# Studio 개발자 안내서

ProObject 7 Fix#1

TMAXSOFT

## 저작권 공지



## 제한된 권리



## 상표 공지



## 오픈소스 소프트웨어 공지

본 제품의 일부 파일 또는 모듈은 다음의 라이선스를 준수합니다. : APACHE2.0, CDDL1.0, EDL1.0, EDL1.0, BSD, MIT, SIL OPEN FONT1.1, CPL1.0, EPL1.0

관련 상세 정보는 제품의 다음 디렉터리에 기재된 사항을 참고하시기 바랍니다. :
${PROOBJECT_HOME}\proobject\licenses

## 안내서 이력

| 제품 버전 | 안내서 버전 | 발행일 | 비고 |
|---|---|---|---|
| ProObject 7 Fix#1 | 3.1.1 | 2024-08-21 | - |
| ProObject 7 | 2.1.1 | 2020-01-23 | - |

# 목차

# 1. ProObject 소개

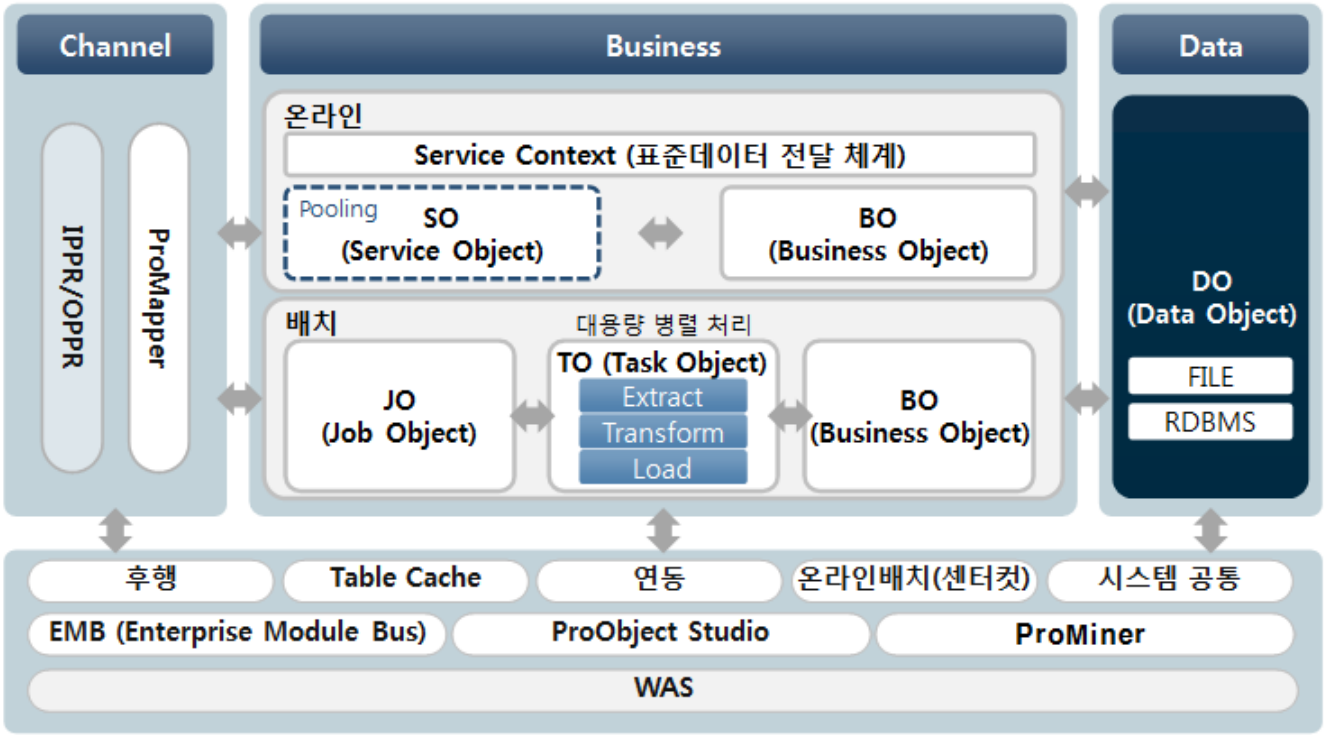
본 장에서는 ProObject의 특징 및 아키텍처 및 구성요소에 대해 설명한다.

## 1.1. 개요

ProObject는 차세대 업무시스템이 요구하는 유연성과 재사용성을 극대화하는 **아키텍처 기반 틀**을 제공하는 애플리케이션 프레임워크(Application Framework)이다. 즉, 온라인 서비스 및 배치 서비스 등을 구현할 수 있는 프레임워크를 제공하므로써 공통적인 구조와 개발 구현으로 빠른 시간에 고객의 요구사항 변화에 맞는 유연하고 기능 확장이 뛰어난 결과를 이끌어 낸다. 이와 같이 ProObject를 사용하여 개발된 서비스는 시스템의 안정적인 성능을 보장하고 유지보수를 쉽게 할 수 있도록 지원한다.

