7단계: 개발/유지보수 가이드

실제로 코드를 수정할 때 알아야 할 주의사항, 패턴, 팁을 정리합니다.

코드 수정 시 핵심 주의사항

1. DI 파라미터 이름 = 의존성 이름

이 프로젝트는 함수 파라미터 이름으로 의존성을 주입합니다. 이름을 바꾸면 DI가 깨집니다.

```
//  이렇게 하면 안 됨 - 'ctx'라는 이름의 의존성은 등록되어 있지 않음
module.exports = function(ctx, dispatcher) { ... }

//  반드시 등록된 이름과 일치해야 함
module.exports = function(applicationContext, pageDispatcher) { ... }
```

등록된 이름은 bin/rodi-x-svc.js의 moduleProvider.constant() / moduleProvider.factory() 호출부에서 확인하세요.

등록된 Constants:

mainConfig, frameworkConfig, assemblyPath, assemblyTempPath, filePath, pythonPath, javaPath

등록된 Factories:

persistedApi, robotApi, uiApi, index, xPluginExceptionHandler, applicationContext, pageDispatcher, sdkCommand, sdkService, extensionPageService, programNodePageService, daemonService, widgetNodePageService, serviceInfoProvider, utils, virtualCaller, ipcLogger

2. externals/ 수정 시 영향 범위

externals/는 Git 서브모듈이며 다른 rodi 서비스와 공유됩니다.

```
externals/ 수정 영향 범위:
  core-framework    → 모든 rodi 서비스
  persisted-domain  → rodi-data-svc, rodi-control-svc 등
  robot-api         → rodi-control-svc 등
  ui-api            → rodi-web-svc 등
  script-interpreter → rodi-control-svc
  data-migration    → rodi-data-svc
```

수정 전 반드시:

1. 어떤 서비스가 이 서브모듈을 사용하는지 확인

2. 변경이 역호환되는지 검토

3. 영향받는 서비스에서 테스트

3. MQTT 토픽은 3곳을 동시에 수정

새 이벤트를 추가하거나 토픽을 변경할 때:

```
config/framework.json          ← 토픽 정의
modules/interface/sdkCommand.js ← 핸들러 구현
modules/interface/sdkService.js ← 구독 등록
```

하나라도 빠지면 이벤트가 동작하지 않거나 무시됩니다.

4. 전역 변수 의존

다음 전역 변수들이 코드 곳곳에서 사용됩니다:

```
$logger       // Winston 로거 (전역)
$lodash       // Lodash 유틸리티 (전역)
$loggerData   // 데이터 로거 (전역)
require_global // 전역 모듈 require (core-framework이 등록)
TGOS_CODES    // 에러 코드 레지스트리 (전역)
Exception     // 커스텀 예외 클래스 (전역)
```

이것들은 core-framework 초기화 시 global에 등록됩니다. 일반 Node.js 방식(require)이 아니므로 주의하세요.

5. self 바인딩 패턴

모듈 내에서 self는 core-framework의 컨텍스트를 가리킵니다:

```
module.exports = function(dependency1, dependency2) {
  var self = this;  // ← core-framework가 바인딩한 컨텍스트

  // self를 통해 전역 서비스 접근
  self.$eventManager.subscribe(...);
  self.$aggregator.emit(...);
  self.$logger.info(...);
};
```

화살표 함수(=>)를 사용하면 this 바인딩이 달라지므로, 최상위 레벨에서는 function 키워드를 사용해야 합니다.

자주 하는 작업별 가이드

새 MQTT 커맨드 추가

Step 1: config/framework.json
→ eventManager.sub_list에 추가

```
"rodix_my_new_command": {
  "run": true,
  "topic": "rodix/my/new/command",
  "qos": 2,
  "log": false
}
```

Step 2: modules/interface/sdkCommand.js
→ commands 객체에 핸들러 추가

```
commands.rodix_my_new_command = function(data, finishCb) {
    // 비즈니스 로직
    let result = someService.doSomething(data);
    if(finishCb) finishCb(result);
};
```

Step 3: modules/interface/sdkService.js
→ runServices() 내에 구독 추가

```
self.$eventManager.subscribe('rodix_my_new_command', subscribeEvent);
```

새 도메인 모델 추가

Step 1: modules/applicationContext/inject/domain/api/ 아래에 모델 파일 생성
→ myNewModel/index.js

Step 2: modules/applicationContext/inject/index.js
→ APIObject에 새 모델 등록 (get 메서드에 lazy-loading 추가)

Step 3: 플러그인에서 rodiAPI를 통해 접근
→ rodiAPI.getMyNewModel()

플러그인 디버깅

방법 1: plugins.debug/ 디렉토리 사용
→ plugins.debug/ 아래에 플러그인 폴더를 직접 배치
→ ASAR 추출 없이 바로 로딩 (핫 리로드 가능)

방법 2: 로그 확인
→ $logger로 출력된 로그는 {repository}/logs/rodi-x-svc/ 에 저장
→ 콘솔: debug 레벨
→ 파일: verbose 레벨 (더 상세)
→ 최대 파일 크기: 5MB, 최대 10개 파일 로테이션

방법 3: virtualCaller (테스트 모드)
→ modules/test/virtualCaller.js를 통해 이벤트 직접 발생
→ sdkService.js의 runTestService()에서 aggregator로 테스트 이벤트 수신

