

Interactive X-Sheet\interactive xsheet working v1-06_01.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Interactive Animation X-Sheet with Audio Waveform</title>
8   <script src="https://cdnjs.cloudflare.com/ajax/libs/jspdf/2.5.1/jspdf.umd.min.js">
9 </script>
10  <script
11    src="https://cdnjs.cloudflare.com/ajax/libs/html2canvas/1.4.1/html2canvas.min.js"></script>
12
13 <style>
14   /* General styling */
15   body {
16     font-family: Arial, sans-serif;
17     font-size: 10pt;
18     line-height: 1.2;
19     margin: 0;
20     padding: 10px;
21   }
22
23   .controls {
24     background-color: #f5f5f5;
25     padding: 10px;
26     margin-bottom: 15px;
27     border-radius: 5px;
28     display: flex;
29     flex-wrap: wrap;
30     gap: 10px;
31     align-items: center;
32   }
33
34   .controls select,
35   .controls input,
36   .controls button {
37     padding: 6px 10px;
38     border: 1px solid #ccc;
39     border-radius: 4px;
40   }
41
42   .controls button {
43     background-color: #4CAF50;
44     color: white;
45     border: none;
46     cursor: pointer;
47     transition: background-color 0.3s;
48   }
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
```

```
47     .controls button:hover {
48         background-color: #45a049;
49     }
50
51     #pdf-button {
52         background-color: #f44336;
53     }
54
55     #pdf-button:hover {
56         background-color: #d32f2f;
57     }
58
59     #print-button {
60         background-color: #2196F3;
61     }
62
63     #print-button:hover {
64         background-color: #0b7dda;
65     }
66
67     #audio-button {
68         background-color: #9c27b0;
69     }
70
71     #audio-button:hover {
72         background-color: #7b1fa2;
73     }
74
75     .header {
76         text-align: center;
77         margin-bottom: 5px;
78     }
79
80     .title {
81         font-size: 14pt;
82         font-weight: bold;
83         margin-bottom: 5px;
84     }
85
86     .metadata {
87         display: grid;
88         grid-template-columns: repeat(3, 1fr);
89         gap: 5px;
90         margin-bottom: 10px;
91     }
92
93     .metadata div {
94         border: 1px solid #000;
95         padding: 3px 5px;
96     }
```

```
97
98     .metadata span {
99         font-weight: bold;
100        margin-right: 5px;
101    }
102
103    .metadata input {
104        border: none;
105        width: 70%;
106        font-family: Arial, sans-serif;
107        font-size: 9pt;
108    }
109
110    table {
111        width: 100%;
112        border-collapse: collapse;
113        table-layout: fixed;
114    }
115
116    th,
117    td {
118        border: 1px solid #000;
119        padding: 2px 4px;
120        vertical-align: top;
121        height: 20px;
122        overflow: hidden;
123    }
124
125    th {
126        background-color: #eee;
127        font-weight: bold;
128        text-align: center;
129        font-size: 9pt;
130    }
131
132    .action-col {
133        width: 16%;
134    }
135
136    .frame-col {
137        width: 4%;
138        text-align: center;
139    }
140
141    .waveform-col {
142        width: 10%;
143        padding: 0;
144        position: relative;
145    }
146
```

```
147     .dialogue-col {
148         width: 10%;
149         text-align: center;
150     }
151
152     .sound-col {
153         width: 9%;
154         text-align: center;
155     }
156
157     .technical-col {
158         width: 9%;
159     }
160
161     .extra1-col {
162         width: 8%;
163     }
164
165     .extra2-col {
166         width: 8%;
167     }
168
169     .camera-col {
170         width: 12%;
171     }
172
173     .waveform-container {
174         position: absolute;
175         width: 100%;
176         top: 0;
177         left: 0;
178         z-index: 10;
179         pointer-events: none;
180     }
181
182     .waveform-overlay {
183         position: absolute;
184         width: 100%;
185         height: 100%;
186         top: 0;
187         left: 0;
188         z-index: 11;
189         pointer-events: auto;
190         cursor: crosshair;
191     }
192
193     .waveform-marker {
194         position: absolute;
195         width: 100%;
196         height: 20px;
```

```
197     background-color: rgba(255, 255, 0, 0.2);  
198     pointer-events: none;  
199     text-align: center;  
200     font-size: 7pt;  
201     line-height: 18px;  
202     color: #666;  
203     z-index: 20;  
204 }  
205  
206 .waveform-canvas {  
207     position: absolute;  
208     top: 0;  
209     left: 0;  
210     width: 100%;  
211     z-index: 15;  
212     pointer-events: none;  
213 }  
214  
215 .waveform-col-container {  
216     position: relative;  
217 }  
218  
219 .phonetic-label {  
220     position: absolute;  
221     background-color: rgba(255, 255, 255, 0.8);  
222     border: 1px solid #ccc;  
223     border-radius: 2px;  
224     font-size: 7pt;  
225     padding: 1px 2px;  
226     z-index: 3;  
227     pointer-events: none;  
228 }  
229  
230 .footer {  
231     font-size: 8pt;  
232     text-align: center;  
233     margin-top: 5px;  
234     color: #333;  
235     font-style: italic;  
236 }  
237  
238 [contenteditable="true"] {  
239     min-height: 18px;  
240     cursor: text;  
241 }  
242  
243 [contenteditable="true"]:focus {  
244     background-color: #f0f7ff;  
245     outline: none;  
246 }
```

```
247
248     [contenteditable="true"]::empty:before {
249         content: attr(data-placeholder);
250         color: #888;
251         font-style: italic;
252     }
253
254     .frame-number {
255         background-color: #eee;
256         font-weight: bold;
257         text-align: center;
258     }
259
260     .modified {
261         background-color: #ffffacd;
262     }
263
264     .selected-cell {
265         background-color: rgba(0, 123, 255, 0.2) !important;
266         outline: 2px solid #0d6efd;
267     }
268
269     .status {
270         margin-top: 10px;
271         padding: 5px;
272         background-color: #f0f0f0;
273         border-radius: 4px;
274         font-style: italic;
275         color: #555;
276     }
277
278     #audio-controls {
279         display: flex;
280         flex-wrap: wrap;
281         gap: 5px;
282         align-items: center;
283         margin-top: 5px;
284         padding: 5px;
285         background-color: #f9f9f9;
286         border-radius: 4px;
287     }
288
289     #audio-controls button {
290         padding: 4px 8px;
291         background-color: #673ab7;
292         color: white;
293         border: none;
294         border-radius: 3px;
295         cursor: pointer;
296     }
```

```
297  
298     #audio-controls button:hover {  
299         background-color: #5e35b1;  
300     }  
301  
302     #audio-info {  
303         font-size: 8pt;  
304         color: #333;  
305     }  
306  
307     #phonetic-input {  
308         position: absolute;  
309         z-index: 100;  
310         background: white;  
311         border: 1px solid #ccc;  
312         padding: 5px;  
313         border-radius: 4px;  
314         box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);  
315         display: none;  
316     }  
317  
318     #audio-upload {  
319         display: none;  
320     }  
321  
322     /* Drawing system specific styles */  
323     .drawing-toolbar button {  
324         transition: background-color 0.2s, color 0.2s;  
325     }  
326  
327     .drawing-toolbar button:hover {  
328         background-color: #e6e6e6 !important;  
329     }  
330  
331     .drawing-toolbar button.active {  
332         background-color: #4CAF50 !important;  
333         color: white !important;  
334     }  
335  
336     .drawing-layer-container {  
337         pointer-events: none;  
338     }  
339  
340     .drawing-layer-container canvas {  
341         pointer-events: none;  
342         touch-action: none;  
343     }  
344  
345     /* Print specific styles */  
346     @media print {
```

```
347     .controls,  
348     button,  
349     #frame-count-container,  
350     .status,  
351     #audio-controls,  
352     #phonetic-input,  
353     .drawing-toolbar {  
354         display: none !important;  
355     }  
356  
357     body {  
358         margin: 0;  
359         padding: 0;  
360     }  
361  
362     @page {  
363         size: auto;  
364         margin: 0.5cm;  
365     }  
366  
367     thead {  
368         display: table-header-group;  
369     }  
370  
371     tfoot {  
372         display: table-footer-group;  
373     }  
374  
375     tr {  
376         page-break-inside: avoid;  
377     }  
378  
379     /* Better waveform printing */  
380     .waveform-col {  
381         position: relative !important;  
382         overflow: hidden !important;  
383     }  
384  
385     .print-waveform-container {  
386         display: block !important;  
387         position: absolute !important;  
388         z-index: 1000 !important;  
389         pointer-events: none !important;  
390         overflow: hidden !important;  
391     }  
392  
393     .waveform-col * {  
394         page-break-inside: avoid !important;  
395     }  
396
```

```
397     /* ensure the waveform clone prints correctly */
398     .print-waveform-clone {
399         display: block !important;
400         position: absolute !important;
401         z-index: 1000 !important;
402         pointer-events: none !important;
403     }
404
405     .cell-waveform-window {
406         position: relative !important;
407         width: 100% !important;
408         height: 100% !important;
409         overflow: hidden !important;
410     }
411
412     /* Hide original waveform during print */
413     body>.waveform-container {
414         display: none !important;
415     }
416
417     .drawing-layer-container {
418         display: block !important;
419         position: absolute !important;
420         z-index: 1000 !important;
421         pointer-events: none !important;
422         page-break-inside: avoid !important;
423     }
424
425     .drawing-layer-container.printing {
426         transform: translate(0, 0) !important; /* Prevent any transforms during print */
427     }
428
429     .drawing-layer-container canvas {
430         display: block !important;
431         position: absolute !important;
432     }
433     }
434     </style>
435 </head>
436
437 <body>
438
439     <script>
440         function updateLayoutSize() {
441             // Preserve any existing functionality in this function
442
443             // Update drawing layer position and size based on table position
444             if (window.xsheetDrawing && window.xsheetDrawing.layerSystem) {
445                 const tableRect = document.getElementById('xsheet-
table').getBoundingClientRect();
```

```
446     const drawingContainer = document.querySelector('.drawing-layer-container');
447
448     if (drawingContainer) {
449         drawingContainer.style.left = tableRect.left + 'px';
450         drawingContainer.style.top = tableRect.top + 'px';
451         drawingContainer.style.width = tableRect.width + 'px';
452         drawingContainer.style.height = tableRect.height + 'px';
453
454         // Trigger redraw of the drawing layer
455         window.xsheetDrawing.layerSystem.resize(tableRect.width,
456         tableRect.height);
457     }
458 }
459
460 // Add event listener for window resize if not already present
461 window.addEventListener('resize', updateLayoutSize);
462 </script>
463 <div class="controls">
464     <div>
465         <label for="template-selector">Template:</label>
466         <select id="template-selector">
467             <option value="large">11"x17" (96 Frames)</option>
468             <option value="small">8"x10" (48 Frames)</option>
469         </select>
470     </div>
471
472     <div id="frame-count-container">
473         <label for="frame-count">Frames:</label>
474         <input type="number" id="frame-count" min="24" step="8" value="96">
475     </div>
476
477     <button id="audio-button">Import Audio</button>
478     <input type="file" id="audio-upload" accept="audio/*">
479
480     <button id="save-button">Save Project</button>
481     <button id="load-button">Load Project</button>
482     <button id="pdf-button">Export PDF</button>
483     <button id="print-button">Print</button>
484     <button id="add-rows-button">Add 8 Rows</button>
485     <button id="clear-button">Clear All</button>
486 </div>
487
488 <script>
489     // Initialize the drawing layer system
490     const xsheetTable = document.getElementById('xsheet-table');
491     window.xsheetDrawing = {
492         layerSystem: new DrawingLayerSystem(xsheetTable)
493     };
494 
```

```
495     // Initialize the drawing toolbar and connect it to the layer system
496     const drawingToolbar = new DrawingToolSystem(window.xsheetDrawing.layerSystem);
497
498     // Append the toolbar to the controls section
499     document.querySelector('.controls').appendChild(drawingToolbar.element);
500
501     // Call updateLayoutSize initially to set correct position
502     updateLayoutSize();
503 </script>
504
505 <div id="audio-controls">
506     <button id="play-audio">Play/Pause</button>
507     <button id="stop-audio">Stop</button>
508     <input type="range" id="audio-scrubber" min="0" max="100" value="0" style="width:
200px;">
509     <span id="audio-info">No audio loaded</span>
510     <button id="add-phonetic">Add Phonetic Marker</button>
511     <div style="margin-left: 10px; color: #666; font-style: italic;">
512         ♦ TIP: Drag down the waveform column while holding the left mouse button to
scrub audio frame-by-frame<br>
513         ⚡ TIP: Click and drag to select multiple cells (use Ctrl+C to copy, Delete to
clear)
514     </div>
515 </div>
516
517 <div id="phonetic-input">
518     <input type="text" id="phonetic-text" placeholder="Enter phonetic sound">
519     <button id="save-phonetic">Save</button>
520     <button id="cancel-phonetic">Cancel</button>
521 </div>
522
523 <div id="printable-area">
524     <div class="header">
525         <div class="title">3D ANIMATION X-SHEET</div>
526     </div>
527
528     <div class="metadata">
529         <div><span>Project #:</span><input type="text" id="project-number"></div>
530         <div><span>DATE:</span><input type="date" id="project-date"></div>
531         <div><span>PAGE #:</span><input type="text" id="page-number"></div>
532         <div><span>ANIMATOR:</span><input type="text" id="animator-name"></div>
533         <div><span>VERSION:</span><input type="text" id="version-number"></div>
534         <div><span>Shot #:</span><input type="text" id="shot-number"></div>
535     </div>
536
537     <table id="xsheet-table">
538         <thead>
539             <tr>
540                 <th class="action-col">Action/Description</th>
541                 <th class="frame-col">Fr</th>
542                 <th class="waveform-col">Audio Waveform</th>
```

```
543     <th class="dialogue-col">Dialogue</th>
544     <th class="sound-col">Sound FX</th>
545     <th class="technical-col">Tech. Notes</th>
546     <th class="extra1-col">Extra 1</th>
547     <th class="extra2-col">Extra 2</th>
548     <th class="frame-col">Fr</th>
549     <th class="camera-col">Camera Moves</th>
550   </tr>
551 </thead>
552 <tbody id="xsheet-body">
553   <!-- Rows will be generated via JavaScript -->
554 </tbody>
555 </table>
556
557 <div class="footer">
558   Bold lines mark 8-frame intervals. Double lines mark 24-frame intervals (24fps).
559   Left columns track character actions, middle columns for technical notes, right
for camera moves.
560 </div>
561 </div>
562
563 <div class="status" id="status-message"></div>
564
565 <script>
566 /**
567 * INTERACTIVE X-SHEET DRAWING TOOLS IMPLEMENTATION
568 *
569 * This code adds comprehensive drawing capabilities to the animation X-Sheet.
570 * Features include:
571 * - Multiple drawing layers
572 * - Various drawing tools (pen, line, arrows, shapes, text, images, animation
symbols)
573 * - Grid-aware annotation that can span multiple frames
574 * - Object selection and manipulation
575 * - Integration with saving, loading, and printing
576 */
577
578 /**
579 * DRAWING LAYER SYSTEM
580 * Manages the canvas layers that contain drawing objects
581 */
582 class DrawingLayerSystem {
583   constructor(xsheetTable) {
584     this.xsheetTable = xsheetTable;
585     this.layers = [];
586     this.activeLayerIndex = 0;
587     this.container = null;
588
589     this.init();
590   }
}
```

```
591
592     init() {
593         // Create container aligned with table
594         const tableRect = this.xsheetTable.getBoundingClientRect();
595         this.container = document.createElement('div');
596         this.container.className = 'drawing-layer-container';
597         this.container.style.position = 'absolute';
598         this.container.style.left = `${tableRect.left}px`;
599         this.container.style.top = `${tableRect.top}px`;
600         this.container.style.width = `${tableRect.width}px`;
601         this.container.style.height = `${tableRect.height}px`;
602         this.container.style.pointerEvents = 'none'; // Initially pass events
603         through
604         this.container.style.zIndex = '5';
605         this.container.style.overflow = 'hidden'; // Prevent drawings from
606         overflowing
607         document.body.appendChild(this.container);
608
609         // Create default background and foreground layers
610         this.addLayer('background');
611         this.addLayer('foreground');
612         this.setActiveLayer(1); // Set foreground as active by default
613
614     }
615
616     addLayer(name) {
617         const canvas = document.createElement('canvas');
618         canvas.className = `drawing-layer-${name}`;
619         canvas.width = this.container.clientWidth;
620         canvas.height = this.container.clientHeight;
621         canvas.style.position = 'absolute';
622         canvas.style.left = '0';
623         canvas.style.top = '0';
624         canvas.style.pointerEvents = 'none';
625
626         this.container.appendChild(canvas);
627
628         const layer = {
629             name: name,
630             canvas: canvas,
631             context: canvas.getContext('2d'),
632             objects: [],
633             visible: true
634         };
635
636         this.layers.push(layer);
637         return this.layers.length - 1; // Return index of new layer
638     }

```

```
639
640     setActiveLayer(index) {
641         if (index >= 0 && index < this.layers.length) {
642             this.activeLayerIndex = index;
643             return true;
644         }
645         return false;
646     }
647
648     getActiveLayer() {
649         return this.layers[this.activeLayerIndex];
650     }
651     updateLayoutSize() {
652         const tableRect = this.xsheetTable.getBoundingClientRect();
653
654         // Store current drawings from each layer
655         const tempCanvases = this.layers.map(layer => {
656             const tempCanvas = document.createElement('canvas');
657             tempCanvas.width = layer.canvas.width;
658             tempCanvas.height = layer.canvas.height;
659             const tempCtx = tempCanvas.getContext('2d');
660             tempCtx.drawImage(layer.canvas, 0, 0);
661             return tempCanvas;
662         });
663
664         // Update container position and dimensions
665         this.container.style.position = 'absolute';
666         this.container.style.left = `${tableRect.left}px`;
667         this.container.style.top = `${tableRect.top}px`;
668         this.container.style.width = `${tableRect.width}px`;
669         this.container.style.height = `${tableRect.height}px`;
670
671         // Update each layer canvas
672         this.layers.forEach((layer, i) => {
673             const scaleX = tableRect.width / layer.canvas.width;
674             const scaleY = tableRect.height / layer.canvas.height;
675
676             layer.canvas.width = tableRect.width;
677             layer.canvas.height = tableRect.height;
678
679             // Redraw with scaling
680             layer.context.save();
681             layer.context.scale(scaleX, scaleY);
682             layer.context.drawImage(tempCanvases[i], 0, 0);
683             layer.context.restore();
684         });
685
686         // Force redraw of all objects
687         this.redrawAll();
688     }
```

```
689         // Dispatch a custom event that the layout was updated
690         document.dispatchEvent(new Event('drawing-layout-updated'));
691     }
692
693     enableDrawing() {
694         this.layers.forEach(layer => {
695             layer.canvas.style.pointerEvents = 'auto';
696         });
697     }
698
699     disableDrawing() {
700         this.layers.forEach(layer => {
701             layer.canvas.style.pointerEvents = 'none';
702         });
703     }
704
705     clearLayer(layerIndex) {
706         if (layerIndex >= 0 && layerIndex < this.layers.length) {
707             const layer = this.layers[layerIndex];
708             layer.context.clearRect(0, 0, layer.canvas.width, layer.canvas.height);
709             layer.objects = [];
710         }
711     }
712
713     clearAllLayers() {
714         this.layers.forEach((layer, index) => {
715             this.clearLayer(index);
716         });
717     }
718
719     // Convert screen coordinates to canvas coordinates
720     screenToCanvas(screenX, screenY) {
721         const containerRect = this.container.getBoundingClientRect();
722         return {
723             x: screenX - containerRect.left,
724             y: screenY - containerRect.top
725         };
726     }
727
728     // Convert frame/column to canvas coordinates (for multi-frame spanning)
729     gridToCanvas(frame, column) {
730         const cell = document.querySelector(`tr.frame-${frame} td:nth-child(${column})`);
731         if (!cell) return null;
732
733         const cellRect = cell.getBoundingClientRect();
734         const containerRect = this.container.getBoundingClientRect();
735
736         return {
737             x: cellRect.left - containerRect.left + cellRect.width / 2,
```

```
738         y: cellRect.top - containerRect.top + cellRect.height / 2
739     );
740 }
741
742 // Add a drawing object to the active layer
743 addObject(object) {
744     const layer = this.getActiveLayer();
745     layer.objects.push(object);
746     this.redrawLayer(this.activeLayerIndex);
747     return object;
748 }
749
750 // Redraw a specific layer
751 redrawLayer(layerIndex) {
752     if (layerIndex >= 0 && layerIndex < this.layers.length) {
753         const layer = this.layers[layerIndex];
754         layer.context.clearRect(0, 0, layer.canvas.width, layer.canvas.height);
755
756         // Draw all objects in this layer
757         layer.objects.forEach(obj => {
758             if (obj.visible) {
759                 obj.draw(layer.context);
760             }
761         });
762     }
763 }
764
765 // Redraw all layers
766 redrawAll() {
767     this.layers.forEach(_, index) => {
768         this.redrawLayer(index);
769     });
770 }
771
772 // Find object under point
773 findObjectAt(x, y) {
774     // Check active layer first, then others in reverse order (top to bottom)
775     const activeLayer = this.getActiveLayer();
776
777     // Check active layer
778     for (let i = activeLayer.objects.length - 1; i >= 0; i--) {
779         const obj = activeLayer.objects[i];
780         if (obj.containsPoint(x, y)) {
781             return { object: obj, layerIndex: this.activeLayerIndex };
782         }
783     }
784
785     // Check other layers from top to bottom
786     for (let l = this.layers.length - 1; l >= 0; l--) {
```

```

787     if (l === this.activeLayerIndex) continue; // Skip active layer (already
checked)

788
789     const layer = this.layers[l];
790     if (!layer.visible) continue;

791
792     for (let i = layer.objects.length - 1; i >= 0; i--) {
793         const obj = layer.objects[i];
794         if (obj.containsPoint(x, y)) {
795             return { object: obj, layerIndex: l };
796         }
797     }
798 }

799
800     return null;
801 }

802
803     // Remove object
804     removeObject(object, layerIndex) {
805         const layer = layerIndex !== undefined ? this.layers[layerIndex] :
this.getActiveLayer();
806         const index = layer.objects.indexOf(object);
807         if (index !== -1) {
808             layer.objects.splice(index, 1);
809             this.redrawLayer(layerIndex !== undefined ? layerIndex :
this.activeLayerIndex);
810             return true;
811         }
812         return false;
813     }
814 }

815
816 /**
817 * DRAWING OBJECT MODEL
818 * Defines the object classes for different types of drawings
819 */
820
821 // Base class for all drawing objects
822 class DrawingObject {
823     constructor(props = {}) {
824         this.x = props.x || 0;
825         this.y = props.y || 0;
826         this.color = props.color || '#000000';
827         this.lineWidth = props.lineWidth || 2;
828         this.visible = props.visible !== undefined ? props.visible : true;
829         this.selected = false;
830         this.type = 'drawingObject'; // Base type
831     }
832
833     draw(context) {
834         // Base drawing functionality

```

```
835     if (this.selected) {
836         this.drawSelectionMarkers(context);
837     }
838 }
839
840 drawSelectionMarkers(context) {
841     // Draw selection handles (default implementation)
842     context.save();
843     context.strokeStyle = '#0099ff';
844     context.lineWidth = 1;
845     context.setLineDash([5, 3]);
846
847     // Default is to draw a box around the object
848     // This should be overridden by subclasses with specific bounds
849     const bounds = this.getBounds();
850     context.strokeRect(
851         bounds.x - 2,
852         bounds.y - 2,
853         bounds.width + 4,
854         bounds.height + 4
855     );
856
857     context.restore();
858 }
859
860 getBounds() {
861     // Default implementation - should be overridden
862     return { x: this.x, y: this.y, width: 0, height: 0 };
863 }
864
865 containsPoint(x, y) {
866     // Default implementation - should be overridden
867     return false;
868 }
869
870 move(dx, dy) {
871     this.x += dx;
872     this.y += dy;
873 }
874
875 toJSON() {
876     return {
877         type: this.type,
878         x: this.x,
879         y: this.y,
880         color: this.color,
881         lineWidth: this.lineWidth,
882         visible: this.visible
883     };
884 }
```

```
885
886     static fromJSON(data) {
887         // Factory method to create objects from JSON
888         // This will be overridden by subclasses
889         return new DrawingObject(data);
890     }
891 }
892
893 // Line object
894 class LineObject extends DrawingObject {
895     constructor(props = {}) {
896         super(props);
897         this.x2 = props.x2 || 0;
898         this.y2 = props.y2 || 0;
899         this.type = 'line';
900         this.dashPattern = props.dashPattern || [];
901     }
902
903     draw(context) {
904         context.save();
905         context.beginPath();
906         context.strokeStyle = this.color;
907         context.lineWidth = this.lineWidth;
908
909         if (this.dashPattern.length > 0) {
910             context.setLineDash(this.dashPattern);
911         }
912
913         context.moveTo(this.x, this.y);
914         context.lineTo(this.x2, this.y2);
915         context.stroke();
916         context.restore();
917
918         super.draw(context);
919     }
920
921     getBounds() {
922         const minX = Math.min(this.x, this.x2);
923         const minY = Math.min(this.y, this.y2);
924         const width = Math.abs(this.x2 - this.x);
925         const height = Math.abs(this.y2 - this.y);
926
927         return { x: minX, y: minY, width, height };
928     }
929
930     containsPoint(x, y) {
931         // Check if point is near the line
932         const lineLength = Math.sqrt(
933             Math.pow(this.x2 - this.x, 2) + Math.pow(this.y2 - this.y, 2)
934         );

```

```
935
936    // If line is too short, use a minimum distance
937    if (lineLength < 1) {
938        const dx = x - this.x;
939        const dy = y - this.y;
940        return Math.sqrt(dx * dx + dy * dy) <= 5;
941    }
942
943    // Calculate distance from point to line segment
944    const t = ((x - this.x) * (this.x2 - this.x) + (y - this.y) * (this.y2 -
945    this.y)) / (lineLength * lineLength);
946
947    if (t < 0) {
948        // Point is beyond start point
949        const dx = x - this.x;
950        const dy = y - this.y;
951        return Math.sqrt(dx * dx + dy * dy) <= 5;
952    }
953
954    if (t > 1) {
955        // Point is beyond end point
956        const dx = x - this.x2;
957        const dy = y - this.y2;
958        return Math.sqrt(dx * dx + dy * dy) <= 5;
959    }
960
961    // Calculate perpendicular distance
962    const px = this.x + t * (this.x2 - this.x);
963    const py = this.y + t * (this.y2 - this.y);
964    const dx = x - px;
965    const dy = y - py;
966    return Math.sqrt(dx * dx + dy * dy) <= 5;
967
968    move(dx, dy) {
969        super.move(dx, dy);
970        this.x2 += dx;
971        this.y2 += dy;
972    }
973
974    toJSON() {
975        const json = super.toJSON();
976        return {
977            ...json,
978            x2: this.x2,
979            y2: this.y2,
980            dashPattern: this.dashPattern
981        };
982    }
983}
```

```
984     static fromJSON(data) {
985         return new LineObject(data);
986     }
987 }
988
989 // Arrow object (extends Line)
990 class ArrowObject extends LineObject {
991     constructor(props = {}) {
992         super(props);
993         this.arrowSize = props.arrowSize || 10;
994         this.type = 'arrow';
995     }
996
997     draw(context) {
998         // Draw the line part
999         super.draw(context);
1000
1001         // Draw the arrowhead
1002         const angle = Math.atan2(this.y2 - this.y, this.x2 - this.x);
1003         context.save();
1004         context.fillStyle = this.color;
1005         context.beginPath();
1006         context.moveTo(this.x2, this.y2);
1007         context.lineTo(
1008             this.x2 - this.arrowSize * Math.cos(angle - Math.PI/6),
1009             this.y2 - this.arrowSize * Math.sin(angle - Math.PI/6)
1010         );
1011         context.lineTo(
1012             this.x2 - this.arrowSize * Math.cos(angle + Math.PI/6),
1013             this.y2 - this.arrowSize * Math.sin(angle + Math.PI/6)
1014         );
1015         context.closePath();
1016         context.fill();
1017         context.restore();
1018     }
1019
1020     toJSON() {
1021         const json = super.toJSON();
1022         return {
1023             ...json,
1024             arrowSize: this.arrowSize
1025         };
1026     }
1027
1028     static fromJSON(data) {
1029         return new ArrowObject(data);
1030     }
1031 }
1032
1033 // Rectangle object
```