## 1.2. 아키텍처

개발자는 **온라인 업무 처리 영역**과 **배치 업무 처리 영역**만 개발하고 나머지는 시스템에서 제공한다.




ProObject 아키텍처

ProObject는 크게 10개 영역 구조가 유기적으로 연결되어 있다.

- 온라인 입출력 처리 영역

  다양한 입력 채널로부터 요청되는 입력 데이터를 업무 처리 영역에서 사용할 DO로 변환 하거나 업무 처리 영역에서 처리된 결과를 채널로 전달할 출력 데이터로 변환 및 매핑을 담당한다.

- 온라인 거래 제어 영역

  운영 환경에서 발생할 수 있는 다양한 서비스별 거래 제어 요건을 수용 및 지원하여 탄력적인 운영환경 제공한다.

- 배치 입력 처리 영역

  배치 입력 처리 영역은 다양한 프로토콜 및 전문 형식을 지원한다.

- 온라인 업무처리 영역

  온라인 거래 처리 흐름 제어 환경 및 프레임 제공하며, 객체 LifeCycle 관리 및 DI(Dependency Injection)를 통해 최적의 온라인 서비스 환경을 제공한다.

- 배치 업무처리 영역

  대량의 업무 처리를 효과적으로 처리하기 위한 환경을 제공한다.

  ProObject의 배치는, Flow관리를 위한 Block과 실제 처리를 하는 Task로 나눠진다. Block에는 Condition Block, Step Block, Partitioner Block이 있으며, 실제 업무 처리를 하는 Task에는 ETL Task, Online Task, Normal Task가있다.

- 데이터 처리 영역

  Function base 데이터 처리 방식 지양 하고 OOP 극대화하는 Entity 가능 처리 구조이다. DO Optimizer의 성능 최적화 기술을 통해 바람직하지 않게 구현된 애플리케이션의 수행 시간을 단축 처리할 수 있다.

- 후행 처리 영역

  ProObject는 메인 업무 서비스에 대한 빠른 회신을 위해 메인 서비스와 분리할 수 있는 업무를 메인 업무 서비스 응답 이후 처리할 수 있는 아키텍처를 제공한다.

- Cache 영역

  ProObject는 Cache 인프라를 제공하며, DB IO를 최소화하여 DB 서버 부하량을 감소시키며, 애플리케이션의 성능 극대화 환경을 제공한다. 자세한 내용은 ProObject 런타임 엔진 개발자 안내서의 "데이터 오브젝트 캐시 "를 참고한다.

- 연동 영역

  연동 아키텍처를 통해 다양한 연동 상황에 대한 서비스간 연동을 지원한다. 개발자는 연동 API를 통해 손쉽게 애플리케이션을 구현할 수 있다. 자세한 설명은 ProObject 런타임 엔진 개발자 안내서의 "서비스 개발"을 참고한다.

- 시스템 처리 영역

  ProObject의 객체 Pooling, 캐시, 메시지, 연동 프레임, 트랜잭션, 예외처리, Logging을 AOP(관점 지향 프로그래밍 : Aspect-Oriented Programming) 관점에서 처리하여 한 애플리케이션 내의 다양한 모듈에서 공통적으로 이용되는 기능을 분리시켜 핵심 기능 외의 기능을 응집되지 않도록 성능, 안정성 및 장애처리를 위해 제공되는 영역이다.

## 1.3. 특징

ProObject는 기존의 강점인 시스템 소프트웨어 기반의 기능과 성능은 더욱 강화하고 다른 프레임워크의 장점을 수용한 아키텍처적으로 최적화한 고성능 애플리케이션 플랫폼이다.

다음은 ProObject의 주요 특장점에 대한 설명이다.

- 획기적인 고성능 아키텍처
  - 데이터 오브젝트 캐시는 데이터베이스 시스템의 IO를 획기적으로 줄이며, 고성능 애플리케이션 처리를 보장하는 기술이다. 캐시 사용율 증가 및 DB 서버의 IO 발생의 최소화 지원을 통해 애플리케이션의 성능을 향상시킬 수 있다.

    다음은 데이터 오브젝트 캐시의 주요 특징이다.

