Cross-File “Go to Definition” Design

Overview

This document describes the design for implementing cross-file “Go to Definition” functionality in the PowerBuilder Language Server.

Current Limitations

The current implementation only supports go-to-definition within the same file:
- `findDefinition()` searches only within the current document’s AST
- No workspace-wide symbol index exists
- Cross-file references cannot be resolved

Proposed Solution

1. WorkspaceIndex Class

A new `WorkspaceIndex` class will maintain a global symbol index across all documents in the workspace.

Location: `packages/pb-language-service/src/index/workspace-index.ts`

Responsibilities:
- Index symbols from all parsed documents
- Provide efficient lookup by symbol name
- Track symbol locations (file URI + range)
- Handle incremental updates when documents change
- Support querying by symbol name and kind

Data Structure:

```
interface IndexedSymbol {
  name: string;
  kind: SymbolKind;
  uri: string;
  range: Range;
  selectionRange: Range;
  detail?: string;
}

// Main index: Map<symbolName, IndexedSymbol[]>
// Multiple symbols can have the same name (e.g., same function in different files)
```

2. Integration with TreeSitterManager

The `TreeSitterManager` will be enhanced to:
- Accept a reference to `WorkspaceIndex` in its constructor
- Update the index whenever a document is parsed/updated
- Remove symbols from index when document is closed

3. Enhanced findDefinition

The `findDefinition` function will implement a two-phase search:

Phase 1: Local Search (current behavior)
- Search for definition within the current file
- Fast path for local symbols

Phase 2: Workspace Search (new)
- If not found locally, query the WorkspaceIndex
- Return first matching symbol from other files
- Prefer exact matches by kind if possible

4. PowerBuilder-Specific Handling

The implementation will handle PowerBuilder constructs:

Supported Symbol Types:
- Functions (global and object methods)
- Variables (instance and global)
- Types/Objects (user objects, custom types)
- Events
- DataWindows (if present in grammar)

Name Resolution Strategy:
1. Exact name match
2. Case-insensitive fallback (PowerBuilder is case-insensitive)
3. Qualified name resolution (e.g., `object.method`)

Implementation Plan

Phase 1: Core Infrastructure

1. Create `WorkspaceIndex` class
2. Integrate with `TreeSitterManager`
3. Update `PowerBuilderLanguageService` to manage the index

Phase 2: Enhanced Definition Provider

1. Modify `findDefinition()` to support cross-file lookup
2. Implement workspace-wide symbol resolution
3. Handle edge cases (multiple definitions, ambiguous names)

Phase 3: Testing & Refinement

1. Create test fixtures with multiple PowerBuilder files
2. Test cross-file navigation scenarios
3. Performance testing with large workspaces
4. Documentation

API Changes

PowerBuilderLanguageService

```
class PowerBuilderLanguageService {
  private workspaceIndex: WorkspaceIndex;

  // New: Get all symbols in workspace
  getWorkspaceSymbols(): IndexedSymbol[]

  // New: Find symbols by name across workspace
  findSymbolsByName(name: string): IndexedSymbol[]

  // Enhanced: Now searches across files
  findDefinition(uri: string, position: Position): Location | null
}
```

WorkspaceIndex

```
class WorkspaceIndex {
  // Index symbols from a document
  indexDocument(uri: string, symbols: Symbol[]): void

  // Remove document from index
  removeDocument(uri: string): void

  // Find symbols by name
  findSymbols(name: string): IndexedSymbol[]

  // Get all symbols
  getAllSymbols(): IndexedSymbol[]

  // Clear entire index
  clear(): void
}
```

Performance Considerations

1. **Incremental Updates**: Only re-index changed documents
2. **Efficient Data Structures**: Use Map for O(1) lookups
3. **Lazy Indexing**: Index documents only when opened/changed
4. **Memory Management**: Consider weak references for very large workspaces

Backward Compatibility

- Existing single-file functionality remains unchanged
- Graceful degradation if index is empty or unavailable
- No breaking changes to public API

Future Enhancements

1. **Workspace symbol search** (LSP `workspace/symbol`)
2. **Find all references** (LSP `textDocument/references`)

3. **Rename refactoring** (LSP `textDocument/rename`)
4. **Call hierarchy** (LSP `textDocument/prepareCallHierarchy`)
5. **Type definitions** (LSP `textDocument/typeDefinition`)

Testing Strategy

1. **Unit Tests**:
   - WorkspaceIndex operations
   - Symbol indexing and lookup
2. **Integration Tests**:
   - Cross-file navigation
   - Index updates on document changes
3. **Test Fixtures**:
   - Multiple .sru files with cross-references
   - Functions calling each other across files
   - Object inheritance scenarios

Risks & Mitigations

Risk	Mitigation
Performance with large workspaces	Implement lazy indexing and consider pagination
Memory usage	Monitor and optimize data structures, use weak references
Ambiguous symbol names	Return multiple locations, let user choose
Index out of sync	Implement validation and recovery mechanisms

Timeline Estimate

- Core Infrastructure: 2-3 hours
- Enhanced Definition Provider: 1-2 hours
- Testing & Documentation: 1-2 hours
- **Total**: ~4-7 hours

Success Criteria

1. ✅ Users can navigate from a symbol usage to its definition in another file
2. ✅ Index stays in sync with document changes
3. ✅ No performance degradation for single-file operations
4. ✅ Comprehensive test coverage
5. ✅ Clear documentation for maintainers