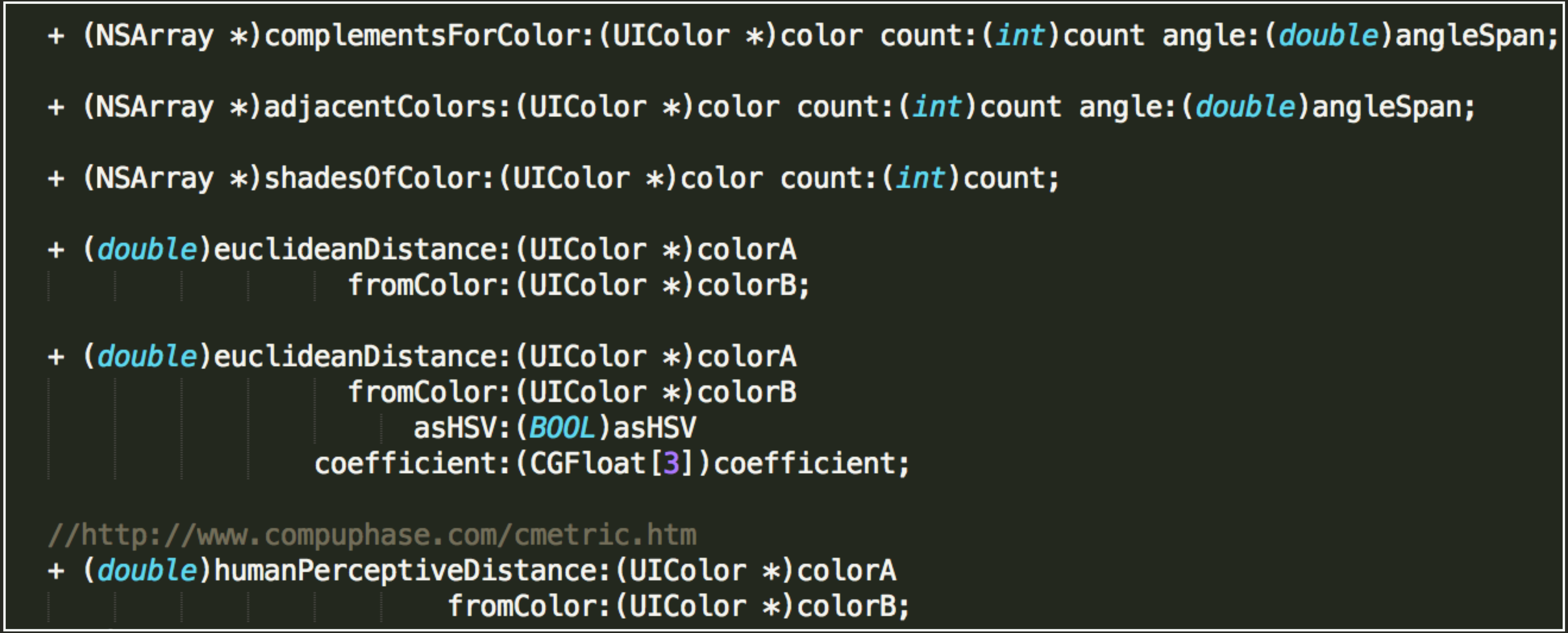
- Scans video or image for prominent colors
- In video it constantly searches for new colors as the camera moves
- Initialization of view, delegate, and image/video are identical to `CBRemodelingView`
- Implement `CBColorFinderDelegate`
- Delegate method `colorsFound(_ results: [CBColorResult])` returns 5 colors every ~3 seconds
    - Contains `UIColor` and `CGPoint`, representing the view-space position that the color was found
- Details on specific implementation, such as working with a DB, exist in the demo app



```swift

// CBColorFinderDelegate method
func colorsFound(_ results: [CBColorResult]) {
    // Transform views and do DB queries as desired
}


// Manual query at view position (when dragging an eyedropper, for example)
func getColorAtPosition(_ pos: CGPoint) -> UIColor {
    return self.colorFinderView.getColorAt(position)
}

```

CBColoring

- Class for common color match and differentiation methods
- Simplifies commonly used color algorithms, such as those used in match tools



```objc
+ (NSArray *)complementsForColor:(UIColor *)color count:(int)count angle:(double)angleSpan;

+ (NSArray *)adjacentColors:(UIColor *)color count:(int)count angle:(double)angleSpan;

+ (NSArray *)shadesOfColor:(UIColor *)color count:(int)count;

+ (double)euclideanDistance:(UIColor *)colorA
                  fromColor:(UIColor *)colorB;

+ (double)euclideanDistance:(UIColor *)colorA
                  fromColor:(UIColor *)colorB
                      asHSV:(BOOL)asHSV
                coefficient:(CGFloat[3])coefficient;

//http://www.compuphase.com/cmetric.htm
+ (double)humanPerceptiveDistance:(UIColor *)colorA
                        fromColor:(UIColor *)colorB;
```

