Paired Comments - VS Code Extension MVP Specification

Version: 0.1.0
Date: October 2025
Author: Greg Barker
Target: Claude Code Implementation

🎯 Vision & Long-Term Goal

Create a universal standard for separating code commentary from source files, enabling cleaner codebases, better version control, and enhanced collaboration through synchronized sidecar `.comments` files.

The Big Picture

- **Phase 1 (MVP):** VS Code extension proving the concept
- **Phase 2:** Multi-editor support (JetBrains, Vim, Neovim)
- **Phase 3:** GitHub/GitLab native rendering
- **Phase 4:** Industry standard adoption (like `.d.ts` for TypeScript)

🧩 Core Problem

Inline code comments create fundamental issues:

1. **Visual Clutter:** Comments interrupt code flow and reduce readability
2. **Git Noise:** Comment changes create unnecessary diffs and merge conflicts
3. **Staleness:** Developers hesitate to update comments, leading to outdated documentation
4. **Poor Organization:** Hard to filter, search, or categorize comments independently
5. **Collaboration Friction:** Multiple people can't easily annotate the same code sections

---

💡 The Solution: Sidecar Comment Files

Concept

Each source file pairs with a structured `.comments` file containing all annotations:

```
src/
├── main.ts
├── main.ts.comments        # Sidecar file
├── utils.py
└── utils.py.comments
```

Key Innovation

Display both files side-by-side in VS Code with **synchronized scrolling**, like a diff viewer:

- **Left pane:** Clean source code (no comment clutter)
- **Right pane:** Structured comments aligned to specific lines
- **Scroll sync:** Moving in one pane automatically scrolls the other

---

🎯 MVP Goals (Version 0.1)

Build the **minimum viable product** that proves the concept is valuable and usable.

Success Criteria

- [ ] Users can open paired comments for any file with one command
- [ ] Comments scroll in perfect sync with source code
- [ ] Adding/editing comments feels natural and fast
- [ ] `.comments` files are human-readable and git-friendly
- [ ] Zero configuration required to start using

Non-Goals for MVP

- ❌ AI-assisted comment generation
- ❌ Multi-user live collaboration
- ❌ Advanced filtering/tagging system
- ❌ Export to documentation formats
- ❌ Automatic comment migration from inline

🏗️ Technical Architecture

File Format Specification

Option A: JSON (Recommended for MVP)

json

```json
{
  "file": "src/main.ts",
  "version": "1.0",
  "comments": [
    {
      "id": "c1",
      "line": 42,
      "text": "This function should return a Promise for async operations",
      "author": "Greg",
      "created": "2025-10-16T20:00:00Z",
      "updated": "2025-10-16T20:00:00Z"
    },
    {
      "id": "c2",
      "line": 67,
      "text": "TODO: Refactor this to use the new API endpoint",
      "author": "Greg",
      "created": "2025-10-16T21:30:00Z",
      "updated": "2025-10-16T21:30:00Z"
    }
  ]
}
```

Schema Definition

typescript

```
interface CommentFile {
  file: string;          // Relative path to source file
  version: string;       // Schema version
  comments: Comment[];
}

interface Comment {
  id: string;            // Unique identifier
  line: number;          // Line number in source file
  text: string;          // Comment content
  author: string;        // Comment author
  created: string;       // ISO 8601 timestamp
  updated: string;       // ISO 8601 timestamp
}
```

🚀 MVP Implementation Plan

Phase 1: Core Infrastructure (Week 1)

1.1 Project Setup

bash

```bash
yo code  # VS Code extension generator
# Choose TypeScript
# Extension name: paired-comments
```

Files to create:

- `src/extension.ts` - Main entry point
- `src/commentManager.ts` - Comment CRUD operations
- `src/pairedViewManager.ts` - Dual-pane and scroll sync
- `src/types.ts` - TypeScript interfaces

1.2 Comment File Manager

typescript

```typescript
// src/commentManager.ts
export class CommentManager {
  // Read .comments file (create if doesn't exist)
  async loadComments(sourceFile: string): Promise<CommentFile>

  // Write comments back to disk
  async saveComments(comments: CommentFile): Promise<void>

  // CRUD operations
  addComment(line: number, text: string): Comment
  updateComment(id: string, text: string): void
  deleteComment(id: string): void
  getCommentsForLine(line: number): Comment[]
}
```