```
1034     class RectangleObject extends DrawingObject {
1035         constructor(props = {}) {
1036             super(props);
1037             this.width = props.width || 0;
1038             this.height = props.height || 0;
1039             this.fill = props.fill || false;
1040             this.fillColor = props.fillColor || this.color;
1041             this.type = 'rectangle';
1042         }
1043
1044         draw(context) {
1045             context.save();
1046             context.strokeStyle = this.color;
1047             context.lineWidth = this.lineWidth;
1048
1049             // Draw rectangle
1050             if (this.fill) {
1051                 context.fillStyle = this.fillColor;
1052                 context.fillRect(this.x, this.y, this.width, this.height);
1053             }
1054             context.strokeRect(this.x, this.y, this.width, this.height);
1055             context.restore();
1056
1057             super.draw(context);
1058         }
1059
1060         getBounds() {
1061             return {
1062                 x: this.x,
1063                 y: this.y,
1064                 width: this.width,
1065                 height: this.height
1066             };
1067         }
1068
1069         containsPoint(x, y) {
1070             // Check if point is inside or near the edge of the rectangle
1071             if (this.fill) {
1072                 // For filled rectangles, check if point is inside
1073                 return (
1074                     x >= this.x && x <= this.x + this.width &&
1075                     y >= this.y && y <= this.y + this.height
1076                 );
1077             } else {
1078                 // For unfilled rectangles, check if point is near the edges
1079                 const nearLeft = Math.abs(x - this.x) <= 5;
1080                 const nearRight = Math.abs(x - (this.x + this.width)) <= 5;
1081                 const nearTop = Math.abs(y - this.y) <= 5;
1082                 const nearBottom = Math.abs(y - (this.y + this.height)) <= 5;
1083             }
1084         }
1085     }
1086 
```

```
1084         return (
1085             (nearLeft || nearRight) && (y >= this.y && y <= this.y +
1086                 this.height) ||
1087                 (nearTop || nearBottom) && (x >= this.x && x <= this.x + this.width)
1088             );
1089         }
1090     }
1091
1092     toJSON() {
1093         const json = super.toJSON();
1094         return {
1095             ...json,
1096             width: this.width,
1097             height: this.height,
1098             fill: this.fill,
1099             fillColor: this.fillColor
1100         };
1101     }
1102
1103     static fromJSON(data) {
1104         return new RectangleObject(data);
1105     }
1106
1107 // Circle/Ellipse object
1108 class EllipseObject extends DrawingObject {
1109     constructor(props = {}) {
1110         super(props);
1111         this.radiusX = props.radiusX || 0;
1112         this.radiusY = props.radiusY || 0;
1113         this.fill = props.fill || false;
1114         this.fillColor = props.fillColor || this.color;
1115         this.type = 'ellipse';
1116     }
1117
1118     draw(context) {
1119         context.save();
1120         context.beginPath();
1121         context.strokeStyle = this.color;
1122         context.lineWidth = this.lineWidth;
1123
1124         // Draw ellipse
1125         context.ellipse(
1126             this.x,
1127             this.y,
1128             this.radiusX,
1129             this.radiusY,
1130             0,
1131             0,
1132             2 * Math.PI
```

```
1133     );
1134
1135     if (this.fill) {
1136         context.fillStyle = this.fillColor;
1137         context.fill();
1138     }
1139     context.stroke();
1140     context.restore();
1141
1142     super.draw(context);
1143 }
1144
1145 getBounds() {
1146     return {
1147         x: this.x - this.radiusX,
1148         y: this.y - this.radiusY,
1149         width: this.radiusX * 2,
1150         height: this.radiusY * 2
1151     };
1152 }
1153
1154 containsPoint(x, y) {
1155     // Check if point is inside or near the edge of the ellipse
1156     const normalizedX = (x - this.x) / this.radiusX;
1157     const normalizedY = (y - this.y) / this.radiusY;
1158     const distance = Math.sqrt(normalizedX * normalizedX + normalizedY *
normalizedY);
1159
1160     if (this.fill) {
1161         // For filled ellipses, check if point is inside
1162         return distance <= 1.0;
1163     } else {
1164         // For unfilled ellipses, check if point is near the edge
1165         return Math.abs(distance - 1.0) <= 5 / this.radiusX;
1166     }
1167 }
1168
1169 toJSON() {
1170     const json = super.toJSON();
1171     return {
1172         ...json,
1173         radiusX: this.radiusX,
1174         radiusY: this.radiusY,
1175         fill: this.fill,
1176         fillColor: this.fillColor
1177     };
1178 }
1179
1180 static fromJSON(data) {
1181     return new EllipseObject(data);
```

```
1182     }
1183 }
1184
1185 // Text object
1186 class TextObject extends DrawingObject {
1187     constructor(props = {}) {
1188         super(props);
1189         this.text = props.text || '';
1190         this.fontSize = props.fontSize || 14;
1191         this.fontFamily = props.fontFamily || 'Arial, sans-serif';
1192         this.align = props.align || 'left';
1193         this.type = 'text';
1194     }
1195
1196     draw(context) {
1197         context.save();
1198         context.fillStyle = this.color;
1199         context.font = `${this.fontSize}px ${this.fontFamily}`;
1200         context.textAlign = this.align;
1201
1202         // Draw text
1203         context.fillText(this.text, this.x, this.y);
1204         context.restore();
1205
1206         super.draw(context);
1207     }
1208
1209     getBounds() {
1210         // Estimate text dimensions
1211         const dummyCanvas = document.createElement('canvas');
1212         const ctx = dummyCanvas.getContext('2d');
1213         ctx.font = `${this.fontSize}px ${this.fontFamily}`;
1214         const metrics = ctx.measureText(this.text);
1215         const height = this.fontSize; // Approximation
1216
1217         return {
1218             x: this.align === 'center' ? this.x - metrics.width / 2 :
1219                 this.align === 'right' ? this.x - metrics.width : this.x,
1220             y: this.y - height,
1221             width: metrics.width,
1222             height: height
1223         };
1224     }
1225
1226     containsPoint(x, y) {
1227         const bounds = this.getBounds();
1228         return (
1229             x >= bounds.x && x <= bounds.x + bounds.width &&
1230             y >= bounds.y && y <= bounds.y + bounds.height
1231         );
1232     }
1233 }
```

```
1232 }
1233
1234     toJSON() {
1235         const json = super.toJSON();
1236         return {
1237             ...json,
1238             text: this.text,
1239             fontSize: this.fontSize,
1240             fontFamily: this.fontFamily,
1241             align: this.align
1242         };
1243     }
1244
1245     static fromJSON(data) {
1246         return new TextObject(data);
1247     }
1248 }
1249
1250 // Image object
1251 class ImageObject extends DrawingObject {
1252     constructor(props = {}) {
1253         super(props);
1254         this.width = props.width || 0;
1255         this.height = props.height || 0;
1256         this.imageUrl = props.imageUrl || '';
1257         this.image = null;
1258         this.loaded = false;
1259         this.type = 'image';
1260
1261         // Load the image if provided
1262         if (this.imageUrl) {
1263             this.loadImage(this.imageUrl);
1264         }
1265     }
1266
1267     loadImage(url) {
1268         this.image = new Image();
1269         this.image.onload = () => {
1270             this.loaded = true;
1271
1272             // If dimensions not specified, use image dimensions
1273             if (this.width === 0 || this.height === 0) {
1274                 this.width = this.image.width;
1275                 this.height = this.image.height;
1276             }
1277
1278             // Trigger redraw
1279             document.dispatchEvent(new Event('xsheet-redraw'));
1280         };
1281         this.image.src = url;
```

```
1282 }
1283
1284     draw(context) {
1285         if (this.loaded && this.image) {
1286             context.save();
1287             context.drawImage(this.image, this.x, this.y, this.width, this.height);
1288             context.restore();
1289         } else if (!this.loaded) {
1290             // Draw placeholder while loading
1291             context.save();
1292             context.strokeStyle = '#999999';
1293             context.lineWidth = 1;
1294             context.strokeRect(this.x, this.y, this.width, this.height);
1295             context.font = '10px Arial';
1296             context.fillStyle = '#999999';
1297             context.fillText('Loading Image...', this.x + 5, this.y + 15);
1298             context.restore();
1299         }
1300
1301         super.draw(context);
1302     }
1303
1304     getBounds() {
1305         return {
1306             x: this.x,
1307             y: this.y,
1308             width: this.width,
1309             height: this.height
1310         };
1311     }
1312
1313     containsPoint(x, y) {
1314         return (
1315             x >= this.x && x <= this.x + this.width &&
1316             y >= this.y && y <= this.y + this.height
1317         );
1318     }
1319
1320     toJSON() {
1321         const json = super.toJSON();
1322         return {
1323             ...json,
1324             width: this.width,
1325             height: this.height,
1326             imageUrl: this.imageUrl
1327         };
1328     }
1329
1330     static fromJSON(data) {
1331         return new ImageObject(data);
```

```
1332     }
1333 }
1334
1335 // Symbol object (predefined animation symbols)
1336 class SymbolObject extends DrawingObject {
1337     constructor(props = {}) {
1338         super(props);
1339         this.symbolType = props.symbolType || 'default';
1340         this.scale = props.scale || 1.0;
1341         this.type = 'symbol';
1342     }
1343
1344     draw(context) {
1345         context.save();
1346         context.strokeStyle = this.color;
1347         context.fillStyle = this.color;
1348         context.lineWidth = this.lineWidth;
1349
1350         // Draw based on symbol type
1351         switch (this.symbolType) {
1352             case 'anticipation':
1353                 this.drawAnticipation(context);
1354                 break;
1355             case 'impact':
1356                 this.drawImpact(context);
1357                 break;
1358             case 'keyframe':
1359                 this.drawKeyframe(context);
1360                 break;
1361             case 'inbetween':
1362                 this.drawInbetween(context);
1363                 break;
1364             case 'hold':
1365                 this.drawHold(context);
1366                 break;
1367             default:
1368                 this.drawDefault(context);
1369             }
1370
1371         context.restore();
1372         super.draw(context);
1373     }
1374
1375     drawAnticipation(context) {
1376         context.save();
1377         context.translate(this.x, this.y);
1378         context.scale(this.scale, this.scale);
1379
1380         // Draw curved arrow going back
1381         context.beginPath();
```

```
1382     context.moveTo(0, 0);
1383     context.bezierCurveTo(-20, -5, -25, 10, -10, 15);
1384     context.stroke();
1385
1386     // Draw arrowhead
1387     context.beginPath();
1388     context.moveTo(-10, 15);
1389     context.lineTo(-5, 10);
1390     context.lineTo(-15, 5);
1391     context.closePath();
1392     context.fill();
1393
1394     context.restore();
1395 }
1396
1397 drawImpact(context) {
1398     context.save();
1399     context.translate(this.x, this.y);
1400     context.scale(this.scale, this.scale);
1401
1402     // Draw impact star
1403     for (let i = 0; i < 8; i++) {
1404         const angle = (i / 8) * Math.PI * 2;
1405         const innerRadius = 5;
1406         const outerRadius = 15;
1407
1408         context.beginPath();
1409         context.moveTo(
1410             innerRadius * Math.cos(angle),
1411             innerRadius * Math.sin(angle)
1412         );
1413         context.lineTo(
1414             outerRadius * Math.cos(angle),
1415             outerRadius * Math.sin(angle)
1416         );
1417         context.stroke();
1418     }
1419
1420     context.restore();
1421 }
1422
1423 drawKeyframe(context) {
1424     context.save();
1425     context.translate(this.x, this.y);
1426     context.scale(this.scale, this.scale);
1427
1428     // Draw diamond
1429     context.beginPath();
1430     context.moveTo(0, -10);
1431     context.lineTo(10, 0);
```

```
1432         context.lineTo(0, 10);
1433         context.lineTo(-10, 0);
1434         context.closePath();
1435         context.stroke();
1436         context.fill();
1437
1438         context.restore();
1439     }
1440
1441     drawInbetween(context) {
1442         context.save();
1443         context.translate(this.x, this.y);
1444         context.scale(this.scale, this.scale);
1445
1446         // Draw circle
1447         context.beginPath();
1448         context.arc(0, 0, 7, 0, Math.PI * 2);
1449         context.stroke();
1450
1451         context.restore();
1452     }
1453
1454     drawHold(context) {
1455         context.save();
1456         context.translate(this.x, this.y);
1457         context.scale(this.scale, this.scale);
1458
1459         // Draw horizontal bar
1460         context.beginPath();
1461         context.moveTo(-15, 0);
1462         context.lineTo(15, 0);
1463         context.lineWidth = this.lineWidth * 2;
1464         context.stroke();
1465
1466         context.restore();
1467     }
1468
1469     drawDefault(context) {
1470         context.save();
1471         context.translate(this.x, this.y);
1472         context.scale(this.scale, this.scale);
1473
1474         // Draw square
1475         context.beginPath();
1476         context.rect(-7, -7, 14, 14);
1477         context.stroke();
1478
1479         context.restore();
1480     }
1481 }
```

```
1482     getBounds() {
1483         // Approximate bounds based on symbol type
1484         const size = 20 * this.scale;
1485         return {
1486             x: this.x - size / 2,
1487             y: this.y - size / 2,
1488             width: size,
1489             height: size
1490         };
1491     }
1492
1493     containsPoint(x, y) {
1494         const bounds = this.getBounds();
1495         const dx = x - this.x;
1496         const dy = y - this.y;
1497         const distance = Math.sqrt(dx * dx + dy * dy);
1498
1499         // Use a radius-based check as most symbols are roughly circular
1500         return distance <= bounds.width / 2;
1501     }
1502
1503     toJSON() {
1504         const json = super.toJSON();
1505         return {
1506             ...json,
1507             symbolType: this.symbolType,
1508             scale: this.scale
1509         };
1510     }
1511
1512     static fromJSON(data) {
1513         return new SymbolObject(data);
1514     }
1515 }
1516
1517 // Free-form path (for pen/brush tools)
1518 class PathObject extends DrawingObject {
1519     constructor(props = {}) {
1520         super(props);
1521         this.points = props.points || [];
1522         this.smoothing = props.smoothing !== undefined ? props.smoothing : true;
1523         this.closed = props.closed || false;
1524         this.fill = props.fill || false;
1525         this.fillColor = props.fillColor || this.color;
1526         this.type = 'path';
1527     }
1528
1529     addPoint(x, y) {
1530         this.points.push({ x, y });
1531     }
```

```
1532
1533     draw(context) {
1534         if (this.points.length < 2) return;
1535
1536         context.save();
1537         context.beginPath();
1538         context.strokeStyle = this.color;
1539         context.lineWidth = this.lineWidth;
1540         context.lineJoin = 'round';
1541         context.lineCap = 'round';
1542
1543         // Start from first point
1544         context.moveTo(this.points[0].x, this.points[0].y);
1545
1546         if (this.smoothing && this.points.length > 2) {
1547             // Draw using bezier curves for smoothing
1548             for (let i = 1; i < this.points.length - 1; i++) {
1549                 const p1 = this.points[i];
1550                 const p2 = this.points[i + 1];
1551
1552                 const xc = (p1.x + p2.x) / 2;
1553                 const yc = (p1.y + p2.y) / 2;
1554
1555                 context.quadraticCurveTo(p1.x, p1.y, xc, yc);
1556             }
1557
1558             // Connect to the last point
1559             const last = this.points[this.points.length - 1];
1560             context.lineTo(last.x, last.y);
1561         } else {
1562             // Simple line segments
1563             for (let i = 1; i < this.points.length; i++) {
1564                 context.lineTo(this.points[i].x, this.points[i].y);
1565             }
1566         }
1567
1568         if (this.closed) {
1569             context.closePath();
1570         }
1571
1572         if (this.fill) {
1573             context.fillStyle = this.fillColor;
1574             context.fill();
1575         }
1576
1577         context.stroke();
1578         context.restore();
1579
1580         super.draw(context);
1581     }
```

```
1582
1583     getBounds() {
1584         if (this.points.length === 0) {
1585             return { x: this.x, y: this.y, width: 0, height: 0 };
1586         }
1587
1588         let minX = this.points[0].x;
1589         let maxX = this.points[0].x;
1590         let minY = this.points[0].y;
1591         let maxY = this.points[0].y;
1592
1593         // Find min/max coordinates
1594         for (let i = 1; i < this.points.length; i++) {
1595             const point = this.points[i];
1596             minX = Math.min(minX, point.x);
1597             maxX = Math.max(maxX, point.x);
1598             minY = Math.min(minY, point.y);
1599             maxY = Math.max(maxY, point.y);
1600         }
1601
1602         return {
1603             x: minX,
1604             y: minY,
1605             width: maxX - minX,
1606             height: maxY - minY
1607         };
1608     }
1609
1610     containsPoint(x, y) {
1611         if (this.points.length < 2) return false;
1612
1613         // Check if point is near any line segment
1614         for (let i = 0; i < this.points.length - 1; i++) {
1615             const p1 = this.points[i];
1616             const p2 = this.points[i + 1];
1617
1618             const lineLength = Math.sqrt(
1619                 Math.pow(p2.x - p1.x, 2) + Math.pow(p2.y - p1.y, 2)
1620             );
1621
1622             // If line is too short, check distance to point
1623             if (lineLength < 1) {
1624                 const dx = x - p1.x;
1625                 const dy = y - p1.y;
1626                 if (Math.sqrt(dx * dx + dy * dy) <= 5) {
1627                     return true;
1628                 }
1629                 continue;
1630             }
1631         }
1632     }
1633 }
```

```
1632 // Calculate distance from point to line segment
1633     const t = ((x - p1.x) * (p2.x - p1.x) + (y - p1.y) * (p2.y - p1.y)) /
1634     (lineLength * lineLength);
1635
1636     if (t < 0) {
1637         // Point is beyond start point
1638         const dx = x - p1.x;
1639         const dy = y - p1.y;
1640         if (Math.sqrt(dx * dx + dy * dy) <= 5) {
1641             return true;
1642         }
1643     } else if (t > 1) {
1644         // Point is beyond end point
1645         const dx = x - p2.x;
1646         const dy = y - p2.y;
1647         if (Math.sqrt(dx * dx + dy * dy) <= 5) {
1648             return true;
1649         }
1650     } else {
1651         // Calculate perpendicular distance
1652         const px = p1.x + t * (p2.x - p1.x);
1653         const py = p1.y + t * (p2.y - p1.y);
1654         const dx = x - px;
1655         const dy = y - py;
1656         if (Math.sqrt(dx * dx + dy * dy) <= 5) {
1657             return true;
1658         }
1659     }
1660
1661 // If closed and filled, also check if point is inside
1662 if (this.closed && this.fill) {
1663     // Use point-in-polygon algorithm
1664     let inside = false;
1665     for (let i = 0, j = this.points.length - 1; i < this.points.length; j =
1666     i++) {
1667         const xi = this.points[i].x;
1668         const yi = this.points[i].y;
1669         const xj = this.points[j].x;
1670         const yj = this.points[j].y;
1671
1672         const intersect = ((yi > y) !== (yj > y)) &&
1673             (x < (xj - xi) * (y - yi) / (yj - yi) + xi);
1674
1675         if (intersect) inside = !inside;
1676     }
1677     return inside;
1678 }
1679 return false;
```

```
1680    }
1681
1682    move(dx, dy) {
1683        super.move(dx, dy);
1684
1685        // Move all points
1686        this.points.forEach(point => {
1687            point.x += dx;
1688            point.y += dy;
1689        });
1690    }
1691
1692    toJSON() {
1693        const json = super.toJSON();
1694        return {
1695            ...json,
1696            points: this.points,
1697            smoothing: this.smoothing,
1698            closed: this.closed,
1699            fill: this.fill,
1700            fillColor: this.fillColor
1701        };
1702    }
1703
1704    static fromJSON(data) {
1705        return new PathObject(data);
1706    }
1707 }
1708
1709 // Frame-Spanning Line (connects specific cells in the grid)
1710 class FrameSpanningLineObject extends DrawingObject {
1711     constructor(props = {}) {
1712         super(props);
1713         this.startFrame = props.startFrame || 1;
1714         this.startColumn = props.startColumn || 1;
1715         this.endFrame = props.endFrame || 1;
1716         this.endColumn = props.endColumn || 1;
1717         this.type = 'frameSpanningLine';
1718         this.dashPattern = props.dashPattern || [];
1719         this.arrowStart = props.arrowStart || false;
1720         this.arrowEnd = props.arrowEnd || false;
1721         this.arrowSize = props.arrowSize || 10;
1722     }
1723
1724     draw(context) {
1725         // Calculate actual coordinates from frame and column
1726         const layerSystem = window.xsheetDrawing.layerSystem;
1727
1728         const startPos = layerSystem.gridToCanvas(this.startFrame,
this.startColumn);
```

```
1729     const endPos = layerSystem.gridToCanvas(this.endFrame, this.endColumn);
1730
1731     if (!startPos || !endPos) return; // Skip if cells not found
1732
1733     context.save();
1734     context.beginPath();
1735     context.strokeStyle = this.color;
1736     context.lineWidth = this.lineWidth;
1737
1738     if (this.dashPattern.length > 0) {
1739         context.setLineDash(this.dashPattern);
1740     }
1741
1742     context.moveTo(startPos.x, startPos.y);
1743     context.lineTo(endPos.x, endPos.y);
1744     context.stroke();
1745
1746     // Draw arrows if needed
1747     if (this.arrowStart) {
1748         this.drawArrow(context, endPos.x, endPos.y, startPos.x, startPos.y);
1749     }
1750
1751     if (this.arrowEnd) {
1752         this.drawArrow(context, startPos.x, startPos.y, endPos.x, endPos.y);
1753     }
1754
1755     context.restore();
1756
1757     // Store computed coordinates for selection	hit testing
1758     this.computedStart = startPos;
1759     this.computedEnd = endPos;
1760
1761     super.draw(context);
1762 }
1763
1764 drawArrow(context, fromX, fromY, toX, toY) {
1765     const angle = Math.atan2(toY - fromY, toX - fromX);
1766
1767     context.save();
1768     context.fillStyle = this.color;
1769     context.beginPath();
1770     context.moveTo(toX, toY);
1771     context.lineTo(
1772         toX - this.arrowSize * Math.cos(angle - Math.PI/6),
1773         toY - this.arrowSize * Math.sin(angle - Math.PI/6)
1774     );
1775     context.lineTo(
1776         toX - this.arrowSize * Math.cos(angle + Math.PI/6),
1777         toY - this.arrowSize * Math.sin(angle + Math.PI/6)
1778     );
}
```

```
1779         context.closePath();
1780         context.fill();
1781         context.restore();
1782     }
1783
1784     getBounds() {
1785         if (!this.computedStart || !this.computedEnd) return { x: 0, y: 0, width: 0,
height: 0 };
1786
1787         const minX = Math.min(this.computedStart.x, this.computedEnd.x);
1788         const minY = Math.min(this.computedStart.y, this.computedEnd.y);
1789         const width = Math.abs(this.computedEnd.x - this.computedStart.x);
1790         const height = Math.abs(this.computedEnd.y - this.computedStart.y);
1791
1792         return { x: minX, y: minY, width, height };
1793     }
1794
1795     containsPoint(x, y) {
1796         if (!this.computedStart || !this.computedEnd) return false;
1797
1798         // Same algorithm as LineObject
1799         const lineLength = Math.sqrt(
1800             Math.pow(this.computedEnd.x - this.computedStart.x, 2) +
1801             Math.pow(this.computedEnd.y - this.computedStart.y, 2)
1802         );
1803
1804         if (lineLength < 1) {
1805             const dx = x - this.computedStart.x;
1806             const dy = y - this.computedStart.y;
1807             return Math.sqrt(dx * dx + dy * dy) <= 5;
1808         }
1809
1810         const t = ((x - this.computedStart.x) * (this.computedEnd.x -
this.computedStart.x) +
1811                     (y - this.computedStart.y) * (this.computedEnd.y -
this.computedStart.y)) /
1812                     (lineLength * lineLength);
1813
1814         if (t < 0) {
1815             const dx = x - this.computedStart.x;
1816             const dy = y - this.computedStart.y;
1817             return Math.sqrt(dx * dx + dy * dy) <= 5;
1818         }
1819
1820         if (t > 1) {
1821             const dx = x - this.computedEnd.x;
1822             const dy = y - this.computedEnd.y;
1823             return Math.sqrt(dx * dx + dy * dy) <= 5;
1824         }
1825     }
```

```
1826         const px = this.computedStart.x + t * (this.computedEnd.x -  
1827             this.computedStart.x);  
1828         const py = this.computedStart.y + t * (this.computedEnd.y -  
1829             this.computedStart.y);  
1830         const dx = x - px;  
1831         const dy = y - py;  
1832         return Math.sqrt(dx * dx + dy * dy) <= 5;  
1833     }  
1834  
1835     // This object type doesn't use the regular move method  
1836     // Instead, it updates the frame/column values  
1837  
1838     toJSON() {  
1839         const json = super.toJSON();  
1840         return {  
1841             ...json,  
1842             startFrame: this.startFrame,  
1843             startColumn: this.startColumn,  
1844             endFrame: this.endFrame,  
1845             endColumn: this.endColumn,  
1846             dashPattern: this.dashPattern,  
1847             arrowStart: this.arrowStart,  
1848             arrowEnd: this.arrowEnd,  
1849             arrowSize: this.arrowSize  
1850         };  
1851     }  
1852  
1853     static fromJSON(data) {  
1854         return new FrameSpanningLineObject(data);  
1855     }  
1856  
1857     // Register all object types in a factory  
1858     const DrawingObjectFactory = {  
1859         types: {  
1860             'drawingObject': DrawingObject,  
1861             'line': LineObject,  
1862             'arrow': ArrowObject,  
1863             'rectangle': RectangleObject,  
1864             'ellipse': EllipseObject,  
1865             'text': TextObject,  
1866             'image': ImageObject,  
1867             'symbol': SymbolObject,  
1868             'path': PathObject,  
1869             'frameSpanningLine': FrameSpanningLineObject  
1870         },  
1871         createFromJSON(data) {  
1872             const Type = this.types[data.type];  
1873             if (Type) {
```

```
1874             return Type.fromJson(data);
1875         }
1876         return null;
1877     }
1878 }
1879
1880 /**
1881 * DRAWING TOOL SYSTEM
1882 * Manages the drawing tools and interfaces with the layer system
1883 */
1884 class DrawingToolSystem {
1885     constructor(layerSystem) {
1886         this.layerSystem = layerSystem;
1887         this.activeTool = null;
1888         this.toolSettings = {
1889             color: '#ff0000',
1890             lineWidth: 2,
1891             fill: false,
1892             fillColor: '#ff8080',
1893             fontSize: 16,
1894             fontFamily: 'Arial, sans-serif',
1895             textAlign: 'left',
1896             symbolType: 'default',
1897             symbolScale: 1.0
1898         };
1899
1900         this.availableTools = {
1901             'select': new SelectTool(this),
1902             'pen': new PenTool(this),
1903             'line': new LineTool(this),
1904             'arrow': new ArrowTool(this),
1905             'rectangle': new RectangleTool(this),
1906             'ellipse': new EllipseTool(this),
1907             'text': new TextTool(this),
1908             'image': new ImageTool(this),
1909             'symbol': new SymbolTool(this),
1910             'frameLine': new FrameSpanningLineTool(this),
1911             'eraser': new EraserTool(this)
1912         };
1913
1914         // Default to select tool
1915         this.setActiveTool('select');
1916
1917         // Set up toolbar UI
1918         this.createToolbar();
1919     }
1920
1921     setActiveTool(toolName) {
1922         if (this.activeTool) {
1923             this.activeTool.deactivate();
```

