HCP Portfolio Tracker Technical Specification

Version: 1.1

File: hcp_tracker_technical_spec_v1.1.md

Last Updated: 2025-09-01 19:15:00 UTC

Status: Production Ready

Target Audience: Developers, Technical Implementers

Version Compatibility Matrix

Component	Current Series	Compatibility Notes
Tracker Release	6.5.x series	Production stable
Core Architecture	TrackerCore v1.x	Foundation module
Theme Calculator	v2.9+ series	IPS v3.10 compliant
File Handler	v1.5+ series	Momentum-aware generation
Data Collector	v3.8+ series	Independent system

---

1. System Architecture

1.1 Core Design Principles

Modular Architecture: Each functional area implemented as independent module with versioned API

Single-File Deployment: All modules embedded in HTML for zero-dependency operation

Browser-Native: Pure JavaScript/HTML/CSS with no external libraries or frameworks

State Persistence: localStorage-based state management with JSON serialization

Progressive Enhancement: Graceful degradation when features unavailable

1.2 Module Dependencies

```
TrackerCore v1.x (Foundation)
├── Navigation and state management
├── Step validation and workflow
└── Browser storage abstraction

FileHandler v1.5+ (Data Layer)
├── Sample data generation with momentum patterns
├── File validation and parsing
```

```
└── Nested theme data structure support

ThemeCalculator v2.9+ (Analysis Engine)
├── IPS v3.10 mathematical compliance
├── Enhanced trigger detection algorithms
└── 16-scenario probability generation

DataEditor v1.0+ (User Interface)
├── Modal-based editing system
├── Manual override tracking
└── Data table display and validation

Indicators v1.0+ (Configuration)
├── 13-indicator framework definitions
├── Three-tier signal classification
└── Theme organization and weighting
```

---

2. Data Architecture

2.1 State Structure

javascript

```
TrackerCore.state = {
  // Step validation
  philosophyAcknowledged: boolean,

  // Data layer
  monthlyData: {
    indicators: {
      usd: { dxy: {current, history, ...}, ... },
      innovation: { qqq_spy: {current, history, ...}, ... },
      pe: { forward_pe: {current, history, ...}, ... },
      intl: { acwx_spy: {current, history, ...}, ... }
    },
    trading_status: string,
    timestamp: string
  },

  // Analysis results
  themeProbabilities: { usd: number, ai: number, pe: number, intl: number },
  scenarioProbabilities: Array<{id, name, probability, binary, themes}>,

  // User modifications
  manualOverrides: { [indicatorKey]: {value, reason, timestamp} },

  // Metadata
  lastDataGeneration: string,
  dataScenario: string,
  calculationResults: object
}
```

2.2 Indicator Data Format

Each indicator follows standardized structure:

javascript

```
{
  current: number,          // Current value
  history: number[],        // Historical values (6+ periods minimum)
  freshness: 'fresh'|'stale', // Data quality indicator
  source: string,           // Data source identifier
  name: string,             // Display name
  manual_override?: boolean, // Manual edit flag
  override_reason?: string,  // Edit justification
  override_timestamp?: string // Edit timestamp
}
```

---

3. Calculation Algorithms

3.1 Theme Probability Calculation

Algorithm: IPS v3.10 Enhanced Transition Probability Framework

Key Innovation: Dual approach based on indicator state:

- **Triggered indicators:** Signal strength methodology (higher probability range)
- **Non-triggered indicators:** Time-to-trigger methodology (traditional approach)

Mathematical Framework:

```
For triggered indicators:
  base_probability = 0.70  # Higher baseline
  distance_bonus = min(0.30, abs(distance_to_trigger) * 3)
  momentum_boost = favorable_momentum * 0.25
  result = base_probability + distance_bonus + momentum_boost

For non-triggered indicators:
  months_to_trigger = abs(distance_to_trigger) / momentum_rate
  base_probability = time_decay_function(months_to_trigger)
  direction_adjustment = momentum_direction_factor
  result = base_probability * direction_adjustment
```

3.2 Momentum Calculation

Critical Implementation: 6-period baseline comparison

```
javascript
```

```
calculateMomentum(indicator) {
  const current = indicator.current;
  const baseline = indicator.history[indicator.history.length - 6];
  const percentChange = (current - baseline) / Math.abs(baseline);
  return Math.max(-1, Math.min(1, percentChange * 10));
}
```

3.3 Moving Average Specifications

Frequency-Matched Calculations:

- Daily indicators: Use calendar days for MA periods
- Monthly indicators: Use month-end standardized values
- Quarterly indicators: Interpolation for monthly alignment

Example Specifications:

- DXY: 200D MA vs 400D MA comparison
- QQQ/SPY: 50D MA vs 200D MA comparison
- Forward P/E: 12M MA vs 36M MA comparison
- Productivity: 2Q MA vs 6Q MA comparison