Phase 2: Dual-Pane View (Week 1-2)

2.1 Open Paired View Command

typescript

```typescript
// Command: "Paired Comments: Open"
vscode.commands.registerCommand('pairedComments.open', async () => {
  const activeEditor = vscode.window.activeTextEditor;
  if (!activeEditor) return;

  const sourceUri = activeEditor.document.uri;
  const commentsUri = Uri.file(sourceUri.fsPath + '.comments');

  // Open side-by-side
  await vscode.commands.executeCommand('vscode.openWith',
    commentsUri,
    'default',
    ViewColumn.Beside
  );

  // Initialize scroll sync
  startScrollSync(sourceUri, commentsUri);
});
```

2.2 Scroll Synchronization

typescript

```
// src/pairedViewManager.ts
export class ScrollSyncManager {
  private syncEnabled = true;

  setupScrollSync(sourceEditor: TextEditor, commentsEditor: TextEditor) {
    // Listen to scroll events in source editor
    vscode.window.onDidChangeTextEditorVisibleRanges(event => {
      if (event.textEditor === sourceEditor && this.syncEnabled) {
        this.syncScrollPosition(sourceEditor, commentsEditor);
      }
    });

    // Listen to scroll events in comments editor
    vscode.window.onDidChangeTextEditorVisibleRanges(event => {
      if (event.textEditor === commentsEditor && this.syncEnabled) {
        this.syncScrollPosition(commentsEditor, sourceEditor);
      }
    });
  }

  private syncScrollPosition(from: TextEditor, to: TextEditor) {
    const visibleRange = from.visibleRanges[0];
    const topLine = visibleRange.start.line;

    // Temporarily disable sync to prevent ping-pong effect
    this.syncEnabled = false;

    to.revealRange(
      new Range(topLine, 0, topLine + 1, 0),
      TextEditorRevealType.AtTop
    );

    setTimeout(() => this.syncEnabled = true, 100);
  }
}
```

Phase 3: Comment Visualization (Week 2)

3.1 Gutter Icons

Show indicators in the source file where comments exist:

typescript

```typescript
// src/decorationManager.ts
export class DecorationManager {
  private commentDecoration: TextEditorDecorationType;

  constructor() {
    this.commentDecoration = vscode.window.createTextEditorDecorationType({
      gutterIconPath: path.join(__dirname, 'icons', 'comment.svg'),
      gutterIconSize: '16px',
      overviewRulerColor: 'rgba(100, 150, 255, 0.5)',
      overviewRulerLane: OverviewRulerLane.Left
    });
  }

  updateDecorations(editor: TextEditor, comments: Comment[]) {
    const decorations: DecorationOptions[] = comments.map(c => ({
      range: new Range(c.line - 1, 0, c.line - 1, 0),
      hoverMessage: c.text
    }));

    editor.setDecorations(this.commentDecoration, decorations);
  }
}
```

Phase 4: Comment Editing (Week 2-3)

4.1 Add Comment Command

typescript

```
vscode.commands.registerCommand('pairedComments.addComment', async () => {
  const editor = vscode.window.activeTextEditor;
  if (!editor) return;

  const line = editor.selection.active.line + 1;
  const text = await vscode.window.showInputBox({
    prompt: `Add comment for line ${line}`,
    placeHolder: 'Enter your comment...'
  });

  if (!text) return;

  const commentManager = getCommentManager(editor.document.uri);
  await commentManager.addComment(line, text);

  // Refresh view
  refreshDecorations(editor);
});
```

4.2 Edit/Delete via Context Menu

json

```
// package.json
{
 "contributes": {
  "menus": {
   "editor/context": [
    {
     "command": "pairedComments.addComment",
     "group": "pairedComments",
     "when": "editorHasComments"
    },
    {
     "command": "pairedComments.editComment",
     "group": "pairedComments",
     "when": "editorLineHasComment"
    },
    {
     "command": "pairedComments.deleteComment",
     "group": "pairedComments",
     "when": "editorLineHasComment"
    }
   ]
  }
 }
}
```

📋 Complete Feature List for MVP

✅ Must Have (Launch Blockers)