```
1924     }
1925
1926     if (this.availableTools[toolName]) {
1927         this.activeTool = this.availableTools[toolName];
1928         this.activeTool.activate();
1929         return true;
1930     }
1931
1932     return false;
1933 }
1934
1935 createToolbar() {
1936     // Create toolbar container
1937     const toolbar = document.createElement('div');
1938     toolbar.className = 'drawing-toolbar';
1939     toolbar.style.display = 'flex';
1940     toolbar.style.flexWrap = 'wrap';
1941     toolbar.style.gap = '5px';
1942     toolbar.style.padding = '10px';
1943     toolbar.style.backgroundColor = '#f5f5f5';
1944     toolbar.style.marginBottom = '10px';
1945     toolbar.style.borderRadius = '5px';
1946
1947     // Add tool buttons
1948     this.addButton(toolbar, 'select', 'Select', '👉');
1949     this.addButton(toolbar, 'pen', 'Freehand Drawing', '📝');
1950     this.addButton(toolbar, 'line', 'Line', '-');
1951     this.addButton(toolbar, 'arrow', 'Arrow', '→');
1952     this.addButton(toolbar, 'rectangle', 'Rectangle', '□');
1953     this.addButton(toolbar, 'ellipse', 'Circle/Ellipse', '○');
1954     this.addButton(toolbar, 'text', 'Text', 'T');
1955     this.addButton(toolbar, 'image', 'Insert Image', '🖼️');
1956     this.addButton(toolbar, 'symbol', 'Animation Symbol', '⭐');
1957     this.addButton(toolbar, 'frameLine', 'Multi-Frame Line', '💠');
1958
1959     // Add separator
1960     toolbar.appendChild(document.createElement('div')).style.borderLeft = '1px
solid #ccc';
1961     toolbar.lastChild.style.height = '30px';
1962
1963     // Add color picker
1964     const colorContainer = document.createElement('div');
1965     colorContainer.style.display = 'flex';
1966     colorContainer.style.alignItems = 'center';
1967     colorContainer.style.gap = '5px';
1968
1969     const colorLabel = document.createElement('label');
1970     colorLabel.textContent = 'Color:';
1971     colorLabel.htmlFor = 'drawing-color';
1972 }
```

```
1973 const colorPicker = document.createElement('input');
1974 colorPicker.type = 'color';
1975 colorPicker.id = 'drawing-color';
1976 colorPicker.value = this.toolSettings.color;
1977 colorPicker.addEventListener('input', (e) => {
1978     this.toolSettings.color = e.target.value;
1979 });
1980
1981     colorContainer.appendChild(colorLabel);
1982     colorContainer.appendChild(colorPicker);
1983     toolbar.appendChild(colorContainer);
1984
1985     // Add line width selector
1986     const lineWidthContainer = document.createElement('div');
1987     lineWidthContainer.style.display = 'flex';
1988     lineWidthContainer.style.alignItems = 'center';
1989     lineWidthContainer.style.gap = '5px';
1990
1991     const lineWidthLabel = document.createElement('label');
1992     lineWidthLabel.textContent = 'Width:';
1993     lineWidthLabel.htmlFor = 'drawing-line-width';
1994
1995     const lineWidthSelect = document.createElement('select');
1996     lineWidthSelect.id = 'drawing-line-width';
1997
1998     const widths = [1, 2, 3, 5, 8, 12];
1999     widths.forEach(width => {
2000         const option = document.createElement('option');
2001         option.value = width;
2002         option.textContent = width + 'px';
2003         if (width === this.toolSettings.lineWidth) {
2004             option.selected = true;
2005         }
2006         lineWidthSelect.appendChild(option);
2007     });
2008
2009     lineWidthSelect.addEventListener('change', (e) => {
2010         this.toolSettings.lineWidth = parseInt(e.target.value);
2011     });
2012
2013     lineWidthContainer.appendChild(lineWidthLabel);
2014     lineWidthContainer.appendChild(lineWidthSelect);
2015     toolbar.appendChild(lineWidthContainer);
2016
2017     // Add fill option
2018     const fillContainer = document.createElement('div');
2019     fillContainer.style.display = 'flex';
2020     fillContainer.style.alignItems = 'center';
2021     fillContainer.style.gap = '5px';
2022 
```

```
2023 const fillCheck = document.createElement('input');
2024 fillCheck.type = 'checkbox';
2025 fillCheck.id = 'drawing-fill';
2026 fillCheck.checked = this.toolSettings.fill;
2027
2028 const fillLabel = document.createElement('label');
2029 fillLabel.textContent = 'Fill';
2030 fillLabel.htmlFor = 'drawing-fill';
2031
2032 fillCheck.addEventListener('change', (e) => {
2033     this.toolSettings.fill = e.target.checked;
2034     fillColorPicker.disabled = !e.target.checked;
2035 });
2036
2037 const fillColorPicker = document.createElement('input');
2038 fillColorPicker.type = 'color';
2039 fillColorPicker.id = 'drawing-fill-color';
2040 fillColorPicker.value = this.toolSettings.fillColor;
2041 fillColorPicker.disabled = !this.toolSettings.fill;
2042
2043 fillColorPicker.addEventListener('input', (e) => {
2044     this.toolSettings.fillColor = e.target.value;
2045 });
2046
2047 fillContainer.appendChild(fillCheck);
2048 fillContainer.appendChild(fillLabel);
2049 fillContainer.appendChild(fillColorPicker);
2050 toolbar.appendChild(fillContainer);
2051
2052 // Add separator
2053 toolbar.appendChild(document.createElement('div')).style.borderLeft = '1px
solid #ccc';
2054 toolbar.lastChild.style.height = '30px';
2055
2056 // Add layer selector
2057 const layerSelector = document.createElement('select');
2058 layerSelector.id = 'drawing-layer-selector';
2059
2060 // Add options for each layer
2061 this.layerSystem.layers.forEach((layer, index) => {
2062     const option = document.createElement('option');
2063     option.value = index;
2064     option.textContent = layer.name;
2065     if (index === this.layerSystem.activeLayerIndex) {
2066         option.selected = true;
2067     }
2068     layerSelector.appendChild(option);
2069 });
2070
2071 layerSelector.addEventListener('change', (e) => {
```

```
2072         this.layerSystem.setActiveLayer(parseInt(e.target.value));
2073     });
2074
2075     const layerLabel = document.createElement('label');
2076     layerLabel.textContent = 'Layer:';
2077     layerLabel.htmlFor = 'drawing-layer-selector';
2078     layerLabel.style.marginRight = '5px';
2079
2080     toolbar.appendChild(layerLabel);
2081     toolbar.appendChild(layerSelector);
2082
2083     // Add separator
2084     toolbar.appendChild(document.createElement('div')).style.borderLeft = '1px
solid #ccc';
2085     toolbar.lastChild.style.height = '30px';
2086
2087     // Add eraser tool
2088     this.addToolButton(toolbar, 'eraser', 'Eraser', '✗');
2089
2090     // Add clear button
2091     const clearButton = document.createElement('button');
2092     clearButton.textContent = 'Clear All Drawings';
2093     clearButton.style.backgroundColor = '#ff5555';
2094     clearButton.style.color = 'white';
2095     clearButton.style.border = 'none';
2096     clearButton.style.borderRadius = '4px';
2097     clearButton.style.padding = '5px 10px';
2098     clearButton.style.cursor = 'pointer';
2099
2100     clearButton.addEventListener('click', () => {
2101         if (confirm('Are you sure you want to clear all drawings?')) {
2102             this.layerSystem.clearAllLayers();
2103         }
2104     });
2105
2106     toolbar.appendChild(clearButton);
2107
2108     // Find controls div and add toolbar before it
2109     const controls = document.querySelector('.controls');
2110     if (controls) {
2111         controls.parentNode.insertBefore(toolbar, controls.nextSibling);
2112     } else {
2113         document.body.insertBefore(toolbar, document.body.firstChild);
2114     }
2115 }
2116
2117 addToolButton(toolbar, toolName, tooltip, icon) {
2118     const button = document.createElement('button');
2119     button.textContent = icon;
2120     button.title = tooltip;
```

```
2121     button.style.width = '36px';
2122     button.style.height = '36px';
2123     button.style.fontSize = '16px';
2124     button.style.margin = '0';
2125     button.style.padding = '5px';
2126     button.style.borderRadius = '4px';
2127     button.style.border = '1px solid #ccc';
2128     button.style.backgroundColor = 'white';
2129     button.style.cursor = 'pointer';

2130
2131     button.addEventListener('click', () => {
2132         this.setActiveTool(toolName);

2133         // Update active button styling
2134         document.querySelectorAll('.drawing-toolbar button').forEach(btn => {
2135             btn.style.backgroundColor = 'white';
2136             btn.style.color = 'black';
2137         });

2138         button.style.backgroundColor = '#4CAF50';
2139         button.style.color = 'white';
2140     });

2141
2142     // Set active state for default tool
2143     if (toolName === 'select') {
2144         button.style.backgroundColor = '#4CAF50';
2145         button.style.color = 'white';
2146     }

2147     toolbar.appendChild(button);
2148 }
2149
2150 /**
2151 * DRAWING TOOLS
2152 * Individual tool implementations
2153 */
2154
2155
2156
2157
2158
2159 // Base tool class
2160 class DrawingTool {
2161     constructor(toolSystem) {
2162         this.toolSystem = toolSystem;
2163         this.layerSystem = toolSystem.layerSystem;
2164         this.active = false;
2165         this.settings = toolSystem.toolSettings;
2166     }
2167
2168     activate() {
2169         this.active = true;
2170         this.layerSystem.enableDrawing();
```

```
2171     this.attachEvents();
2172 }
2173
2174 deactivate() {
2175     this.active = false;
2176     // turn pointer-events back off so clicks go to the table cells
2177     this.toolSystem.layerSystem.disableDrawing();
2178     this.detachEvents();
2179 }
2180
2181 attachEvents() {
2182     // Override in subclasses
2183 }
2184
2185 detachEvents() {
2186     // Override in subclasses
2187 }
2188 }
2189
2190 // Select tool for manipulating objects
2191 class SelectTool extends DrawingTool {
2192
2193     constructor(toolSystem) {
2194         super(toolSystem);
2195         this.selectedObject = null;
2196         this.selectedLayer = null;
2197         this.dragging = false;
2198         this.dragStart = { x: 0, y: 0 };
2199         this.objectStart = { x: 0, y: 0 };
2200     }
2201     activate() {
2202         this.active = true;
2203         // let pointer-events go to the table cells instead of the canvases
2204         this.layerSystem.disableDrawing();
2205         this.attachEvents();
2206     }
2207
2208     deactivate() {
2209         this.active = false;
2210         this.layerSystem.disableDrawing();
2211         this.detachEvents();
2212     }
2213     attachEvents() {
2214         const canvas = this.layerSystem.getActiveLayer().canvas;
2215
2216         canvas.addEventListener('mousedown', this.handleMouseDown);
2217         canvas.addEventListener('pointerdown', this.handleMouseDown);
2218         document.addEventListener('mousemove', this.handleMouseMove);
2219         document.addEventListener('pointermove', this.handleMouseMove);
2220         document.addEventListener('mouseup', this.handleMouseUp);
```

```
2221     document.addEventListener('pointerup', this.handleMouseUp);
2222     document.addEventListener('keydown', this.handleKeyDown);
2223 }
2224
2225 detachEvents() {
2226     const canvas = this.layerSystem.getActiveLayer().canvas;
2227
2228     canvas.removeEventListener('mousedown', this.handleMouseDown);
2229     canvas.removeEventListener('pointerdown', this.handleMouseDown);
2230     document.removeEventListener('mousemove', this.handleMouseMove);
2231     document.removeEventListener('pointermove', this.handleMouseMove);
2232     document.removeEventListener('mouseup', this.handleMouseUp);
2233     document.removeEventListener('pointerup', this.handleMouseUp);
2234     document.removeEventListener('keydown', this.handleKeyDown);
2235
2236     // Clear selection
2237     this.clearSelection();
2238 }
2239
2240 handleMouseDown = (e) => {
2241     // Convert to canvas coordinates
2242     const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2243
2244     // Check if clicked on an object
2245     const hit = this.layerSystem.findObjectAt(coords.x, coords.y);
2246
2247     if (hit) {
2248         // Select the object
2249         this.selectObject(hit.object, hit.layerIndex);
2250
2251         // Start drag
2252         this.dragging = true;
2253         this.dragStart = { x: coords.x, y: coords.y };
2254         this.objectStart = { x: hit.object.x, y: hit.object.y };
2255     } else {
2256         // Clear selection if clicked empty space
2257         this.clearSelection();
2258     }
2259 }
2260
2261 handleMouseMove = (e) => {
2262     if (!this.dragging || !this.selectedObject) return;
2263
2264     const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2265
2266     // Calculate move distance
2267     const dx = coords.x - this.dragStart.x;
2268     const dy = coords.y - this.dragStart.y;
2269
2270     // Move the selected object
```

```
2271     this.selectedObject.x = this.objectStart.x + dx;
2272     this.selectedObject.y = this.objectStart.y + dy;
2273
2274     // For objects with special move handling
2275     this.selectedObject.move(dx, dy);
2276     this.selectedObject.x = this.objectStart.x; // Reset x as move() already
2277 handled it
2278
2279 handled it
2280
2281         // Redraw
2282         this.layerSystem.redrawAll();
2283
2284
2285     handleMouseUp = () => {
2286         this.dragging = false;
2287     }
2288
2289
2290     handleKeyDown = (e) => {
2291         if (!this.selectedObject) return;
2292
2293         // Delete key
2294         if (e.key === 'Delete' || e.key === 'Backspace') {
2295             this.layerSystem.removeObject(this.selectedObject, this.selectedLayer);
2296             this.clearSelection();
2297         }
2298
2299         // Arrow keys for fine movement
2300         const moveDistance = e.shiftKey ? 10 : 1;
2301
2302         if (e.key === 'ArrowLeft') {
2303             this.selectedObject.move(-moveDistance, 0);
2304             this.layerSystem.redrawAll();
2305         } else if (e.key === 'ArrowRight') {
2306             this.selectedObject.move(moveDistance, 0);
2307             this.layerSystem.redrawAll();
2308         } else if (e.key === 'ArrowUp') {
2309             this.selectedObject.move(0, -moveDistance);
2310             this.layerSystem.redrawAll();
2311         } else if (e.key === 'ArrowDown') {
2312             this.selectedObject.move(0, moveDistance);
2313             this.layerSystem.redrawAll();
2314         }
2315
2316     selectObject(object, layerIndex) {
2317         // Clear previous selection
2318         this.clearSelection();
2319
2320         // Set new selection
2321     }
2322 }
```

```
2319     this.selectedObject = object;
2320     this.selectedLayer = layerIndex;
2321     object.selected = true;
2322
2323     // Redraw with selection visual
2324     this.layerSystem.redrawAll();
2325 }
2326
2327     clearSelection() {
2328         if (this.selectedObject) {
2329             this.selectedObject.selected = false;
2330             this.selectedObject = null;
2331             this.selectedLayer = null;
2332             this.layerSystem.redrawAll();
2333         }
2334     }
2335 }
2336
2337 // Pen tool for free drawing
2338 class PenTool extends DrawingTool {
2339     constructor(toolSystem) {
2340         super(toolSystem);
2341         this.currentPath = null;
2342         this.drawing = false;
2343     }
2344
2345     attachEvents() {
2346         const canvas = this.layerSystem.getActiveLayer().canvas;
2347
2348         canvas.addEventListener('mousedown', this.handleMouseDown);
2349         canvas.addEventListener('pointerdown', this.handleMouseDown);
2350         document.addEventListener('mousemove', this.handleMouseMove);
2351         document.addEventListener('pointermove', this.handleMouseMove);
2352         document.addEventListener('mouseup', this.handleMouseUp);
2353         document.addEventListener('pointerup', this.handleMouseUp);
2354     }
2355
2356     detachEvents() {
2357         const canvas = this.layerSystem.getActiveLayer().canvas;
2358
2359         canvas.removeEventListener('mousedown', this.handleMouseDown);
2360         canvas.removeEventListener('pointerdown', this.handleMouseDown);
2361         document.removeEventListener('mousemove', this.handleMouseMove);
2362         document.removeEventListener('pointermove', this.handleMouseMove);
2363         document.removeEventListener('mouseup', this.handleMouseUp);
2364         document.removeEventListener('pointerup', this.handleMouseUp);
2365
2366         // Finish any in-progress drawing
2367         this.finishDrawing();
2368     }
}
```

```
2369
2370     handleMouseDown = (e) => {
2371         e.preventDefault();
2372         if (e.pointerId != null) {
2373             e.target.setPointerCapture(e.pointerId);
2374         }
2375
2376         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2377
2378         // Start a new path
2379         this.currentPath = new PathObject({
2380             color: this.settings.color,
2381             lineWidth: this.settings.lineWidth,
2382             x: coords.x,
2383             y: coords.y
2384         });
2385
2386         this.currentPath.addPoint(coords.x, coords.y);
2387         this.drawing = true;
2388
2389         // Add to layer
2390         this.layerSystem.addObject(this.currentPath);
2391     }
2392
2393     handleMouseMove = (e) => {
2394         if (!this.drawing || !this.currentPath) return;
2395
2396         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2397
2398         // Add point to path
2399         this.currentPath.addPoint(coords.x, coords.y);
2400
2401         // Redraw
2402         this.layerSystem.redrawLayer(this.layerSystem.activeLayerIndex);
2403     }
2404
2405     handleMouseUp = (e) => {
2406         if (e.pointerId != null) {
2407             e.target.releasePointerCapture(e.pointerId);
2408         }
2409         this.finishDrawing();
2410     }
2411
2412     finishDrawing() {
2413         if (this.drawing && this.currentPath) {
2414             // Finish the path
2415             this.drawing = false;
2416             this.currentPath = null;
2417         }
2418     }
}
```

```
2419 }
2420
2421 // Line tool
2422 class LineTool extends DrawingTool {
2423     constructor(toolSystem) {
2424         super(toolSystem);
2425         this.startPoint = null;
2426         this.currentLine = null;
2427         this.drawing = false;
2428     }
2429
2430     attachEvents() {
2431         const canvas = this.layerSystem.getActiveLayer().canvas;
2432
2433         canvas.addEventListener('mousedown', this.handleMouseDown);
2434         document.addEventListener('mousemove', this.handleMouseMove);
2435         document.addEventListener('mouseup', this.handleMouseUp);
2436     }
2437
2438     detachEvents() {
2439         const canvas = this.layerSystem.getActiveLayer().canvas;
2440
2441         canvas.removeEventListener('mousedown', this.handleMouseDown);
2442         document.removeEventListener('mousemove', this.handleMouseMove);
2443         document.removeEventListener('mouseup', this.handleMouseUp);
2444
2445         // Finish any in-progress drawing
2446         this.finishDrawing();
2447     }
2448
2449     handleMouseDown = (e) => {
2450         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2451
2452         this.startPoint = coords;
2453
2454         // Create temporary line
2455         this.currentLine = new LineObject({
2456             x: coords.x,
2457             y: coords.y,
2458             x2: coords.x,
2459             y2: coords.y,
2460             color: this.settings.color,
2461             lineWidth: this.settings.lineWidth
2462         });
2463
2464         this.drawing = true;
2465
2466         // Add to layer
2467         this.layerSystem.addObject(this.currentLine);
2468     }
```

```
2469
2470     handleMouseMove = (e) => {
2471         if (!this.drawing || !this.currentLine) return;
2472
2473         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2474
2475         // Update end point
2476         this.currentLine.x2 = coords.x;
2477         this.currentLine.y2 = coords.y;
2478
2479         // Redraw
2480         this.layerSystem.redrawLayer(this.layerSystem.activeLayerIndex);
2481     }
2482
2483     handleMouseUp = () => {
2484         this.finishDrawing();
2485     }
2486
2487     finishDrawing() {
2488         if (this.drawing && this.currentLine) {
2489             // Finish the line
2490             this.drawing = false;
2491             this.startPoint = null;
2492             this.currentLine = null;
2493         }
2494     }
2495 }
2496
2497 // Arrow tool (extends Line tool)
2498 class ArrowTool extends LineTool {
2499     handleMouseDown = (e) => {
2500         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2501
2502         this.startPoint = coords;
2503
2504         // Create temporary arrow
2505         this.currentLine = new ArrowObject({
2506             x: coords.x,
2507             y: coords.y,
2508             x2: coords.x,
2509             y2: coords.y,
2510             color: this.settings.color,
2511             lineWidth: this.settings.lineWidth
2512         });
2513
2514         this.drawing = true;
2515
2516         // Add to layer
2517         this.layerSystem.addObject(this.currentLine);
2518     }
}
```

```
2519     }
2520
2521     // Rectangle tool
2522     class RectangleTool extends DrawingTool {
2523         constructor(toolSystem) {
2524             super(toolSystem);
2525             this.startPoint = null;
2526             this.currentRect = null;
2527             this.drawing = false;
2528         }
2529
2530         attachEvents() {
2531             const canvas = this.layerSystem.getActiveLayer().canvas;
2532
2533             canvas.addEventListener('mousedown', this.handleMouseDown);
2534             document.addEventListener('mousemove', this.handleMouseMove);
2535             document.addEventListener('mouseup', this.handleMouseUp);
2536         }
2537
2538         detachEvents() {
2539             const canvas = this.layerSystem.getActiveLayer().canvas;
2540
2541             canvas.removeEventListener('mousedown', this.handleMouseDown);
2542             document.removeEventListener('mousemove', this.handleMouseMove);
2543             document.removeEventListener('mouseup', this.handleMouseUp);
2544
2545             // Finish any in-progress drawing
2546             this.finishDrawing();
2547         }
2548
2549         handleMouseDown = (e) => {
2550             const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2551
2552             this.startPoint = coords;
2553
2554             // Create temporary rectangle
2555             this.currentRect = new RectangleObject({
2556                 x: coords.x,
2557                 y: coords.y,
2558                 width: 0,
2559                 height: 0,
2560                 color: this.settings.color,
2561                 lineWidth: this.settings.lineWidth,
2562                 fill: this.settings.fill,
2563                 fillColor: this.settings.fillColor
2564             });
2565
2566             this.drawing = true;
2567
2568             // Add to layer
```

```
2569         this.layerSystem.addObject(this.currentRect);
2570     }
2571
2572     handleMouseMove = (e) => {
2573         if (!this.drawing || !this.currentRect) return;
2574
2575         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2576
2577         // Update dimensions
2578         const width = coords.x - this.startPoint.x;
2579         const height = coords.y - this.startPoint.y;
2580
2581         if (width < 0) {
2582             this.currentRect.x = coords.x;
2583             this.currentRect.width = Math.abs(width);
2584         } else {
2585             this.currentRect.x = this.startPoint.x;
2586             this.currentRect.width = width;
2587         }
2588
2589         if (height < 0) {
2590             this.currentRect.y = coords.y;
2591             this.currentRect.height = Math.abs(height);
2592         } else {
2593             this.currentRect.y = this.startPoint.y;
2594             this.currentRect.height = height;
2595         }
2596
2597         // Redraw
2598         this.layerSystem.redrawLayer(this.layerSystem.activeLayerIndex);
2599     }
2600
2601     handleMouseUp = () => {
2602         this.finishDrawing();
2603     }
2604
2605     finishDrawing() {
2606         if (this.drawing && this.currentRect) {
2607             // Finish the rectangle
2608             this.drawing = false;
2609             this.startPoint = null;
2610             this.currentRect = null;
2611         }
2612     }
2613 }
2614
2615 // Ellipse tool
2616 class EllipseTool extends DrawingTool {
2617     constructor(toolSystem) {
2618         super(toolSystem);
```

```
2619     this.center = null;
2620     this.currentEllipse = null;
2621     this.drawing = false;
2622 }
2623
2624 attachEvents() {
2625     const canvas = this.layerSystem.getActiveLayer().canvas;
2626
2627     canvas.addEventListener('mousedown', this.handleMouseDown);
2628     document.addEventListener('mousemove', this.handleMouseMove);
2629     document.addEventListener('mouseup', this.handleMouseUp);
2630 }
2631
2632 detachEvents() {
2633     const canvas = this.layerSystem.getActiveLayer().canvas;
2634
2635     canvas.removeEventListener('mousedown', this.handleMouseDown);
2636     document.removeEventListener('mousemove', this.handleMouseMove);
2637     document.removeEventListener('mouseup', this.handleMouseUp);
2638
2639     // Finish any in-progress drawing
2640     this.finishDrawing();
2641 }
2642
2643 handleMouseDown = (e) => {
2644     const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2645
2646     this.center = coords;
2647
2648     // Create temporary ellipse
2649     this.currentEllipse = new EllipseObject({
2650         x: coords.x,
2651         y: coords.y,
2652         radiusX: 0,
2653         radiusY: 0,
2654         color: this.settings.color,
2655         lineWidth: this.settings.lineWidth,
2656         fill: this.settings.fill,
2657         fillColor: this.settings.fillColor
2658     });
2659
2660     this.drawing = true;
2661
2662     // Add to layer
2663     this.layerSystem.addObject(this.currentEllipse);
2664 }
2665
2666 handleMouseMove = (e) => {
2667     if (!this.drawing || !this.currentEllipse) return;
2668 }
```

```
2669     const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2670
2671     // Update radii
2672     this.currentEllipse.radiusX = Math.abs(coords.x - this.center.x);
2673     this.currentEllipse.radiusY = Math.abs(coords.y - this.center.y);
2674
2675     // Redraw
2676     this.layerSystem.redrawLayer(this.layerSystem.activeLayerIndex);
2677 }
2678
2679 handleMouseUp = () => {
2680     this.finishDrawing();
2681 }
2682
2683 finishDrawing() {
2684     if (this.drawing && this.currentEllipse) {
2685         // Finish the ellipse
2686         this.drawing = false;
2687         this.center = null;
2688         this.currentEllipse = null;
2689     }
2690 }
2691
2692
2693 // Text tool
2694 class TextTool extends DrawingTool {
2695     constructor(toolSystem) {
2696         super(toolSystem);
2697         this.textInput = null;
2698     }
2699
2700     attachEvents() {
2701         const canvas = this.layerSystem.getActiveLayer().canvas;
2702
2703         canvas.addEventListener('click', this.handleClick);
2704     }
2705
2706     detachEvents() {
2707         const canvas = this.layerSystem.getActiveLayer().canvas;
2708
2709         canvas.removeEventListener('click', this.handleClick);
2710
2711         // Remove any active text input
2712         this.removeTextInput();
2713     }
2714
2715     handleClick = (e) => {
2716         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2717
2718         // Show text input at click position
```

```
2719     this.showTextInput(coords.x, coords.y);
2720 }
2721
2722     showTextInput(x, y) {
2723         // Remove any existing text input
2724         this.removeTextInput();
2725
2726         // Create text input element
2727         this.textInput = document.createElement('div');
2728         this.textInput.style.position = 'absolute';
2729         this.textInput.style.zIndex = '100';
2730
2731         // Position relative to canvas
2732         const canvasRect = this.layerSystem.container.getBoundingClientRect();
2733         this.textInput.style.left = (canvasRect.left + x) + 'px';
2734         this.textInput.style.top = (canvasRect.top + y) + 'px';
2735
2736         // Style the input
2737         this.textInput.style.backgroundColor = 'white';
2738         this.textInput.style.border = '1px solid #ccc';
2739         this.textInput.style.padding = '5px';
2740         this.textInput.style.borderRadius = '3px';
2741         this.textInput.style.boxShadow = '0 2px 5px rgba(0,0,0,0.1)';
2742
2743         // Create the actual input
2744         const input = document.createElement('input');
2745         input.type = 'text';
2746         input.placeholder = 'Enter text...';
2747         input.style.width = '200px';
2748         input.style.padding = '5px';
2749         input.style.border = '1px solid #ddd';
2750         input.style.borderRadius = '3px';
2751
2752         // Create font size selector
2753         const sizeSelect = document.createElement('select');
2754         [8, 10, 12, 14, 16, 18, 20, 24, 36].forEach(size => {
2755             const option = document.createElement('option');
2756             option.value = size;
2757             option.textContent = size + 'px';
2758             if (size === this.settings.fontSize) {
2759                 option.selected = true;
2760             }
2761             sizeSelect.appendChild(option);
2762         });
2763
2764         sizeSelect.addEventListener('change', (e) => {
2765             this.settings.fontSize = parseInt(e.target.value);
2766         });
2767
2768         // Create alignment options
```

```
2769     const alignmentDiv = document.createElement('div');
2770     alignmentDiv.style.display = 'flex';
2771     alignmentDiv.style.marginTop = '5px';
2772
2773     ['left', 'center', 'right'].forEach((align => {
2774         const button = document.createElement('button');
2775         button.textContent = align[0].toUpperCase();
2776         button.style.flex = '1';
2777         button.style.padding = '2px 5px';
2778         button.style.backgroundColor = this.settings.textAlign === align ?
2779             '#4CAF50' : '#f1f1f1';
2780         button.style.color = this.settings.textAlign === align ? 'white' :
2781             'black';
2782         button.style.border = '1px solid #ccc';
2783         button.style.cursor = 'pointer';
2784
2785         button.addEventListener('click', () => {
2786             this.settings.textAlign = align;
2787             alignmentDiv.querySelectorAll('button').forEach(btn => {
2788                 btn.style.backgroundColor = '#f1f1f1';
2789                 btn.style.color = 'black';
2790             });
2791             button.style.backgroundColor = '#4CAF50';
2792             button.style.color = 'white';
2793         });
2794
2795
2796         // Add UI elements to the container
2797         this.textInput.appendChild(input);
2798         this.textInput.appendChild(document.createElement('br'));
2799         this.textInput.appendChild(document.createTextNode('Size: '));
2800         this.textInput.appendChild(sizeSelect);
2801         this.textInput.appendChild(document.createElement('br'));
2802         this.textInput.appendChild(alignmentDiv);
2803
2804         // Add buttons container
2805         const buttons = document.createElement('div');
2806         buttons.style.display = 'flex';
2807         buttons.style.marginTop = '5px';
2808         buttons.style.gap = '5px';
2809
2810         // Add button
2811         const addButton = document.createElement('button');
2812         addButton.textContent = 'Add Text';
2813         addButton.style.flex = '1';
2814         addButton.style.padding = '5px';
2815         addButton.style.backgroundColor = '#4CAF50';
2816         addButton.style.color = 'white';
```