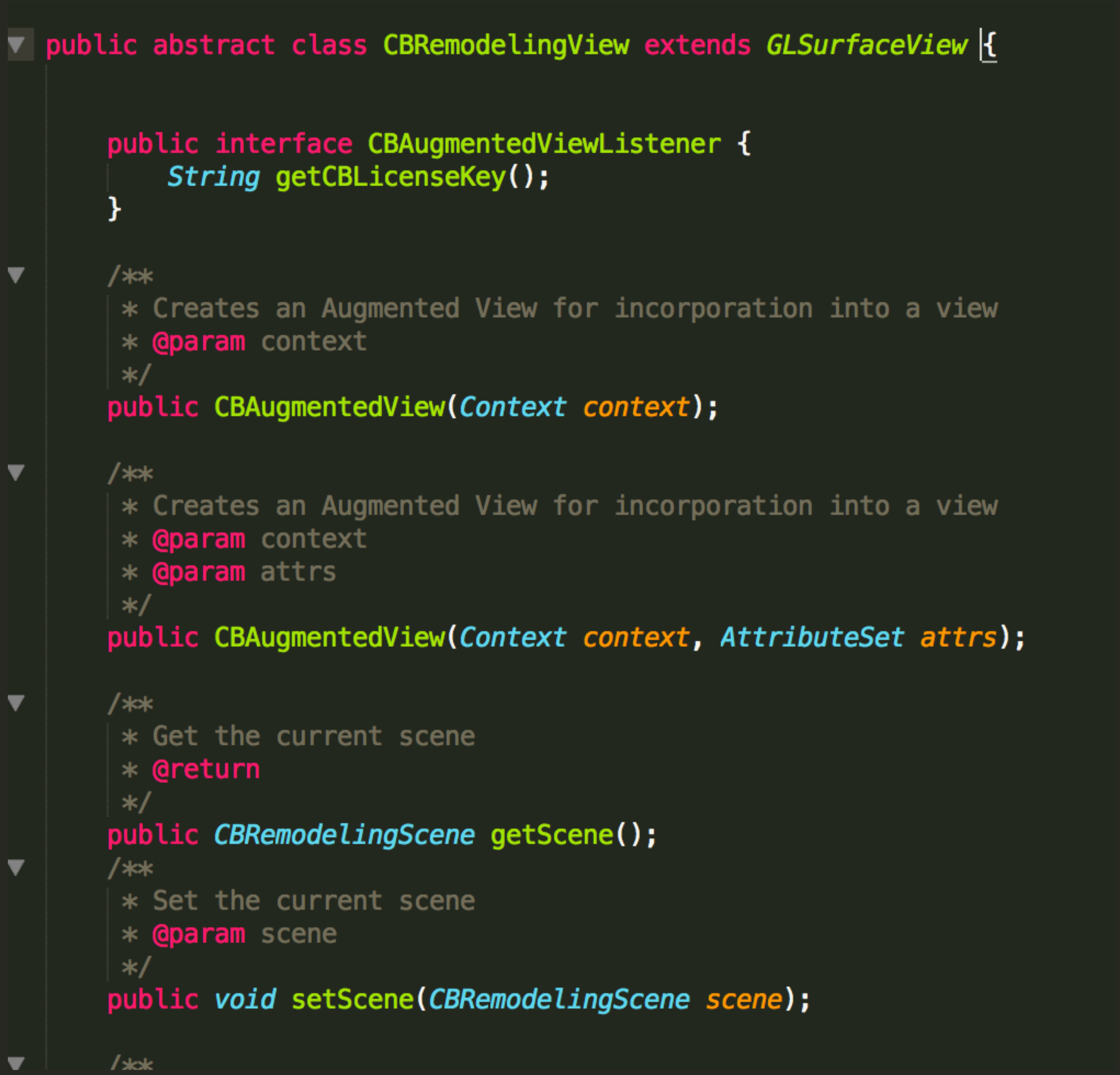
Cambrian Android SDK

Identical to iOS SDK in terms of most objects and interfaces. Relies on the same binary SDK rendering engine. Utilizes a GLSurfaceView as its primary interface and bitmaps instead of UIImage, Fully customizable in user interface..

