Health and Wellness Lifestyle Solver
Complete Reference Manual

Shyamal Chandra

November 18, 2025

Contents

1 Introduction

This reference manual provides comprehensive API documentation for the Health and Wellness Lifestyle Solver system. It includes detailed information about all data models, classes, methods, and usage examples for working with diet optimization, exercise planning, medical test analysis, cognitive assessment, sensory health monitoring, and time-based planning.

2 Getting Started

2.1 Installation

The system is a Swift Package. Add it to your project:

```
// Package.swift
dependencies: [
    .package(path: "path/to/diet-solver")
]
```

2.2 Basic Import

```
import DietSolver
```

3 Core Data Models

3.1 HealthData

The central health information repository.

3.1.1 Structure

```
struct HealthData: Codable {
    // Basic Health Metrics
    var glucose: Double?              // mg/dL
    var hemoglobin: Double?          // g/dL
    var cholesterol: Double?          // mg/dL
    var bloodPressure: BloodPressure?
    var age: Int
    var gender: Gender
    var weight: Double                // kg
    var height: Double                // cm
    var activityLevel: ActivityLevel

    // Exercise & Fitness
    var exerciseLogs: [DailyExerciseLog] = []
    var exerciseGoals: ExerciseGoals?
    var muscleMass: Double?           // kg
    var bodyFatPercentage: Double?    // %

    // Sexual Health
    var sexualHealth: SexualHealth?

    // Mental Health
    var mentalHealth: MentalHealth?

    // Medical Tests
    var medicalTests: MedicalTestCollection
    var medicalAnalysis: MedicalAnalysisReport?

```

```
    // Cognitive Assessment
    var cognitiveAssessments: [CognitiveAssessment] = []

    // Medical Specialties
    var medicalSpecialties: MedicalSpecialtyCollection

    // Sleep Data
    var sleepRecords: [SleepRecord] = []
    var sleepAnalysis: SleepAnalysis?

    // Journal Data
    var journalCollection: JournalCollection

    // Vision Data
    var visionPrescription: VisionPrescription?
    var dailyVisionChecks: [DailyVisionCheck] = []
    var visionGameSessions: [VisionGameSession] = []
    var visionAnalysis: VisionAnalysisReport?

    // Hearing Data
    var hearingPrescription: HearingPrescription?
    var dailyAudioHearingTests: [DailyAudioHearingTest] = []
    var musicHearingSessions: [MusicHearingSession] = []
    var hearingAnalysis: HearingAnalysisReport?

    // Tactile Data
    var tactilePrescription: TactilePrescription?
    var dailyTactileTests: [DailyTactileTest] = []
    var tactileVitalitySessions: [TactileVitalitySession] = []
    var tactileAnalysis: TactileAnalysisReport?

    // Tongue Data
    var tonguePrescription: TonguePrescription?
    var dailyTongueTests: [DailyTongueTest] = []
    var tongueVitalitySessions: [TongueVitalitySession] = []
    var tongueAnalysis: TongueAnalysisReport?

    // HealthKit Data
    var healthKitBiomarkers: [HealthKitBiomarkers] = []

    // Eating Metrics
    var eatingMetrics: [EatingMetrics] = []

    // Emotional Health
    var emotionalHealth: [EmotionalHealth] = []
}
```

3.1.2 Key Methods

```
// Calculate adjusted nutrient requirements based on health data
func adjustedNutrientRequirements() -> NutrientRequirements

// Calculate BMR (Basal Metabolic Rate)
func calculateBMR() -> Double

// Calculate TDEE (Total Daily Energy Expenditure)
func calculateTDEE() -> Double
```

3.1.3 Example Usage

```swift
// Initialize health data
var healthData = HealthData(
    glucose: 95.0,
    hemoglobin: 14.5,
    cholesterol: 180.0,
    bloodPressure: HealthData.BloodPressure(systolic: 120, diastolic: 80),
    age: 35,
    gender: .male,
    weight: 75.0,
    height: 175.0,
    activityLevel: .moderate
)

// Get adjusted nutrient requirements
let requirements = healthData.adjustedNutrientRequirements()
print("Daily calories: \(requirements.calories)")
```

3.2 Food and Nutrient Models

3.2.1 Food

```swift
struct Food: Codable, Identifiable, Hashable {
    let id: UUID
    let name: String
    let category: FoodCategory
    let nutrientContent: NutrientContent
    let properties: FoodProperties
    let seasonalAvailability: [Season]
}

enum FoodCategory: String, Codable {
    case vegetable, fruit, grain, protein
    case legume, nut, herb, spice, dairy
}
```