```
2817     addButton.style.border = 'none';
2818     addButton.style.borderRadius = '3px';
2819     addButton.style.cursor = 'pointer';
2820
2821     addButton.addEventListener('click', () => {
2822         this.addText(x, y, input.value);
2823     });
2824
2825     // Cancel button
2826     const cancelButton = document.createElement('button');
2827     cancelButton.textContent = 'Cancel';
2828     cancelButton.style.flex = '1';
2829     cancelButton.style.padding = '5px';
2830     cancelButton.style.backgroundColor = '#f44336';
2831     cancelButton.style.color = 'white';
2832     cancelButton.style.border = 'none';
2833     cancelButton.style.borderRadius = '3px';
2834     cancelButton.style.cursor = 'pointer';
2835
2836     cancelButton.addEventListener('click', () => {
2837         this.removeTextInput();
2838     });
2839
2840     buttons.appendChild(addButton);
2841     buttons.appendChild(cancelButton);
2842     this.textInput.appendChild(buttons);
2843
2844     // Add to document
2845     document.body.appendChild(this.textInput);
2846
2847     // Focus the input
2848     input.focus();
2849
2850     // Handle enter key
2851     input.addEventListener('keydown', (e) => {
2852         if (e.key === 'Enter') {
2853             this.addText(x, y, input.value);
2854         } else if (e.key === 'Escape') {
2855             this.removeTextInput();
2856         }
2857     });
2858 }
2859
2860 removeTextInput() {
2861     if (this.textInput && this.textInput.parentNode) {
2862         this.textInput.parentNode.removeChild(this.textInput);
2863         this.textInput = null;
2864     }
2865 }
2866 }
```

```
2867 |     addText(x, y, text) {
2868 |         if (!text.trim()) {
2869 |             this.removeTextInput();
2870 |             return;
2871 |         }
2872 |
2873 |         // Create text object
2874 |         const textObj = new TextObject({
2875 |             x: x,
2876 |             y: y,
2877 |             text: text,
2878 |             color: this.settings.color,
2879 |             fontSize: this.settings.fontSize,
2880 |             fontFamily: this.settings.fontFamily,
2881 |             align: this.settings.textAlign
2882 |         });
2883 |
2884 |         // Add to layer
2885 |         this.layerSystem.addObject(textObj);
2886 |
2887 |         // Remove text input
2888 |         this.removeTextInput();
2889 |     }
2890 |
2891 |     // Image tool
2892 |     class ImageTool extends DrawingTool {
2893 |         constructor(toolSystem) {
2894 |             super(toolSystem);
2895 |             this.fileInput = null;
2896 |         }
2897 |
2898 |
2899 |         attachEvents() {
2900 |             const canvas = this.layerSystem.getActiveLayer().canvas;
2901 |
2902 |             canvas.addEventListener('click', this.handleClick);
2903 |         }
2904 |
2905 |         detachEvents() {
2906 |             const canvas = this.layerSystem.getActiveLayer().canvas;
2907 |
2908 |             canvas.removeEventListener('click', this.handleClick);
2909 |         }
2910 |
2911 |         handleClick = (e) => {
2912 |             // Show file upload dialog
2913 |             this.showFileDialog();
2914 |         }
2915 |
2916 |         showFileDialog() {
```

```
2917     // Create hidden file input if it doesn't exist
2918     if (!this.fileInput) {
2919         this.fileInput = document.createElement('input');
2920         this.fileInput.type = 'file';
2921         this.fileInput.accept = 'image/*';
2922         this.fileInput.style.display = 'none';
2923         document.body.appendChild(this.fileInput);
2924
2925         this.fileInput.addEventListener('change', (e) => {
2926             if (e.target.files && e.target.files[0]) {
2927                 this.handleFileSelect(e.target.files[0]);
2928             }
2929         });
2930     }
2931
2932     // Trigger file dialog
2933     this.fileInput.click();
2934 }
2935
2936 handleFileSelect(file) {
2937     // Read the file and create a data URL
2938     const reader = new FileReader();
2939
2940     reader.onload = (e) => {
2941         const imageUrl = e.target.result;
2942
2943         // Show image placement UI
2944         this.showImagePlacementUI(imageUrl);
2945     };
2946
2947     reader.readAsDataURL(file);
2948 }
2949
2950 showImagePlacementUI(imageUrl) {
2951     // Create a preview image to get dimensions
2952     const img = new Image();
2953
2954     img.onload = () => {
2955         // Calculate dimensions (max size 300px width/height while maintaining
2956         aspect ratio)
2957         let width = img.width;
2958         let height = img.height;
2959
2960         const maxSize = 300;
2961         if (width > maxSize || height > maxSize) {
2962             if (width > height) {
2963                 height = (height / width) * maxSize;
2964                 width = maxSize;
2965             } else {
2966                 width = (width / height) * maxSize;
```

```
2966             height = maxSize;
2967         }
2968     }
2969
2970     // Create placement UI
2971     const placementUI = document.createElement('div');
2972     placementUI.style.position = 'fixed';
2973     placementUI.style.top = '50%';
2974     placementUI.style.left = '50%';
2975     placementUI.style.transform = 'translate(-50%, -50%)';
2976     placementUI.style.backgroundColor = 'white';
2977     placementUI.style.padding = '20px';
2978     placementUI.style.borderRadius = '5px';
2979     placementUI.style.boxShadow = '0 0 10px rgba(0,0,0,0.3)';
2980     placementUI.style.zIndex = '1000';
2981
2982     // Add heading
2983     const heading = document.createElement('h3');
2984     heading.textContent = 'Place Image';
2985     heading.style.margin = '0 0 10px 0';
2986
2987     // Add image preview
2988     const preview = document.createElement('img');
2989     preview.src = imageUrl;
2990     preview.style maxWidth = '300px';
2991     preview.style maxHeight = '300px';
2992     preview.style display = 'block';
2993     preview.style marginBottom = '10px';
2994
2995     // Size controls
2996     const sizeControls = document.createElement('div');
2997     sizeControls.style marginBottom = '10px';
2998
2999     const widthLabel = document.createElement('label');
3000     widthLabel.textContent = 'Width: ';
3001     const widthInput = document.createElement('input');
3002     widthInput.type = 'number';
3003     widthInput.value = Math.round(width);
3004     widthInput.style.width = '60px';
3005
3006     const heightLabel = document.createElement('label');
3007     heightLabel.textContent = 'Height: ';
3008     heightLabel.style marginLeft = '10px';
3009     const heightInput = document.createElement('input');
3010     heightInput.type = 'number';
3011     heightInput.value = Math.round(height);
3012     heightInput.style.width = '60px';
3013
3014     // Maintain aspect ratio
3015     const aspectRatio = img.width / img.height;
```

```
3016
3017     widthInput.addEventListener('input', () => {
3018         const newWidth = parseInt(widthInput.value);
3019         if (!isNaN(newWidth)) {
3020             heightInput.value = Math.round(newWidth / aspectRatio);
3021         }
3022     });
3023
3024     heightInput.addEventListener('input', () => {
3025         const newHeight = parseInt(heightInput.value);
3026         if (!isNaN(newHeight)) {
3027             widthInput.value = Math.round(newHeight * aspectRatio);
3028         }
3029     });
3030
3031     sizeControls.appendChild(widthLabel);
3032     sizeControls.appendChild(widthInput);
3033     sizeControls.appendChild(heightLabel);
3034     sizeControls.appendChild(heightInput);
3035
3036     // Buttons container
3037     const buttons = document.createElement('div');
3038     buttons.style.display = 'flex';
3039     buttons.style.justifyContent = 'space-between';
3040     buttons.style.marginTop = '15px';
3041
3042     // Place button
3043     const placeButton = document.createElement('button');
3044     placeButton.textContent = 'Place Image';
3045     placeButton.style.padding = '8px 15px';
3046     placeButton.style.backgroundColor = '#4CAF50';
3047     placeButton.style.color = 'white';
3048     placeButton.style.border = 'none';
3049     placeButton.style.borderRadius = '4px';
3050     placeButton.style.cursor = 'pointer';
3051
3052     placeButton.addEventListener('click', () => {
3053         // Get center of view as placement position
3054         const containerRect =
3055             this.layerSystem.container.getBoundingClientRect();
3056         const x = containerRect.width / 2;
3057         const y = containerRect.height / 2;
3058
3059         // Get dimensions from inputs
3060         const finalWidth = parseInt(widthInput.value);
3061         const finalHeight = parseInt(heightInput.value);
3062
3063         // Create and add image object
3064         const imageObj = new ImageObject({
3065             x: x - finalWidth / 2,
```

```
3065     y: y - finalHeight / 2,  
3066     width: finalWidth,  
3067     height: finalHeight,  
3068     imageUrl: imageUrl  
3069   );  
3070  
3071   this.layerSystem.addObject(imageObj);  
3072  
3073   // Remove UI  
3074   document.body.removeChild(placementUI);  
3075 );  
3076  
3077 // Cancel button  
3078 const cancelButton = document.createElement('button');  
3079 cancelButton.textContent = 'Cancel';  
3080 cancelButton.style.padding = '8px 15px';  
3081 cancelButton.style.backgroundColor = '#f44336';  
3082 cancelButton.style.color = 'white';  
3083 cancelButton.style.border = 'none';  
3084 cancelButton.style.borderRadius = '4px';  
3085 cancelButton.style.cursor = 'pointer';  
3086  
3087 cancelButton.addEventListener('click', () => {  
3088   document.body.removeChild(placementUI);  
3089 });  
3090  
3091 buttons.appendChild(cancelButton);  
3092 buttons.appendChild(placeButton);  
3093  
3094 // Assemble UI  
3095 placementUI.appendChild(heading);  
3096 placementUI.appendChild(preview);  
3097 placementUI.appendChild(sizeControls);  
3098 placementUI.appendChild(buttons);  
3099  
3100 // Add to document  
3101 document.body.appendChild(placementUI);  
3102 );  
3103  
3104 img.src = imageUrl;  
3105 }  
3106 }  
3107  
3108 // Symbol tool  
3109 class SymbolTool extends DrawingTool {  
3110   constructor(toolSystem) {  
3111     super(toolSystem);  
3112     this.symbolSelector = null;  
3113   }  
3114 }
```

```
3115 |     attachEvents() {
3116 |         const canvas = this.layerSystem.getActiveLayer().canvas;
3117 |
3118 |         canvas.addEventListener('click', this.handleClick);
3119 |     }
3120 |
3121 |     detachEvents() {
3122 |         const canvas = this.layerSystem.getActiveLayer().canvas;
3123 |
3124 |         canvas.removeEventListener('click', this.handleClick);
3125 |
3126 |         // Remove any active symbol selector
3127 |         this.removeSymbolSelector();
3128 |     }
3129 |
3130 |     handleClick = (e) => {
3131 |         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
3132 |
3133 |         // Show symbol selector
3134 |         this.showSymbolSelector(coords.x, coords.y);
3135 |     }
3136 |
3137 |     showSymbolSelector(x, y) {
3138 |         // Remove existing selector if any
3139 |         this.removeSymbolSelector();
3140 |
3141 |         // Create symbol selector
3142 |         this.symbolSelector = document.createElement('div');
3143 |         this.symbolSelector.style.position = 'absolute';
3144 |         this.symbolSelector.style.zIndex = '100';
3145 |
3146 |         // Position relative to canvas
3147 |         const canvasRect = this.layerSystem.container.getBoundingClientRect();
3148 |         this.symbolSelector.style.left = (canvasRect.left + x + 10) + 'px';
3149 |         this.symbolSelector.style.top = (canvasRect.top + y + 10) + 'px';
3150 |
3151 |         // Style
3152 |         this.symbolSelector.style.backgroundColor = 'white';
3153 |         this.symbolSelector.style.border = '1px solid #ccc';
3154 |         this.symbolSelector.style.padding = '10px';
3155 |         this.symbolSelector.style.borderRadius = '5px';
3156 |         this.symbolSelector.style.boxShadow = '0 2px 10px rgba(0,0,0,0.1)';
3157 |
3158 |         // Add title
3159 |         const title = document.createElement('h4');
3160 |         title.textContent = 'Select Animation Symbol';
3161 |         title.style.margin = '0 0 10px 0';
3162 |         this.symbolSelector.appendChild(title);
3163 |
3164 |         // Symbol grid
```

```
3165 const symbolGrid = document.createElement('div');
3166 symbolGrid.style.display = 'grid';
3167 symbolGrid.style.gridTemplateColumns = 'repeat(3, 1fr)';
3168 symbolGrid.style.gap = '10px';
3169 symbolGrid.style.marginBottom = '10px';
3170
3171 // Available symbols
3172 const symbols = [
3173     { type: 'anticipation', name: 'Anticipation' },
3174     { type: 'impact', name: 'Impact' },
3175     { type: 'keyframe', name: 'Keyframe' },
3176     { type: 'inbetween', name: 'Inbetween' },
3177     { type: 'hold', name: 'Hold' },
3178     { type: 'default', name: 'Default' }
3179 ];
3180
3181 // Create symbol preview for each type
3182 symbols.forEach(symbol => {
3183     const symbolItem = document.createElement('div');
3184     symbolItem.style.display = 'flex';
3185     symbolItem.style.flexDirection = 'column';
3186     symbolItem.style.alignItems = 'center';
3187     symbolItem.style.cursor = 'pointer';
3188     symbolItem.style.padding = '5px';
3189     symbolItem.style.border = '1px solid #ddd';
3190     symbolItem.style.borderRadius = '3px';
3191
3192     // Create canvas for symbol preview
3193     const canvas = document.createElement('canvas');
3194     canvas.width = 50;
3195     canvas.height = 50;
3196     const ctx = canvas.getContext('2d');
3197
3198     // Draw symbol preview
3199     ctx.save();
3200     ctx.strokeStyle = this.settings.color;
3201     ctx.fillStyle = this.settings.color;
3202     ctx.lineWidth = 2;
3203
3204     // Draw centered preview
3205     const symbolObj = new SymbolObject({
3206         x: 25,
3207         y: 25,
3208         symbolType: symbol.type,
3209         color: this.settings.color,
3210         lineWidth: 2,
3211         scale: 1.5
3212     });
3213
3214     symbolObj.draw(ctx);
```

```
3215     ctx.restore();  
3216  
3217     const name = document.createElement('div');  
3218     name.textContent = symbol.name;  
3219     name.style.marginTop = '5px';  
3220     name.style.fontSize = '12px';  
3221  
3222     symbolItem.appendChild(canvas);  
3223     symbolItem.appendChild(name);  
3224  
3225     // Add click handler  
3226     symbolItem.addEventListener('click', () => {  
3227         this.addSymbol(x, y, symbol.type);  
3228         this.removeSymbolSelector();  
3229     });  
3230  
3231     symbolGrid.appendChild(symbolItem);  
3232 });  
3233  
3234     this.symbolSelector.appendChild(symbolGrid);  
3235  
3236     // Scale control  
3237     const scaleContainer = document.createElement('div');  
3238     scaleContainer.style.display = 'flex';  
3239     scaleContainer.style.alignItems = 'center';  
3240     scaleContainer.style.marginBottom = '10px';  
3241  
3242     const scaleLabel = document.createElement('label');  
3243     scaleLabel.textContent = 'Scale: ';  
3244  
3245     const scaleInput = document.createElement('input');  
3246     scaleInput.type = 'range';  
3247     scaleInput.min = '0.5';  
3248     scaleInput.max = '3';  
3249     scaleInput.step = '0.1';  
3250     scaleInput.value = this.settings.symbolScale;  
3251     scaleInput.style.flex = '1';  
3252     scaleInput.style.marginLeft = '5px';  
3253  
3254     const scaleValue = document.createElement('span');  
3255     scaleValue.textContent = this.settings.symbolScale + 'x';  
3256     scaleValue.style.marginLeft = '5px';  
3257     scaleValue.style.width = '30px';  
3258  
3259     scaleInput.addEventListener('input', () => {  
3260         this.settings.symbolScale = parseFloat(scaleInput.value);  
3261         scaleValue.textContent = this.settings.symbolScale + 'x';  
3262     });  
3263  
3264     scaleContainer.appendChild(scaleLabel);
```

```
3265     scaleContainer.appendChild(scaleInput);
3266     scaleContainer.appendChild(scaleValue);
3267
3268     this.symbolSelector.appendChild(scaleContainer);
3269
3270     // Add cancel button
3271     const cancelButton = document.createElement('button');
3272     cancelButton.textContent = 'Cancel';
3273     cancelButton.style.width = '100%';
3274     cancelButton.style.padding = '5px';
3275     cancelButton.style.backgroundColor = '#f44336';
3276     cancelButton.style.color = 'white';
3277     cancelButton.style.border = 'none';
3278     cancelButton.style.borderRadius = '3px';
3279     cancelButton.style.cursor = 'pointer';
3280
3281     cancelButton.addEventListener('click', () => {
3282         this.removeSymbolSelector();
3283     });
3284
3285     this.symbolSelector.appendChild(cancelButton);
3286
3287     // Add to document
3288     document.body.appendChild(this.symbolSelector);
3289 }
3290
3291 removeSymbolSelector() {
3292     if (this.symbolSelector && this.symbolSelector.parentNode) {
3293         this.symbolSelector.parentNode.removeChild(this.symbolSelector);
3294         this.symbolSelector = null;
3295     }
3296 }
3297
3298 addSymbol(x, y, symbolType) {
3299     // Create symbol object
3300     const symbolObj = new SymbolObject({
3301         x: x,
3302         y: y,
3303         symbolType: symbolType,
3304         color: this.settings.color,
3305         lineWidth: this.settings.lineWidth,
3306         scale: this.settings.symbolScale
3307     });
3308
3309     // Add to layer
3310     this.layerSystem.addObject(symbolObj);
3311 }
3312 }
3313
3314 // Frame-spanning line tool (for connecting cells across frames)
```

```
3315 class FrameSpanningLineTool extends DrawingTool {  
3316     constructor(toolSystem) {  
3317         super(toolSystem);  
3318         this.startCell = null;  
3319         this.currentLine = null;  
3320         this.drawing = false;  
3321     }  
3322  
3323     attachEvents() {  
3324         // This tool works with the table cells, not the canvas  
3325         this.setupCellEvents();  
3326     }  
3327  
3328     detachEvents() {  
3329         this.removeCellEvents();  
3330  
3331         // Finish any in-progress drawing  
3332         this.finishDrawing();  
3333     }  
3334  
3335     setupCellEvents() {  
3336         // Find all table cells and add mousedown handler  
3337         const cells = document.querySelectorAll('#xsheet-table td');  
3338         cells.forEach(cell => {  
3339             cell.addEventListener('mousedown', this.handleCellMouseDown);  
3340             cell.addEventListener('mouseup', this.handleCellMouseUp);  
3341  
3342             // Add hover effect  
3343             cell.addEventListener('mouseover', this.handleCellHover);  
3344             cell.addEventListener('mouseout', this.handleCellOut);  
3345  
3346             // Store original background for hover effect  
3347             if (!cell.dataset.originalBg) {  
3348                 cell.dataset.originalBg = cell.style.backgroundColor || '';  
3349             }  
3350         });  
3351     }  
3352  
3353     removeCellEvents() {  
3354         const cells = document.querySelectorAll('#xsheet-table td');  
3355         cells.forEach(cell => {  
3356             cell.removeEventListener('mousedown', this.handleCellMouseDown);  
3357             cell.removeEventListener('mouseup', this.handleCellMouseUp);  
3358             cell.removeEventListener('mouseover', this.handleCellHover);  
3359             cell.removeEventListener('mouseout', this.handleCellOut);  
3360  
3361             // Restore original background  
3362             if (cell.dataset.originalBg) {  
3363                 cell.style.backgroundColor = cell.dataset.originalBg;  
3364             }  
3365         });  
3366     }  
3367 }
```

```
3365     });
3366 }
3367
3368     handleCellHover = (e) => {
3369         const cell = e.currentTarget;
3370         cell.style.backgroundColor = 'rgba(0, 123, 255, 0.2)';
3371     }
3372
3373     handleCellOut = (e) => {
3374         const cell = e.currentTarget;
3375         if (!this.drawing || cell !== this.startCell) {
3376             cell.style.backgroundColor = cell.dataset.originalBg || '';
3377         }
3378     }
3379
3380     handleCellMouseDown = (e) => {
3381         const cell = e.currentTarget;
3382
3383         // Find the row and column index
3384         const row = cell.parentElement;
3385         const frameNumber = parseInt(row.getAttribute('data-frame')) ||
3386         row.className.replace('frame-', '');
3387         const columnIndex = Array.from(row.children).indexOf(cell) + 1;
3388
3389         if (isNaN(frameNumber) || frameNumber <= 0) return;
3390
3391         // Store as start cell
3392         this.startCell = cell;
3393         this.drawing = true;
3394
3395         // Highlight the cell
3396         cell.style.backgroundColor = 'rgba(0, 123, 255, 0.4)';
3397
3398         // Create temporary frame-spanning line object
3399         this.currentLine = new FrameSpanningLineObject({
3400             startFrame: frameNumber,
3401             startColumn: columnIndex,
3402             endFrame: frameNumber,
3403             endColumn: columnIndex,
3404             color: this.settings.color,
3405             lineWidth: this.settings.lineWidth,
3406             arrowEnd: true
3407         });
3408
3409         // Add to layer
3410         this.layerSystem.addObject(this.currentLine);
3411     }
3412
3413     handleCellMouseUp = (e) => {
3414         if (!this.drawing || !this.currentLine) return;
```

```
3414
3415     const cell = e.currentTarget;
3416
3417     // Skip if this is the same cell
3418     if (cell === this.startCell) {
3419         this.finishDrawing();
3420         return;
3421     }
3422
3423     // Find the row and column index
3424     const row = cell.parentElement;
3425     const frameNumber = parseInt(row.getAttribute('data-frame') || row.className.replace('frame-', ''));  
3426     const columnIndex = Array.from(row.children).indexOf(cell) + 1;  
3427
3428     if (isNaN(frameNumber) || frameNumber <= 0) {
3429         this.finishDrawing();
3430         return;
3431     }
3432
3433     // Update the end point of the line
3434     this.currentLine.endFrame = frameNumber;
3435     this.currentLine.endColumn = columnIndex;
3436
3437     // Redraw
3438     this.layerSystem.redrawLayer(this.layerSystem.activeLayerIndex);
3439
3440     // Finish drawing
3441     this.finishDrawing();
3442 }
3443
3444 finishDrawing() {
3445     if (this.drawing) {
3446         // Reset flags
3447         this.drawing = false;
3448
3449         // If the line is too short (same start and end), remove it
3450         if (this.currentLine &&
3451             this.currentLine.startFrame === this.currentLine.endFrame &&
3452             this.currentLine.startColumn === this.currentLine.endColumn) {
3453             this.layerSystem.removeObject(this.currentLine);
3454         }
3455
3456         // Clear current line reference
3457         this.currentLine = null;
3458
3459         // Reset cell highlights
3460         if (this.startCell) {
3461             this.startCell.style.backgroundColor =
3461             this.startCell.dataset.originalBg || '';
```

```
3462         this.startCell = null;
3463     }
3464   }
3465 }
3466 }
3467
3468 // Eraser tool
3469 class EraserTool extends DrawingTool {
3470   constructor(toolSystem) {
3471     super(toolSystem);
3472   }
3473
3474   attachEvents() {
3475     const canvas = this.layerSystem.getActiveLayer().canvas;
3476
3477     canvas.addEventListener('mousedown', this.handleMouseDown);
3478   }
3479
3480   detachEvents() {
3481     const canvas = this.layerSystem.getActiveLayer().canvas;
3482
3483     canvas.removeEventListener('mousedown', this.handleMouseDown);
3484   }
3485
3486   handleMouseDown = (e) => {
3487     const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
3488
3489     // Find object at click position
3490     const hit = this.layerSystem.findObjectAt(coords.x, coords.y);
3491
3492     if (hit) {
3493       // Remove the object
3494       this.layerSystem.removeObject(hit.object, hit.layerIndex);
3495     }
3496   }
3497 }
3498
3499 /**
3500 * INTEGRATION WITH X-SHEET APPLICATION
3501 * Initialize the drawing system, save/load integration, and PDF/print handling
3502 */
3503
3504 // Initialize drawing system when document is loaded
3505 function initDrawingSystem() {
3506   // Setup after the table is rendered
3507   setTimeout(() => {
3508     // Get the X-Sheet table element
3509     const xsheetTable = document.getElementById('xsheet-table');
3510     if (!xsheetTable) {
3511       console.error('X-Sheet table not found');
```

```
3512         return;
3513     }
3514
3515     // Create drawing systems
3516     const drawingLayerSystem = new DrawingLayerSystem(xsheetTable);
3517     const drawingToolSystem = new DrawingToolSystem(drawingLayerSystem);
3518
3519     // Store global reference
3520     window.xsheetDrawing = {
3521         layerSystem: drawingLayerSystem,
3522         toolSystem: drawingToolSystem
3523     };
3524
3525     // Add to status message
3526     const statusElement = document.getElementById('status-message');
3527     if (statusElement) {
3528         statusElement.textContent = 'Drawing tools initialized';
3529     }
3530
3531     // Integrate with X-Sheet save/load system
3532     integrateWithXSheetSaveLoad();
3533
3534     // Integrate with X-Sheet PDF and printing
3535     integrateWithXSheetExport();
3536
3537     // Custom event for when drawing objects change
3538     document.addEventListener('xsheet-redraw', function() {
3539         drawingLayerSystem.redrawAll();
3540     });
3541
3542         console.log('Drawing tools initialized successfully');
3543     }, 500);
3544 }
3545
3546 // Integrate drawing data with X-Sheet save/load system
3547 function integrateWithXSheetSaveLoad() {
3548     // Store original functions
3549     const originalCollectData = window.collectData;
3550     const originalRestoreData = window.restoreData;
3551
3552     // Override collectData to include drawings
3553     window.collectData = function() {
3554         // Call original function to get base data
3555         const data = originalCollectData ? originalCollectData() : {};
3556
3557         // Add drawing data if drawing system is initialized
3558         if (window.xsheetDrawing && window.xsheetDrawing.layerSystem) {
3559             data.drawingLayers = [];
3560
3561             // Collect objects from each layer
```