1. **Command: "Open Paired Comments"** - Opens `.comments` file beside source
2. **Auto-create `.comments` file** - If it doesn't exist, create empty one
3. **Scroll synchronization** - Bidirectional, smooth, no lag
4. **Add comment** - Via command palette or right-click menu
5. **View comments** - Visual indicators (gutter icons) on lines with comments
6. **Edit comment** - Update existing comment text
7. **Delete comment** - Remove comment
8. **Persist comments** - Auto-save to `.comments` file
9. **Hover preview** - Show comment when hovering over gutter icon

🎨 Should Have (Quality of Life)

10. **Keyboard shortcuts** - Fast access to common operations
11. **Comment counter** - Show total comments in status bar

12. **Toggle sync** - Button to enable/disable scroll sync
13. **Comment list** - Quick panel showing all comments in file
14. **Search comments** - Find text within comments

🚀 Nice to Have (Future Iterations)

15. **Tags/categories** - Organize comments by type (TODO, NOTE, etc.)
16. **Multi-line comments** - Support for longer annotations
17. **Line range comments** - Comment on sections, not just single lines
18. **Export to Markdown** - Generate documentation from comments
19. **Git integration** - Commit `.comments` files alongside code

🎨 User Interface Design

Command Palette

```
> Paired Comments: Open
> Paired Comments: Add Comment
> Paired Comments: Edit Comment
> Paired Comments: Delete Comment
> Paired Comments: Toggle Scroll Sync
> Paired Comments: Show All Comments
```

Status Bar Item

```
[💬 5 comments] [🔄 Sync: ON]
```

Gutter Icons

- 💬 Regular comment
- ⚠️ TODO/Warning comment (future)
- ❌ Deleted/Resolved comment (future)

🗂️ File Structure

```
paired-comments/
├── src/
│   ├── extension.ts           # Main entry point
│   ├── commentManager.ts       # CRUD operations
│   ├── pairedViewManager.ts     # Dual-pane logic
│   ├── scrollSyncManager.ts     # Scroll synchronization
│   ├── decorationManager.ts     # Visual indicators
│   ├── commands/
│   │   ├── openPaired.ts
│   │   ├── addComment.ts
│   │   ├── editComment.ts
│   │   └── deleteComment.ts
│   ├── types.ts               # TypeScript interfaces
│   └── utils.ts               # Helper functions
├── icons/
│   └── comment.svg            # Gutter icon
├── package.json              # Extension manifest
├── tsconfig.json
└── README.md
```

🧪 Testing Strategy

Manual Testing Checklist

- [ ] Open `.comments` for file without comments (creates new file)
- [ ] Add comment to line 10, verify it appears in both panes
- [ ] Scroll source file, verify comments pane follows
- [ ] Scroll comments pane, verify source file follows
- [ ] Edit comment, verify change persists after reload
- [ ] Delete comment, verify it's removed from file
- [ ] Close and reopen VS Code, verify comments still load
- [ ] Test with large file (1000+ lines)
- [ ] Test with multiple files open simultaneously
- [ ] Test with files in different languages (.ts, .py, .js, etc.)

Performance Targets

- Scroll sync latency: < 50ms
- Comment file load time: < 100ms for files with 100 comments
- No visible lag when switching between files

📦 Package.json Configuration

json

```
{
  "name": "paired-comments",
  "displayName": "Paired Comments",
  "description": "Sidecar comment files with synchronized viewing",
  "version": "0.1.0",
  "engines": {
    "vscode": "^1.80.0"
  },
  "categories": ["Other"],
  "activationEvents": [
    "onCommand:pairedComments.open",
    "onLanguage:*"
  ],
  "main": "./out/extension.js",
  "contributes": {
    "commands": [
      {
        "command": "pairedComments.open",
        "title": "Paired Comments: Open",
        "icon": "$(comment-discussion)"
      },
      {
        "command": "pairedComments.addComment",
        "title": "Paired Comments: Add Comment"
      },
      {
        "command": "pairedComments.editComment",
        "title": "Paired Comments: Edit Comment"
      },
      {
        "command": "pairedComments.deleteComment",
        "title": "Paired Comments: Delete Comment"
      }
    ],
    "keybindings": [
      {
        "command": "pairedComments.open",
        "key": "ctrl+shift+c",
        "mac": "cmd+shift+c"
      },
      {
        "command": "pairedComments.addComment",
        "key": "ctrl+shift+a",
        "mac": "cmd+shift+a",
        "when": "editorTextFocus"
      }
    ]
  }
}
```

🚀 Development Workflow with Claude Code

Initial Setup

bash

```bash
# Claude will handle this:
1. Initialize VS Code extension project
2. Set up TypeScript configuration
3. Create file structure
4. Install dependencies (vscode, @types/node)
```

Implementation Order

1. **Start with CommentManager** - Get file I/O working first
2. **Build PairedViewManager** - Open side-by-side views
3. **Add ScrollSyncManager** - Get sync working smoothly
4. **Implement DecorationManager** - Visual feedback
5. **Create Commands** - User-facing actions
6. **Polish UX** - Keyboard shortcuts, status bar, etc.