    - DB 부하량 감소
    - 애플리 케이션 처리 속도 향상
  - **Dynamic DataSource**는 트랜잭션 상황에 따른 최적의 데이터소스를 동적으로 판단하여 사용하는 트랜잭션 최적화 기술이다. 서비스가 호출되면 AP에 트랜잭션을 분석해서 XA 데이터소스와 NON-XA 데이터소스로 동적으로 선택한 두 DB 영역에 접근한다. 이 기능은 불필요한 XA 데이터소스의 사용을 방지하여 빠른 트랜잭션을 처리하여 최적화된 관리환경을 제공한다.
  - ProObject는 강력한 미들웨어 기반의 배치 엔진을 구성하여 다양한 유형의 대용량 배치를 처리한다. Task Object 및 ETL을 병렬로 처리할 수 있다.
- 데이터 프레임워크 포함
  - Data Object(DO)는 DB 또는 File의 데이터를 객체화(Entity)하여 물리적으로 떨어진 데이터를 애플리케이션 서버 영역에 캐시하는 기능을 제공한다. 이 기능은 비즈니스에서는 데이터의 저장소가 DB나 File로 저장되어 있든지 플랫폼에서 데이터 처리를 표준화해서 제공하기 때문에 그에 따른 처리를 하지 않아도 된다.
  - DO Optimizer는 비효율적으로 구현된 애플리케이션을 보다 빠른 속도로 처리 할 수 있는 기능을 제공한다. DO Optimizer는 컴파일러 기술을 활용하여 업부 비즈니스 처리의 효율성을 극대화할 수 있는 환경을 제공한다. 또한 비효율적인 애플리케이션의 관리를 Smart하게 처리하여 DB 부하를 감소시킬수 있다.
- JVM 메모리 안정성 극대화
  - 메모리는 시스템 안정성과 밀접한 관계가 있는데, ProObject는 객체 라이프 사이클 관리(Object Pooling)를 제공하여 안정적인 JVM의 메모리 사용 환경을 제공한다. 따라서 런타임 객체를 생성하는데 오베헤드를 최소화할 수 있다. ProObject에서 IoC(Inversion of Control) / DI(Dependency Injection) 기술을 직접 구현하여 JVM GC의 최적화를 가능하도록 한다. 또한 ProObject에서는 업무 서비스 객체에 대한 최대 메모리 사용량 제약 기능 제공을 통해 잠재적인 메모리 장애 상황을 차단할 수 있다. 따라서 안정적인 시스템 운영기반을 마련하고 개발자의 실수로 인한 시스템 메모리 장애가 발생하는 문제를 방지할 수 있다.
  - ProObject는 업무 서비스 단위 클래스를 Class Loader를 사용해서 배포할 수 있다. Hot-Deploy 기능은 서비스 단위 동적 반영 구조 지원을 통해 안정적인 시스템 무중단 운영 환경을 제공한다.
- 최적화된 개발 유연성
  - ProObject는 Service, Business, Data Tier 별 고도화된 기능을 제공하며, 업무 서비스 실행에 최적화된 애플리케이션 아키텍처를 지원한다.

| 구분 | 설명 |
| --- | --- |
| Service Tier | EMB(Enterprise Module bus) 방식으로 기존의 기존 EMB 기반의 개발 환경을 제공한다. Service Object (이하 SO) 단위의 객체 관리를 통해 객체 관리의 효율화를 극대화시키고 SO에서 참조하는 모든 BO, DO의 객체 관리 또한 가능하게 한다.<br><br>SO를 통해 비즈니스 로직과 프로그램 Flow의 분리를 통한 유연한 개발환경 제공하여 비즈니스 변화의 대응에 용이하고, 비즈니스 흐름을 가시적으로 표현할 수 있다. |
| Business Tier | EMB 방식, POJO(Plain Old Java Object) 방식으로 EMB 모델 중심의 소스 생성 방식을 제공하므로, 개발에 용이하고 변경에 유연하게 적용이 가능하다. |
| Data Tier | DO Editor를 제공한다. Data Entity 객체 개념의 도입하여 DB/File 등의 I/O 접근의 일원화하고, DO가 중심이 되어 모듈간 데이터를 전달한다. |

- ProObject는 데이터 전달에 레퍼런스 참조 구조를 제공한다. 레퍼런스 참조 구조는 처리에 필요한 데이터는 객체의 래퍼런스 참조를 통해 사용하도록 하는 구조로 객체를 재사용하도록 하여 업무 구현의 유연성을 증대시킬 수 있다. 객체의 재사용성으로 불필요한 객체의 생성을 방지하고 매핑에 대한 오버헤드를 줄일 수 있다.

# Part I. Studio 사용법

# 2. ProStudio 소개

본 장에서는 ProStudio의 전반적인 기능과 사용 예제를 통해 개발 과정을 설명한다.

## 2.1. 개요

ProObject Studio(이하 ProStudio)는 ProObject 기반에서 서비스 또는 업무 시스템을 개발하기 위한 툴로서 시스템의 서비스 설계, 데이터 설계, 업무 개발을 위한 코딩 등을 편리하게 한다. 또한 안정되고 성능이 검증된 컴포넌트를 사용하기 때문에 개발자가 임의로 코딩하여 생기는 실수를 최소화할 수 있다.

ProStudio는 다음과 같은 특징을 포함한다.

- Layered Object Model

  OOP 관점으로 플로우, 업무 로직, 데이터 Layer별 역할을 정의하여 재사용 가능한 애플리케이션 Object의 조립을 통해 서비스 Object 개발할 수 있는 환경을 제공한다.

- 통합 GUI 기반의 애플리케이션 프로그램 개발

  모든 리소스 개발을 GUI 기반의 개발 도구로 개발 가능하며 Eclipse 기반의 툴을 통합하고 툴 간의 상호연동을 통해 입출력 정보, 서비스 플로우 정보, 업무 룰, 데이터 접근 정보 등 다양한 유형의 애플리케이션 프로그램 개발 환경을 제공한다.