```
3562 |         window.xsheetDrawing.layerSystem.layers.forEach(layer => {
3563 |             const layerData = {
3564 |                 name: layer.name,
3565 |                 visible: layer.visible,
3566 |                 objects: layer.objects.map(obj => obj.toJSON())
3567 |             };
3568 |
3569 |             data.drawingLayers.push(layerData);
3570 |         });
3571 |
3572 |
3573 |         return data;
3574 |     };
3575 |
3576 |     // Override restoreData to handle drawings
3577 |     window.restoreData = function(data) {
3578 |         // Call original function to restore base data
3579 |         if (originalRestoreData) {
3580 |             originalRestoreData(data);
3581 |         }
3582 |
3583 |         // Restore drawing data if available
3584 |         if (data.drawingLayers && window.xsheetDrawing &&
3585 |             window.xsheetDrawing.layerSystem) {
3586 |             const layerSystem = window.xsheetDrawing.layerSystem;
3587 |
3588 |             // Clear existing layers
3589 |             layerSystem.clearAllLayers();
3590 |
3591 |             // Restore each layer
3592 |             data.drawingLayers.forEach((layerData, index) => {
3593 |                 // Create layer if needed
3594 |                 if (index >= layerSystem.layers.length) {
3595 |                     layerSystem.addLayer(layerData.name);
3596 |                 } else {
3597 |                     layerSystem.layers[index].name = layerData.name;
3598 |                     layerSystem.layers[index].visible = layerData.visible;
3599 |                 }
3600 |
3601 |                 // Restore objects
3602 |                 layerData.objects.forEach(objData => {
3603 |                     const newObj = DrawingObjectFactory.createFromJSON(objData);
3604 |                     if (newObj) {
3605 |                         layerSystem.layers[index].objects.push(newObj);
3606 |                     }
3607 |                 });
3608 |
3609 |             });
3610 |
3611 |             // Redraw
3612 |             layerSystem.redrawAll();
3613 |         }
3614 |     }
3615 |
3616 |     // Set up event listeners
3617 |     window.addEventListener('load', () => {
3618 |         // ...
3619 |     });
3620 |
3621 |     // ...
3622 |
3623 |     // ...
3624 |
3625 |     // ...
3626 |
3627 |     // ...
3628 |
3629 |     // ...
3630 |
3631 |     // ...
3632 |
3633 |     // ...
3634 |
3635 |     // ...
3636 |
3637 |     // ...
3638 |
3639 |     // ...
3640 |
3641 |     // ...
3642 |
3643 |     // ...
3644 |
3645 |     // ...
3646 |
3647 |     // ...
3648 |
3649 |     // ...
3650 |
3651 |     // ...
3652 |
3653 |     // ...
3654 |
3655 |     // ...
3656 |
3657 |     // ...
3658 |
3659 |     // ...
3660 |
3661 |     // ...
3662 |
3663 |     // ...
3664 |
3665 |     // ...
3666 |
3667 |     // ...
3668 |
3669 |     // ...
3670 |
3671 |     // ...
3672 |
3673 |     // ...
3674 |
3675 |     // ...
3676 |
3677 |     // ...
3678 |
3679 |     // ...
3680 |
3681 |     // ...
3682 |
3683 |     // ...
3684 |
3685 |     // ...
3686 |
3687 |     // ...
3688 |
3689 |     // ...
3690 |
3691 |     // ...
3692 |
3693 |     // ...
3694 |
3695 |     // ...
3696 |
3697 |     // ...
3698 |
3699 |     // ...
3700 |
3701 |     // ...
3702 |
3703 |     // ...
3704 |
3705 |     // ...
3706 |
3707 |     // ...
3708 |
3709 |     // ...
3710 |
3711 |     // ...
3712 |
3713 |     // ...
3714 |
3715 |     // ...
3716 |
3717 |     // ...
3718 |
3719 |     // ...
3720 |
3721 |     // ...
3722 |
3723 |     // ...
3724 |
3725 |     // ...
3726 |
3727 |     // ...
3728 |
3729 |     // ...
3730 |
3731 |     // ...
3732 |
3733 |     // ...
3734 |
3735 |     // ...
3736 |
3737 |     // ...
3738 |
3739 |     // ...
3740 |
3741 |     // ...
3742 |
3743 |     // ...
3744 |
3745 |     // ...
3746 |
3747 |     // ...
3748 |
3749 |     // ...
3750 |
3751 |     // ...
3752 |
3753 |     // ...
3754 |
3755 |     // ...
3756 |
3757 |     // ...
3758 |
3759 |     // ...
3760 |
3761 |     // ...
3762 |
3763 |     // ...
3764 |
3765 |     // ...
3766 |
3767 |     // ...
3768 |
3769 |     // ...
3770 |
3771 |     // ...
3772 |
3773 |     // ...
3774 |
3775 |     // ...
3776 |
3777 |     // ...
3778 |
3779 |     // ...
3780 |
3781 |     // ...
3782 |
3783 |     // ...
3784 |
3785 |     // ...
3786 |
3787 |     // ...
3788 |
3789 |     // ...
3790 |
3791 |     // ...
3792 |
3793 |     // ...
3794 |
3795 |     // ...
3796 |
3797 |     // ...
3798 |
3799 |     // ...
3800 |
3801 |     // ...
3802 |
3803 |     // ...
3804 |
3805 |     // ...
3806 |
3807 |     // ...
3808 |
3809 |     // ...
3810 |
3811 |     // ...
3812 |
3813 |     // ...
3814 |
3815 |     // ...
3816 |
3817 |     // ...
3818 |
3819 |     // ...
3820 |
3821 |     // ...
3822 |
3823 |     // ...
3824 |
3825 |     // ...
3826 |
3827 |     // ...
3828 |
3829 |     // ...
3830 |
3831 |     // ...
3832 |
3833 |     // ...
3834 |
3835 |     // ...
3836 |
3837 |     // ...
3838 |
3839 |     // ...
3840 |
3841 |     // ...
3842 |
3843 |     // ...
3844 |
3845 |     // ...
3846 |
3847 |     // ...
3848 |
3849 |     // ...
3850 |
3851 |     // ...
3852 |
3853 |     // ...
3854 |
3855 |     // ...
3856 |
3857 |     // ...
3858 |
3859 |     // ...
3860 |
3861 |     // ...
3862 |
3863 |     // ...
3864 |
3865 |     // ...
3866 |
3867 |     // ...
3868 |
3869 |     // ...
3870 |
3871 |     // ...
3872 |
3873 |     // ...
3874 |
3875 |     // ...
3876 |
3877 |     // ...
3878 |
3879 |     // ...
3880 |
3881 |     // ...
3882 |
3883 |     // ...
3884 |
3885 |     // ...
3886 |
3887 |     // ...
3888 |
3889 |     // ...
3890 |
3891 |     // ...
3892 |
3893 |     // ...
3894 |
3895 |     // ...
3896 |
3897 |     // ...
3898 |
3899 |     // ...
3900 |
3901 |     // ...
3902 |
3903 |     // ...
3904 |
3905 |     // ...
3906 |
3907 |     // ...
3908 |
3909 |     // ...
3910 |
3911 |     // ...
3912 |
3913 |     // ...
3914 |
3915 |     // ...
3916 |
3917 |     // ...
3918 |
3919 |     // ...
3920 |
3921 |     // ...
3922 |
3923 |     // ...
3924 |
3925 |     // ...
3926 |
3927 |     // ...
3928 |
3929 |     // ...
3930 |
3931 |     // ...
3932 |
3933 |     // ...
3934 |
3935 |     // ...
3936 |
3937 |     // ...
3938 |
3939 |     // ...
3940 |
3941 |     // ...
3942 |
3943 |     // ...
3944 |
3945 |     // ...
3946 |
3947 |     // ...
3948 |
3949 |     // ...
3950 |
3951 |     // ...
3952 |
3953 |     // ...
3954 |
3955 |     // ...
3956 |
3957 |     // ...
3958 |
3959 |     // ...
3960 |
3961 |     // ...
3962 |
3963 |     // ...
3964 |
3965 |     // ...
3966 |
3967 |     // ...
3968 |
3969 |     // ...
3970 |
3971 |     // ...
3972 |
3973 |     // ...
3974 |
3975 |     // ...
3976 |
3977 |     // ...
3978 |
3979 |     // ...
3980 |
3981 |     // ...
3982 |
3983 |     // ...
3984 |
3985 |     // ...
3986 |
3987 |     // ...
3988 |
3989 |     // ...
3990 |
3991 |     // ...
3992 |
3993 |     // ...
3994 |
3995 |     // ...
3996 |
3997 |     // ...
3998 |
3999 |     // ...
3999 | }
```

```
3611         }
3612     };
3613 }
3614
3615 // Integrate drawing layers with PDF export and printing
3616 function integrateWithXSheetExport() {
3617     // Store original functions
3618     const originalExportToPDF = window.exportToPDF;
3619     const originalPrintSheet = window.printSheet;
3620
3621     // Override PDF export to include drawings
3622     window.exportToPDF = function () {
3623         if (!window.xsheetDrawing || !window.xsheetDrawing.layerSystem) {
3624             return originalExportToPDF ? originalExportToPDF() : null;
3625         }
3626
3627         // Save current state
3628         const layerSystem = window.xsheetDrawing.layerSystem;
3629         const layerContainer = layerSystem.container;
3630
3631         // 1) Save original CSS & DOM properties
3632         const originalDisplay = layerContainer.style.display;
3633         const originalPos = layerContainer.style.position;
3634         const originalTop = layerContainer.style.top;
3635         const originalLeft = layerContainer.style.left;
3636         const originalZIndex = layerContainer.style.zIndex;
3637         const originalWidth = layerContainer.style.width;
3638         const originalHeight = layerContainer.style.height;
3639         const originalParent = layerContainer.parentNode;
3640
3641         // 2) Make it visible for capture
3642         layerContainer.style.display = 'block';
3643
3644         // 3) Compute offsets - FIXED POSITIONING LOGIC
3645         const printableArea = document.getElementById('printable-area');
3646         const printableRect = printableArea.getBoundingClientRect();
3647         const tableRect = document.getElementById('xsheet-
3648 table').getBoundingClientRect();
3649
3650         // 4) Reparent into the printable area
3651         printableArea.appendChild(layerContainer);
3652
3653         // 5) Force absolute positioning & explicit size - FIXED POSITIONING LOGIC
3654         layerContainer.style.position = 'absolute';
3655         layerContainer.style.top = `${tableRect.top - printableRect.top}px`;
3656         layerContainer.style.left = `${tableRect.left - printableRect.left}px`;
3657         layerContainer.style.width = `${tableRect.width}px`;
3658         layerContainer.style.height = `${tableRect.height}px`;
3659         layerContainer.style.zIndex = '1000';
```

```
3660 // Force a redraw of all drawing objects
3661 layerSystem.redrawAll();
3662
3663 // 6) Wait a brief moment to ensure drawings are rendered
3664 setTimeout(() => {
3665     // 7) Generate the PDF
3666     const result = originalExportToPDF ? originalExportToPDF() : null;
3667
3668     // 8) Restore everything
3669     layerContainer.style.display = originalDisplay;
3670     layerContainer.style.position = originalPos;
3671     layerContainer.style.top = originalTop;
3672     layerContainer.style.left = originalLeft;
3673     layerContainer.style.zIndex = originalZIndex;
3674     layerContainer.style.width = originalWidth;
3675     layerContainer.style.height = originalHeight;
3676     originalParent.appendChild(layerContainer);
3677 }, 50);
3678
3679     return true; // Will be async due to setTimeout
3680 };
3681
3682 // Override print to include drawings
3683 window.printSheet = function () {
3684     // 1) Fallback if no drawing system
3685     if (!window.xsheetDrawing || !window.xsheetDrawing.layerSystem) {
3686         return originalPrintSheet ? originalPrintSheet() : null;
3687     }
3688
3689     // 2) Grab the container and save its original state
3690     const layerSystem = window.xsheetDrawing.layerSystem;
3691     const layerContainer = layerSystem.container;
3692     const originalDisplay = layerContainer.style.display;
3693     const originalPos = layerContainer.style.position;
3694     const originalTop = layerContainer.style.top;
3695     const originalLeft = layerContainer.style.left;
3696     const originalZIndex = layerContainer.style.zIndex;
3697     const originalWidth = layerContainer.style.width;
3698     const originalHeight = layerContainer.style.height;
3699     const originalParent = layerContainer.parentNode;
3700
3701     // 3) Make it visible for print capture
3702     layerContainer.style.display = 'block';
3703
3704     // 4) Compute offsets relative to the sheet - FIXED POSITIONING LOGIC
3705     const printableArea = document.getElementById('printable-area');
3706     const printableRect = printableArea.getBoundingClientRect();
3707     const tableRect = document.getElementById('xsheet-
table').getBoundingClientRect();
3708 }
```

```
3709         // 5) Reparent into the printable-area element
3710         printableArea.appendChild(layerContainer);
3711
3712         // 6) Force absolute positioning & lock size - FIXED POSITIONING LOGIC
3713         layerContainer.style.position = 'absolute';
3714         layerContainer.style.top = `${tableRect.top - printableRect.top}px`;
3715         layerContainer.style.left = `${tableRect.left - printableRect.left}px`;
3716         layerContainer.style.width = `${tableRect.width}px`;
3717         layerContainer.style.height = `${tableRect.height}px`;
3718         layerContainer.style.zIndex = '1000';
3719
3720         // 7) Force a redraw of all drawing objects
3721         layerSystem.redrawAll();
3722
3723         // 8) Add a print class to the container for print-specific CSS
3724         layerContainer.classList.add('printing');
3725
3726         // 9) Call the original print with a slight delay to ensure drawing render
3727         setTimeout(() => {
3728             const result = originalPrintSheet ? originalPrintSheet() :
3729             window.print();
3730
3731             // 10) Restore everything back
3732             layerContainer.style.display = originalDisplay;
3733             layerContainer.style.position = originalPos;
3734             layerContainer.style.top = originalTop;
3735             layerContainer.style.left = originalLeft;
3736             layerContainer.style.zIndex = originalZIndex;
3737             layerContainer.style.width = originalWidth;
3738             layerContainer.style.height = originalHeight;
3739             layerContainer.classList.remove('printing');
3740             originalParent.appendChild(layerContainer);
3741         }, 50);
3742
3743         return true; // Will be async due to setTimeout
3744     };
3745
3746
3747     // Setup event listeners for table updates
3748     function setupXSheetUpdateHandling() {
3749         // Hook into existing update functions
3750         if (typeof window.generateTable === 'function') {
3751             const originalGenerateTable = window.generateTable;
3752             window.generateTable = function() {
3753                 // Call original function
3754                 const result = originalGenerateTable.apply(this, arguments);
3755
3756                 // Fire update event
3757                 document.dispatchEvent(new Event('xsheet-updated'));
3758             };
3759         }
3760     }
3761
3762     // Create a new table
3763     const table = document.createElement('table');
3764     table.id = 'xsheet-table';
3765     table.innerHTML = `<thead><tr><th>Column 1</th><th>Column 2</th></tr></thead><tbody><tr><td>Row 1, Col 1</td><td>Row 1, Col 2</td></tr></tbody>`;
3766
3767     // Append table to the page
3768     document.body.appendChild(table);
3769
3770     // Call the update handling function
3771     setupXSheetUpdateHandling();
3772
3773     // Test the update handling
3774     const button = document.createElement('button');
3775     button.textContent = 'Update Table';
3776     button.addEventListener('click', () => {
3777         // Simulate a table update
3778         const newTableContent = `<thead><tr><th>Column 1</th><th>Column 2</th></tr></thead><tbody><tr><td>Row 1, Col 1</td><td>Row 1, Col 2</td></tr><tr><td>Row 2, Col 1</td><td>Row 2, Col 2</td></tr></tbody>`;
3779         table.innerHTML = newTableContent;
3780
3781         // Trigger the update event
3782         document.dispatchEvent(new Event('xsheet-updated'));
3783     });
3784
3785     // Append button to the page
3786     document.body.appendChild(button);
3787
3788     // Test the print functionality
3789     const printButton = document.createElement('button');
3790     printButton.textContent = 'Print';
3791     printButton.addEventListener('click', () => {
3792         print();
3793     });
3794
3795     // Append print button to the page
3796     document.body.appendChild(printButton);
3797
3798     // Test the XSheet API
3799     const xsheet = new XSheet();
3800     xsheet.createTable();
3801
3802     // Append XSheet API button to the page
3803     const xsheetButton = document.createElement('button');
3804     xsheetButton.textContent = 'XSheet API';
3805     xsheetButton.addEventListener('click', () => {
3806         xsheet.print();
3807     });
3808
3809     // Append XSheet API button to the page
3810     document.body.appendChild(xsheetButton);
3811
3812     // Test the XSheet API
3813     const xsheet2 = new XSheet();
3814     xsheet2.createTable();
3815
3816     // Append XSheet API button to the page
3817     const xsheet3 = document.createElement('button');
3818     xsheet3.textContent = 'XSheet API';
3819     xsheet3.addEventListener('click', () => {
3820         xsheet2.print();
3821     });
3822
3823     // Append XSheet API button to the page
3824     document.body.appendChild(xsheet3);
3825
3826     // Test the XSheet API
3827     const xsheet4 = new XSheet();
3828     xsheet4.createTable();
3829
3830     // Append XSheet API button to the page
3831     const xsheet5 = document.createElement('button');
3832     xsheet5.textContent = 'XSheet API';
3833     xsheet5.addEventListener('click', () => {
3834         xsheet4.print();
3835     });
3836
3837     // Append XSheet API button to the page
3838     document.body.appendChild(xsheet5);
3839
3840     // Test the XSheet API
3841     const xsheet6 = new XSheet();
3842     xsheet6.createTable();
3843
3844     // Append XSheet API button to the page
3845     const xsheet7 = document.createElement('button');
3846     xsheet7.textContent = 'XSheet API';
3847     xsheet7.addEventListener('click', () => {
3848         xsheet6.print();
3849     });
3850
3851     // Append XSheet API button to the page
3852     document.body.appendChild(xsheet7);
3853
3854     // Test the XSheet API
3855     const xsheet8 = new XSheet();
3856     xsheet8.createTable();
3857
3858     // Append XSheet API button to the page
3859     const xsheet9 = document.createElement('button');
3860     xsheet9.textContent = 'XSheet API';
3861     xsheet9.addEventListener('click', () => {
3862         xsheet8.print();
3863     });
3864
3865     // Append XSheet API button to the page
3866     document.body.appendChild(xsheet9);
3867
3868     // Test the XSheet API
3869     const xsheet10 = new XSheet();
3870     xsheet10.createTable();
3871
3872     // Append XSheet API button to the page
3873     const xsheet11 = document.createElement('button');
3874     xsheet11.textContent = 'XSheet API';
3875     xsheet11.addEventListener('click', () => {
3876         xsheet10.print();
3877     });
3878
3879     // Append XSheet API button to the page
3880     document.body.appendChild(xsheet11);
3881
3882     // Test the XSheet API
3883     const xsheet12 = new XSheet();
3884     xsheet12.createTable();
3885
3886     // Append XSheet API button to the page
3887     const xsheet13 = document.createElement('button');
3888     xsheet13.textContent = 'XSheet API';
3889     xsheet13.addEventListener('click', () => {
3890         xsheet12.print();
3891     });
3892
3893     // Append XSheet API button to the page
3894     document.body.appendChild(xsheet13);
3895
3896     // Test the XSheet API
3897     const xsheet14 = new XSheet();
3898     xsheet14.createTable();
3899
3900     // Append XSheet API button to the page
3901     const xsheet15 = document.createElement('button');
3902     xsheet15.textContent = 'XSheet API';
3903     xsheet15.addEventListener('click', () => {
3904         xsheet14.print();
3905     });
3906
3907     // Append XSheet API button to the page
3908     document.body.appendChild(xsheet15);
3909
3910     // Test the XSheet API
3911     const xsheet16 = new XSheet();
3912     xsheet16.createTable();
3913
3914     // Append XSheet API button to the page
3915     const xsheet17 = document.createElement('button');
3916     xsheet17.textContent = 'XSheet API';
3917     xsheet17.addEventListener('click', () => {
3918         xsheet16.print();
3919     });
3920
3921     // Append XSheet API button to the page
3922     document.body.appendChild(xsheet17);
3923
3924     // Test the XSheet API
3925     const xsheet18 = new XSheet();
3926     xsheet18.createTable();
3927
3928     // Append XSheet API button to the page
3929     const xsheet19 = document.createElement('button');
3930     xsheet19.textContent = 'XSheet API';
3931     xsheet19.addEventListener('click', () => {
3932         xsheet18.print();
3933     });
3934
3935     // Append XSheet API button to the page
3936     document.body.appendChild(xsheet19);
3937
3938     // Test the XSheet API
3939     const xsheet20 = new XSheet();
3940     xsheet20.createTable();
3941
3942     // Append XSheet API button to the page
3943     const xsheet21 = document.createElement('button');
3944     xsheet21.textContent = 'XSheet API';
3945     xsheet21.addEventListener('click', () => {
3946         xsheet20.print();
3947     });
3948
3949     // Append XSheet API button to the page
3950     document.body.appendChild(xsheet21);
3951
3952     // Test the XSheet API
3953     const xsheet22 = new XSheet();
3954     xsheet22.createTable();
3955
3956     // Append XSheet API button to the page
3957     const xsheet23 = document.createElement('button');
3958     xsheet23.textContent = 'XSheet API';
3959     xsheet23.addEventListener('click', () => {
3960         xsheet22.print();
3961     });
3962
3963     // Append XSheet API button to the page
3964     document.body.appendChild(xsheet23);
3965
3966     // Test the XSheet API
3967     const xsheet24 = new XSheet();
3968     xsheet24.createTable();
3969
3970     // Append XSheet API button to the page
3971     const xsheet25 = document.createElement('button');
3972     xsheet25.textContent = 'XSheet API';
3973     xsheet25.addEventListener('click', () => {
3974         xsheet24.print();
3975     });
3976
3977     // Append XSheet API button to the page
3978     document.body.appendChild(xsheet25);
3979
3980     // Test the XSheet API
3981     const xsheet26 = new XSheet();
3982     xsheet26.createTable();
3983
3984     // Append XSheet API button to the page
3985     const xsheet27 = document.createElement('button');
3986     xsheet27.textContent = 'XSheet API';
3987     xsheet27.addEventListener('click', () => {
3988         xsheet26.print();
3989     });
3990
3991     // Append XSheet API button to the page
3992     document.body.appendChild(xsheet27);
3993
3994     // Test the XSheet API
3995     const xsheet28 = new XSheet();
3996     xsheet28.createTable();
3997
3998     // Append XSheet API button to the page
3999     const xsheet30 = document.createElement('button');
4000     xsheet30.textContent = 'XSheet API';
4001     xsheet30.addEventListener('click', () => {
4002         xsheet28.print();
4003     });
4004
4005     // Append XSheet API button to the page
4006     document.body.appendChild(xsheet30);
4007
4008     // Test the XSheet API
4009     const xsheet31 = new XSheet();
4010     xsheet31.createTable();
4011
4012     // Append XSheet API button to the page
4013     const xsheet32 = document.createElement('button');
4014     xsheet32.textContent = 'XSheet API';
4015     xsheet32.addEventListener('click', () => {
4016         xsheet31.print();
4017     });
4018
4019     // Append XSheet API button to the page
4020     document.body.appendChild(xsheet32);
4021
4022     // Test the XSheet API
4023     const xsheet33 = new XSheet();
4024     xsheet33.createTable();
4025
4026     // Append XSheet API button to the page
4027     const xsheet34 = document.createElement('button');
4028     xsheet34.textContent = 'XSheet API';
4029     xsheet34.addEventListener('click', () => {
4030         xsheet33.print();
4031     });
4032
4033     // Append XSheet API button to the page
4034     document.body.appendChild(xsheet34);
4035
4036     // Test the XSheet API
4037     const xsheet35 = new XSheet();
4038     xsheet35.createTable();
4039
4040     // Append XSheet API button to the page
4041     const xsheet36 = document.createElement('button');
4042     xsheet36.textContent = 'XSheet API';
4043     xsheet36.addEventListener('click', () => {
4044         xsheet35.print();
4045     });
4046
4047     // Append XSheet API button to the page
4048     document.body.appendChild(xsheet36);
4049
4050     // Test the XSheet API
4051     const xsheet37 = new XSheet();
4052     xsheet37.createTable();
4053
4054     // Append XSheet API button to the page
4055     const xsheet38 = document.createElement('button');
4056     xsheet38.textContent = 'XSheet API';
4057     xsheet38.addEventListener('click', () => {
4058         xsheet37.print();
4059     });
4060
4061     // Append XSheet API button to the page
4062     document.body.appendChild(xsheet38);
4063
4064     // Test the XSheet API
4065     const xsheet39 = new XSheet();
4066     xsheet39.createTable();
4067
4068     // Append XSheet API button to the page
4069     const xsheet40 = document.createElement('button');
4070     xsheet40.textContent = 'XSheet API';
4071     xsheet40.addEventListener('click', () => {
4072         xsheet39.print();
4073     });
4074
4075     // Append XSheet API button to the page
4076     document.body.appendChild(xsheet40);
4077
4078     // Test the XSheet API
4079     const xsheet41 = new XSheet();
4080     xsheet41.createTable();
4081
4082     // Append XSheet API button to the page
4083     const xsheet42 = document.createElement('button');
4084     xsheet42.textContent = 'XSheet API';
4085     xsheet42.addEventListener('click', () => {
4086         xsheet41.print();
4087     });
4088
4089     // Append XSheet API button to the page
4090     document.body.appendChild(xsheet42);
4091
4092     // Test the XSheet API
4093     const xsheet43 = new XSheet();
4094     xsheet43.createTable();
4095
4096     // Append XSheet API button to the page
4097     const xsheet44 = document.createElement('button');
4098     xsheet44.textContent = 'XSheet API';
4099     xsheet44.addEventListener('click', () => {
4100         xsheet43.print();
4101     });
4102
4103     // Append XSheet API button to the page
4104     document.body.appendChild(xsheet44);
4105
4106     // Test the XSheet API
4107     const xsheet45 = new XSheet();
4108     xsheet45.createTable();
4109
4110     // Append XSheet API button to the page
4111     const xsheet46 = document.createElement('button');
4112     xsheet46.textContent = 'XSheet API';
4113     xsheet46.addEventListener('click', () => {
4114         xsheet45.print();
4115     });
4116
4117     // Append XSheet API button to the page
4118     document.body.appendChild(xsheet46);
4119
4120     // Test the XSheet API
4121     const xsheet47 = new XSheet();
4122     xsheet47.createTable();
4123
4124     // Append XSheet API button to the page
4125     const xsheet48 = document.createElement('button');
4126     xsheet48.textContent = 'XSheet API';
4127     xsheet48.addEventListener('click', () => {
4128         xsheet47.print();
4129     });
4130
4131     // Append XSheet API button to the page
4132     document.body.appendChild(xsheet48);
4133
4134     // Test the XSheet API
4135     const xsheet49 = new XSheet();
4136     xsheet49.createTable();
4137
4138     // Append XSheet API button to the page
4139     const xsheet50 = document.createElement('button');
4140     xsheet50.textContent = 'XSheet API';
4141     xsheet50.addEventListener('click', () => {
4142         xsheet49.print();
4143     });
4144
4145     // Append XSheet API button to the page
4146     document.body.appendChild(xsheet50);
4147
4148     // Test the XSheet API
4149     const xsheet51 = new XSheet();
4150     xsheet51.createTable();
4151
4152     // Append XSheet API button to the page
4153     const xsheet52 = document.createElement('button');
4154     xsheet52.textContent = 'XSheet API';
4155     xsheet52.addEventListener('click', () => {
4156         xsheet51.print();
4157     });
4158
4159     // Append XSheet API button to the page
4160     document.body.appendChild(xsheet52);
4161
4162     // Test the XSheet API
4163     const xsheet53 = new XSheet();
4164     xsheet53.createTable();
4165
4166     // Append XSheet API button to the page
4167     const xsheet54 = document.createElement('button');
4168     xsheet54.textContent = 'XSheet API';
4169     xsheet54.addEventListener('click', () => {
4170         xsheet53.print();
4171     });
4172
4173     // Append XSheet API button to the page
4174     document.body.appendChild(xsheet54);
4175
4176     // Test the XSheet API
4177     const xsheet55 = new XSheet();
4178     xsheet55.createTable();
4179
4180     // Append XSheet API button to the page
4181     const xsheet56 = document.createElement('button');
4182     xsheet56.textContent = 'XSheet API';
4183     xsheet56.addEventListener('click', () => {
4184         xsheet55.print();
4185     });
4186
4187     // Append XSheet API button to the page
4188     document.body.appendChild(xsheet56);
4189
4190     // Test the XSheet API
4191     const xsheet57 = new XSheet();
4192     xsheet57.createTable();
4193
4194     // Append XSheet API button to the page
4195     const xsheet58 = document.createElement('button');
4196     xsheet58.textContent = 'XSheet API';
4197     xsheet58.addEventListener('click', () => {
4198         xsheet57.print();
4199     });
4200
4201     // Append XSheet API button to the page
4202     document.body.appendChild(xsheet58);
4203
4204     // Test the XSheet API
4205     const xsheet59 = new XSheet();
4206     xsheet59.createTable();
4207
4208     // Append XSheet API button to the page
4209     const xsheet60 = document.createElement('button');
4210     xsheet60.textContent = 'XSheet API';
4211     xsheet60.addEventListener('click', () => {
4212         xsheet59.print();
4213     });
4214
4215     // Append XSheet API button to the page
4216     document.body.appendChild(xsheet60);
4217
4218     // Test the XSheet API
4219     const xsheet61 = new XSheet();
4220     xsheet61.createTable();
4221
4222     // Append XSheet API button to the page
4223     const xsheet62 = document.createElement('button');
4224     xsheet62.textContent = 'XSheet API';
4225     xsheet62.addEventListener('click', () => {
4226         xsheet61.print();
4227     });
4228
4229     // Append XSheet API button to the page
4230     document.body.appendChild(xsheet62);
4231
4232     // Test the XSheet API
4233     const xsheet63 = new XSheet();
4234     xsheet63.createTable();
4235
4236     // Append XSheet API button to the page
4237     const xsheet64 = document.createElement('button');
4238     xsheet64.textContent = 'XSheet API';
4239     xsheet64.addEventListener('click', () => {
4240         xsheet63.print();
4241     });
4242
4243     // Append XSheet API button to the page
4244     document.body.appendChild(xsheet64);
4245
4246     // Test the XSheet API
4247     const xsheet65 = new XSheet();
4248     xsheet65.createTable();
4249
4250     // Append XSheet API button to the page
4251     const xsheet66 = document.createElement('button');
4252     xsheet66.textContent = 'XSheet API';
4253     xsheet66.addEventListener('click', () => {
4254         xsheet65.print();
4255     });
4256
4257     // Append XSheet API button to the page
4258     document.body.appendChild(xsheet66);
4259
4260     // Test the XSheet API
4261     const xsheet67 = new XSheet();
4262     xsheet67.createTable();
4263
4264     // Append XSheet API button to the page
4265     const xsheet68 = document.createElement('button');
4266     xsheet68.textContent = 'XSheet API';
4267     xsheet68.addEventListener('click', () => {
4268         xsheet67.print();
4269     });
4270
4271     // Append XSheet API button to the page
4272     document.body.appendChild(xsheet68);
4273
4274     // Test the XSheet API
4275     const xsheet69 = new XSheet();
4276     xsheet69.createTable();
4277
4278     // Append XSheet API button to the page
4279     const xsheet70 = document.createElement('button');
4280     xsheet70.textContent = 'XSheet API';
4281     xsheet70.addEventListener('click', () => {
4282         xsheet69.print();
4283     });
4284
4285     // Append XSheet API button to the page
4286     document.body.appendChild(xsheet70);
4287
4288     // Test the XSheet API
4289     const xsheet71 = new XSheet();
4290     xsheet71.createTable();
4291
4292     // Append XSheet API button to the page
4293     const xsheet72 = document.createElement('button');
4294     xsheet72.textContent = 'XSheet API';
4295     xsheet72.addEventListener('click', () => {
4296         xsheet71.print();
4297     });
4298
4299     // Append XSheet API button to the page
4300     document.body.appendChild(xsheet72);
4301
4302     // Test the XSheet API
4303     const xsheet73 = new XSheet();
4304     xsheet73.createTable();
4305
4306     // Append XSheet API button to the page
4307     const xsheet74 = document.createElement('button');
4308     xsheet74.textContent = 'XSheet API';
4309     xsheet74.addEventListener('click', () => {
4310         xsheet73.print();
4311     });
4312
4313     // Append XSheet API button to the page
4314     document.body.appendChild(xsheet74);
4315
4316     // Test the XSheet API
4317     const xsheet75 = new XSheet();
4318     xsheet75.createTable();
4319
4320     // Append XSheet API button to the page
4321     const xsheet76 = document.createElement('button');
4322     xsheet76.textContent = 'XSheet API';
4323     xsheet76.addEventListener('click', () => {
4324         xsheet75.print();
4325     });
4326
4327     // Append XSheet API button to the page
4328     document.body.appendChild(xsheet76);
4329
4330     // Test the XSheet API
4331     const xsheet77 = new XSheet();
4332     xsheet77.createTable();
4333
4334     // Append XSheet API button to the page
4335     const xsheet78 = document.createElement('button');
4336     xsheet78.textContent = 'XSheet API';
4337     xsheet78.addEventListener('click', () => {
4338         xsheet77.print();
4339     });
4340
4341     // Append XSheet API button to the page
4342     document.body.appendChild(xsheet78);
4343
4344     // Test the XSheet API
4345     const xsheet79 = new XSheet();
4346     xsheet79.createTable();
4347
4348     // Append XSheet API button to the page
4349     const xsheet80 = document.createElement('button');
4350     xsheet80.textContent = 'XSheet API';
4351     xsheet80.addEventListener('click', () => {
4352         xsheet79.print();
4353     });
4354
4355     // Append XSheet API button to the page
4356     document.body.appendChild(xsheet80);
4357
4358     // Test the XSheet API
4359     const xsheet81 = new XSheet();
4360     xsheet81.createTable();
4361
4362     // Append XSheet API button to the page
4363     const xsheet82 = document.createElement('button');
4364     xsheet82.textContent = 'XSheet API';
4365     xsheet82.addEventListener('click', () => {
4366         xsheet81.print();
4367     });
4368
4369     // Append XSheet API button to the page
4370     document.body.appendChild(xsheet82);
4371
4372     // Test the XSheet API
4373     const xsheet83 = new XSheet();
4374     xsheet83.createTable();
4375
4376     // Append XSheet API button to the page
4377     const xsheet84 = document.createElement('button');
4378     xsheet84.textContent = 'XSheet API';
4379     xsheet84.addEventListener('click', () => {
4380         xsheet83.print();
4381     });
4382
4383     // Append XSheet API button to the page
4384     document.body.appendChild(xsheet84);
4385
4386     // Test the XSheet API
4387     const xsheet85 = new XSheet();
4388     xsheet85.createTable();
4389
4390     // Append XSheet API button to the page
4391     const xsheet86 = document.createElement('button');
4392     xsheet86.textContent = 'XSheet API';
4393     xsheet86.addEventListener('click', () => {
4394         xsheet85.print();
4395     });
4396
4397     // Append XSheet API button to the page
4398     document.body.appendChild(xsheet86);
4399
4400     // Test the XSheet API
4401     const xsheet87 = new XSheet();
4402     xsheet87.createTable();
4403
4404     // Append XSheet API button to the page
4405     const xsheet88 = document.createElement('button');
4406     xsheet88.textContent = 'XSheet API';
4407     xsheet88.addEventListener('click', () => {
4408         xsheet87.print();
4409     });
4410
4411     // Append XSheet API button to the page
4412     document.body.appendChild(xsheet88);
4413
4414     // Test the XSheet API
4415     const xsheet89 = new XSheet();
4416     xsheet89.createTable();
4417
4418     // Append XSheet API button to the page
4419     const xsheet90 = document.createElement('button');
4420     xsheet90.textContent = 'XSheet API';
4421     xsheet90.addEventListener('click', () => {
4422         xsheet89.print();
4423     });
4424
4425     // Append XSheet API button to the page
4426     document.body.appendChild(xsheet90);
4427
4428     // Test the XSheet API
4429     const xsheet91 = new XSheet();
4430     xsheet91.createTable();
4431
4432     // Append XSheet API button to the page
4433     const xsheet92 = document.createElement('button');
4434     xsheet92.textContent = 'XSheet API';
4435     xsheet92.addEventListener('click', () => {
4436         xsheet91.print();
4437     });
4438
4439     // Append XSheet API button to the page
4440     document.body.appendChild(xsheet92);
4441
4442     // Test the XSheet API
4443     const xsheet93 = new XSheet();
4444     xsheet93.createTable();
4445
4446     // Append XSheet API button to the page
4447     const xsheet94 = document.createElement('button');
4448     xsheet94.textContent = 'XSheet API';
4449     xsheet94.addEventListener('click', () => {
4450         xsheet93.print();
4451     });
4452
4453     // Append XSheet API button to the page
4454     document.body.appendChild(xsheet94);
4455
4456     // Test the XSheet API
4457     const xsheet95 = new XSheet();
4458     xsheet95.createTable();
4459
4460     // Append XSheet API button to the page
4461     const xsheet96 = document.createElement('button');
4462     xsheet96.textContent = 'XSheet API';
4463     xsheet96.addEventListener('click', () => {
4464         xsheet95.print();
4465     });
4466
4467     // Append XSheet API button to the page
4468     document.body.appendChild(xsheet96);
4469
4470     // Test the XSheet API
4471     const xsheet97 = new XSheet();
4472     xsheet97.createTable();
4473
4474     // Append XSheet API button to the page
4475     const xsheet98 = document.createElement('button');
4476     xsheet98.textContent = 'XSheet API';
4477     xsheet98.addEventListener('click', () => {
4478         xsheet97.print();
4479     });
4480
4481     // Append XSheet API button to the page
4482     document.body.appendChild(xsheet98);
4483
4484     // Test the XSheet API
4485     const xsheet99 = new XSheet();
4486     xsheet99.createTable();
4487
4488     // Append XSheet API button to the page
4489     const xsheet100 = document.createElement('button');
4490     xsheet100.textContent = 'XSheet API';
4491     xsheet100.addEventListener('click', () => {
4492         xsheet99.print();
4493     });
4494
4495     // Append XSheet API button to the page
4496     document.body.appendChild(xsheet100);
4497
4498     // Test the XSheet API
4499     const xsheet101 = new XSheet();
4500     xsheet101.createTable();
4501
4502     // Append XSheet API button to the page
4503     const xsheet102 = document.createElement('button');
4504     xsheet102.textContent = 'XSheet API';
4505     xsheet102.addEventListener('click', () => {
4506         xsheet101.print();
4507     });
4508
4509     // Append XSheet API button to the page
4510     document.body.appendChild(xsheet102);
4511
4512     // Test the XSheet API
4513     const xsheet103 = new XSheet();
4514     xsheet103.createTable();
4515
4516     // Append XSheet API button to the page
4517     const xsheet104 = document.createElement('button');
4518     xsheet104.textContent = 'XSheet API';
4519     xsheet104.addEventListener('click', () => {
4520         xsheet103.print();
4521     });
4522
4523     // Append XSheet API button to the page
4524     document.body.appendChild(xsheet104);
4525
4526     // Test the XSheet API
4527     const xsheet105 = new XSheet();
4528     xsheet105.createTable();
4529
4530     // Append XSheet API button to the page
4531     const xsheet106 = document.createElement('button');
4532     xsheet106.textContent = 'XSheet API';
4533     xsheet106.addEventListener('click', () => {
4534         xsheet105.print();
4535     });
4536
4537     // Append XSheet API button to the page
4538     document.body.appendChild(xsheet106);
4539
4540     // Test the XSheet API
4541     const xsheet107 = new XSheet();
4542     xsheet107.createTable();
4543
4544     // Append XSheet API button to the page
4545     const xsheet108 = document.createElement('button');
4546     xsheet108.textContent = 'XSheet API';
4547     xsheet108.addEventListener('click', () => {
4548         xsheet107.print();
4549     });
4550
4551     // Append XSheet API button to the page
4552     document.body.appendChild(xsheet108);
4553
4554     // Test the XSheet API
4555     const xsheet109 = new XSheet();
4556     xsheet109.createTable();
4557
4558     // Append XSheet API button to the page
4559     const xsheet110 = document.createElement('button');
4560     xsheet110.textContent = 'XSheet API';
4561     xsheet110.addEventListener('click', () => {
4562         xsheet109.print();
4563     });
4564
4565     // Append XSheet API button to the page
4566     document.body.appendChild(xsheet110);
4567
4568     // Test the XSheet API
4569     const xsheet111 = new XSheet();
4570     xsheet111.createTable();
4571
4572     // Append XSheet API button to the page
4573     const xsheet112 = document.createElement('button');
4574     xsheet112.textContent = 'XSheet API';
4575     xsheet112.addEventListener('click', () => {
4576         xsheet111.print();
4577     });
4578
4579     // Append XSheet API button to the page
4580     document.body.appendChild(xsheet112);
4581
4582     // Test the XSheet API
4583     const xsheet113 = new XSheet();
4584     xsheet113.createTable();
4585
4586     // Append XSheet API button to the page
4587     const xsheet114 = document.createElement('button');
4588     xsheet114.textContent = 'XSheet API';
4589     xsheet114.addEventListener('click', () => {
4590         xsheet113.print();
4591     });
4592
4593     // Append XSheet API button to the page
4594     document.body.appendChild(xsheet114);
4595
4596     // Test the XSheet API
4597     const xsheet115 = new XSheet();
4598     xsheet115.createTable();
4599
4600     // Append XSheet API button to the page
4601     const xsheet116 = document.createElement('button');
4602     xsheet116.textContent = 'XSheet API';
4603     xsheet116.addEventListener('click', () => {
4604         xsheet115.print();
4605     });
4606
4607     // Append XSheet API button to the page
4608     document.body.appendChild(xsheet116);
4609
4610     // Test the XSheet API
4611     const xsheet117 = new XSheet();
4612     xsheet117.createTable();
4613
4614     // Append XSheet API button to the page
4615     const xsheet118 = document.createElement('button');
4616     xsheet118.textContent = 'XSheet API';
4617     xsheet118.addEventListener('click', () => {
4618         xsheet117.print();
4619     });
4620
4621     // Append XSheet API button to the page
4622     document.body.appendChild(xsheet118);
4623
4624     // Test the XSheet API
4625     const xsheet119 = new XSheet();
4626     xsheet119.createTable();
4627
4628     // Append XSheet API button to the page
4629     const xsheet120 = document.createElement('button');
4630     xsheet120.textContent = 'XSheet API';
4631     xsheet120.addEventListener('click', () => {
4632         xsheet119.print();
4633     });
4634
4635     // Append XSheet API button to the page
4636     document.body.appendChild(xsheet120);
4637
4638     // Test the XSheet API
4639     const xsheet121 = new XSheet();
4640     xsheet121.createTable();
4641
4642     // Append XSheet API button to the page
4643     const xsheet122 = document.createElement('button');
4644     xsheet122.textContent = 'XSheet API';
4645     xsheet122.addEventListener('click', () => {
4646         xsheet121.print();
4647     });
4648
4649     // Append XSheet API button to the page
4650     document.body.appendChild(xsheet12
```