Primary components

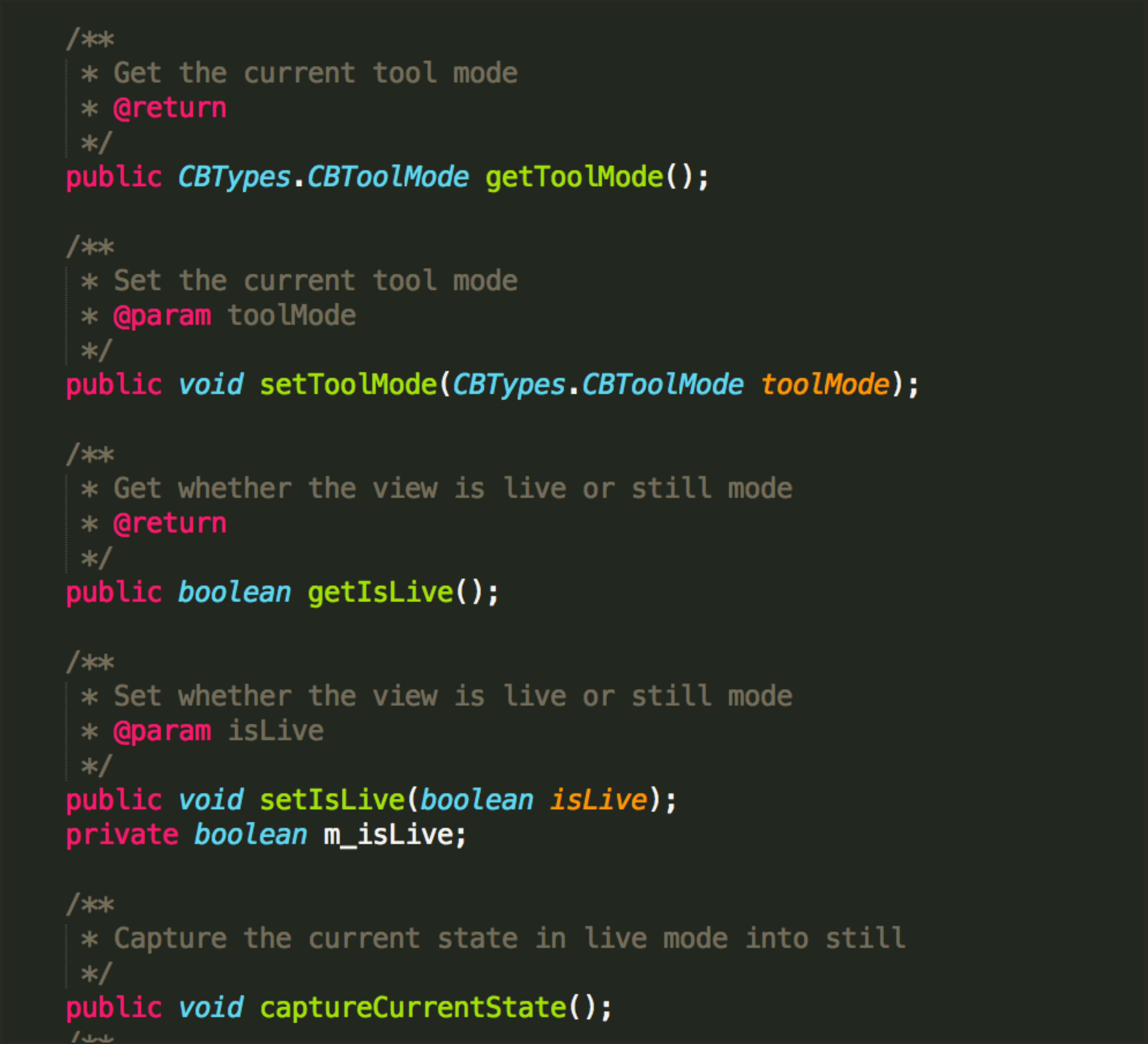
- `View`
  - Holds everything, manages state and lifecycle
- `Scene`
  - Holds information about current image/video being edited
  - Assets, lighting, masks, etc
- `Asset`
  - Generic parent type for remodeling types. Paint, floor, furniture, etc

Initializing CBRemodelingView


```
public abstract class CBRemodelingView extends GLSurfaceView {


    public interface CBAugmentedViewListener {
        String getCBLicenseKey();
    }

    /**
     * Creates an Augmented View for incorporation into a view
     * @param context
     */
    public CBAugmentedView(Context context);

    /**
     * Creates an Augmented View for incorporation into a view
     * @param context
     * @param attrs
     */
    public CBAugmentedView(Context context, AttributeSet attrs);

    /**
     * Get the current scene
     * @return
     */
    public CBRemodelingScene getScene();
    /**
     * Set the current scene
     * @param scene
     */
    public void setScene(CBRemodelingScene scene);

    /**
```

- Add CBRemodelingView instance to layout
- Set listener for callbacks
- Choose appropriate initialization method
  - New image
  - Previously saved CBScene
  - Live video
- Ensure camera permissions are available before starting video
- Class must implement CBRemodelingViewListener