- Source Generation 방식

  ProObject 기반에서 개발자의 하드 코딩을 최소화한다. 또한 검증된 소스 생성을 통해 개발자의 오류를 최소화한다.

- 개발 리소스 통합 관리

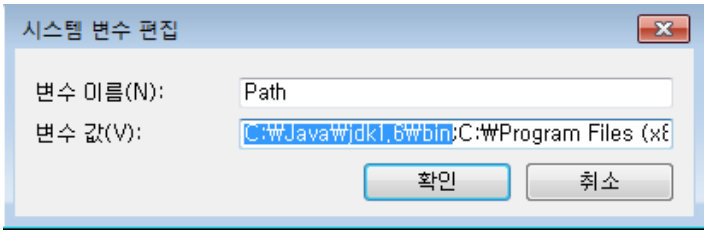
  통합 개발 서버는 ProStudio에서 개발한 소스의 버전관리, 이력관리, 권한관리, 리소스 영향도 분석 기능, 서비스 테스트, 빌드, 디플로이 기능 등을 제공한다.

## 2.2. 사용 환경 확인

ProStudio 구동 전 제품 사용을 위한 최소 요구사항을 확인한 후 JDK를 설정한다. ProStudio 구동 후 DB Connection을 위한 JDBC, ProStudio 메모리 사이즈 등을 설정하여 환경을 구성한다. 시스템 요구사항은 ProObject 설치 안내서의 "시스템 요구사항"을 참고한다.

- **JDK 설정**

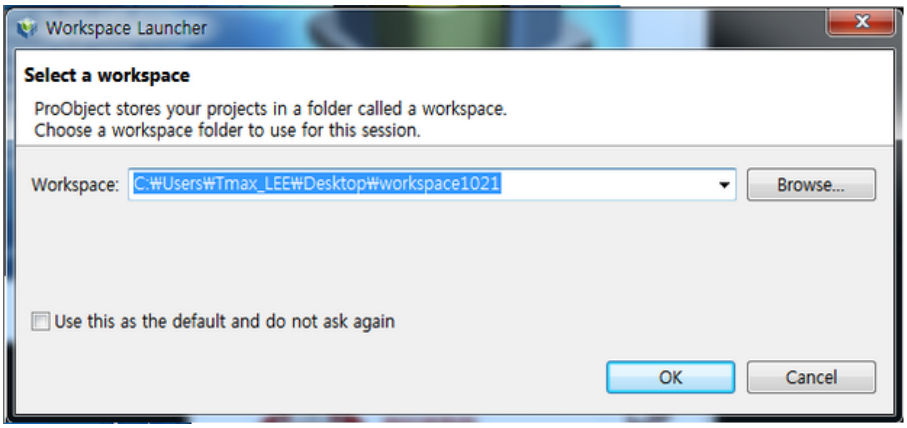
  ProStudio를 설치할 때 JDK가 환경변수의 설정 정보를 반드시 확인한다. 다음과 같이 설치한 JDK의 bin 디렉터리 경로를 PATH의 맨 앞에 설정되어 있어야 한다.



Java Path

- **ProStudio 작업 공간 설정**

  ProStudio 설치된 후에 **"ProObject.exe 바로가기"** 아이콘이 생성된다. 바로가기 아이콘을 실행하면 로컬에 작업 공간을 설정하는 팝업창이 나타난다.



ProStudio 작업공간 실행

ProObject.exe 바로 가기 속성에서 다음과 같이 메모리 사이즈를 추가한다.