Testing During Development

bash

```
# Run extension in development mode
F5 in VS Code

# Test on real files in your project
# Use the extension host window that opens
```

📊 Success Metrics

Technical Metrics

- Extension activates in < 200ms
- Scroll sync works smoothly (no jitter or lag)
- Comment files remain under 1MB for typical projects
- Zero crashes or data loss

User Experience Metrics

- Users can add their first comment within 30 seconds
- Scroll sync feels "natural" (no perceptible delay)
- Comments persist correctly 100% of the time
- Clear visual feedback for all actions

🔮 Future Roadmap (Post-MVP)

v0.2 - Enhanced Comments

- Tags and categories (TODO, NOTE, WARNING)
- Multi-line comments
- Comment threading (replies)
- Resolved/unresolved status

v0.3 - Collaboration Features

- Author attribution with Git integration
- Team comment filtering (show only my comments)
- Export comments to Markdown/HTML

v0.4 - Smart Features

- Line anchoring (comments follow code through refactors)
- Orphan detection (warn when commented code is deleted)
- Comment search across workspace

v1.0 - Public Release

- Polish UI/UX
- Comprehensive documentation

- Video tutorials
- Submit to VS Code Marketplace

v2.0 - Universal Standard

- JetBrains IDEs plugin
- Vim/Neovim support
- GitHub native rendering
- Language server protocol integration

⚠️ Known Challenges & Solutions

Challenge 1: Scroll Sync Precision

Problem: Source and comments files have different line counts
Solution: Use percentage-based scrolling, not absolute line numbers

Challenge 2: Comment Line Anchoring

Problem: Line numbers shift when code is edited
Solution: For MVP, accept that comments may drift. Post-MVP: AST-aware anchoring

Challenge 3: Large Files

Problem: Slow performance with 10,000+ line files
Solution: Lazy load comments, only render visible range

Challenge 4: Git Conflicts

Problem: `.comments` files might conflict in merges
Solution: Clear documentation, consider conflict resolution UI post-MVP

📚 References & Inspiration

- VS Code Extension API: https://code.visualstudio.com/api
- Diff Viewer Implementation: Study VS Code's built-in diff view
- `.d.ts` files: Precedent for successful sidecar file format
- Source maps (`.map`): Another successful sidecar standard

✅ Pre-Launch Checklist

- [ ] All core commands working
- [ ] Scroll sync smooth and bidirectional
- [ ] Comments persist across restarts
- [ ] Gutter icons display correctly
- [ ] README.md with clear usage instructions
- [ ] Demo GIF showing key features

- [ ] No console errors or warnings
- [ ] Tested on Windows, Mac, and Linux
- [ ] Package extension as `.vsix`
- [ ] Get 3 people to test and provide feedback

🎯 First User Test Script

Goal: Validate that core UX is intuitive

Test Script

1. Open any source file
2. Run command "Paired Comments: Open"
3. Add a comment to line 20
4. Scroll the source file up and down
5. Edit the comment you just added
6. Close both files and reopen
7. Verify comment is still there

Success: User completes all steps without asking questions

💬 Claude Code Instructions

When implementing this MVP, focus on:

1. **Start Simple:** Get basic file I/O working before tackling scroll sync
2. **Test Continuously:** Run the extension after each major component
3. **Use VS Code APIs Properly:** Follow official examples for scroll sync
4. **Keep It Fast:** Performance is critical for user adoption
5. **Error Handling:** Gracefully handle missing files, corrupted JSON, etc.

Priority Order:

1. CommentManager (CRUD + persistence)
2. Basic dual-pane opening
3. Scroll synchronization
4. Visual indicators (gutter icons)
5. Commands and keyboard shortcuts

Questions? Issues? Ideas? Document everything as you build - this will become the foundation for v0.2!

Good luck! 🚀