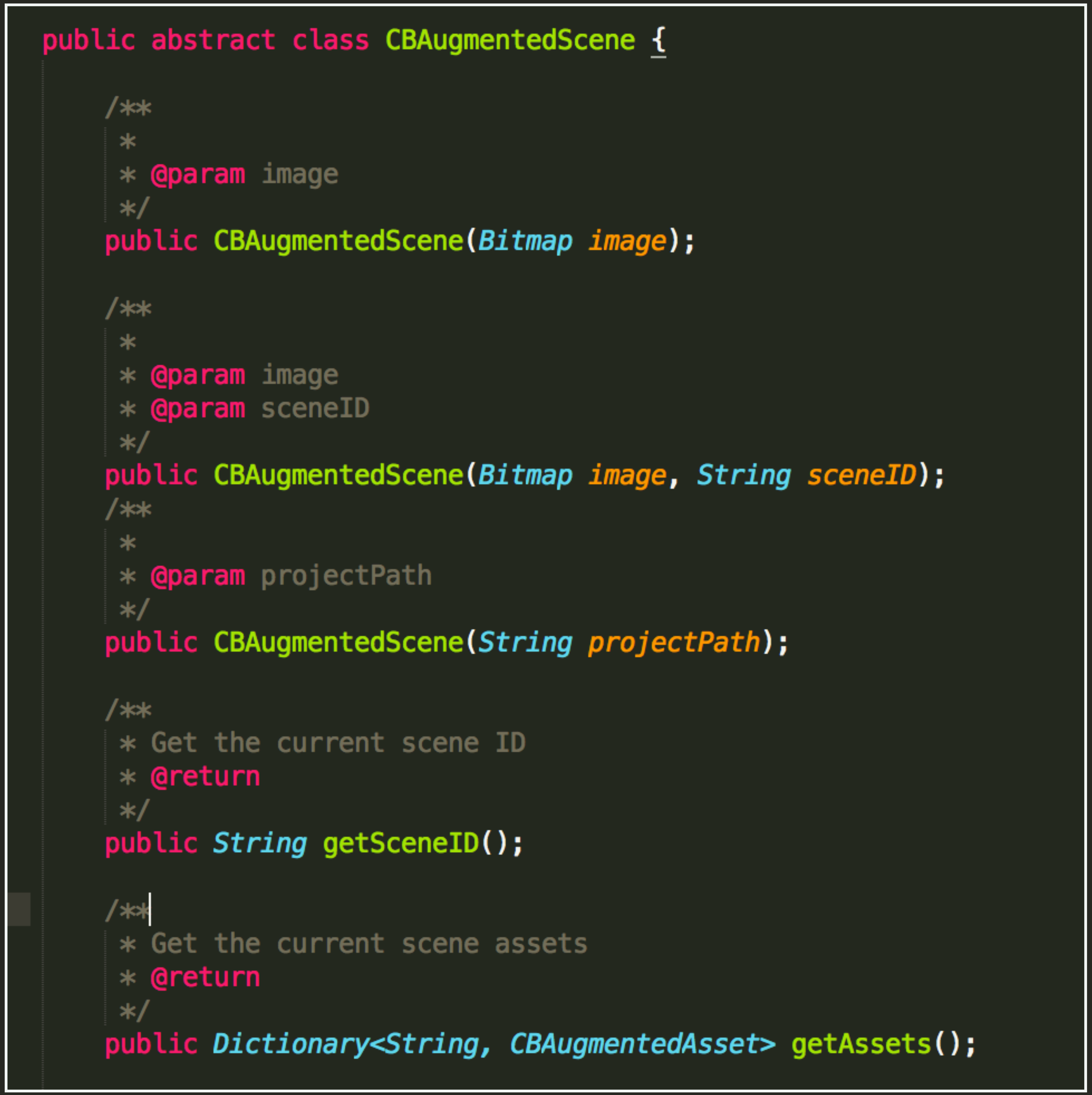
CBRemodelingView Methods



```java
/**
 * Get the current tool mode
 * @return
 */
public CBTypes.CBToolMode getToolMode();

/**
 * Set the current tool mode
 * @param toolMode
 */
public void setToolMode(CBTypes.CBToolMode toolMode);

/**
 * Get whether the view is live or still mode
 * @return
 */
public boolean getIsLive();

/**
 * Set whether the view is live or still mode
 * @param isLive
 */
public void setIsLive(boolean isLive);
private boolean m_isLive;

/**
 * Capture the current state in live mode into still
 */
public void captureCurrentState();
/**
```