3.2.2 NutrientContent

```swift
struct NutrientContent: Codable {
    // Macronutrients
    var calories: Double = 0
    var protein: Double = 0         // grams
    var carbohydrates: Double = 0   // grams
    var fat: Double = 0             // grams
    var fiber: Double = 0           // grams

    // Vitamins
    var vitaminA: Double = 0         // IU
    var vitaminC: Double = 0        // mg
    var vitaminD: Double = 0        // IU
    var vitaminE: Double = 0        // mg
    var vitaminK: Double = 0        // mcg
    var thiamine: Double = 0         // mg (B1)
    var riboflavin: Double = 0       // mg (B2)
    var niacin: Double = 0           // mg (B3)
    var vitaminB6: Double = 0        // mg
    var folate: Double = 0           // mcg
    var vitaminB12: Double = 0      // mcg
    var biotin: Double = 0          // mcg
    var pantothenicAcid: Double = 0 // mg

    // Minerals
```

```
    var calcium: Double = 0         // mg
    var iron: Double = 0            // mg
    var magnesium: Double = 0       // mg
    var phosphorus: Double = 0      // mg
    var potassium: Double = 0       // mg
    var sodium: Double = 0          // mg
    var zinc: Double = 0            // mg
    var copper: Double = 0          // mg
    var manganese: Double = 0       // mg
    var selenium: Double = 0        // mcg
    var iodine: Double = 0         // mcg
    var chromium: Double = 0        // mcg
    var molybdenum: Double = 0     // mcg

    // Operators
    static func + (lhs: NutrientContent, rhs: NutrientContent) ->
    NutrientContent
    func scaled(by factor: Double) -> NutrientContent
}
```

4 Diet Solver

4.1 DietSolver Class

Main optimization engine for diet planning.

4.1.1 Initialization

```
let solver = DietSolver()
```

4.1.2 Main Method

```
func solve(healthData: HealthData, season: Season) -> DailyDietPlan
```

4.1.3 Example Usage

```
// Create solver
let solver = DietSolver()

// Solve diet plan
let season = Season.spring
let dietPlan = solver.solve(healthData: healthData, season: season)

// Access results
print("Meals: \(dietPlan.meals.count)")
print("Total calories: \(dietPlan.totalNutrients.calories)")
print("Taste score: \(dietPlan.overallTasteScore)")
print("Digestion score: \(dietPlan.overallDigestionScore)")

// Access individual meals
for meal in dietPlan.meals {
    print("\(meal.name): \(meal.mealType)")
    for item in meal.items {
        print("  - \(item.food.name): \(item.amount)g")
    }
}
```

4.2 DailyDietPlan

Result structure from diet solver.

```
struct DailyDietPlan: Codable {
    var meals: [Meal]
    let season: Season
    let healthData: HealthData

    var totalNutrients: NutrientContent
    var overallTasteScore: Double
    var overallDigestionScore: Double
}
```

5 Exercise Planning

5.1 ExercisePlanner Class

Generates personalized weekly exercise plans.

5.1.1 Initialization

```
let planner = ExercisePlanner()
```

5.1.2 Key Methods

```
// Generate weekly exercise plan
func generateWeeklyPlan(
    for healthData: HealthData,
    goals: ExerciseGoals
) -> ExercisePlan

// Get exercise recommendations for specific day
func recommendExercises(
    for healthData: HealthData,
    dayOfWeek: Int  // 0 = Sunday, 6 = Saturday
) -> [ExerciseActivity]
```

5.1.3 ExerciseGoals

```
struct ExerciseGoals: Codable {
    var weeklyCardioMinutes: Int = 150
    var weeklyStrengthSessions: Int = 2
    var weeklyFlexibilityMinutes: Int = 60
    var weeklyMindBodyMinutes: Int = 120
}
```

5.1.4 Example Usage

```
// Create planner
let planner = ExercisePlanner()

// Define goals
let goals = ExerciseGoals(
    weeklyCardioMinutes: 150,
    weeklyStrengthSessions: 2,
    weeklyFlexibilityMinutes: 60,
```

```
    weeklyMindBodyMinutes: 120
)

// Generate plan
let plan = planner.generateWeeklyPlan(for: healthData, goals: goals)

// Access weekly plan
for dayPlan in plan.weeklyPlan {
    print("\(dayPlan.dayOfWeek):")
    for activity in dayPlan.activities {
        print("  - \(activity.activity.name): \(activity.duration) min")
    }
}

// Get recommendations
let recommendations = planner.recommendExercises(
    for: healthData,
    dayOfWeek: 0  // Sunday
)
```

6 Medical Test Analysis

6.1 MedicalAnalyzer Class

Analyzes medical test results and generates recommendations.