```
3758         return result;
3759     );
3760 }
3761
3762 if (typeof window.addEightRows === 'function') {
3763     const originalAddEightRows = window.addEightRows;
3764     window.addEightRows = function() {
3765         // Call original function
3766         const result = originalAddEightRows.apply(this, arguments);
3767
3768         // Fire update event
3769         document.dispatchEvent(new Event('xsheet-updated'));
3770
3771         return result;
3772     };
3773 }
3774
3775 if (typeof window.updateTemplate === 'function') {
3776     const originalUpdateTemplate = window.updateTemplate;
3777     window.updateTemplate = function() {
3778         // Call original function
3779         const result = originalUpdateTemplate.apply(this, arguments);
3780
3781         // Fire update event
3782         document.dispatchEvent(new Event('xsheet-updated'));
3783
3784         return result;
3785     };
3786 }
3787
3788 }
3789
3790 // Original X-Sheet code
3791 // Wait for page load and libraries to initialize
3792 document.addEventListener('DOMContentLoaded', function () {
3793     // Global variables
3794     let frameCount = 96;
3795     let projectName = 'Animation_XSheet_' + new Date().toISOString().split('T')[0];
3796     let modified = false;
3797     let currentTemplate = 'large';
3798
3799     // Audio variables
3800     let audioContext = null;
3801     let audioBuffer = null;
3802     let audioSource = null;
3803     let audioData = null;
3804     let isPlaying = false;
3805     let startTime = 0;
3806     let startOffset = 0;
3807     let audioFileName = '';
```

```
3808 let waveformData = [];
3809 let waveformCanvases = [];
3810 let phonetics = [];
3811 let frameDuration = 1 / 24; // 24fps
3812 let currentFrame = 0;
3813 let phoneticEditPosition = null;
3814
3815 // Initialize UI elements
3816 const templateSelector = document.getElementById('template-selector');
3817 const frameCountInput = document.getElementById('frame-count');
3818 const saveButton = document.getElementById('save-button');
3819 const loadButton = document.getElementById('load-button');
3820 const pdfButton = document.getElementById('pdf-button');
3821 const printButton = document.getElementById('print-button');
3822 const addRowsButton = document.getElementById('add-rows-button');
3823 const clearButton = document.getElementById('clear-button');
3824 const tableBody = document.getElementById('xsheet-body');
3825 const statusMessage = document.getElementById('status-message');
3826 const audioButton = document.getElementById('audio-button');
3827 const audioUpload = document.getElementById('audio-upload');
3828 const playAudioButton = document.getElementById('play-audio');
3829 const stopAudioButton = document.getElementById('stop-audio');
3830 const audioScrubber = document.getElementById('audio-scrubber');
3831 const audioInfo = document.getElementById('audio-info');
3832 const addPhoneticButton = document.getElementById('add-phonetic');
3833 const phoneticInput = document.getElementById('phonetic-input');
3834 const phoneticText = document.getElementById('phonetic-text');
3835 const savePhoneticButton = document.getElementById('save-phonetic');
3836 const cancelPhoneticButton = document.getElementById('cancel-phonetic');
3837
3838 // Date field - set today by default
3839 const dateField = document.getElementById('project-date');
3840 dateField.valueAsDate = new Date();
3841
3842 // Initialize the table
3843 generateTable(frameCount);
3844
3845 // Initialize the drawing system
3846 initDrawingSystem();
3847 setupXSheetUpdateHandling();
3848
3849 // Event Listeners
3850 templateSelector.addEventListener('change', function () {
3851     currentTemplate = this.value;
3852     updateTemplate();
3853 });
3854
3855 frameCountInput.addEventListener('change', function () {
3856     frameCount = parseInt(this.value);
3857     if (frameCount < 8) frameCount = 8;
```

```
3858     generateTable(frameCount);
3859     updateStatusMessage('Frame count updated to ' + frameCount);
3860
3861     // Re-render waveform if audio is loaded
3862     if (audioBuffer) {
3863         renderWaveform();
3864     }
3865 });
3866
3867 saveButton.addEventListener('click', saveProject);
3868 loadButton.addEventListener('click', loadProject);
3869 pdfButton.addEventListener('click', exportToPDF);
3870 printButton.addEventListener('click', printSheet);
3871 addRowsButton.addEventListener('click', addEightRows);
3872 clearButton.addEventListener('click', clearSheet);
3873
3874 // Audio related event listeners
3875 audioButton.addEventListener('click', function () {
3876     audioUpload.click();
3877 });
3878
3879 audioUpload.addEventListener('change', function (e) {
3880     if (e.target.files.length > 0) {
3881         const file = e.target.files[0];
3882         audioFileName = file.name;
3883         loadAudioFile(file);
3884     }
3885 });
3886
3887 playAudioButton.addEventListener('click', togglePlayAudio);
3888 stopAudioButton.addEventListener('click', stopAudio);
3889 audioScrubber.addEventListener('input', scrubAudio);
3890
3891 addPhoneticButton.addEventListener('click', function () {
3892     if (audioBuffer) {
3893         showPhoneticInput(null);
3894     } else {
3895         updateStatusMessage('Please load audio first');
3896     }
3897 });
3898
3899 savePhoneticButton.addEventListener('click', savePhoneticMarker);
3900 cancelPhoneticButton.addEventListener('click', function () {
3901     phoneticInput.style.display = 'none';
3902 });
3903
3904 // Monitor for changes to set modified flag
3905 document.addEventListener('input', function (e) {
3906     if (!e.target.matches('input, [contenteditable="true"]')) return;
3907     modified = true;
```

```
3908
3909     if (e.target.matches('[contenteditable="true"]')) {
3910         // Add the modified class when content is added
3911         e.target.classList.add('modified');
3912
3913         // Remove the modified class when the cell is empty
3914         if (e.target.textContent.trim() === '') {
3915             e.target.classList.remove('modified');
3916         }
3917     }
3918
3919     updateStatusMessage('Changes detected - not saved');
3920 });
3921
3922 // Variables for cell selection
3923 let selectedCells = [];
3924 let isSelecting = false;
3925 let selectionStart = null;
3926 let hasMovedDuringSelection = false;
3927
3928 // Set up multi-cell selection functionality
3929 function setupCellSelection() {
3930     const editableCells = document.querySelectorAll('[contenteditable="true"]');
3931
3932     // Clear selected cells when clicking outside
3933     document.addEventListener('click', function (e) {
3934         // If this was part of a drag operation, skip the click handler
3935         if (hasMovedDuringSelection) {
3936             // We removed this line to fix the selection issue
3937             // hasMovedDuringSelection = false;
3938             return;
3939         }
3940
3941         // Get the clicked element and check if it's inside an editable cell or
3942         // a selected cell
3943         const clickedCell = e.target.closest('[contenteditable="true"]');
3944         const clickedInSelection = e.target.closest('.selected-cell');
3945
3946         // If not clicking in a cell or selection and not using modifier keys
3947         if (!clickedCell && !clickedInSelection && !e.ctrlKey && !e.metaKey) {
3948             clearCellSelection();
3949         }
3950     });
3951
3952     // Handle clicks on the table for better cell focusing
3953     document.getElementById('xsheet-table').addEventListener('click', function
3954     (e) {
3955         // Find the closest editable cell from the click point
3956         const targetCell = e.target.closest('[contenteditable="true"]');
```

```
3956 // If we found a cell and we're not in multi-select mode
3957 if (targetCell && !hasMovedDuringSelection && !e.ctrlKey && !e.metaKey)
3958 {
3959     // Focus the cell and position cursor at the end
3960     if (selectedCells.length === 1 && selectedCells[0] === targetCell) {
3961         // If already selected, just ensure focus
3962         targetCell.focus();
3963         placeCaretAtEnd(targetCell);
3964     }
3965 });
3966
3967 // Helper function to place caret at the end of content
3968 function placeCaretAtEnd(el) {
3969     if (document.createRange) {
3970         const range = document.createRange();
3971         range.selectNodeContents(el);
3972         range.collapse(false); // false means collapse to end
3973         const selection = window.getSelection();
3974         selection.removeAllRanges();
3975         selection.addRange(range);
3976     }
3977 }
3978
3979 // Handle mousedown for selection start
3980 editableCells.forEach(cell => {
3981     // Mousedown - start potential selection
3982     cell.addEventListener('mousedown', function (e) {
3983         // Only handle left mouse button
3984         if (e.button !== 0) return;
3985
3986         // If using modifier keys for multi-select
3987         if (e.ctrlKey || e.metaKey) {
3988             toggleCellSelection(cell);
3989             e.preventDefault(); // Prevent text cursor
3990             return;
3991         }
3992
3993         // Clear previous selection unless clicking on already selected cell
3994         if (!cell.classList.contains('selected-cell')) {
3995             clearCellSelection();
3996         }
3997
3998         // Store starting cell and reset movement flag
3999         selectionStart = cell;
4000         hasMovedDuringSelection = false;
4001         isSelecting = true;
4002
4003         // Initially add this cell to selection
4004         if (!cell.classList.contains('selected-cell')) {
```

```
4005         toggleCellSelection(cell, true);
4006     }
4007
4008     // If it's a single click (not start of drag), focus the cell
4009     cell.focus();
4010
4011     // Don't prevent default here to allow normal focus behavior
4012   });
4013
4014     // Additional keyboard events for selected cells
4015   cell.addEventListener('keydown', function (e) {
4016     // If Delete or Backspace key and we have selected cells
4017     if ((e.key === 'Delete' || e.key === 'Backspace') &&
4018       selectedCells.length > 1) {
4019       e.preventDefault();
4020       clearSelectedCellsContent();
4021     }
4022
4023     // Copy with Ctrl+C or Cmd+C
4024     if ((e.ctrlKey || e.metaKey) && e.key === 'c' &&
4025       selectedCells.length > 0) {
4026       e.preventDefault();
4027       copySelectedCells();
4028     }
4029
4030     // Select all cells in row with Ctrl+A or Cmd+A
4031     if ((e.ctrlKey || e.metaKey) && e.key === 'a') {
4032       e.preventDefault();
4033       selectRowCells(cell.parentElement);
4034     }
4035   });
4036
4037   // Global mousemove handler for extending selection
4038   document.addEventListener('mousemove', function (e) {
4039     if (!isSelecting) return;
4040
4041     // Set the flag that we've moved during this selection
4042     hasMovedDuringSelection = true;
4043
4044     // Find the element under the cursor
4045     const elementUnderCursor = document.elementFromPoint(e.clientX,
4046     e.clientY);
4047     if (!elementUnderCursor) return;
4048
4049     // Find the closest editable cell
4050     const targetCell =
4051       elementUnderCursor.closest('[contenteditable="true"]');
4052     if (targetCell) {
4053       // Add to selection
4054       toggleCellSelection(targetCell, true);
4055     }
4056   });
4057 }
```

```
4052
4053          // Prevent text selection during drag
4054          e.preventDefault();
4055          if (window.getSelection) {
4056              window.getSelection().removeAllRanges();
4057          }
4058      });
4059
4060
4061 // Global mouseup to end selection - THIS IS THE FIXED VERSION
4062 document.addEventListener('mouseup', function (e) {
4063     // End selection mode but keep selected cells
4064     if (isSelecting) {
4065         isSelecting = false;
4066
4067         // FIX: Use setTimeout to delay resetting the hasMovedDuringSelection flag
4068         // This gives the click handler a chance to see the flag first
4069         if (hasMovedDuringSelection) {
4070             setTimeout(function () {
4071                 hasMovedDuringSelection = false;
4072             }, 10);
4073         }
4074
4075         // If this wasn't a drag and there's only one cell selected, focus it
4076         if (!hasMovedDuringSelection && selectedCells.length === 1) {
4077             const cell = selectedCells[0];
4078             cell.focus();
4079             placeCaretAtEnd(cell);
4080         }
4081     }
4082
4083     selectionStart = null;
4084 });
4085
4086 // Listen for paste events
4087 document.addEventListener('paste', function (e) {
4088     if (selectedCells.length > 0) {
4089         handlePaste(e);
4090     }
4091 });
4092
4093
4094 function toggleCellSelection(cell, addOnly = false) {
4095     const index = selectedCells.indexOf(cell);
4096
4097     if (index === -1) {
4098         // Add to selection
4099         selectedCells.push(cell);
```

```
4100         cell.classList.add('selected-cell');
4101     } else if (!addOnly) {
4102         // Remove from selection if not add-only mode
4103         selectedCells.splice(index, 1);
4104         cell.classList.remove('selected-cell');
4105     }
4106 }
4107
4108 function clearCellSelection() {
4109     selectedCells.forEach(cell => {
4110         cell.classList.remove('selected-cell');
4111     });
4112     selectedCells = [];
4113 }
4114
4115 function clearSelectedCellsContent() {
4116     selectedCells.forEach(cell => {
4117         cell.textContent = '';
4118         cell.classList.remove('modified');
4119     });
4120     modified = true;
4121     updateStatusMessage('Cleared selected cells');
4122 }
4123
4124 function copySelectedCells() {
4125     if (selectedCells.length === 0) return;
4126
4127     // Create a text representation of selected cells
4128     const cellData = selectedCells.map(cell => cell.textContent || ''
4129         ).join('\t');
4130
4131     // Copy to clipboard using Clipboard API
4132     navigator.clipboard.writeText(cellData)
4133         .then(() => {
4134             updateStatusMessage('Copied selected cells to clipboard');
4135         })
4136         .catch(err => {
4137             updateStatusMessage('Failed to copy: ' + err);
4138         });
4139
4140     function handlePaste(e) {
4141         // Prevent default paste behavior
4142         e.preventDefault();
4143
4144         // Get clipboard data
4145         const clipboardData = e.clipboardData || window.clipboardData;
4146         const pastedText = clipboardData.getData('text');
4147
4148         // If we have a focused element within selection, paste there
```

```
4149     const activeElement = document.activeElement;
4150     if (activeElement && selectedCells.includes(activeElement)) {
4151         activeElement.textContent = pastedText;
4152         activeElement.classList.add('modified');
4153         modified = true;
4154     } else if (selectedCells.length > 0) {
4155         // Otherwise paste into first selected cell
4156         selectedCells[0].textContent = pastedText;
4157         selectedCells[0].classList.add('modified');
4158         modified = true;
4159     }
4160
4161     if (pastedText.trim() === '') {
4162         selectedCells.forEach(cell => {
4163             cell.classList.remove('modified');
4164         });
4165     }
4166
4167     updateStatusMessage('Pasted content into selected cell');
4168 }
4169
4170 function selectRowCells(row) {
4171     // Clear previous selection
4172     clearCellSelection();
4173
4174     // Select all editable cells in the row
4175     const cells = Array.from(row.cells).filter(cell => cell.contentEditable ===
4176 'true');
4177     cells.forEach(cell => {
4178         toggleCellSelection(cell, true);
4179     });
4180
4181     // Functions
4182     function generateTable(frames) {
4183         tableBody.innerHTML = '';
4184         waveformCanvases = [];
4185
4186         // Create a column container for the waveform
4187         const waveformColContainer = document.createElement('div');
4188         waveformColContainer.className = 'waveform-col-container';
4189
4190         for (let i = 1; i <= frames; i++) {
4191             const row = document.createElement('tr');
4192             row.className = `frame-${i}`;
4193             row.setAttribute('data-frame', i);
4194
4195             // Create standard cell
4196             function createCell(className, isEditable = true, frameNum = null) {
4197                 const cell = document.createElement('td');
```

