Widow and Orphan Control Test Document [□]

This document is designed to stress-test page breaking behavior with headers of various levels. It contains realistic content structures that commonly cause orphaned headers.

Executive Summary [*]

This section has just enough content to potentially end up at the bottom of a page, with its header orphaned from its content. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Section One: Multi-Paragraph Content [□]

Subsection 1.1: Short Content After Header

This subsection has minimal content that could easily get separated from its header if page breaks occur at the wrong point.

Subsection 1.2: Long Content [...]

This subsection has substantial content to push boundaries. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.

Subsection 1.3: Lists After Headers [□]

This header is followed by a list, which is a common pattern:

- □ First item with enough content to potentially cause issues when breaking across pages
- □ Second item that continues the list and adds more vertical space
- □ Third item to ensure we have substantial content
- □ Fourth item to push the boundaries further
- □ Fifth item to really test the limits

Subsection 1.4: Code Blocks

This section demonstrates code blocks after headers:

```
function exampleFunction() {
  console.log("This code block could get orphaned from its header");
  const data = {
    key: "value",
    nested: {
      property: "test"
    }
  };
  return data;
}
```

The code block above should stay with its header.

Section Two: Nested Headers [□]

Subsection 2.1: Introduction [□]

Brief introduction to nested content structures.

Deep Subsection 2.1.1: Fourth Level This is a fourth-level header that's particularly vulnerable to orphaning.

Fifth Level Header: Even More Nested Fifth-level headers are rare but should still be handled correctly when they appear near page boundaries.

Sixth Level: Maximum Nesting

This is the deepest level of heading supported by Markdown.

Subsection 2.2: Tables After Headers

This header is followed by a table:

Column 1	Column 2	Column 3
Data A	Data B	Data C
Data D	Data E	Data F
Data G	Data H	Data I

Subsection 2.3: Blockquotes

This header is followed by a blockquote:

> This is a blockquote that demonstrates another common pattern where headers can get separated from their content. The blockquote contains enough text to potentially cause pagination issues if not handled correctly.
>
> Multiple paragraphs in blockquotes are also common and should be tested.

Section Three: Mixed Content Patterns [≈]

Subsection 3.1: Paragraph Then List [!]

First we have a paragraph of text that provides context for the list below. This paragraph has enough content to potentially push the list to the next page if it appears near a page boundary.

Now the list follows:

1. Numbered item one with substantial content
2. Numbered item two continuing the pattern
3. Numbered item three adding more vertical space
4. Numbered item four to ensure adequate testing

Subsection 3.2: Multiple Short Paragraphs

This header is followed by multiple short paragraphs, which is a different pagination challenge.

First paragraph is brief.

Second paragraph is also brief.

Third paragraph continues the pattern.

Fourth paragraph should stay with the section.

Placeholder Image Description

Figure 1: Placeholder Image Description

Subsection 3.3: Image Reference

This section would typically contain an image:

The image caption and surrounding text should stay with the header.

Section Four: Edge Cases [!]

Subsection 4.1: Very Long Single Paragraph [—]

This subsection contains a very long single paragraph that will definitely span multiple lines and potentially multiple pages. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.

Subsection 4.2: Alternating Content Types

This section alternates between different content types:

Bold text followed by *italic text* and `inline code` to test mixed formatting.

- A list item
- Another list item

Then back to paragraph text with more content to fill the space and test boundaries.

```
# A code block
def another_example():
    return "Testing alternating content"
```

Subsection 4.3: Dense Technical Content

This section simulates dense technical documentation:

The algorithm complexity is O(n log n) where n represents the input size. The space complexity is O(n) due to the auxiliary data structures required for processing. Implementation details:

```
interface Config {
  maxRetries: number;
  timeout: number;
  backoff: 'exponential' | 'linear';
}

class ServiceHandler {
  private config: Config;

  constructor(config: Config) {
    this.config = config;
  }
```

```
  async process(): Promise<void> {
    // Implementation here
  }
}
```

The above implementation provides a foundation for further development.

Section Five: Realistic Documentation Structure [□ □]

Overview [□]

This section mimics a real documentation page with typical content patterns.

Prerequisites □

Before proceeding, ensure you have:

- Node.js 18.x or higher
- npm or yarn package manager
- Basic understanding of TypeScript
- Familiarity with async/await patterns

Installation

Install the package using npm:

```
npm install example-package
```

Or using yarn:

```
yarn add example-package
```

Configuration

Create a configuration file:

```
{
  "version": "1.0.0",
  "settings": {
    "debug": false,
    "timeout": 5000
  }
}
```

Usage Examples

Basic Example Here's a simple usage example:

```
const example = require('example-package');

example.init({
  apiKey: 'your-api-key',
  endpoint: 'https://api.example.com'
});
```

Advanced Example For more complex scenarios:

```
const example = require('example-package');

example.configure({
  apiKey: process.env.API_KEY,
  endpoint: process.env.API_ENDPOINT,
  retry: {
    maxAttempts: 3,
    backoff: 'exponential'
  },
  timeout: 10000
});
```

API Reference

Methods

`init(options)` Initializes the service with the provided options.

Parameters: - `options` (Object): Configuration options - `apiKey` (String): Your API key - `endpoint` (String): API endpoint URL

Returns: Promise

`process(data)` Processes the provided data.

Parameters: - `data` (Object): Data to process

Returns: Promise

Troubleshooting

Common Issues **Problem:** Connection timeout errors

Solution: Check your network connection and ensure the API endpoint is accessible.

Problem: Authentication failures

Solution: Verify your API key is correct and has the necessary permissions.

Debugging Enable debug mode in your configuration:

```
{
  "settings": {
    "debug": true
  }
}
```

Best Practices

Follow these recommendations:

1. **Always handle errors gracefully** - Use try/catch blocks around async operations
2. **Set appropriate timeouts** - Don't let requests hang indefinitely
3. **Use environment variables** - Keep sensitive data out of your code
4. **Monitor API usage** - Track your quota to avoid service interruptions
5. **Cache responses** - Reduce API calls when possible

Performance Considerations

When working with large datasets:

- Use pagination to limit response sizes
- Implement request throttling
- Consider using a queue for batch operations
- Monitor memory usage during processing

Security Notes

Important security considerations:

- Never commit API keys to version control
- Use HTTPS endpoints only
- Validate all input data
- Implement rate limiting
- Keep dependencies updated

Migration Guide

If upgrading from version 1.x:

1. Update your package.json
2. Review breaking changes in CHANGELOG.md
3. Update configuration format
4. Test thoroughly before deploying

Contributing

We welcome contributions! Please:

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Add tests
5. Submit a pull request

License

MIT License - see LICENSE file for details.

Support

For support:

- GitHub Issues: https://github.com/example/issues
- Email: support@example.com
- Discord: https://discord.gg/example

Conclusion

This test document contains various header levels and content patterns designed to expose pagination issues. Headers should never be orphaned at the bottom of pages, and content should flow naturally across page boundaries.

Final Notes

The test is complete. Review the generated PDF to identify any orphaned headers or awkward page breaks.