---

4. Module Integration Patterns

4.1 Core Integration Flow

javascript

```
// 1. TrackerCore initialization
TrackerCore.init() → {
  loadState(),
  setupEventListeners(),
  navigateToStep(currentStep)
}

// 2. Data loading integration
FileHandler.generateSampleData() →
TrackerCore.processFileHandlerData() → {
  validateFileHandlerData(),
  displayDataEditor(),
  triggerThemeCalculation()
}

// 3. Analysis integration
ThemeCalculator.calculateThemeAnalysis() →
TrackerCore.state.themeProbabilities →
displayResults()
```

4.2 Event-Driven Architecture

State Change Events:

- Philosophy acknowledgment → Enable step 2 navigation
- Data load → Validate and trigger analysis
- Theme calculation → Enable step 4 navigation
- Manual override → Update calculations and display

Error Handling Pattern:

```
javascript
```

```
try {
  const result = moduleFunction(data);
  if (result.error) {
    displayError(result.error);
    return false;
  }
  updateState(result);
  return true;
} catch (error) {
  logError(error);
  displayFallbackUI();
  return false;
}
```

---

5. Performance Specifications

5.1 Memory Management

State Size Limits:

- Maximum indicator history: 450 data points per indicator
- State object target: <2MB serialized
- localStorage quota: Monitor and warn at 80% usage

Garbage Collection:

- Clear intermediate calculation objects
- Debounce user input events
- Lazy load non-critical display elements

5.2 Computation Optimization

Theme Calculation Performance:

- Target: <200ms for full 4-theme analysis
- Caching: Store intermediate MA calculations
- Parallelization: Process themes independently where possible

5.3 File Size Management

Current Production Metrics:

- Core HTML file: ~112KB (acceptable threshold: <150KB)
- Embedded modules add ~30KB total
- Target deployment size: <200KB total

---

6. Critical Display Specifications

6.1 Scenario Probability Color System

5-Tier Color Classification (PRD v3.4 Standard):

Probability Range	Color	CSS Class	Hex Code	RGB	Usage
> 25%	Dark Green	scenario-very-high	#155724	rgb(21, 87, 36)	Extremely likely scenarios
10-25%	Light Green	scenario-high	#28a745	rgb(40, 167, 69)	Likely scenarios
5-10%	Yellow	scenario-medium	#ffc107	rgb(255, 193, 7)	Moderate probability
1-5%	Light Red	scenario-low	#dc3545	rgb(220, 53, 69)	Unlikely scenarios
< 1%	Dark Gray	scenario-very-low	#6c757d	rgb(108, 117, 125)	Extremely unlikely

6.2 Theme Color Assignments

Fixed theme colors for consistency across all displays:

Theme	Color	CSS Class	Hex Code	RGB
USD Dominance Decline	Red	theme-usd	#dc3545	rgb(220, 53, 69)
AI/Innovation	Blue	theme-ai	#007bff	rgb(0, 123, 255)
P/E Reversion	Yellow	theme-pe	#ffc107	rgb(255, 193, 7)
International	Green	theme-intl	#28a745	rgb(40, 167, 69)

6.3 Data Confidence Indicators

Quality indicators separate from probability colors:

Confidence Level	When to Use	Display	Implementation
HIGH	All indicators fresh, complete history	Green dot	●
MEDIUM	Some stale data or interpolation	Yellow dot	●
LOW	Significant missing data	Red dot	●

6.4 UI Display Requirements

Step 3 Theme Display Standards:

- Theme name with theme color bar
- Percentage probability (large, bold typography)
- **NO confidence labels on probabilities** (per PRD v3.4)
- Theme color fill proportional to probability
- All 16 scenarios in binary order (0000-1111), not probability rank

---

7. Critical Browser Restrictions

7.1 Storage API Limitations

CRITICAL: localStorage/sessionStorage NOT SUPPORTED in Claude.ai artifacts environment

```javascript
// ❌ WRONG - Will cause artifacts to fail
localStorage.setItem('data', JSON.stringify(state));
sessionStorage.setItem('temp', value);

// ✅ CORRECT - Use in-memory storage instead
// For React components:
const [state, setState] = useState(initialState);

// For HTML artifacts:
let memoryStorage = {};
function setItem(key, value) { memoryStorage[key] = value; }
function getItem(key) { return memoryStorage[key]; }
```

Exception Handling:
If user explicitly requests localStorage usage, explain Claude.ai limitations and offer in-memory alternatives.