6.1.1 Initialization

```
let analyzer = MedicalAnalyzer()
```

6.1.2 Key Methods

```
// Analyze medical tests
func analyze(medicalTests: MedicalTestCollection) -> MedicalAnalysisReport

// Analyze specific test type
func analyzeBloodTest(_ test: BloodTest) -> BloodTestAnalysis
```

6.2 Medical Test Models

6.2.1 BloodTest

```
struct BloodTest: Codable, Identifiable {
    let id: UUID
    var date: Date
    var testType: BloodTestType
    var bloodSource: BloodSource
    var cbc: CBCResults?
    var metabolicPanel: MetabolicPanelResults?
    var lipidPanel: LipidPanelResults?
    var liverFunction: LiverFunctionResults?
    var kidneyFunction: KidneyFunctionResults?
    var thyroid: ThyroidResults?
    var vitamins: VitaminResults?
    var minerals: MineralResults?
    var hormones: HormoneResults?
    var inflammatoryMarkers: InflammatoryMarkerResults?
}
```

6.2.2 Example Usage

```
// Create blood test
var bloodTest = BloodTest(
    date: Date(),
    testType: .comprehensive,
    bloodSource: BloodSource.venous
)

// Add CBC results
bloodTest.cbc = CBCResults(
    whiteBloodCellCount: 7.0,
    redBloodCellCount: 4.5,
    hemoglobin: 14.5,
    hematocrit: 42.0,
    plateletCount: 250.0
)

// Add to health data
healthData.medicalTests.bloodTests.append(bloodTest)

// Analyze
let analyzer = MedicalAnalyzer()
let report = analyzer.analyze(medicalTests: healthData.medicalTests)
healthData.medicalAnalysis = report

// Access recommendations
if let report = healthData.medicalAnalysis {
    for issue in report.issues {
        print("Issue: \(issue.description)")
        print("Severity: \(issue.severity)")
    }
    for recommendation in report.recommendations {
        print("Recommendation: \(recommendation)")
    }
}
```

7 Cognitive Assessment

7.1 CognitiveAnalyzer Class

Analyzes cognitive assessments.

7.1.1 Initialization

```
let analyzer = CognitiveAnalyzer()
```

7.1.2 Key Methods

```
func analyze(_ assessment: CognitiveAssessment) -> CognitiveAnalysisReport
```

7.2 CognitiveAssessment Model

```
struct CognitiveAssessment: Codable, Identifiable {
    let id: UUID
    var date: Date
    var iq: IQAssessment?
    var eq: EQAssessment?
```

```
    var cq: CQAssessment?
    var spatialReasoning: SpatialReasoningAssessment?
    var temporalReasoning: TemporalReasoningAssessment?
    var tacticalProblemSolving: TacticalProblemSolvingAssessment?
    var strategicProblemSolving: StrategicProblemSolvingAssessment?
    var psychicCapabilities: PsychicCapabilitiesAssessment?
}
```

7.2.1 Example Usage

```
// Create cognitive assessment
var assessment = CognitiveAssessment(date: Date())

// Add IQ assessment
assessment.iq = IQAssessment(
    fullScaleIQ: 115,
    verbalIQ: 120,
    performanceIQ: 110,
    workingMemory: 118,
    processingSpeed: 112
)

// Add to health data
healthData.cognitiveAssessments.append(assessment)

// Analyze
let analyzer = CognitiveAnalyzer()
let report = analyzer.analyze(assessment)

// Access results
print("Strengths: \(report.strengths)")
print("Areas for improvement: \(report.areasForImprovement)")
print("Recommendations: \(report.recommendations)")
```

8 Sensory Health

8.1 Hearing Health

8.1.1 HearingPrescription

```
struct HearingPrescription: Codable {
    var date: Date
    var expirationDate: Date?
    var rightEar: EarPrescription
    var leftEar: EarPrescription
    var hearingAidSettings: HearingAidSettings?
    var notes: String?
}
```

