5단계: 플러그인 시스템

rodi-x-svc의 핵심 기능은 플러그인(Assembly)을 로드하고 실행하는 것입니다. 이 문서에서는 플러그인의 구조, 로딩 과정, 서비스 타입, 그리고 플러그인에 제공되는 API를 설명합니다.

플러그인 패키지 구조

플러그인은 .asar 아카이브(Electron 아카이브 포맷) 형태로 배포됩니다.

```
MyPlugin.asar
├── package.json              ← 메타데이터 (필수)
├── Activator.js              ← 진입점 (필수)
└── [플러그인 소스 코드]        ← 자유 구성
    ├── services/
    │   ├── MyProgramNode.js
    │   └── MyExtension.js
    ├── views/
    │   └── myPage.html
    └── ...
```

package.json (플러그인)

```
{
  "name": "MyPlugin",
  "version": "1.0.0",
  "minimumRequiredVersion": "2.003.000",
  "minimumUserLevel": "Engineer"
}
```

| 필드 | 설명 |

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

| name | 플러그인 식별자 (어셈블리명으로 사용) |

| minimumRequiredVersion | 이 플러그인이 요구하는 RODI 최소 버전 |

| minimumUserLevel | 이 플러그인을 사용할 수 있는 최소 사용자 레벨 |

Activator.js (진입점)

```
module.exports = {
    start: function(serviceContext) {
        // serviceContext를 통해 서비스 등록
        serviceContext.registerService('myCommand', new MyProgramNodeService());
        serviceContext.registerService('myExtension', new MyExtensionService());
    }
};
```

- start(serviceContext)는 플러그인 로딩 시 자동 호출됩니다.
- serviceContext를 통해 서비스를 등록합니다.

- 등록된 서비스 key는 {어셈블리명}-{serviceKey} 형식으로 저장됩니다.

4가지 서비스 타입

플러그인이 등록할 수 있는 서비스 타입은 4가지입니다. 모두 rodix_api(전역)에 정의되어 있습니다.

1. ProgramNodeService (프로그램 노드)

로봇 프로그래밍 트리에 커스텀 명령어를 추가합니다.

역할: 프로그램 에디터에서 사용하는 명령 블록
예시: "Move to Position", "Wait 3 seconds", "Set IO"

주요 메서드:
- generateScript() → 로봇 실행 스크립트 생성
- openView() → 설정 페이지 열기
- closeView() → 설정 페이지 닫기
- initializeNode() → 새 노드 생성 시 초기화

2. ExtensionNodeService (확장)

UI에 독립적인 페이지/패널을 추가합니다.

역할: 메인 UI에 확장 페이지를 제공
예시: "Arc Welding 설정", "Vision 카메라 설정"

주요 메서드:
- generateScript() → 확장의 스크립트 코드 생성
- openView() → 확장 페이지 열기
- closeView() → 확장 페이지 닫기

3. DaemonService (데몬)

백그라운드 프로세스를 실행합니다.

역할: 지속적으로 실행되는 백그라운드 작업
예시: 외부 장비 통신, 데이터 수집, 모니터링

실행 가능한 프로세스:
- Python 스크립트
- Java 애플리케이션
- 네이티브 실행파일 (.exe)

주요 메서드:
- start(args) → 프로세스 시작
- stop() → 프로세스 종료
- showWindow() → 윈도우 표시 (Windows API)
- hideWindow() → 윈도우 숨기기

4. WidgetNodeService (위젯)

UI 대시보드에 위젯을 추가합니다.

역할: 대시보드에 실시간 정보 표시
예시: 로봇 상태 모니터, I/O 상태 표시

주요 메서드:
 - openView() → 위젯 렌더링
 - closeView() → 위젯 제거

플러그인 로딩 과정 상세

plugins/
├── AdvancedHelloworld.asar
├── Components.asar
└── RobotTeaching.asar

Step 1: 디렉토리 스캔

파일: modules/applicationContext/assembly/assemblyDirectoryInfo.js

plugins/ 디렉토리를 스캔하여:
 - .asar 파일 목록 수집
 - 각 파일의 { name, absolutePath } 반환

Step 2: ASAR 추출

파일: modules/applicationContext/assembly/extractor/asarAssemblyExtractor.js

각 .asar 파일에 대해:
 1. plugins.temp/{임시폴더}/ 아래에 추출
 2. asar-fs 라이브러리로 아카이브 내용 풀기
 3. 추출된 디렉토리 구조 생성

Step 3: JS 어셈블리 로딩

파일: modules/applicationContext/assembly/loader/jsAssemblyLoader.js

추출된 각 디렉토리에 대해:
 1. package.json 읽기
 2. Activator.js를 require()로 로딩
 3. AssemblyCatalog(Map)에 저장:
    key: 어셈블리명
    value: { name, activator, package, errorMessage }