```
4198     cell.className = className;
4199
4200     if (isEditable) {
4201         cell.contentEditable = true;
4202         cell.setAttribute('data-placeholder', '');
4203         cell.setAttribute('tabindex', '0');
4204     } else if (frameNum !== null) {
4205         cell.textContent = frameNum;
4206         cell.className += ' frame-number';
4207
4208         // Color the first frame green for reference (like in the
4209         // example image)
4210         if (frameNum === 1) {
4211             cell.style.backgroundColor = '#00cc00';
4212         } else {
4213             cell.style.backgroundColor = '#cccccc';
4214         }
4215
4216         return cell;
4217     }
4218
4219     // Create waveform cell (placeholder for the vertical waveform)
4220     function createWaveformCell(frameNum) {
4221         const cell = document.createElement('td');
4222         cell.className = 'waveform-col';
4223         cell.setAttribute('data-frame', frameNum);
4224         return cell;
4225     }
4226
4227     // Add all cells to the row
4228     row.appendChild(createCell('action-col'));
4229     row.appendChild(createCell('frame-col', false, i));
4230     row.appendChild(createWaveformCell(i));
4231     row.appendChild(createCell('dialogue-col'));
4232     row.appendChild(createCell('sound-col'));
4233     row.appendChild(createCell('technical-col'));
4234     row.appendChild(createCell('extra1-col'));
4235     row.appendChild(createCell('extra2-col'));
4236     row.appendChild(createCell('frame-col', false, i));
4237     row.appendChild(createCell('camera-col'));
4238
4239     // Add special styling for 8-frame and 24-frame intervals
4240     if (i % 24 === 0) {
4241         row.style.borderBottom = '4px double #000';
4242         for (let cell of row.cells) {
4243             cell.style.borderBottom = '4px double #000';
4244             cell.style.fontWeight = 'bold';
4245         }
4246     } else if (i % 8 === 0) {
```

```
4247         row.style.borderBottom = '2px solid #000';
4248         for (let cell of row.cells) {
4249             cell.style.borderBottom = '2px solid #000';
4250             cell.style.fontWeight = 'bold';
4251         }
4252     }
4253
4254     tableBody.appendChild(row);
4255 }
4256
4257 setupCellNavigation();
4258 updateStatusMessage('Table generated with ' + frames + ' frames');
4259
4260 // Fire custom event for drawing system to update
4261 document.dispatchEvent(new Event('xsheet-updated'));
4262 }
4263
4264 function updateTemplate() {
4265     if (currentTemplate === 'large') {
4266         // 11"x17" template (96 frames)
4267         document.body.style.maxWidth = '11in';
4268         document.body.style.maxHeight = '17in';
4269         document.body.style.fontSize = '9pt';
4270         frameCountInput.value = 96;
4271         frameCount = 96;
4272     } else {
4273         // 8"x10" template (48 frames)
4274         document.body.style.maxWidth = '8in';
4275         document.body.style.maxHeight = '10in';
4276         document.body.style.fontSize = '8pt';
4277         frameCountInput.value = 48;
4278         frameCount = 48;
4279     }
4280
4281     generateTable(frameCount);
4282     updateStatusMessage('Template switched to ' + (currentTemplate === 'large' ?
4283 '11"x17"' : '8"x10"'));
4284
4285     // Re-render waveform if audio is loaded
4286     if (audioBuffer) {
4287         renderWaveform();
4288     }
4289
4290     function setupCellNavigation() {
4291         const editableCells = document.querySelectorAll('[contenteditable="true"]');
4292
4293         editableCells.forEach(cell => {
4294             cell.addEventListener('keydown', function (e) {
4295                 // Tab navigation
```

```
4296 if (e.key === 'Tab') {
4297     e.preventDefault();
4298     const currentRow = this.parentElement;
4299     const currentIndex = Array.from(currentRow.cells).indexOf(this);
4300
4301     if (e.shiftKey) {
4302         // Shift+Tab moves backward
4303         let prevCell = null;
4304         if (currentIndex > 0) {
4305             // Find previous editable cell in same row
4306             for (let i = currentIndex - 1; i >= 0; i--) {
4307                 if (currentRow.cells[i].contentEditable === 'true')
4308                     prevCell = currentRow.cells[i];
4309                 break;
4310             }
4311         }
4312     }
4313
4314     if (!prevCell) {
4315         // Move to previous row, last cell
4316         const prevRow = currentRow.previousElementSibling;
4317         if (prevRow) {
4318             const cells = Array.from(prevRow.cells).filter(c =>
c.contentEditable === 'true');
4319             prevCell = cells[cells.length - 1];
4320         }
4321     }
4322
4323     if (prevCell) {
4324         prevCell.focus();
4325         // Place cursor at end of content
4326         const range = document.createRange();
4327         const sel = window.getSelection();
4328         range.selectNodeContents(prevCell);
4329         range.collapse(false);
4330         sel.removeAllRanges();
4331         sel.addRange(range);
4332     }
4333 } else {
4334     // Tab moves forward
4335     let nextCell = null;
4336     if (currentIndex < currentRow.cells.length - 1) {
4337         // Find next editable cell in same row
4338         for (let i = currentIndex + 1; i <
currentRow.cells.length; i++)
4339             if (currentRow.cells[i].contentEditable === 'true')
4340                 nextCell = currentRow.cells[i];
4341             break;
4342 }
```

```
4343         }
4344     }
4345
4346     if (!nextCell) {
4347         // Move to next row, first cell
4348         const nextRow = currentRow.nextElementSibling;
4349         if (nextRow) {
4350             nextCell = Array.from(nextRow.cells).find(c =>
4351             c.contentEditable === 'true');
4352         }
4353     }
4354     if (nextCell) {
4355         nextCell.focus();
4356     }
4357 }
4358 }
4359
4360 // Enter key to move down
4361 if (e.key === 'Enter' && !e.shiftKey) {
4362     e.preventDefault();
4363     const currentRow = this.parentElement;
4364     const nextRow = currentRow.nextElementSibling;
4365     const currentIndex = Array.from(currentRow.cells).indexOf(this);
4366
4367     if (nextRow) {
4368         const cells = Array.from(nextRow.cells);
4369         const samePositionCell = cells[currentIndex];
4370         if (samePositionCell && samePositionCell.contentEditable ===
4371             'true') {
4372             samePositionCell.focus();
4373         }
4374     }
4375 });
4376
4377 // Add listener to remove the modified class when cell is emptied
4378 cell.addEventListener('keyup', function (e) {
4379     if ((e.key === 'Delete' || e.key === 'Backspace') &&
4380         this.textContent.trim() === '') {
4381         this.classList.remove('modified');
4382     }
4383 });
4384
4385 // Setup cell selection functionality
4386 setupCellSelection();
4387 }
4388
4389 // Audio functions
4390 function loadAudioFile(file) {
```

```
4391     if (!audioContext) {
4392         try {
4393             window.AudioContext = window.AudioContext || 
4394             window.webkitAudioContext;
4395             audioContext = new AudioContext();
4396         } catch (e) {
4397             updateStatusMessage('Web Audio API is not supported in this
4398             browser');
4399         }
4400     }
4401
4402     const reader = new FileReader();
4403
4404     reader.onload = function (e) {
4405         const audioData = e.target.result;
4406
4407         // Update status
4408         updateStatusMessage('Decoding audio...');

4409         // Decode the audio data
4410         audioContext.decodeAudioData(audioData, function (buffer) {
4411             audioBuffer = buffer;

4412             // Update audio info
4413             const duration = buffer.duration;
4414             const minutes = Math.floor(duration / 60);
4415             const seconds = Math.floor(duration % 60);
4416             const frameCount = Math.ceil(duration * 24); // Assuming 24fps

4417             audioInfo.textContent = `${audioFileName}
4418             (${minutes}:${seconds.toString().padStart(2, '0')}, ${frameCount} frames @ 24fps)`;

4419
4420             // Enable audio controls
4421             playAudioButton.disabled = false;
4422             stopAudioButton.disabled = false;
4423             audioScrubber.disabled = false;

4424             // Generate waveform visualization
4425             generateWaveformData(buffer);

4426             // Update status
4427             updateStatusMessage('Audio loaded: ' + audioFileName);

4428
4429             // If the x-sheet has fewer frames than the audio, suggest
4430             increasing
4431             if (frameCount > frameCountInput.value) {
4432                 if (confirm('This audio is ${frameCount} frames long at 24fps,
4433                 but your X-sheet only has ${frameCountInput.value} frames. Do you want to increase the frame
4434                 count?')) {
4435                     frameCountInput.value = frameCount;
```

```
4436         frameCount = frameCount;
4437         generateTable(frameCount);
4438         renderWaveform();
4439     } else {
4440         renderWaveform();
4441     }
4442 } else {
4443     renderWaveform();
4444 }
4445 }, function (e) {
4446     updateStatusMessage('Error decoding audio: ' + e.message);
4447 });
4448 };
4449
4450 reader.onerror = function () {
4451     updateStatusMessage('Error reading audio file');
4452 };
4453
4454 reader.readAsArrayBuffer(file);
4455 }
4456
4457 function generateWaveformData(buffer) {
4458     // Get the raw audio data from the buffer
4459     const rawData = buffer.getChannelData(0); // Use first channel
4460
4461     // Calculate how many samples we need for our visualization
4462     const totalSamples = rawData.length;
4463
4464     // Process for visualization - we need to reduce the resolution
4465     // to make it efficient to display
4466     waveformData = [];
4467
4468     // Calculate desired number of points for the visualization
4469     // For vertical waveform, we want more detail
4470     const pointsPerSecond = 100; // Increase detail for vertical display
4471     const totalPoints = Math.ceil(buffer.duration * pointsPerSecond);
4472
4473     // Calculate step size
4474     const step = Math.floor(totalSamples / totalPoints);
4475
4476     // Build the waveform data array
4477     for (let i = 0; i < totalPoints; i++) {
4478         const index = Math.floor(i * step);
4479         if (index < totalSamples) {
4480             // Get the absolute value for a nicer visual
4481             waveformData.push(Math.abs(rawData[index]));
4482         }
4483     }
4484 }
4485 }
```

```
4486 function renderWaveform() {
4487     if (!audioBuffer || waveformData.length === 0) return;
4488
4489     // Clear existing markers and components
4490     const existingMarkers = document.querySelectorAll('.waveform-marker');
4491     existingMarkers.forEach(marker => marker.remove());
4492
4493     const existingLabels = document.querySelectorAll('.phonetic-label');
4494     existingLabels.forEach(label => label.remove());
4495
4496     const existingContainer = document.querySelector('.waveform-container');
4497     if (existingContainer) {
4498         existingContainer.remove();
4499     }
4500
4501     // Get the table and first waveform column cell for positioning
4502     const table = document.getElementById('xsheet-table');
4503     const firstCell = document.querySelector('.waveform-col[data-frame="1"]');
4504     if (!firstCell) return;
4505
4506     // Calculate total height needed for the waveform
4507     const totalRows = document.querySelectorAll('tr[data-frame]').length;
4508     const rowHeight = firstCell.offsetHeight;
4509     const totalHeight = totalRows * rowHeight;
4510
4511     // Create a container for the vertical waveform that spans the entire table
4512     const waveformContainer = document.createElement('div');
4513     waveformContainer.className = 'waveform-container';
4514
4515     // Create a canvas for the waveform
4516     const canvas = document.createElement('canvas');
4517     canvas.className = 'waveform-canvas';
4518     canvas.width = firstCell.offsetWidth;
4519     canvas.height = totalHeight;
4520     waveformContainer.appendChild(canvas);
4521
4522     // Add overlay for event handling
4523     const overlay = document.createElement('div');
4524     overlay.className = 'waveform-overlay';
4525     overlay.style.height = totalHeight + 'px';
4526
4527     // Add event listener for clicking on the waveform
4528     overlay.addEventListener('click', function (e) {
4529         if (!audioBuffer) return;
4530
4531         const rect = overlay.getBoundingClientRect();
4532         const y = e.clientY - rect.top;
4533         const percentage = y / rect.height;
4534
4535         // Calculate time point in the audio
```

```
4536 |         const timePoint = percentage * audioBuffer.duration;
4537 |
4538 |         // Update audio position
4539 |         audioScrubber.value = (timePoint / audioBuffer.duration) * 100;
4540 |         if (isPlaying) {
4541 |             stopAudio();
4542 |             startOffset = timePoint;
4543 |             playAudio();
4544 |         } else {
4545 |             startOffset = timePoint;
4546 |         }
4547 |
4548 |         // Update the UI to show frame position
4549 |         updateFrameMarker();
4550 |     });
4551 |
4552 |     // Variables for scrubbing
4553 |     let isScrubbing = false;
4554 |
4555 |     // Add scrubbing functionality (dragging while holding mouse button)
4556 |     overlay.addEventListener('mousedown', function (e) {
4557 |         if (!audioBuffer) return;
4558 |         if (e.button !== 0) return; // Only respond to left mouse button
4559 |
4560 |         isScrubbing = true;
4561 |
4562 |         // Pause any playing audio
4563 |         if (isPlaying) {
4564 |             pauseAudio();
4565 |         }
4566 |         // - NEW: allow pen (pointerType="pen") to scrub as well -
4567 |         overlay.addEventListener('pointerdown', function (e) {
4568 |             if (!audioBuffer || e.pointerType !== 'pen') return;
4569 |             isScrubbing = true;
4570 |             if (isPlaying) pauseAudio(); // pause continuous playback
4571 |             updateScrubPosition(e); // scrub to this location
4572 |             e.preventDefault(); // prevent default pen
4573 |         });
4574 |         overlay.addEventListener('pointermove', function (e) {
4575 |             // - FIX: only scrub while the pen is physically pressing
4576 |             if (isScrubbing && e.pointerType === 'pen' && e.pressure > 0) {
4577 |                 updateScrubPosition(e);
4578 |             }
4579 |         });
4580 |         overlay.addEventListener('pointerup', function (e) {
4581 |             if (e.pointerType === 'pen') {
4582 |                 isScrubbing = false;
4583 |             }
behavior
(pressure>0)
```

```
4584     });
4585
4586     // Create a scrub audio context if needed
4587     if (!audioContext) {
4588         try {
4589             window.AudioContext = window.AudioContext || window.webkitAudioContext;
4590             audioContext = new AudioContext();
4591         } catch (e) {
4592             console.error('Web Audio API not supported');
4593             return;
4594         }
4595     }
4596
4597     // Initial scrub to current position
4598     updateScrubPosition(e);
4599
4600     // Prevent text selection during drag
4601     e.preventDefault();
4602 });
4603
4604     // Handle scrubbing movement
4605     overlay.addEventListener('mousemove', function (e) {
4606         if (!isScrubbing || !audioBuffer) return;
4607         updateScrubPosition(e);
4608     });
4609
4610     // Function to update position during scrubbing
4611     function updateScrubPosition(e) {
4612         const rect = overlay.getBoundingClientRect();
4613         const y = e.clientY - rect.top;
4614         const percentage = Math.max(0, Math.min(1, y / rect.height));
4615
4616         // Calculate time point in the audio
4617         const timePoint = percentage * audioBuffer.duration;
4618         startOffset = timePoint;
4619
4620         // Update audio scrubber
4621         audioScrubber.value = percentage * 100;
4622
4623         // Play a short snippet of audio at this position
4624         playScrubAudio(timePoint);
4625
4626         // Update the marker
4627         updateFrameMarker();
4628     }
4629
4630     // End scrubbing when mouse is released or leaves element
4631     overlay.addEventListener('mouseup', function () {
4632         isScrubbing = false;
```

```
4633     });
4634
4635     overlay.addEventListener('mouseleave', function () {
4636         isScrubbing = false;
4637     });
4638
4639     // Function to play a short snippet at the scrub position
4640     let scrubSource = null;
4641     function playScrubAudio(timePoint) {
4642         if (scrubSource) {
4643             try {
4644                 scrubSource.stop();
4645             } catch (e) {
4646                 // Ignore errors when stopping already stopped source
4647             }
4648         }
4649
4650         // Play a very short snippet at the current position
4651         scrubSource = audioContext.createBufferSource();
4652         scrubSource.buffer = audioBuffer;
4653         scrubSource.connect(audioContext.destination);
4654
4655         // Play just a short snippet (equivalent to 1-2 frames at 24fps)
4656         const snippetDuration = 1 / 12; // 1/12 of a second (2 frames at 24fps)
4657         scrubSource.start(0, timePoint, snippetDuration);
4658     }
4659
4660     // Right-click to add phonetic marker
4661     overlay.addEventListener('contextmenu', function (e) {
4662         e.preventDefault();
4663         if (!audioBuffer) return;
4664
4665         const rect = overlay.getBoundingClientRect();
4666         const y = e.clientY - rect.top;
4667         const percentage = y / rect.height;
4668
4669         // Calculate time point in the audio
4670         const timePoint = percentage * audioBuffer.duration;
4671
4672         showPhoneticInput(timePoint);
4673         return false;
4674     });
4675
4676     waveformContainer.appendChild(overlay);
4677
4678     // Add the container to the table
4679     document.body.appendChild(waveformContainer);
4680
4681     // Position the container over the waveform column
4682     const tableRect = table.getBoundingClientRect();
```

```
4683 const cellRect = firstCell.getBoundingClientRect();
4684
4685     waveformContainer.style.left = (cellRect.left - 1) + 'px';
4686     waveformContainer.style.top = (cellRect.top) + 'px';
4687     waveformContainer.style.width = (cellRect.width) + 'px';
4688     waveformContainer.style.height = totalHeight + 'px';
4689
4690     // Draw the waveform on the canvas
4691     const ctx = canvas.getContext('2d');
4692     const width = canvas.width;
4693     const height = canvas.height;
4694
4695     // Clear and set background
4696     ctx.clearRect(0, 0, width, height);
4697     ctx.fillStyle = '#ffffff';
4698     ctx.fillRect(0, 0, width, height);
4699
4700     // Calculate scaling factors
4701     const totalDuration = audioBuffer.duration;
4702     const framesPerSecond = 24;
4703     const totalFrames = Math.ceil(totalDuration * framesPerSecond);
4704
4705     // Draw center line
4706     ctx.beginPath();
4707     ctx.strokeStyle = '#cccccc';
4708     ctx.moveTo(width / 2, 0);
4709     ctx.lineTo(width / 2, height);
4710     ctx.stroke();
4711
4712     // Draw the waveform
4713     ctx.beginPath();
4714     ctx.strokeStyle = '#000000';
4715     ctx.lineWidth = 1;
4716
4717     // Map waveform data to vertical height
4718     for (let i = 0; i < waveformData.length; i++) {
4719         const y = (i / waveformData.length) * height;
4720         const amplitude = waveformData[i] * (width * 0.4); // Scale amplitude to
40% of width
4721         const x = (width / 2) + amplitude;
4722
4723         if (i === 0) {
4724             ctx.moveTo(x, y);
4725         } else {
4726             ctx.lineTo(x, y);
4727         }
4728     }
4729
4730     // Draw mirrored waveform for visual effect
4731     for (let i = waveformData.length - 1; i >= 0; i--) {
```

```
4732     const y = (i / waveformData.length) * height;
4733     const amplitude = waveformData[i] * (width * 0.4);
4734     const x = (width / 2) - amplitude;
4735
4736     ctx.lineTo(x, y);
4737 }
4738
4739     ctx.stroke();
4740
4741     // Draw frame markers
4742     for (let i = 1; i <= totalFrames && i <= totalRows; i++) {
4743         const y = (i / totalRows) * height;
4744
4745         // Draw horizontal line at frame boundary
4746         if (i % 8 === 0) {
4747             ctx.beginPath();
4748             ctx.strokeStyle = '#999999';
4749             ctx.lineWidth = 1;
4750             ctx.moveTo(0, y);
4751             ctx.lineTo(width, y);
4752             ctx.stroke();
4753         }
4754     }
4755
4756     // Render phonetic markers
4757     renderPhoneticMarkers();
4758
4759     // Add the moving marker for current frame
4760     updateFrameMarker();
4761 }
4762
4763 function renderPhoneticMarkers() {
4764     if (!phonetics || !audioBuffer) return;
4765
4766     // Get the waveform container
4767     const waveformContainer = document.querySelector('.waveform-container');
4768     if (!waveformContainer) return;
4769
4770     // Get container dimensions
4771     const containerHeight = waveformContainer.offsetHeight;
4772
4773     // Add phonetic markers
4774     phonetics.forEach(phonic => {
4775         // Calculate vertical position based on time
4776         const percentage = phonic.time / audioBuffer.duration;
4777         const yPosition = percentage * containerHeight;
4778
4779         // Create and position the marker
4780         const label = document.createElement('div');
4781         label.className = 'phonetic-label';
```

```
4782     label.textContent = phonetic.text;
4783     label.style.position = 'absolute';
4784     label.style.left = '2px';
4785     label.style.top = yPosition + 'px';
4786     label.style.zIndex = '25';
4787
4788     // Add event listener to edit the phonetic marker
4789     label.addEventListener('dblclick', function () {
4790       showPhoneticInput(phonetic.time, phonetic.text,
4791       phonetics.indexOf(phonetic));
4792     });
4793
4794     waveformContainer.appendChild(label);
4795   }
4796
4797   function updateFrameMarker() {
4798     if (!audioBuffer) return;
4799
4800     // Remove existing markers
4801     const existingMarkers = document.querySelectorAll('.waveform-marker');
4802     existingMarkers.forEach(marker => marker.remove());
4803
4804     // Calculate which frame we're on
4805     const time = isPlaying ?
4806       (audioContext.currentTime - startTime + startOffset) :
4807       startOffset;
4808
4809     const frame = Math.floor(time / frameDuration) + 1;
4810     currentFrame = frame;
4811
4812     // Get the waveform container
4813     const waveformContainer = document.querySelector('.waveform-container');
4814     if (!waveformContainer) return;
4815
4816     // Calculate vertical position based on time
4817     const containerHeight = waveformContainer.offsetHeight;
4818     const percentage = time / audioBuffer.duration;
4819     const yPosition = percentage * containerHeight;
4820
4821     // Create horizontal marker line
4822     const marker = document.createElement('div');
4823     marker.className = 'waveform-marker';
4824     marker.style.position = 'absolute';
4825     marker.style.top = yPosition + 'px';
4826     marker.style.left = '0';
4827     marker.style.width = '100%';
4828     marker.style.height = '2px';
4829     marker.style.backgroundColor = 'rgba(255, 0, 0, 0.7)';
4830     marker.style.zIndex = '30';
```

```
4831
4832         // Add frame number
4833         marker.textContent = `Frame ${frame}`;
4834         marker.style.lineHeight = '0';
4835         marker.style.textAlign = 'right';
4836         marker.style.color = 'red';
4837         marker.style.fontWeight = 'bold';
4838         marker.style.fontSize = '8pt';
4839
4840         waveformContainer.appendChild(marker);
4841
4842         // Scroll to the marker if it's not visible and if we're playing
4843         if (isPlaying) {
4844             const table = document.getElementById('xsheet-table');
4845             const tableRect = table.getBoundingClientRect();
4846             const markerY = tableRect.top + yPosition;
4847
4848             // Check if marker is outside visible area
4849             if (markerY < window.scrollY || markerY > window.scrollY +
4850                 window.innerHeight) {
4851                 window.scrollTo({
4852                     top: markerY - (window.innerHeight / 2),
4853                     behavior: 'smooth'
4854                 });
4855             }
4856
4857             // Update scrubber position
4858             if (audioBuffer) {
4859                 const scrubPercentage = (time / audioBuffer.duration) * 100;
4860                 audioScrubber.value = scrubPercentage;
4861             }
4862
4863             // If playing, schedule the next update
4864             if (isPlaying) {
4865                 requestAnimationFrame(updateFrameMarker);
4866             }
4867         }
4868
4869         function togglePlayAudio() {
4870             if (isPlaying) {
4871                 pauseAudio();
4872             } else {
4873                 playAudio();
4874             }
4875         }
4876
4877         function playAudio() {
4878             if (!audioBuffer) return;
```

```
4880     try {
4881         // Create a new source node
4882         audioSource = audioContext.createBufferSource();
4883         audioSource.buffer = audioBuffer;
4884         audioSource.connect(audioContext.destination);
4885
4886         // Calculate start position
4887         startTime = audioContext.currentTime;
4888
4889         // Start playback from the current offset
4890         audioSource.start(0, startOffset);
4891         isPlaying = true;
4892
4893         // Set up animation loop
4894         updateFrameMarker();
4895
4896         // Update UI
4897         playAudioButton.textContent = 'Pause';
4898
4899         // Set up ended event
4900         audioSource.onended = function () {
4901             if (isPlaying) {
4902                 stopAudio();
4903             }
4904         };
4905     } catch (e) {
4906         updateStatusMessage('Error playing audio: ' + e.message);
4907     }
4908 }
4909
4910 function pauseAudio() {
4911     if (!isPlaying || !audioSource) return;
4912
4913     // Stop the audio
4914     audioSource.stop();
4915
4916     // Calculate current position
4917     startOffset += audioContext.currentTime - startTime;
4918
4919     isPlaying = false;
4920
4921     // Update UI
4922     playAudioButton.textContent = 'Play';
4923 }
4924
4925 function stopAudio() {
4926     if (audioSource) {
4927         try {
4928             audioSource.stop();
4929         } catch (e) {
```

```
4930          // Ignore errors when stopping already stopped source
4931      }
4932  }
4933
4934  isPlaying = false;
4935  startOffset = 0;
4936
4937  // Update UI
4938  playAudioButton.textContent = 'Play';
4939  audioScrubber.value = 0;
4940
4941  // Clear frame marker
4942  updateFrameMarker();
4943 }
4944
4945 function scrubAudio() {
4946  if (!audioBuffer) return;
4947
4948  // Calculate time from scrubber position
4949  const percentage = audioScrubber.value / 100;
4950  const newTime = percentage * audioBuffer.duration;
4951
4952  // If playing, restart from new position
4953  if (isPlaying) {
4954    pauseAudio();
4955    startOffset = newTime;
4956    playAudio();
4957  } else {
4958    startOffset = newTime;
4959    updateFrameMarker();
4960  }
4961 }
4962
4963 function showPhoneticInput(time, initialText = '', editIndex = -1) {
4964  if (time === null) {
4965    // If no time provided, use current position
4966    time = isPlaying ?
4967      (audioContext.currentTime - startTime + startOffset) :
4968      startOffset;
4969  }
4970
4971  // Store position for later use
4972  phoneticEditPosition = {
4973    time: time,
4974    editIndex: editIndex
4975  };
4976
4977  // Set initial text if editing existing marker
4978  phoneticText.value = initialText;
4979 }
```

```
4980 // Get the waveform container and calculate position
4981 const waveformContainer = document.querySelector('.waveform-container');
4982 if (waveformContainer) {
4983     const containerRect = waveformContainer.getBoundingClientRect();
4984     const containerHeight = waveformContainer.offsetHeight;
4985     const percentage = time / audioBuffer.duration;
4986     const yPosition = percentage * containerHeight;
4987
4988     // Position the input near the click position
4989     phoneticInput.style.top = (containerRect.top + yPosition +
window.scrollY) + 'px';
4990     phoneticInput.style.left = (containerRect.right + window.scrollX + 5) +
'px';
4991 } else {
4992     // Default position if container not found
4993     phoneticInput.style.top = '200px';
4994     phoneticInput.style.left = '200px';
4995 }
4996
4997 // Show the input and focus it
4998 phoneticInput.style.display = 'block';
4999 phoneticText.focus();
5000 }
5001
5002 function savePhoneticMarker() {
5003     if (!phoneticEditPosition) return;
5004
5005     const text = phoneticText.value.trim();
5006     if (text === '') {
5007         // If empty text and editing an existing marker, remove it
5008         if (phoneticEditPosition.editIndex >= 0) {
5009             phonetics.splice(phoneticEditPosition.editIndex, 1);
5010         }
5011     } else {
5012         // Save or update the phonetic marker
5013         if (phoneticEditPosition.editIndex >= 0) {
5014             // Update existing
5015             phonetics[phoneticEditPosition.editIndex].text = text;
5016         } else {
5017             // Add new
5018             phonetics.push({
5019                 time: phoneticEditPosition.time,
5020                 text: text
5021             });
5022         }
5023     }
5024
5025     // Hide input
5026     phoneticInput.style.display = 'none';
5027 }
```