8.1.2 DailyAudioHearingTest

```
struct DailyAudioHearingTest: Codable, Identifiable {
    let id: UUID
    var date: Date
    var time: Date
    var rightEar: EarTest
    var leftEar: EarTest
    var bothEars: BothEarsTest
    var testType: TestType
```

```
    var device: TestDevice
    var environment: TestEnvironment
    var notes: String?
}
```

8.1.3 MusicHearingSession

```
struct MusicHearingSession: Codable, Identifiable {
    let id: UUID
    var date: Date
    var startTime: Date
    var endTime: Date?
    var duration: Double  // minutes
    var musicType: MusicType
    var genre: String?
    var volumeLevel: VolumeLevel
    var device: MusicDevice
    var listeningMode: ListeningMode
    var hearingProtection: Bool
    var hearingFatigue: HearingFatigueLevel?
    var enjoyment: EnjoymentLevel
    var notes: String?
}
```

8.1.4 HearingAnalyzer

```
let analyzer = HearingAnalyzer()
let report = analyzer.analyze(hearingData: healthData.hearingData)
```

8.2 Vision Health

8.2.1 VisionPrescription

```
struct VisionPrescription: Codable {
    var date: Date
    var expirationDate: Date?
    var rightEye: EyePrescription
    var leftEye: EyePrescription
    var notes: String?
}
```

8.2.2 DailyVisionCheck

```
struct DailyVisionCheck: Codable, Identifiable {
    let id: UUID
    var date: Date
    var time: Date
    var rightEye: EyeCheck
    var leftEye: EyeCheck
    var bothEyes: BothEyesCheck
    var testType: VisionTestType
    var device: VisionTestDevice
    var environment: VisionTestEnvironment
    var notes: String?
}
```

8.3 Tactile Health

8.3.1 TactilePrescription

```
struct TactilePrescription: Codable {
    var date: Date
    var expirationDate: Date?
    var bodyParts: [BodyPartTactilePrescription]
    var notes: String?
}
```

8.3.2 DailyTactileTest

```
struct DailyTactileTest: Codable, Identifiable {
    let id: UUID
    var date: Date
    var time: Date
    var bodyParts: [BodyPartTactileTest]
    var testType: TactileTestType
    var notes: String?
}
```

8.4 Tongue Health

8.4.1 TonguePrescription

```
struct TonguePrescription: Codable {
    var date: Date
    var expirationDate: Date?
    var appearance: TongueAppearance
    var tasteSensitivity: TasteSensitivityAssessment
    var mobility: TongueMobilityAssessment
    var notes: String?
}
```

8.4.2 DailyTongueTest

```
struct DailyTongueTest: Codable, Identifiable {
    let id: UUID
    var date: Date
    var time: Date
    var appearance: TongueAppearance
    var tasteSensitivity: TasteSensitivityTest
    var mobility: TongueMobilityTest
    var symptoms: [TongueSymptom]
    var notes: String?
}
```

9 Time-Based Planning

9.1 TimeBasedPlanner Class

Generates planning sessions for day/week/month.

9.1.1 Initialization

```
let planner = TimeBasedPlanner()
```

9.1.2 Key Methods

```
// Generate day start planning session
func generateDayStartSession(
    for date: Date,
    healthData: HealthData,
    journalAnalysis: JournalAnalysisReport?
) -> TimeBasedPlanningSession

// Generate day end planning session
func generateDayEndSession(
    for date: Date,
    healthData: HealthData,
    journalAnalysis: JournalAnalysisReport?
) -> TimeBasedPlanningSession

// Generate week start planning session
func generateWeekStartSession(
    for date: Date,
    healthData: HealthData,
    journalAnalysis: JournalAnalysisReport?
) -> TimeBasedPlanningSession

// Generate week end planning session
func generateWeekEndSession(
    for date: Date,
    healthData: HealthData,
    journalAnalysis: JournalAnalysisReport?
) -> TimeBasedPlanningSession

// Generate month start planning session
func generateMonthStartSession(
    for date: Date,
    healthData: HealthData,
    journalAnalysis: JournalAnalysisReport?
) -> TimeBasedPlanningSession

// Generate month end planning session
func generateMonthEndSession(
    for date: Date,
    healthData: HealthData,
    journalAnalysis: JournalAnalysisReport?
) -> TimeBasedPlanningSession
```

9.1.3 Example Usage

```
let planner = TimeBasedPlanner()

// Generate day start session
let dayStart = planner.generateDayStartSession(
    for: Date(),
    healthData: healthData,
    journalAnalysis: journalAnalysis
)

// Access planning content
print("Tasks: \(dayStart.tasks)")
print("Priorities: \(dayStart.priorities)")
print("Reflections: \(dayStart.reflections)")

// Generate week start session
let weekStart = planner.generateWeekStartSession(
```

```
    for: Date(),
    healthData: healthData,
    journalAnalysis: journalAnalysis
)
```

10 Long-Term Planning

10.1 LongTermPlanner Class

Generates transformation plans.