  - Local PC 메모리 2GB 이하인 경우

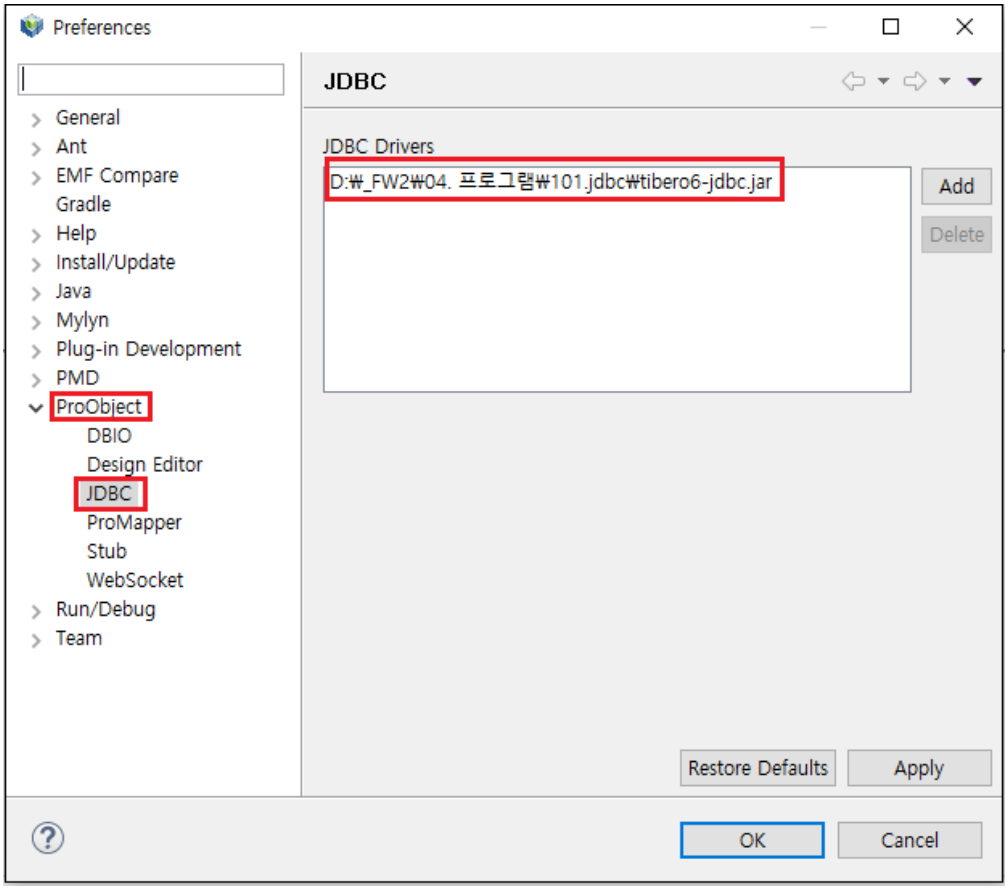
```
-clean -vmargs -Xverify:none -XX:+UseParallelGC -XX:PermSize=10M -XX:MaxNewSize=16M
-XX:NewSize=16M –Xms256m -Xms512m
```

  - Local PC 메모리 4GB인 경우

```
-clean -vmargs -Xverify:none -XX:+UseParallelGC -XX:PermSize=20M -XX:MaxNewSize=32M
-XX:NewSize=32M –Xms512m -Xmx1024m
```

- **JDBC 설정**

  **[Window] > [Preferences] > [ProObject] > [JDBC]** 메뉴를 선택해서 ProStudio에서 사용 가능한 JDBC 드라이버 파일을 등록한다.




JDBC 설정

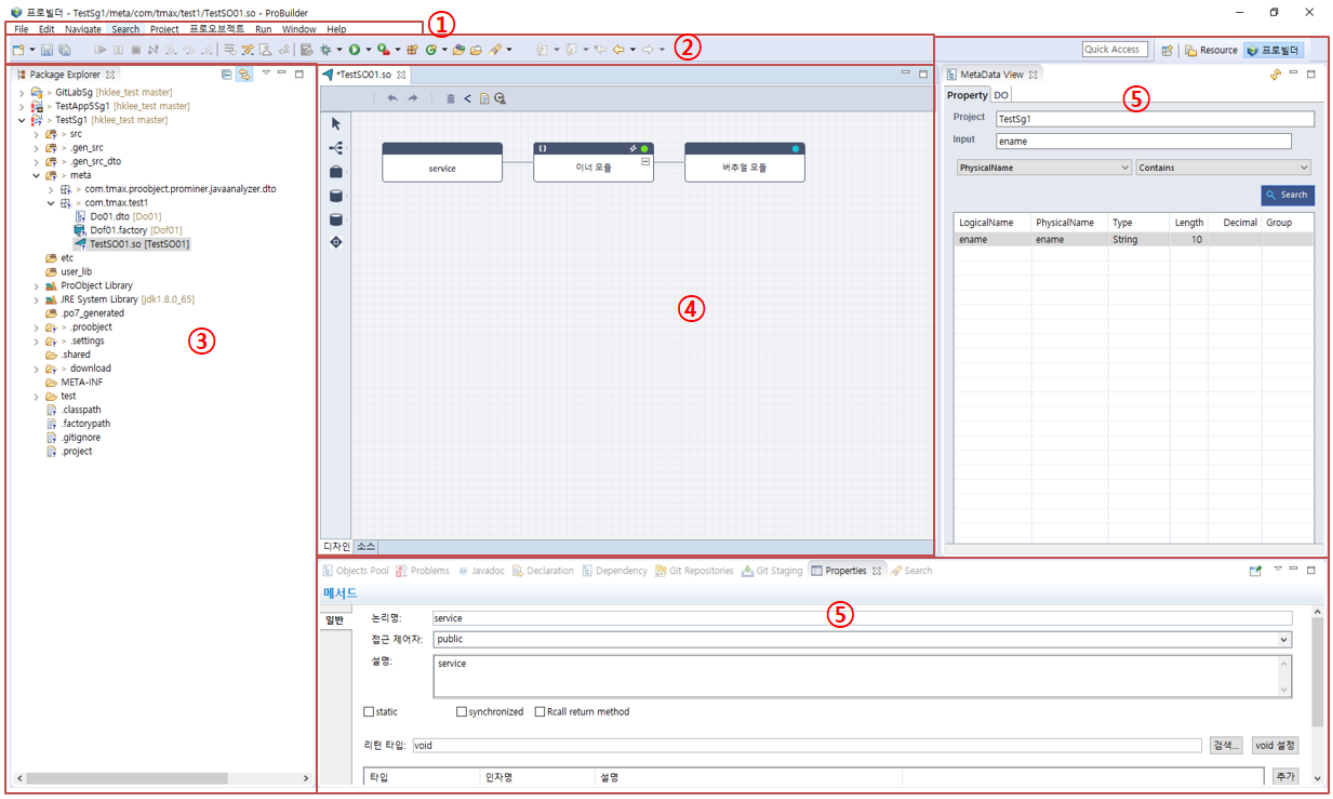
**Preferences 화면**의 JDBC 영역에 **[추가]** 버튼을 클릭해서 사용할 DB의 JDBC 드라이버를 추가한다.