- Select tool mode
- Go live, still
- capture current state to still mode

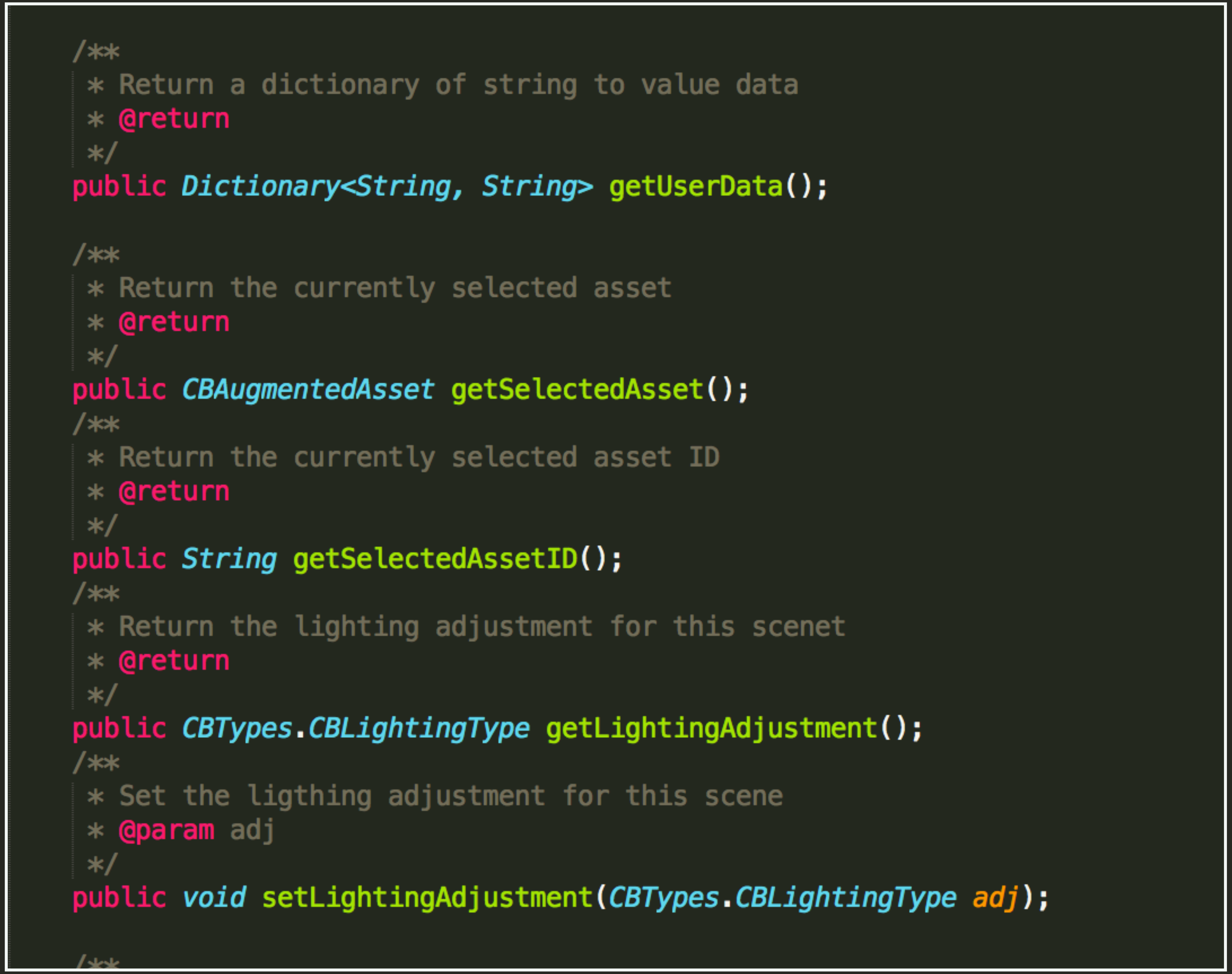
CBRemodelingScene



```
public abstract class CBAugmentedScene {

    /**
     *
     * @param image
     */
    public CBAugmentedScene(Bitmap image);

    /**
     *
     * @param image
     * @param sceneID
     */
    public CBAugmentedScene(Bitmap image, String sceneID);
    /**
     *
     * @param projectPath
     */
    public CBAugmentedScene(String projectPath);

    /**
     * Get the current scene ID
     * @return
     */
    public String getSceneID();

    /**
     * Get the current scene assets
     * @return
     */
    public Dictionary<String, CBAugmentedAsset> getAssets();
```