10.1.1 Initialization

```
let planner = LongTermPlanner()
```

10.1.2 Key Methods

```
// Generate long-term plan
func generatePlan(
    duration: PlanDuration,
    difficulty: PlanDifficulty,
    healthData: HealthData,
    goals: TransformationGoals
) -> LongTermPlan
```

10.1.3 PlanDuration

```
enum PlanDuration: String, Codable {
    case threeMonths = "3 Months"
    case sixMonths = "6 Months"
    case oneYear = "1 Year"
    case twoYears = "2 Years"
    case fiveYears = "5 Years"
    case tenYears = "10 Years"
}
```

10.1.4 PlanDifficulty

```
enum PlanDifficulty: String, Codable {
    case gentle = "Gentle"
    case moderate = "Moderate"
    case aggressive = "Aggressive"
    case extreme = "Extreme"
}
```

10.1.5 Example Usage

```
let planner = LongTermPlanner()

let goals = TransformationGoals(
    weightGoal: 70.0,
    muscleMassGoal: 20.0,
    cardiovascularGoal: .improve,
    mentalHealthGoal: .improve
)
```

```

let plan = planner.generatePlan(
    duration: .oneYear,
    difficulty: .moderate,
    healthData: healthData,
    goals: goals
)

// Access plan
print("Plan duration: \(plan.duration)")
print("Phases: \(plan.phases.count)")
for phase in plan.phases {
    print("Phase: \(phase.name)")
    print("Duration: \(phase.duration) days")
}
```

11 Journal Analysis

11.1 JournalAnalyzer Class

Analyzes journal entries.

11.1.1 Initialization

```
let analyzer = JournalAnalyzer()
```

11.1.2 Key Methods

```
// Analyze journal collection
func analyze(_ journalCollection: JournalCollection) -> JournalAnalysisReport
```

11.1.3 Example Usage

```
// Add journal entry
var entry = JournalEntry(
    date: Date(),
    type: .unstructured,
    content: "Feeling great today after morning workout"
)
healthData.journalCollection.entries.append(entry)

// Analyze
let analyzer = JournalAnalyzer()
let report = analyzer.analyze(healthData.journalCollection)

// Access insights
print("Themes: \(report.themes)")
print("Emotional trends: \(report.emotionalTrends)")
print("Insights: \(report.insights)")
```

12 Generators

12.1 RecipeGenerator

Generates cooking instructions.

```
let recipe = RecipeGenerator.generateRecipe(for: meal)
print(recipe.instructions)
```

12.2 NutritionFactsGenerator

Generates nutrition labels.

```
let facts = NutritionFactsGenerator.generateNutritionFacts(
    for: dietPlan,
    requirements: healthData.adjustedNutrientRequirements()
)
print(facts.label)
```

12.3 SongGenerator

Generates songs about meals.

```
let song = SongGenerator.generateSong(for: dietPlan)
print(song.lyrics)
```

13 Data Persistence

13.1 Encoding

```
// Encode health data
func saveHealthData(_ healthData: HealthData) throws {
    let encoder = JSONEncoder()
    encoder.dateEncodingStrategy = .iso8601
    encoder.outputFormatting = .prettyPrinted

    let data = try encoder.encode(healthData)
    try data.write(to: fileURL)
}
```

13.2 Decoding

```
// Decode health data
func loadHealthData() throws -> HealthData {
    let data = try Data(contentsOf: fileURL)
    let decoder = JSONDecoder()
    decoder.dateDecodingStrategy = .iso8601

    return try decoder.decode(HealthData.self, from: data)
}
```

14 Calendar Integration

14.1 CalendarScheduler

Manages calendar event creation.