| 구분 | JDBC 드라이버 |
|---|---|
| Oracle 11g | ojdbc5.jar, ojdbc6.jar |
| Tibero | tibero-jdbc.jar |

## 2.3. 화면 구성

ProStudio 메인 화면은 (1)메인 메뉴, (2)툴바, (3)탐색기 영역, (4)에디터 영역, (5)View 영역으로 나뉜다.

다음은 ProStudio 화면 구성에 대한 설명이다.




ProStudio 화면구성

- **(1) 메인 메뉴**

  메인 메뉴는 **[File]**, **[Edit]**, **[Navigate]**, **[Search]**, **[Project]**, **[프로오브젝트]**, **[Run]**, **[Window]**, **[Help]**로 구성된다. **[프로오브젝트]**와 **[Window] > [Show View]**, **[Window] > [Preferences]**를 제외한 메뉴는 Eclipse의 메뉴이다.

  **[프로오브젝트]** 하위 기능은 현재 제공하고 있지 않으므로 본 안내서에는 **[Window] > [Show View]**, **[Window] > [Preferences]** 메뉴에 대해서만 설명한다.

  - **[Window]**

| 메뉴 | 설명 |
|---|---|
| **[Show View]** | **[Show View]**에서는 **[Objects Pool]**, **[Show View] > [Other] > [프로오브젝트]** 메뉴를 제공한다. **[프로오브젝트]** 메뉴는 Dependency, MetaData View, Objects Pool 등 View 영역에서 사용하는 지원 기능을 제공한다. 지원 기능의 자세한 내용은 지원 기능에서 설명한다.<br><br>그 외 Declaration, JavaDoc, Navigator, Outline, Package Explorer, Problems, Properties, Tasks, Type Hierarchy는 Eclipse에서 제공하는 메뉴이다. |
| **[Preferences]** | ProStudio 환경을 개발자가 변경할 수 있도록 하는 메뉴이다. |

- **(2) 툴바**

  특정 기능의 빠른 실행을 위해 메인 메뉴에 있는 일부 주요 메뉴를 표시한다.

  아래 설명한 아이콘을 제외한 기능은 Eclipse의 메뉴이므로 설명하지 않는다. 자세한 설명은 해당 제품의

안내서를 참고한다.

| 아이콘 | 설명 |
| --- | --- |
|  | **[Load Configuration from Server]** 메뉴로 ProStudio 환경설정은 서버의 $PROOBJECT_HOME/system/config 값이다. 폴더에 저장되어 있으며 ProStudio에서 서버 파일을 다운받아 사용한다.<br><br>해당 파일을 통해 서버에서 각 개발자들에게 일괄적인 설정 관리가 가능하다. 다운로드된 파일은 프로젝트 목록의 .settings 폴더에 저장되며 이를 확인하기 위해서는 필터에서 ".*resource"의 체크를 해제한다.<br><br>상세한 파일 내역은 다음과 같다.<br><br>• **.settings 폴더 파일**<br>◦ site_config.setting : SiteConfig.xml 값과 동일하다.<br>• **download/config 파일**<br>Mapper 상세 옵션 및 리소스 생성 관련 설정이다.<br>◦ SiteConfig.xml : 각 개발자들에게 일괄적인 설정 관리를 위한 파일로 site_config.setting의 값과 동일하다.<br>◦ dbio_config.xml : BO/SO의 DOF 데이터소스 내역 표시 정보이다.<br>◦ restriction_codes.xml : 금지 함수 정의 내역이다. |