```
5028         // Re-render markers
5029         renderWaveform();
5030
5031         // Mark as modified
5032         modified = true;
5033     }
5034
5035     function collectData() {
5036         const data = {
5037             template: currentTemplate,
5038             frameCount: frameCount,
5039             metadata: {
5040                 projectNumber: document.getElementById('project-number').value,
5041                 date: document.getElementById('project-date').value,
5042                 pageNumber: document.getElementById('page-number').value,
5043                 animatorName: document.getElementById('animator-name').value,
5044                 versionNumber: document.getElementById('version-number').value,
5045                 shotNumber: document.getElementById('shot-number').value
5046             },
5047             audio: {
5048                 fileName: audioFileName,
5049                 phonetics: phonetics
5050             },
5051             rows: []
5052         };
5053
5054         // Collect data from all rows
5055         const rows = tableBody.querySelectorAll('tr');
5056         rows.forEach(row => {
5057             const rowData = {
5058                 action: row.cells[0].innerText,
5059                 frame: row.cells[1].innerText,
5060                 // Skip waveform cell (2)
5061                 dialogue: row.cells[3].innerText,
5062                 sound: row.cells[4].innerText,
5063                 technical: row.cells[5].innerText,
5064                 extra1: row.cells[6].innerText,
5065                 extra2: row.cells[7].innerText,
5066                 frameRepeat: row.cells[8].innerText,
5067                 camera: row.cells[9].innerText
5068             };
5069             data.rows.push(rowData);
5070         });
5071
5072         return data;
5073     }
5074
5075     function restoreData(data) {
5076         if (!data) return;
5077     }
```

```
5078 // Update template and frame count
5079 currentTemplate = data.template || 'large';
5080 frameCount = data.frameCount || 96;
5081
5082 // Update UI to match
5083 templateSelector.value = currentTemplate;
5084 frameCountInput.value = frameCount;
5085
5086 // Restore metadata
5087 if (data.metadata) {
5088     document.getElementById('project-number').value =
5089     data.metadata.projectNumber || '';
5090     document.getElementById('project-date').value = data.metadata.date ||
5091     '';
5092     document.getElementById('page-number').value = data.metadata.pageNumber
5093     || '';
5094     document.getElementById('animator-name').value =
5095     data.metadata.animatorName || '';
5096     document.getElementById('version-number').value =
5097     data.metadata.versionNumber || '';
5098     document.getElementById('shot-number').value = data.metadata.shotNumber
5099     || '';
5100 }
5101
5102 // Generate the table with the right frame count
5103 generateTable(frameCount);
5104 updateTemplate();
5105
5106 // Restore audio data
5107 if (data.audio) {
5108     audioFileName = data.audio.fileName || '';
5109     phonetics = data.audio.phonetics || [];
5110
5111     if (audioBuffer && phonetics.length > 0) {
5112         renderWaveform();
5113     }
5114 }
5115
5116 // Restore row data
5117 if (data.rows && data.rows.length > 0) {
5118     const rows = tableBody.querySelectorAll('tr');
5119     data.rows.forEach((rowData, index) => {
5120         if (index < rows.length) {
5121             rows[index].cells[0].innerText = rowData.action || '';
5122             // Don't restore frame number cells as they're auto-generated
5123             rows[index].cells[3].innerText = rowData.dialogue || '';
5124             rows[index].cells[4].innerText = rowData.sound || '';
5125             rows[index].cells[5].innerText = rowData.technical || '';
5126             rows[index].cells[6].innerText = rowData.extra1 || '';
5127             rows[index].cells[7].innerText = rowData.extra2 || '';
5128             // Don't restore the second frame number cell
5129         }
5130     });
5131 }
```

```
5123             rows[index].cells[9].innerText = rowData.camera || '';
5124         }
5125     });
5126 }
5127
5128     modified = false;
5129     updateStatusMessage('Project loaded successfully');
5130 }
5131
5132 function saveProject() {
5133     const data = collectData();
5134
5135     // Save to localStorage
5136     try {
5137         localStorage.setItem('animationXSheet', JSON.stringify(data));
5138         modified = false;
5139         updateStatusMessage('Project saved successfully');
5140
5141         // Also download as JSON file for backup
5142         const filename = projectName + '.json';
5143         const dataStr = "data:text/json;charset=utf-8," +
5144 encodeURIComponent(JSON.stringify(data));
5145         const downloadAnchorNode = document.createElement('a');
5146         downloadAnchorNode.setAttribute("href", dataStr);
5147         downloadAnchorNode.setAttribute("download", filename);
5148         document.body.appendChild(downloadAnchorNode);
5149         downloadAnchorNode.click();
5150         downloadAnchorNode.remove();
5151     } catch (e) {
5152         updateStatusMessage('Error saving project: ' + e.message);
5153     }
5154 }
5155
5156 function loadProject() {
5157     // First try to load from localStorage
5158     try {
5159         const savedData = localStorage.getItem('animationXSheet');
5160         if (savedData) {
5161             restoreData(JSON.parse(savedData));
5162             return;
5163         }
5164     } catch (e) {
5165         updateStatusMessage('Error loading saved project: ' + e.message);
5166     }
5167
5168     // If no localStorage data, create file input for uploading JSON
5169     const fileInput = document.createElement('input');
5170     fileInput.type = 'file';
5171     fileInput.accept = '.json';
5172     fileInput.style.display = 'none';
```

```
5172     document.body.appendChild(fileInput);  
5173  
5174     fileInput.addEventListener('change', function (e) {  
5175         const file = e.target.files[0];  
5176         if (!file) return;  
5177  
5178         const reader = new FileReader();  
5179         reader.onload = function (e) {  
5180             try {  
5181                 const data = JSON.parse(e.target.result);  
5182                 restoreData(data);  
5183             } catch (error) {  
5184                 updateStatusMessage('Error parsing file: ' + error.message);  
5185             }  
5186         };  
5187         reader.readAsText(file);  
5188     });  
5189  
5190     fileInput.click();  
5191     fileInput.remove();  
5192 }  
5193  
5194 function exportToPDF() {  
5195     // Save the current state before PDF export  
5196     const savedData = collectData();  
5197  
5198     updateStatusMessage('Preparing PDF. Please wait...');  
5199  
5200     // Remove controls temporarily for PDF generation  
5201     const controls = document.querySelector('.controls');  
5202     const audioControls = document.querySelector('#audio-controls');  
5203     const statusMsg = document.querySelector('.status');  
5204     const phoneticInputEl = document.querySelector('#phonetic-input');  
5205  
5206     controls.style.display = 'none';  
5207     audioControls.style.display = 'none';  
5208     statusMsg.style.display = 'none';  
5209     phoneticInputEl.style.display = 'none';  
5210  
5211     // Clean up any previous waveform elements  
5212     const previousContainers = document.querySelectorAll('.cell-waveform-  
window');  
5213     previousContainers.forEach(container => container.remove());  
5214  
5215     // Draw the waveform directly into the cells  
5216     if (audioBuffer && waveformData.length > 0) {  
5217         drawWaveformInCells();  
5218     }  
5219  
5220     // Wait a moment for DOM updates to complete
```

```
5221     setTimeout(() => {
5222       try {
5223         // Force a redraw of the drawing layer before capture
5224         if (window.xsheetDrawing && window.xsheetDrawing.layerSystem) {
5225           window.xsheetDrawing.layerSystem.redrawAll();
5226         }
5227
5228         // Use html2canvas to capture the printable area
5229         html2canvas(document.getElementById('printable-area'), {
5230           scale: 2, // Higher resolution
5231           useCORS: true,
5232           logging: false,
5233           allowTaint: true,
5234           onclone: function (clonedDoc) {
5235             // Ensure drawing elements are visible in the cloned
5236             document
5237               const clonedDrawingLayer =
5238                 clonedDoc.querySelector('.drawing-layer-container');
5239                 if (clonedDrawingLayer) {
5240                   clonedDrawingLayer.style.display = 'block';
5241                   clonedDrawingLayer.style.visibility = 'visible';
5242                   clonedDrawingLayer.style.opacity = '1';
5243                 }
5244               }).then(canvas => {
5245                 const imgData = canvas.toDataURL('image/png');
5246
5247                 // Determine PDF size based on template
5248                 let pdfWidth, pdfHeight;
5249                 if (currentTemplate === 'large') {
5250                   // 11"x17"
5251                   pdfWidth = 279.4; // mm
5252                   pdfHeight = 431.8; // mm
5253                 } else {
5254                   // 8.5"x11"
5255                   pdfWidth = 215.9; // mm
5256                   pdfHeight = 279.4; // mm
5257                 }
5258
5259                 // Create PDF with jsPDF
5260                 const { jsPDF } = window.jspdf;
5261                 const pdf = new jsPDF({
5262                   orientation: 'portrait',
5263                   unit: 'mm',
5264                   format: [pdfWidth, pdfHeight]
5265                 });
5266
5267                 // Calculate aspect ratio
5268                 const imgWidth = pdfWidth - 20; // margins
5269                 const imgHeight = canvas.height * imgWidth / canvas.width;
```

```
5269
5270          // Add image to PDF
5271          pdf.addImage(imgData, 'PNG', 10, 10, imgWidth, imgHeight);
5272
5273          // Save PDF
5274          pdf.save(` ${projectName}.pdf` );
5275
5276          // Clean up after PDF generation using saved data instead of
rebuilding
5277          cleanupAfterExport(savedData);
5278      });
5279  } catch (e) {
5280      // Clean up if there was an error
5281      cleanupAfterExport(savedData);
5282      updateStatusMessage('Error exporting PDF: ' + e.message);
5283  }
5284 }, 100);

5285
5286 function cleanupAfterExport(savedData) {
5287     // Restore controls
5288     controls.style.display = 'flex';
5289     audioControls.style.display = 'flex';
5290     statusMsg.style.display = 'block';
5291
5292     // Restore data instead of regenerating blank table
5293     restoreData(savedData);
5294
5295     updateStatusMessage('PDF exported successfully');
5296 }
5297
5298 function drawWaveformInCells() {
5299     const waveformCells = document.querySelectorAll('.waveform-col');
5300     if (waveformCells.length === 0) return;
5301
5302     // Calculate the number of data points per frame
5303     const totalDuration = audioBuffer.duration;
5304     const pointsPerFrame = waveformData.length / (totalDuration * 24);
5305
5306     // For each cell, draw its portion of the waveform
5307     waveformCells.forEach((cell, index) => {
5308         // Get the frame number (1-based)
5309         const frameNum = index + 1;
5310
5311         // Create a canvas for this cell
5312         const canvas = document.createElement('canvas');
5313         canvas.width = cell.offsetWidth;
5314         canvas.height = cell.offsetHeight;
5315         canvas.style.display = 'block';
5316         canvas.style.width = '100%';
5317         canvas.style.height = '100%';
```

```
5318
5319         const ctx = canvas.getContext('2d');
5320
5321         // Fill with white background
5322         ctx.fillStyle = 'white';
5323         ctx.fillRect(0, 0, canvas.width, canvas.height);
5324
5325         // Draw center line
5326         ctx.beginPath();
5327         ctx.strokeStyle = '#cccccc';
5328         ctx.moveTo(canvas.width / 2, 0);
5329         ctx.lineTo(canvas.width / 2, canvas.height);
5330         ctx.stroke();
5331
5332         // Calculate which section of the waveform to draw
5333         const startPoint = Math.floor((frameNum - 1) * pointsPerFrame);
5334         const endPoint = Math.floor(frameNum * pointsPerFrame);
5335
5336         if (startPoint < waveformData.length) {
5337             // Draw this portion of the waveform
5338             ctx.beginPath();
5339             ctx.strokeStyle = '#000000';
5340
5341             // Check if we have valid data to draw
5342             if (endPoint > startPoint) {
5343                 // Map the waveform section to this cell
5344                 for (let i = startPoint; i <= endPoint && i <
5345                     waveformData.length; i++) {
5346                     // Calculate position within this cell
5347                     const relativePos = (i - startPoint) / (endPoint -
5348                         startPoint);
5349                     const y = relativePos * canvas.height;
5350
5351                     // Draw waveform
5352                     const amplitude = waveformData[i] * (canvas.width *
5353                         0.4);
5354                     const x = (canvas.width / 2) + amplitude;
5355
5356                     if (i === startPoint) {
5357                         ctx.moveTo(x, y);
5358                     } else {
5359                         ctx.lineTo(x, y);
5360                     }
5361
5362                     // Draw left side (mirror)
5363                     for (let i = endPoint; i >= startPoint && i <
5364                         waveformData.length; i--) {
5365                         const relativePos = (i - startPoint) / (endPoint -
5366                             startPoint);
5367                         const y = relativePos * canvas.height;
```

```
5364
5365          const amplitude = waveformData[i] * (canvas.width *
5366              0.4);
5367
5368          const x = (canvas.width / 2) - amplitude;
5369          ctx.lineTo(x, y);
5370      }
5371
5372      ctx.stroke();
5373
5374 // Look for phonetic markers that might be in this frame
5375 if (phonetics && phonetics.length > 0) {
5376     phonetics.forEach(pho => {
5377         // Calculate which frame this phonetic marker belongs to
5378         const frameOfMarker = Math.floor(pho.time /
5379             frameDuration) + 1;
5380
5381         // If it's in this frame, draw it
5382         if (frameOfMarker === frameNum) {
5383             // Calculate position within cell
5384             const posInFrameRatio = (pho.time - ((frameNum -
5385                 1) * frameDuration)) / frameDuration;
5386             const yPos = posInFrameRatio * canvas.height;
5387
5388             // Draw marker line
5389             ctx.beginPath();
5390             ctx.strokeStyle = '#ff0000';
5391             ctx.lineWidth = 1;
5392             ctx.moveTo(0, yPos);
5393             ctx.lineTo(canvas.width, yPos);
5394             ctx.stroke();
5395
5396             // Add label if it fits
5397             if (canvas.width > 30) {
5398                 ctx.fillStyle = 'rgba(255, 255, 255, 0.7)';
5399                 const textWidth =
5400                     ctx.measureText(pho.text).width;
5401
5402                 if (textWidth < canvas.width - 4) {
5403                     ctx.fillRect(2, yPos - 8, textWidth + 4,
5404                         14);
5405
5406                     ctx.fillStyle = '#ff0000';
5407                     ctx.font = '8px Arial';
5408                     ctx.fillText(pho.text, 4, yPos + 4);
5409                 }
5410             }
5411         }
5412     });
5413 }
5414 }
```

```

5410         // Add frame number indicator
5411         ctx.fillStyle = 'rgba(255, 255, 255, 0.7)';
5412         ctx.fillRect(0, 0, 18, 12);
5413         ctx.fillStyle = '#000000';
5414         ctx.font = '8px Arial';
5415         ctx.fillText(frameNum, 2, 8);
5416
5417         // Clear cell content and add the canvas
5418         cell.innerHTML = '';
5419         cell.appendChild(canvas);
5420     });
5421 }
5422 }
5423
5424 function printSheet() {
5425     updateStatusMessage('Preparing to print. Please wait...');
5426
5427     // Save current selection state and cell contents
5428     const tableData = saveSelectionState();
5429
5430     // Hide the waveform container temporarily but don't remove it
5431     const originalWf = document.querySelector('.waveform-container');
5432     if (originalWf) {
5433         originalWf.style.display = 'none';
5434     }
5435
5436     // Draw the waveform directly into the cells before printing
5437     if (audioBuffer && waveformData.length > 0) {
5438         drawWaveformInCells();
5439     }
5440
5441     // Wait a moment for DOM updates to complete
5442     setTimeout(() => {
5443         window.print();
5444
5445         // Clean up after printing
5446         setTimeout(() => {
5447             // Restore the original waveform container
5448             if (originalWf) {
5449                 originalWf.style.display = '';
5450             }
5451
5452             // Restore original cell content for waveform cells only
5453             const waveformCells = document.querySelectorAll('.waveform-col');
5454             waveformCells.forEach(cell => {
5455                 const originalContent = cell.getAttribute('data-original-
content');
5456                 if (originalContent !== null) {
5457                     cell.innerHTML = originalContent;
5458                     cell.removeAttribute('data-original-content');
5459                 }
5460             });
5461         });
5462     });
5463 }
5464
5465 
```

```
5459     }
5460   });
5461
5462   // Restore all cell contents from saved data
5463   if (tableData && tableData.cellContents) {
5464     tableData.cellContents.forEach(cellData => {
5465       if (cellData.rowIndex >= 0 && cellData.rowIndex <
5466         tableBody.children.length) {
5467         const row = tableBody.children[cellData.rowIndex];
5468         if (cellData.colIndex >= 0 && cellData.colIndex <
5469           row.children.length) {
5470           const cell = row.children[cellData.colIndex];
5471
5472           // Skip waveform cells (they were already restored
5473           above)
5474
5475           if (!cell.classList.contains('waveform-col')) {
5476             // Only restore if cell is editable
5477             if (cell.contentEditable === 'true') {
5478               cell.innerHTML = cellData.content;
5479
5480               // Restore modified status
5481               if (cellData.isModified) {
5482                 cell.classList.add('modified');
5483               } else {
5484                 cell.classList.remove('modified');
5485               }
5486             }
5487           }
5488
5489           // Restore metadata fields
5490           if (tableData && tableData.metadata) {
5491             document.getElementById('project-number').value =
5492               tableData.metadata.projectNumber || '';
5493             document.getElementById('project-date').value =
5494               tableData.metadata.date || '';
5495             document.getElementById('page-number').value =
5496               tableData.metadata.pageNumber || '';
5497             document.getElementById('animator-name').value =
5498               tableData.metadata.animatorName || '';
5499             document.getElementById('version-number').value =
5500               tableData.metadata.versionNumber || '';
5501             document.getElementById('shot-number').value =
5502               tableData.metadata.shotNumber || '';
5503           }
5504
5505           // Finally restore cell selection state
5506           restoreSelectionState(tableData);
5507         }
5508       }
5509     }
5510   }
5511 }
```

```
5501
5502             updateStatusMessage('Print complete');
5503         }, 500);
5504     }, 100);
5505
5506     function drawWaveformInCells() {
5507         const waveformCells = document.querySelectorAll('.waveform-col');
5508         if (waveformCells.length === 0) return;
5509
5510         // Calculate the number of data points per frame
5511         const totalDuration = audioBuffer.duration;
5512         const pointsPerFrame = waveformData.length / (totalDuration * 24);
5513
5514         // For each cell, draw its portion of the waveform
5515         waveformCells.forEach((cell, index) => {
5516             // Get the frame number (1-based)
5517             const frameNum = index + 1;
5518
5519             // Create a canvas for this cell
5520             const canvas = document.createElement('canvas');
5521             canvas.width = cell.offsetWidth;
5522             canvas.height = cell.offsetHeight;
5523             canvas.style.display = 'block';
5524             canvas.style.width = '100%';
5525             canvas.style.height = '100%';
5526
5527             const ctx = canvas.getContext('2d');
5528
5529             // Fill with white background
5530             ctx.fillStyle = 'white';
5531             ctx.fillRect(0, 0, canvas.width, canvas.height);
5532
5533             // Draw center line
5534             ctx.beginPath();
5535             ctx.strokeStyle = '#cccccc';
5536             ctx.moveTo(canvas.width / 2, 0);
5537             ctx.lineTo(canvas.width / 2, canvas.height);
5538             ctx.stroke();
5539
5540             // Calculate which section of the waveform to draw
5541             const startPoint = Math.floor((frameNum - 1) * pointsPerFrame);
5542             const endPoint = Math.floor(frameNum * pointsPerFrame);
5543
5544             if (startPoint < waveformData.length) {
5545                 // Draw this portion of the waveform
5546                 ctx.beginPath();
5547                 ctx.strokeStyle = '#000000';
5548
5549                 // Check if we have valid data to draw
5550                 if (endPoint > startPoint) {
```

```

5551
5552
5553
5554
5555
5556
5557
5558
5559
5560
5561
5562
5563
5564
5565
5566
5567
5568
5569
5570
5571
5572
5573
5574
5575
5576
5577
5578
5579
5580
5581
5582
5583
5584
5585
5586
5587
5588
5589
5590
5591
5592
5593

      waveformData.length; i++) {
        // Map the waveform section to this cell
        for (let i = startPoint; i <= endPoint && i <
          // Calculate position within this cell
          const relativePos = (i - startPoint) / (endPoint -
            const y = relativePos * canvas.height;

          // Draw waveform
          const amplitude = waveformData[i] * (canvas.width *
            0.4);
          const x = (canvas.width / 2) + amplitude;

          if (i === startPoint) {
            ctx.moveTo(x, y);
          } else {
            ctx.lineTo(x, y);
          }
        }

        // Draw left side (mirror)
        for (let i = endPoint; i >= startPoint && i <
          const relativePos = (i - startPoint) / (endPoint -
            const y = relativePos * canvas.height;

            const amplitude = waveformData[i] * (canvas.width *
              0.4);
            const x = (canvas.width / 2) - amplitude;

            ctx.lineTo(x, y);
          }

          ctx.stroke();
        }

        // Look for phonetic markers that might be in this frame
        if (phonetics && phonetics.length > 0) {
          phonetics.forEach(phonic => {
            // Calculate which frame this phonetic marker belongs to
            const frameOfMarker = Math.floor(phonic.time /
              frameDuration) + 1;

            // If it's in this frame, draw it
            if (frameOfMarker === frameNum) {
              // Calculate position within cell
              const posInFrameRatio = (phonic.time - ((frameNum -
                1) * frameDuration)) / frameDuration;
              const yPos = posInFrameRatio * canvas.height;
            }
          });
        }
      }
    }
  }
}

```

```

5594         // Draw marker line
5595         ctx.beginPath();
5596         ctx.strokeStyle = '#ff0000';
5597         ctx.lineWidth = 1;
5598         ctx.moveTo(0, yPos);
5599         ctx.lineTo(canvas.width, yPos);
5600         ctx.stroke();
5601
5602         // Add label if it fits
5603         if (canvas.width > 30) {
5604             ctx.fillStyle = 'rgba(255, 255, 255, 0.7)';
5605             const textWidth =
5606                 ctx.measureText(phnetic.text).width;
5607             if (textWidth < canvas.width - 4) {
5608                 ctx.fillRect(2, yPos - 8, textWidth + 4,
5609                             14);
5610                 ctx.fillStyle = '#ff0000';
5611                 ctx.font = '8px Arial';
5612                 ctx.fillText(phnetic.text, 4, yPos + 4);
5613             }
5614         });
5615     }
5616 }
5617
5618     // Add frame number indicator
5619     ctx.fillStyle = 'rgba(255, 255, 255, 0.7)';
5620     ctx.fillRect(0, 0, 18, 12);
5621     ctx.fillStyle = '#000000';
5622     ctx.font = '8px Arial';
5623     ctx.fillText(frameNum, 2, 8);
5624
5625     // Store original content and add the canvas
5626     cell.setAttribute('data-original-content', cell.innerHTML);
5627     cell.innerHTML = '';
5628     cell.appendChild(canvas);
5629   );
5630 }
5631
5632 function saveSelectionState() {
5633   // Save current selection and cell contents
5634   const tableData = {
5635     // Save which cells are selected
5636     selectedIndices: selectedCells.map(cell => {
5637       const row = cell.closest('tr');
5638       const rowIndex = Array.from(tableBody.children).indexOf(row);
5639       const colIndex = Array.from(row.children).indexOf(cell);
5640       return { rowIndex, colIndex };
5641     }),

```

```
5642         // Save metadata fields
5643         metadata: {
5644             projectNumber: document.getElementById('project-number').value,
5645             date: document.getElementById('project-date').value,
5646             pageNumber: document.getElementById('page-number').value,
5647             animatorName: document.getElementById('animator-name').value,
5648             versionNumber: document.getElementById('version-number').value,
5649             shotNumber: document.getElementById('shot-number').value
5650         },
5651         // Store editable cells' content
5652         cellContents: []
5653     };
5654
5655     // Store all editable cells' content
5656     const allEditableCells =
5657         document.querySelectorAll('[contenteditable="true"]');
5658         allEditableCells.forEach(cell => {
5659             const row = cell.closest('tr');
5660             if (row) {
5661                 const rowIndex = Array.from(tableBody.children).indexOf(row);
5662                 const colIndex = Array.from(row.children).indexOf(cell);
5663                 tableData.cellContents.push({
5664                     rowIndex,
5665                     colIndex,
5666                     content: cell.innerHTML,
5667                     isModified: cell.classList.contains('modified')
5668                 });
5669             }
5670         });
5671
5672     return tableData;
5673 }
5674
5675     function restoreSelectionState(tableData) {
5676         if (!tableData || !tableData.selectedIndices) return;
5677
5678         // Clear current selection
5679         clearCellSelection();
5680
5681         // Restore selection
5682         tableData.selectedIndices.forEach(index => {
5683             if (index.rowIndex >= 0 && index.rowIndex <
5684                 tableBody.children.length) {
5685                 const row = tableBody.children[index.rowIndex];
5686                 if (index.colIndex >= 0 && index.colIndex < row.children.length)
5687                 {
5688                     const cell = row.children[index.colIndex];
5689                     if (cell.contentEditable === 'true') {
5690                         toggleCellSelection(cell, true);
5691                     }
5692                 }
5693             }
5694         });
5695     }
5696 }
```

```
5690         }
5691     });
5692   }
5693 }
5694
5695 function addEightRows() {
5696   frameCount += 8;
5697   frameCountInput.value = frameCount;
5698   generateTable(frameCount);
5699   updateStatusMessage('Added 8 rows. Total frames: ' + frameCount);
5700
5701   // Re-render waveform if audio is loaded
5702   if (audioBuffer) {
5703     renderWaveform();
5704   }
5705 }
5706
5707 function clearSheet() {
5708   if (confirm('Are you sure you want to clear all data? This cannot be
undone.')) {
5709     // Clear metadata
5710     document.getElementById('project-number').value = '';
5711     document.getElementById('page-number').value = '';
5712     document.getElementById('animator-name').value = '';
5713     document.getElementById('version-number').value = '';
5714     document.getElementById('shot-number').value = '';
5715
5716     // Reset date to today
5717     document.getElementById('project-date').valueAsDate = new Date();
5718
5719     // Clear all editable cells
5720     const editableCells =
5721       document.querySelectorAll('[contenteditable="true"]');
5722     editableCells.forEach(cell => {
5723       cell.innerText = '';
5724       cell.classList.remove('modified');
5725     });
5726
5727     // Clear audio
5728     audioBuffer = null;
5729     audioSource = null;
5730     waveformData = [];
5731     phonetics = [];
5732     audioFileName = '';
5733     audioInfo.textContent = 'No audio loaded';
5734
5735     // Stop any playing audio
5736     stopAudio();
5737
5738     // Clear waveform visualization
```

```
5738     waveformCanvases.forEach(canvas => {
5739         const ctx = canvas.getContext('2d');
5740         ctx.clearRect(0, 0, canvas.width, canvas.height);
5741     });
5742 
5743     // Clear phonetic markers
5744     const labels = document.querySelectorAll('.phonetic-label');
5745     labels.forEach(label => label.remove());
5746 
5747     // Clear drawings
5748     if (window.xsheetDrawing && window.xsheetDrawing.layerSystem) {
5749         window.xsheetDrawing.layerSystem.clearAllLayers();
5750     }
5751 
5752     modified = false;
5753     updateStatusMessage('Sheet cleared');
5754 }
5755 }
5756 
5757 function updateStatusMessage(message) {
5758     statusMessage.textContent = message;
5759     console.log(message);
5760 
5761     // Clear status message after 3 seconds
5762     setTimeout(() => {
5763         if (statusMessage.textContent === message) {
5764             if (modified) {
5765                 statusMessage.textContent = 'Unsaved changes';
5766             } else {
5767                 statusMessage.textContent = '';
5768             }
5769         }
5770     }, 3000);
5771 }
5772 
5773 // Auto-save timer every 2 minutes
5774 setInterval(() => {
5775     if (modified) {
5776         try {
5777             const data = collectData();
5778             localStorage.setItem('animationXSheet_autosave',
5779             JSON.stringify(data));
5780             updateStatusMessage('Auto-saved');
5781         } catch (e) {
5782             console.error('Auto-save failed:', e);
5783         }
5784     }, 120000);
5785 
5786 // Check for auto-saved data on load
```

```
5787     try {
5788         const autoSavedData = localStorage.getItem('animationXSheet_autosave');
5789         if (autoSavedData && !localStorage.getItem('animationXSheet')) {
5790             if (confirm('Found auto-saved data. Would you like to restore it?')) {
5791                 restoreData(JSON.parse(autoSavedData));
5792             }
5793         }
5794     } catch (e) {
5795         console.error('Error checking for auto-saved data:', e);
5796     }
5797
5798     // Initial status
5799     updateStatusMessage('X-Sheet ready');
5800 });
5801 </script>
5802 </body>
5803
5804 </html>
```