- Initialize with images or existing scenes
- Obtain paint or flooring assets

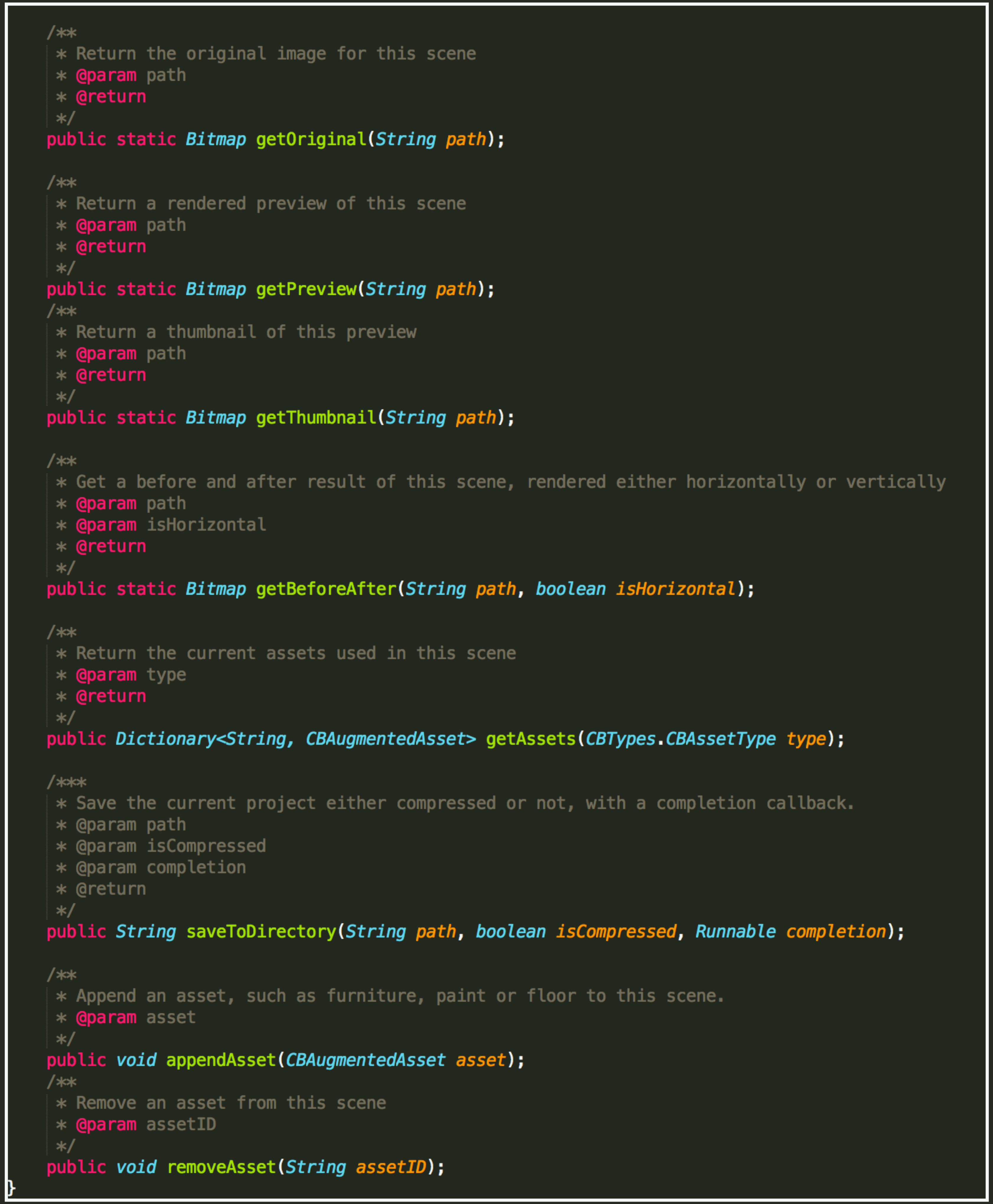
CBRemodelingScene continued



```
/**
 * Return a dictionary of string to value data
 * @return
 */
public Dictionary<String, String> getUserData();

/**
 * Return the currently selected asset
 * @return
 */
public CBAugmentedAsset getSelectedAsset();
/**
 * Return the currently selected asset ID
 * @return
 */
public String getSelectedAssetID();
/**
 * Return the lighting adjustment for this scenet
 * @return
 */
public CBTypes.CBLightingType getLightingAdjustment();
/**
 * Set the ligthing adjustment for this scene
 * @param adj
 */
public void setLightingAdjustment(CBTypes.CBLightingType adj);

/**
```