Step 4: Activator 실행

파일: modules/applicationContext/index.js → startActivator()

AssemblyCatalog의 각 어셈블리에 대해:
 1. ServiceContext 생성

   - ServiceCollection 참조
   - 어셈블리명
   - package.json 정보
2. assembly.activator.start(serviceContext) 호출
3. 플러그인이 serviceContext.registerService()로 서비스 등록
4. 에러 발생 시 assembly.errorMessage에 기록 (다른 플러그인에 영향 없음)

Step 5: 서비스 등록 검증

파일: modules/applicationContext/service/context.js, validator.js

registerService(key, serviceInstance) 호출 시:
1. 서비스 인스턴스의 프로토타입 체인 검사
2. rodix_api의 4가지 타입 중 하나를 상속하는지 검증
   - ProgramNodeService
   - ExtensionNodeService
   - DaemonService
   - WidgetNodeService
3. 키를 "{어셈블리명}-{key}" 형식으로 변환
4. 메타데이터 부착:
   - __assembly__: 어셈블리명
   - __service__: 서비스 키
   - __minUserLevel__: 최소 사용자 레벨
5. ServiceCollection(Map)에 저장

플러그인에 제공되는 API

플러그인 서비스가 초기화되면 PresentationComposite를 통해 다양한 API에 접근할 수 있습니다.

RodiAPI (14개 도메인 모델)

rodiAPI.get(robotId)로 접근하는 모델들:

| 모델 | 설명 | 주요 기능 |

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

| robotModel | 로봇 상태 | 조인트 값, TCP 위치, 상태 조회 |

| ioModel | I/O 포트 | 디지털/아날로그 I/O 읽기/쓰기 |

| eventModel | 이벤트 | 로봇 이벤트 구독/발행 |

| programModel | 프로그램 | 프로그램 노드 생성/삭제/조회 |

| variableModel | 변수 | 프로그램 변수 관리 |

| functionModel | 함수 | 프로그램 함수 관리 |

| commandModel | 명령 | 로봇 명령 실행 |

| coordinateModel | 좌표 | 좌표계 관리 |

| controllerModel | 컨트롤러 | 컨트롤러 설정 |

| tcpModel | TCP | Tool Center Point 관리 |

| systemSettings | 시스템 설정 | 시스템 파라미터 |

| utilsModel | 유틸리티 | 변환, 계산 등 |

| rodiModel | RODI | 플랫폼 정보 |

| identifierBuilder | 식별자 | 고유 ID 생성 |

DataModel (이벤트 기반 KV 저장소)

플러그인별로 독립적인 데이터 저장소가 제공됩니다:

```
// 데이터 설정 → 변경 알림 자동 발행
dataModel.set('myKey', 'myValue');

// 데이터 조회
let value = dataModel.get('myKey', 'defaultValue');

// 데이터 삭제
dataModel.delete('myKey');

// 영속화 (파일 저장)
dataModel.save();
```

PageHandler (JSDOM 기반 UI)

HTML 페이지를 서버사이드에서 관리합니다:

- HTML 파싱 및 DOM 조작
- 이미지 → base64 자동 변환
- UI 이벤트 핸들러 등록 (click, change, select 등)
- updateList를 통한 UI 상태 추적
- active/inactive 상태 관리

플러그인 에러 격리

각 플러그인은 독립적으로 에러가 격리됩니다:

```
플러그인 A 에러 발생:
  → assembly.errorMessage에 기록
  → $logger.error()로 로깅
  → 다른 플러그인(B, C)은 정상 동작 계속

버전 호환성 검증:
  → minimumRequiredVersion과 현재 RODI 버전 비교
  → 버전 불일치 시 에러 메시지 표시 (플러그인 비활성화)
```

에러 핸들링 파일: modules/exceptionHandler/XPluginExceptionHandler.js

- TGOS 에러 코드 체계 사용
- RODIX_PLUGIN_NOT_FOUND: 플러그인을 찾을 수 없음
- RODIX_PLUGIN_MINIMUM_VERSION_VIOLATION: 최소 버전 불일치

디버그 모드

개발 중인 플러그인은 plugins.debug/ 디렉토리에 폴더 형태로 넣으면 ASAR 추출 없이 바로 로딩됩니다.

```
plugins.debug/
└── MyDevPlugin/
    ├── package.json
    ├── Activator.js
    └── ...
```

이 방식은 loadDebugAssemblies() 함수에서 처리됩니다.

다음 단계

플러그인 시스템을 이해했으면, 6단계: 통신과 이벤트에서 서비스 간 MQTT 통신의 상세를 살펴봅니다.