14.1.1 Initialization

```
let scheduler = CalendarScheduler()
```

14.1.2 Key Methods

```
// Request calendar access
func requestCalendarAccess() async -> Bool

// Create calendar event
func createEvent(
    title: String,
    startDate: Date,
    endDate: Date,
    notes: String?
) throws -> EKEvent

// Schedule planning session
func schedulePlanningSession(_ session: TimeBasedPlanningSession) throws
```

15 Best Practices

15.1 Health Data Management

- Always validate health data before optimization
- Update medical tests regularly for accurate recommendations
- Maintain complete exercise logs for better planning
- Track sensory health consistently for trend analysis

15.2 Optimization

- Use appropriate season for food availability
- Review nutrient requirements after medical test updates
- Adjust exercise goals based on progress
- Consider all health factors in planning

15.3 Data Persistence

- Save health data regularly
- Use ISO8601 date encoding for compatibility
- Validate data after loading
- Handle encoding/decoding errors gracefully

16 API Reference Summary

16.1 Core Classes

Class	Purpose
`DietSolver`	Diet optimization engine
`ExercisePlanner`	Exercise plan generation
`MedicalAnalyzer`	Medical test analysis

`CognitiveAnalyzer`	Cognitive assessment analysis
`HearingAnalyzer`	Hearing health analysis
`VisionAnalyzer`	Vision health analysis
`TactileAnalyzer`	Tactile health analysis
`TongueAnalyzer`	Tongue health analysis
`SleepAnalyzer`	Sleep pattern analysis
`JournalAnalyzer`	Journal entry analysis
`TimeBasedPlanner`	Planning session generation
`LongTermPlanner`	Transformation plan generation
`CalendarScheduler`	Calendar event management

16.2 Key Data Models

Model	Purpose
`HealthData`	Central health repository
`Food`	Food database entry
`NutrientContent`	Nutrient information
`DailyDietPlan`	Optimized meal plan
`ExercisePlan`	Weekly exercise plan
`MedicalTestCollection`	Medical test data
`CognitiveAssessment`	Cognitive test results
`HearingData`	Hearing health data
`VisionData`	Vision health data
`TactileData`	Tactile health data
`TongueData`	Tongue health data
`TimeBasedPlanningSession`	Planning session data
`LongTermPlan`	Transformation plan
`JournalCollection`	Journal entries

17 Complete Workflow Example

```
// 1. Initialize health data
var healthData = HealthData(
    age: 35,
    gender: .male,
    weight: 75.0,
    height: 175.0,
    activityLevel: .moderate
)

// 2. Add medical test
var bloodTest = BloodTest(date: Date(), testType: .comprehensive, bloodSource:
    .venous)
bloodTest.cbc = CBCResults(hemoglobin: 14.5, ...)
healthData.medicalTests.bloodTests.append(bloodTest)

// 3. Analyze medical tests
let medicalAnalyzer = MedicalAnalyzer()
healthData.medicalAnalysis = medicalAnalyzer.analyze(medicalTests: healthData.
    medicalTests)

// 4. Solve diet
let solver = DietSolver()
```

```
let dietPlan = solver.solve(healthData: healthData, season: .spring)

// 5. Generate exercise plan
let exercisePlanner = ExercisePlanner()
let goals = ExerciseGoals()
let exercisePlan = exercisePlanner.generateWeeklyPlan(for: healthData, goals:
    goals)

// 6. Generate planning session
let timePlanner = TimeBasedPlanner()
let dayStart = timePlanner.generateDayStartSession(
    for: Date(),
    healthData: healthData,
    journalAnalysis: nil
)

// 7. Generate long-term plan
let longTermPlanner = LongTermPlanner()
let transformationGoals = TransformationGoals(...)
let longTermPlan = longTermPlanner.generatePlan(
    duration: .oneYear,
    difficulty: .moderate,
    healthData: healthData,
    goals: transformationGoals
)

// 8. Save data
let encoder = JSONEncoder()
encoder.dateEncodingStrategy = .iso8601
let data = try encoder.encode(healthData)
try data.write(to: fileURL)
```

18 Troubleshooting

18.1 Common Issues

18.1.1 Diet Optimization Fails

- Check that health data has valid age, weight, height
- Ensure season is appropriate for food availability
- Verify nutrient requirements are reasonable

18.1.2 Exercise Recommendations Missing

- Verify exercise goals are set
- Check that health data includes necessary information
- Ensure exercise database is properly initialized

18.1.3 Medical Analysis Errors

- Validate medical test data format
- Check that test results are within expected ranges
- Ensure test dates are valid

19 Conclusion

This reference manual provides comprehensive documentation for the Health and Wellness Lifestyle Solver system. For additional information, refer to the complete paper and source code documentation.