- Attach information to a scene for later lookup
- Adjust lighting

CBRemodelingScene continued



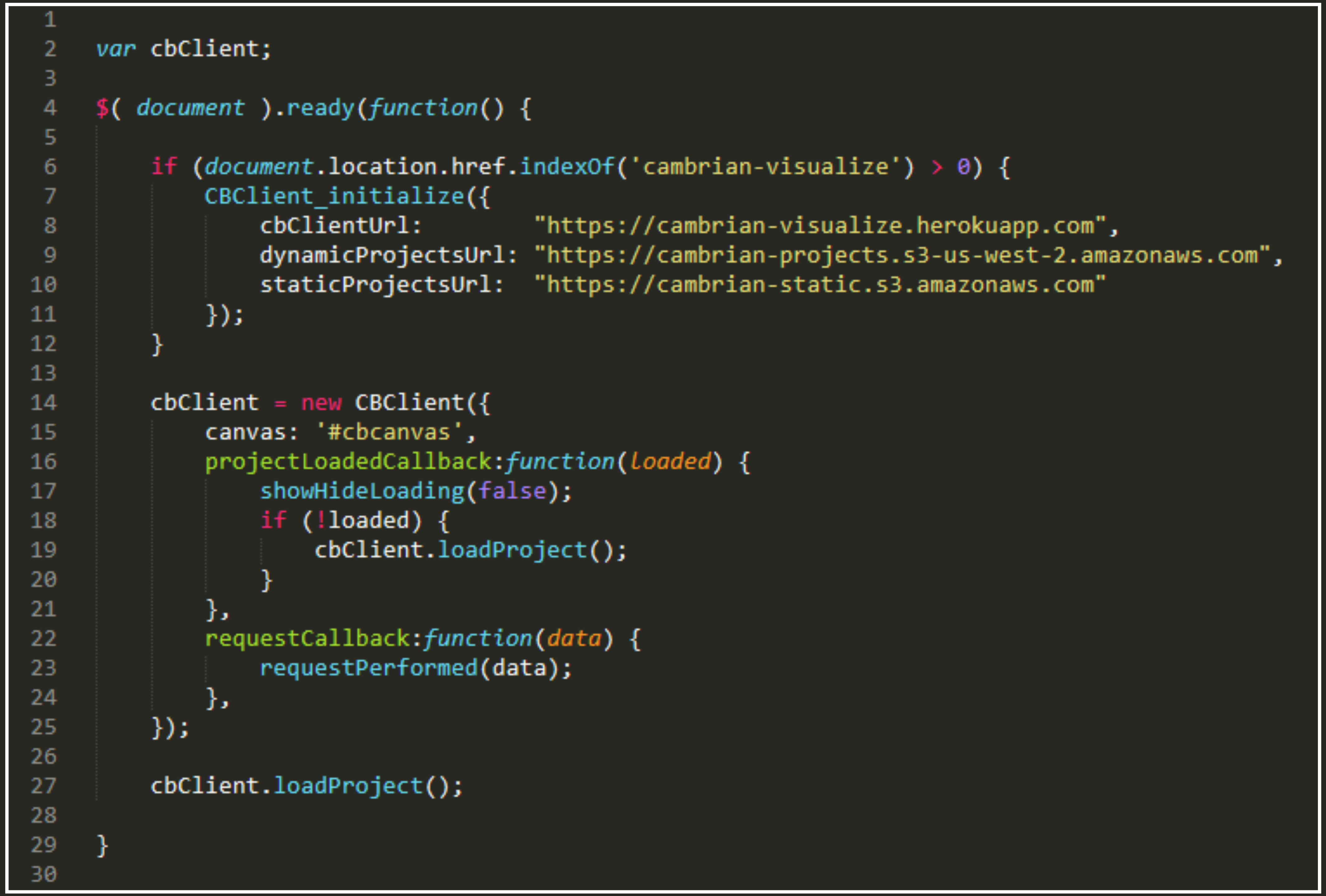
```
    /**
     * Return the original image for this scene
     * @param path
     * @return
     */
    public static Bitmap getOriginal(String path);

    /**
     * Return a rendered preview of this scene
     * @param path
     * @return
     */
    public static Bitmap getPreview(String path);
    /**
     * Return a thumbnail of this preview
     * @param path
     * @return
     */
    public static Bitmap getThumbnail(String path);

    /**
     * Get a before and after result of this scene, rendered either horizontally or vertically
     * @param path
     * @param isHorizontal
     * @return
     */
    public static Bitmap getBeforeAfter(String path, boolean isHorizontal);

    /**
     * Return the current assets used in this scene
     * @param type
     * @return
     */
    public Dictionary<String, CBAugmentedAsset> getAssets(CBTypes.CBAssetType type);

    /***
     * Save the current project either compressed or not, with a completion callback.
     * @param path
     * @param isCompressed
     * @param completion
     * @return
     */
    public String saveToDirectory(String path, boolean isCompressed, Runnable completion);

    /**
     * Append an asset, such as furniture, paint or floor to this scene.
     * @param asset
     */
    public void appendAsset(CBAugmentedAsset asset);
    /**
     * Remove an asset from this scene
     * @param assetID
     */
    public void removeAsset(String assetID);
}
```

- Retrieve Before and after Image information
- Save and load Assets
- Remove Paint Colors