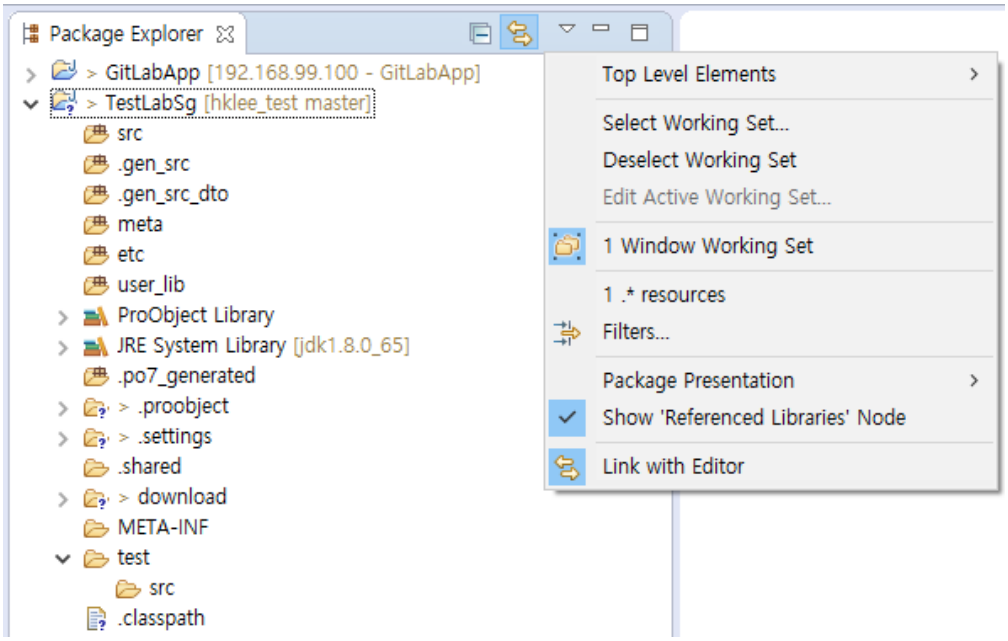
- **(3) 탐색기 영역(Package Explorer)**
  - **리소스 목록**

    로컬에서 개발자가 만든 메타 파일이나 Java 파일 전체 리소스 목록을 트리 형태로 조회한다.

| 폴더 | 설명 |
| --- | --- |
| src | 일반 .java 파일과 BizObject(Java)의 .java 파일이 저장되는 폴더이다. |
| .gen_src | EMB로 만든 Java 파일이 저장되는 폴더이다. |
| .gen_src_dto | DO에 의해 생성된 Java 파일이 저장되는 폴더이다. |
| meta | ProObject 객체를 작성할 때 메타 파일 생성 경로이다. |
| user_lib | 사용자 라이브러리를 추가하는 폴더이다. |
| ProObject Library | ProObject 라이브러리를 조회한다. |
| JRE System Library | JRE 시스템 라이브러리를 조회한다. |
| test/src | BO unit test, Service Test의 테이스 케이스가 저장되는 폴더이다. |
| .project | Application, Service Group 정보를 저장하는 폴더이다. |
| .settings | ProStudio의 설정 파일을 저장하는 폴더이다. |
| .download | Config 등을 다운로드하는 폴더이다. |
| META-INF | ProObject 운영환경에서 수행할 서비스들을 xml 파일에 저장하는 경로이다. |