코드 컨벤션

모듈 패턴

모든 모듈은 팩토리 함수 패턴을 따릅니다:

```
// 표준 모듈 구조
module.exports = function(의존성1, 의존성2) {
    var self = this;  // core-framework 컨텍스트

    // private 변수/함수
    var internalState = {};
    function privateHelper() { ... }
```

```
    // public API 반환
    return {
        publicMethod1: function() { ... },
        publicMethod2: function() { ... }
    };
};
```

네이밍 규칙

| 대상 | 규칙 | 예시 |

|------|------|------|

| 파일명 | camelCase | pageDispatcher.js, sdkCommand.js |

| 모듈 등록명 | camelCase | applicationContext, pageDispatcher |

| MQTT 이벤트명 | snake_case | rodix_open_page, program_play |

| MQTT 토픽 | slash 구분 | rodix/open/page, program/play |

| 내부 이벤트 | dot 구분 | assembly.load, system.start |

| 서비스 키 | hyphen 구분 | {어셈블리명}-{serviceKey} |

| 상수 | PascalCase 또는 UPPER_CASE | AssemblyPath, TGOS_CODES |

비동기 패턴

이 프로젝트는 Q 라이브러리 (레거시 Promise)를 사용합니다:

```
// Q 사용 패턴
var deferred = self.$q.defer();
doAsync(function(result) {
    deferred.resolve(result);
});
return deferred.promise;

// Promise 체이닝
service.runA()
.then(function() { return service.runB(); })
.then(function() { return service.runC(); })
.done();  // ← Q 라이브러리 특유의 마무리 메서드
```

> .done()은 Q 라이브러리에서 체인 끝에 호출하여 에러를 throw하도록 합니다. 일반 Promise의 .catch()와 다릅니다.

로깅 가이드

로그 레벨

| 레벨 | 값 | 용도 |

|------|---|------|

| error | 0 | 심각한 오류, 즉각 대응 필요 |

| warn | 1 | 경고, 잠재적 문제 |

| info | 2 | 주요 동작 기록 |

| verbose | 3 | 상세 동작 기록 (파일 로그 기본) |

| debug | 4 | 디버그 정보 (콘솔 기본) |

| silly | 5 | 매우 상세한 추적 |

로그 출력 채널

| 채널 | 레벨 | 경로 | 로테이션 |

|------|------|------|---------|

| 콘솔 | debug | stdout | - |

| 파일 | verbose | {repository}/logs/rodi-x-svc/ | 5MB x 10파일 |

| 데이터 | info | {repository}/logs/user-log/ | 5MB x 30파일 |

알려진 기술 부채

| 항목 | 상태 | 설명 |

|------|------|------|

| 테스트 없음 | package.json에 "test": "Error: no test specified" | 테스트 프레임워크 미설정 |

| lodash + underscore 중복 | 둘 다 의존성에 포함 | 하나로 통일 필요 |

| Q Promise (레거시) | 현대 Promise/async-await 미사용 | 점진적 마이그레이션 고려 |

| 오래된 의존성 | Express 4.14, lodash 4.17.4 등 | 보안 취약점 가능성 |

| devDependencies 없음 | 빌드/린트/테스트 도구 없음 | 개발 도구 구성 필요 |

| 전역 변수 사용 | $logger, $lodash 등이 global | 모듈 시스템으로 전환 고려 |

| 미사용 커맨드 | rodix_generate_script_program/extension 주석 "현재 호출하는 곳 없음" | 정리 또는 활용 검토 |

트러블슈팅

플러그인이 로딩되지 않을 때

확인 순서:
1. plugins/ 디렉토리에 .asar 파일이 있는지 확인

2. 로그에서 "--- ASAR ---" 이후 파일 목록 확인
3. 로그에서 "error >>> assembly : ..." 에러 메시지 확인
4. package.json의 minimumRequiredVersion이 현재 RODI 버전과 호환되는지 확인
5. Activator.js에서 start() 메서드가 올바른지 확인

MQTT 이벤트가 동작하지 않을 때

확인 순서:
1. MQTT 브로커(rodi-broker-svc)가 실행 중인지 확인 (127.0.0.1:1883)
2. config/framework.json에 토픽이 정의되어 있는지 확인
3. sub_list에서 "run": true 인지 확인
4. sdkService.js에 subscribe가 등록되어 있는지 확인
5. sdkCommand.js에 해당 핸들러가 구현되어 있는지 확인
6. 토폴로지 heartbeat 로그로 연결 상태 확인

DI가 실패할 때

증상: 모듈이 초기화되지 않고 무한 대기

확인 순서:
1. bin/rodi-x-svc.js에서 factory 등록명 확인
2. 해당 모듈의 function 파라미터명이 등록명과 일치하는지 확인
3. 의존하는 다른 모듈이 먼저 등록되어 있는지 확인
4. 순환 의존성이 없는지 확인 (A→B→A)

이 문서 시리즈의 전체 목록

1. 프로젝트 개요 - 역할과 기술 스택
2. 디렉토리 구조 - 파일 배치와 빠른 참조
3. 아키텍처 - 레이어, 디자인 패턴, 데이터 흐름
4. 초기화 흐름 - 부팅 시퀀스 코드 따라가기
5. 플러그인 시스템 - 플러그인 구조, 서비스 타입, API
6. 통신과 이벤트 - MQTT 토픽 전체 목록, 통신 패턴
7. 개발/유지보수 가이드 - 주의사항, 컨벤션, 트러블슈팅