Cambrian Web SDK

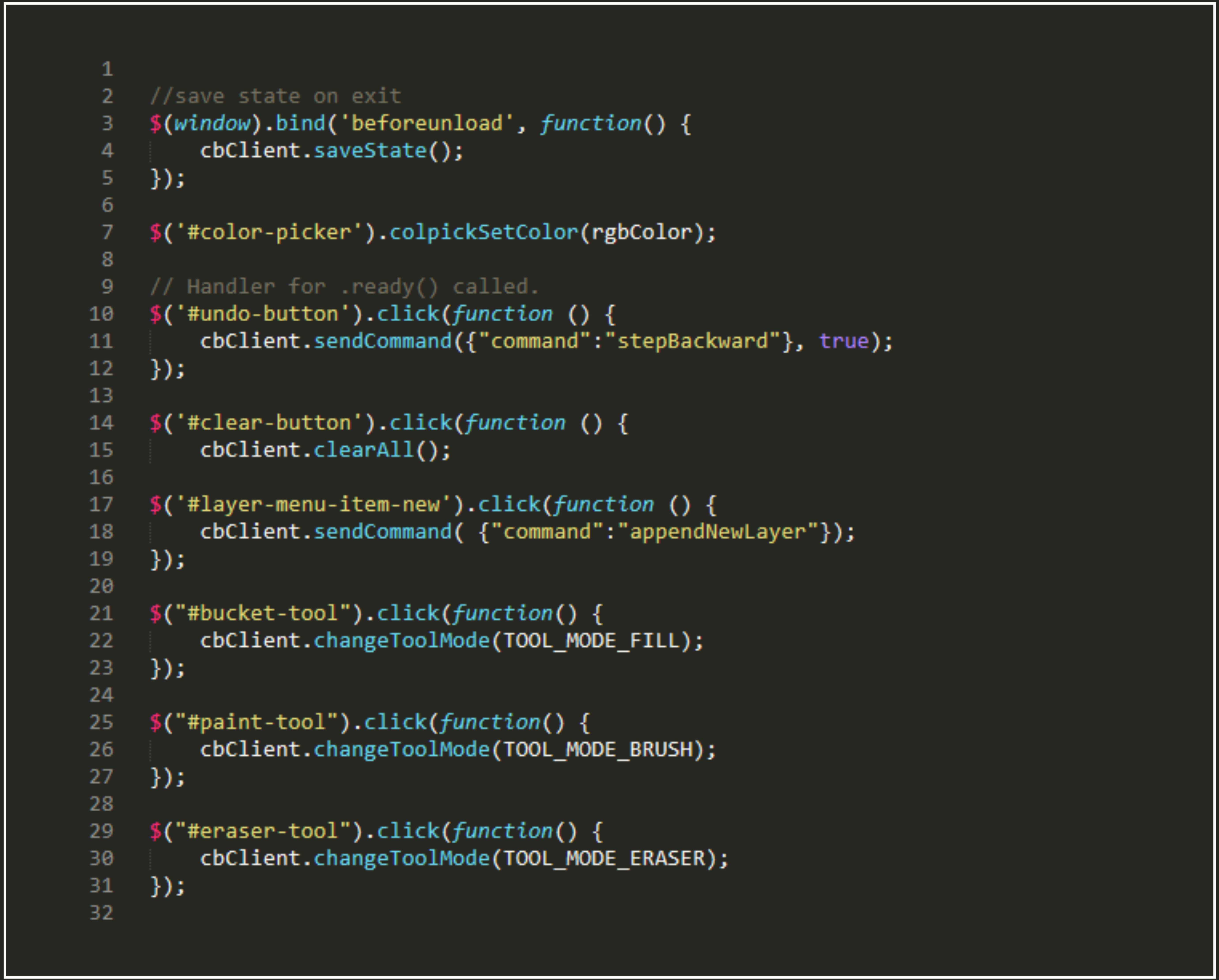
Fully compatible with iOS and android SDK's and utilizes the same rendering engine. Projects generated by mobile devices may be loaded with the web system, and vice-versa. The API requires only a basic javascript interface and will communicate with a scalable binary, such as AWS or Azure, for computer vision.

Web initialization



```

var cbClient;

$( document ).ready(function() {

    if (document.location.href.indexOf('cambrian-visualize') > 0) {
        CBClient_initialize({
            cbClientUrl:        "https://cambrian-visualize.herokuapp.com",
            dynamicProjectsUrl: "https://cambrian-projects.s3-us-west-2.amazonaws.com",
            staticProjectsUrl:  "https://cambrian-static.s3.amazonaws.com"
        });
    }

    cbClient = new CBClient({
        canvas: '#cbcanvas',
        projectLoadedCallback:function(loaded) {
            showHideLoading(false);
            if (!loaded) {
                cbClient.loadProject();
            }
        },
        requestCallback:function(data) {
            requestPerformed(data);
        },
    });

    cbClient.loadProject();

}

```

Web API calls



```javascript

//save state on exit
$(window).bind('beforeunload', function() {
    cbClient.saveState();
});

$('#color-picker').colpickSetColor(rgbColor);

// Handler for .ready() called.
$('#undo-button').click(function () {
    cbClient.sendCommand({"command":"stepBackward"}, true);
});

$('#clear-button').click(function () {
    cbClient.clearAll();

$('#layer-menu-item-new').click(function () {
    cbClient.sendCommand( {"command":"appendNewLayer"});
});

$("#bucket-tool").click(function() {
    cbClient.changeToolMode(TOOL_MODE_FILL);
});

$("#paint-tool").click(function() {
    cbClient.changeToolMode(TOOL_MODE_BRUSH);
});

$("#eraser-tool").click(function() {
    cbClient.changeToolMode(TOOL_MODE_ERASER);
});

```