7.2 Production vs Development Storage

Production Environment (web deployment):

```javascript
// Full localStorage support available
const stateToSave = {
    version: this.version,
    currentStep: this.currentStep,
    state: this.state
};
localStorage.setItem('hcp_tracker_state', JSON.stringify(stateToSave));
```

Development/Artifact Environment:

```javascript
// Memory-only storage pattern
const StorageAdapter = {
    memoryStore: {},
    setItem: function(key, value) {
        this.memoryStore[key] = value;
    },
    getItem: function(key) {
        return this.memoryStore[key] || null;
    }
};
```

---

8. Implementation Error Patterns

8.1 Critical Failure Patterns (From v6.4.1→v6.4.3 Experience)

Pattern 1: Undefined Variable References

Symptom: All theme probabilities show uniform values (15.0%)
Root Cause: Variable scope issues during module embedding

```javascript
```

```
// ❌ WRONG - themeData undefined in embedded function
Object.entries(themeIndicators).forEach(([key, config]) => {
    const indicator = themeData[config.dataKey]; // ❌ themeData undefined
});

// ✅ CORRECT - Proper variable definition
const themeData = dataIndicators[themeName]; // ✅ Define before use
Object.entries(themeIndicators).forEach(([key, config]) => {
    const indicator = themeData[config.dataKey];
});
```

Pattern 2: Momentum Calculation Baseline Errors

Symptom: All theme probabilities show 50.0% (neutral fallback)

Root Cause: Momentum comparison uses wrong historical baseline

```
javascript

// ❌ WRONG - Compare current vs immediate previous (always ≈ same)
const previous = indicator.history[indicator.history.length - 1]; // ❌ Same as current
const momentum = (current - previous) / previous; // ❌ Always ≈ 0

// ✅ CORRECT - Compare current vs 6 periods historical baseline
const previous = indicator.history[indicator.history.length - 6]; // ✅ 6 periods back
const momentum = (current - previous) / previous; // ✅ Shows real momentum
```

Pattern 3: Data Structure Mismatches

Symptom: Module functions fail to find indicators

Root Cause: Different data key naming between modules

```
javascript
```

```
// ❌ WRONG - Inconsistent naming conventions
const indicators = {
    'qqqSpy': data,      // camelCase
    'netMargins': data,  // camelCase
    'qqq_spy': data,     // snake_case - MISMATCH
};

// ✅ CORRECT - Standardize on snake_case for all indicators
const indicators = {
    'qqq_spy': data,     // snake_case
    'net_margins': data, // snake_case
    'forward_pe': data,  // snake_case
};
```

8.2 Debug Console Procedures

Theme Analysis Debugging:

javascript

```
// Enable detailed logging for troubleshooting
console.log('=== THEME MOMENTUM CALCULATION ===');
console.log('Data structure themes:', Object.keys(dataIndicators));
console.log('Individual indicator momentum:', momentum);
console.log('Final theme probabilities:', themeProbabilities);

// Validate data structure
Object.entries(dataIndicators).forEach(([theme, themeData]) => {
    console.log(`${theme}:`, Object.keys(themeData));
});

// Check momentum calculations
Object.values(dataIndicators).forEach(theme => {
    Object.values(theme).forEach(indicator => {
        if (indicator.history && indicator.history.length >= 6) {
            const current = indicator.current;
            const baseline = indicator.history[indicator.history.length - 6];
            console.log(`${indicator.name}: current=${current}, 6-back=${baseline}, momentum=${((current-baseline)/baseli
        }
    });
});
```

Error Detection:

```javascript
// Monitor for common calculation failures
if (Object.values(themeProbabilities).every(p => Math.abs(p - 0.15) < 0.001)) {
    console.error('❌ All themes showing 15% - indicates data validation failure');
}

if (Object.values(themeProbabilities).every(p => Math.abs(p - 0.50) < 0.001)) {
    console.error('❌ All themes showing 50% - indicates momentum calculation failure');
}

if (Object.values(themeProbabilities).some(p => isNaN(p))) {
    console.error('❌ NaN values in theme probabilities - indicates calculation error');
}
```

8.3 Integration Bug Prevention

Pre-Embedding Validation:

```javascript
// Before embedding any new module, test integration
function testModuleIntegration(NewModule) {
    console.log('Testing module:', NewModule.version);

    // Test with known sample data
    const testData = FileHandler.generateSampleData('tech_boom');
    const result = NewModule.processData(testData);

    console.log('Integration test result:', result);
    return result && !result.error;
}
```

Standalone Testing Pattern:

```html
```

```
<!-- Test module functions independently -->
<script src="new_module_v1_0.js"></script>
<script>
// Test module functions independently
console.log('Module loaded:', NewModule.version);
const testResult = NewModule.testFunction();
console.log('Test result:', testResult);
</script>
```

---

9. Browser Compatibility

9.1 Minimum Requirements

JavaScript Features Required:

- ES6+ support (const, let, arrow functions, template literals)
- JSON.parse/stringify with error handling
- localStorage with quota management
- HTML5 file input APIs

Tested Browser Matrix:

- Chrome 90+, Firefox 88+, Safari 14+, Edge 90+
- Mobile: iOS Safari 14+, Chrome Mobile 90+

9.2 Fallback Strategies

localStorage Unavailable:

- Graceful degradation to session-only state
- Warning message to user about persistence loss

File API Unavailable:

- Sample data generation remains functional
- Manual data entry as alternative

---

10. Security Considerations

10.1 Data Privacy

No External Communications: All processing occurs locally in browser
No Persistent Tracking: Only functional localStorage for user convenience
No Data Transmission: User data never leaves local environment

10.2 Input Validation

File Upload Security:

- JSON parsing with try/catch error handling
- Schema validation for expected data structure
- Size limits on uploaded files
- Sanitization of user inputs in manual overrides

---

11. Testing Framework

11.1 Unit Testing Strategy

Module-Level Testing:

```javascript
// Example test structure
testThemeCalculatorV29() {
  const testData = FileHandler.generateSampleData('tech_boom');
  const analysis = ThemeCalculator.calculateThemeAnalysis(testData);

  assert(analysis.themes.ai > 0.70, 'AI theme should be strong in tech boom');
  assert(analysis.scenarios.length === 16, 'Must generate all 16 scenarios');
  assert(Math.abs(totalProbability - 1.0) < 0.01, 'Scenarios must sum to 100%');
}
```

Integration Testing:

- File data flow: FileHandler → TrackerCore → ThemeCalculator → Display
- State persistence: Save → Reload → Verify consistency
- Error recovery: Invalid data → Graceful handling → User notification

11.2 Validation Scenarios

Required Test Cases:

1. All 5 sample scenarios generate expected probability ranges

2. Manual overrides properly update calculations
3. Navigation validation prevents invalid step advancement
4. State persistence survives browser refresh
5. File upload handles malformed JSON gracefully

Expected Result Ranges (Critical for Validation):

- **Tech Boom Scenario:** AI 70-85%, USD 20-35%, P/E 30-45%, International 25-40%
- **USD Strength Scenario:** USD 65-80%, AI 15-30%, P/E 20-35%, International 10-25%
- **P/E Reversion Scenario:** P/E 70-85%, USD 30-45%, AI 25-40%, International 20-35%
- **International Scenario:** International 70-85%, USD 15-30%, AI 20-35%, P/E 25-40%

---

12. Data Collector Integration

12.1 File Format Specifications

Expected JSON Structure:

```javascript
{
  "version": "3.8+",
  "type": "monthly|initialization",
  "timestamp": "ISO_8601_string",
  "indicators": {
    // Nested theme structure as documented in FileHandler v1.5
  },
  "trading_status": "GREEN|YELLOW|RED"
}
```

12.2 Version Compatibility

Backward Compatibility: Support Data Collector v3.6+ output formats
Forward Compatibility: Graceful handling of unknown fields in newer versions
Error Recovery: Clear messaging when file format unsupported

---

13. Deployment Architecture

13.1 Single-File Strategy

Embedded Module Pattern:

```html
<script>
// Module definitions embedded directly
const TrackerCore = { version: '1.2', /* implementation */ };
const FileHandler = { version: '1.5', /* implementation */ };
// etc.
</script>
```

Advantages:

- Zero external dependencies
- Offline functionality
- Simple deployment (single file)
- No CORS issues
- No version synchronization problems

13.2 Development vs Production

Development Mode:

- External module files for easier editing
- Detailed console logging enabled
- Integration test harness available

Production Mode:

- Embedded modules for deployment
- Error logging minimized
- Optimized file size

---

14. Extension Points

14.1 Future Module Integration

Designed Extension Areas:

- Steps 4-10: New modules can hook into existing workflow
- Alternative optimization engines: Modular replacement of portfolio optimization
- Data source adapters: Additional data collector formats
- Export formatters: Multiple output format support

14.2 API Readiness

Module Interfaces:
Each module exposes standardized interface:

```javascript
ModuleName: {
  version: string,
  calculateXXX: function(data) → result,
  displayXXX: function(result, containerId),
  validateXXX: function(data) → validation
}
```

---

15. Change Management

15.1 Version Strategy

Module Independence: Each module maintains separate version numbers
Compatibility Matrix: Document module version compatibility in PRD
Release Coordination: Major releases coordinate compatible module versions

15.2 Update Procedures

Development Updates:

1. Test new module version in isolation
2. Integration test with current module suite
3. Update compatibility matrix
4. Embed in production HTML

Breaking Changes:

- Major version increment required
- Migration guide provided
- Backward compatibility period where possible

End of Technical Specification v1.1