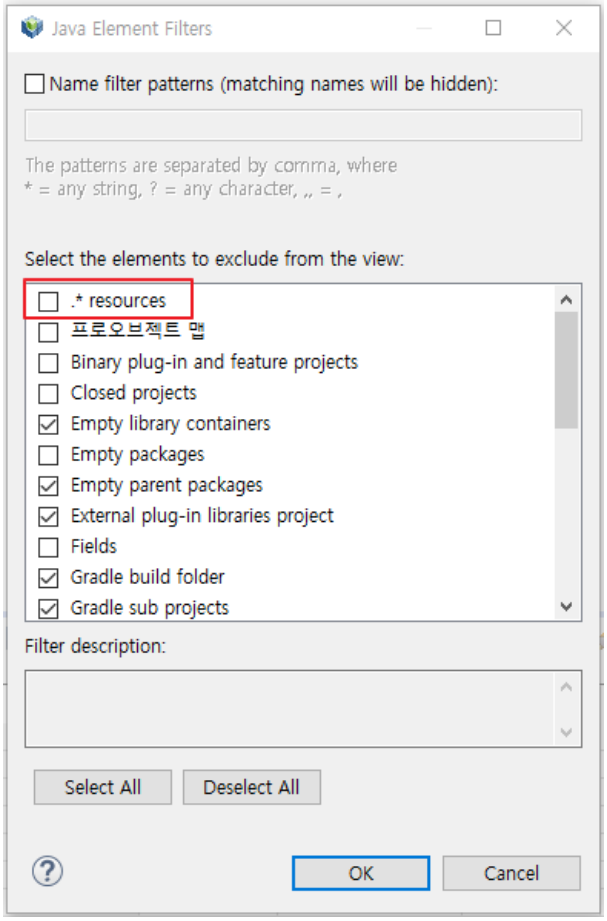
◦ **필터 설정**

**Package Explorer**의 **[▽]** 버튼을 클릭하면 **[필터]** 메뉴를 실행할 수 있다.



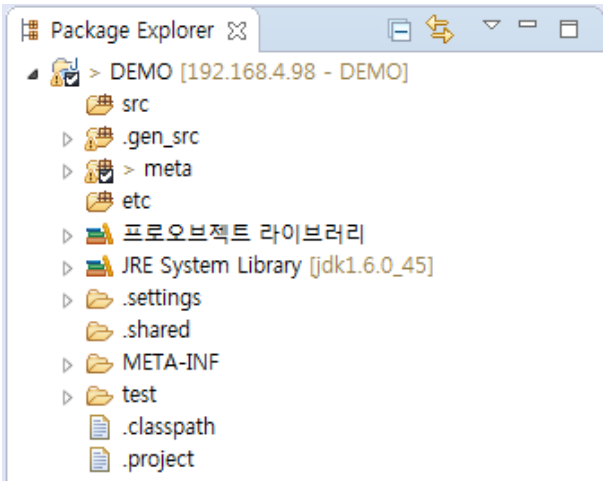

필터 메뉴 열기

**필터 설정 화면**에서 .*resource 체크박스의 체크를 해제하면 **Package Explorer**에 .gen_src, .gen_src_dto 디렉터리가 조회된다.




필터 설정

다음은 필터를 설정한 결과 화면이다.



필터 설정 결과

- **(4) 에디터 영역**

  서비스 및 업무 객체, 데이터 오브젝트, 데이터 오브젝트 팩토리, 잡 오브젝트(Job Object) 객체 등의 세부 내용을 작성하거나 편집한다. 자세한 내용은 오브젝트 플로우 에디터를 참고한다.

- **(5) View 영역**

  객체의 특성 값을 설정하거나 리소스 검색 등의 역할을 한다. View 영역에 보여지는 탭은 **[Winows] > [Show View] > [Other]**에서 설정한다.

  다음은 주요 탭에 대한 설명이다. 각 탭의 사용법은 해당 절의 설명을 함고한다.

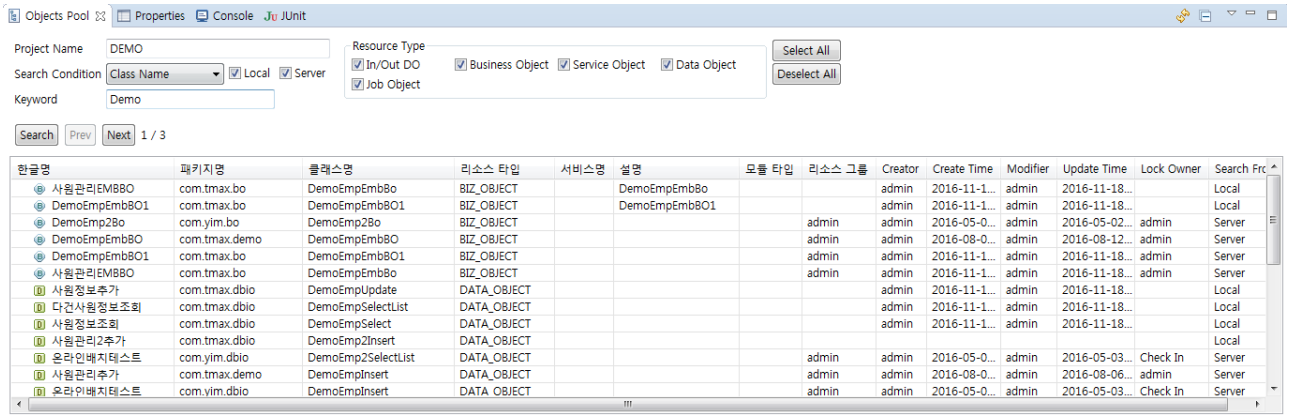
| 탭 | 설명 |
|---|---|
| **[Object Pool]** | 생성한 모든 오브젝트를 관리한다. |
| **[Dependency]** | 통합 서버 DB 정보에 존재하는 ProObject에서 생성된 리소스의 의존 관계를 개발자들이 사용할 수 있도록 한다. |

## View 영역

View 영역은 객체의 특성 값을 설정하거나 리소스 검색 등의 역할을 한다.

- **[Object Pool] 탭**

  **[Object Pool]** 탭은 ProStudio에서 생성한 모든 오브젝트를 관리한다. 사용자는 오브젝트를 검색하여 오브젝트 존재 유무를 확인할 수 있으며 검색한 리소스를 더블클릭하여 **Editor 화면**을 열 수 있다.




View 영역 - [Object Pool] 탭

Object Pool에서는 리소스에 대한 쓰기, 읽기 권한도 설정할 수 있다. 리소스에 대한 쓰기, 읽기 권한은 소유자만 설정 가능하며 쓰기 권한이 있어야 해당 리소스에 대해 커밋, 체크아웃을 받을 수 있다. 또한 리소스에 대해 읽기 권한이 있어야 리소스 조회, 가져오기가 가능하다.

| 권한 | 설명 |
|---|---|
| user right | 리소스의 소유자가 가지는 권한이다. |
| group right | DEV_USER_GROUP 테이블을 기준으로 해당 유저가 속한 그룹 유저들의 권한이다. |
| other right | 그 이외의 유저들이 가지게 되는 권한이다. |

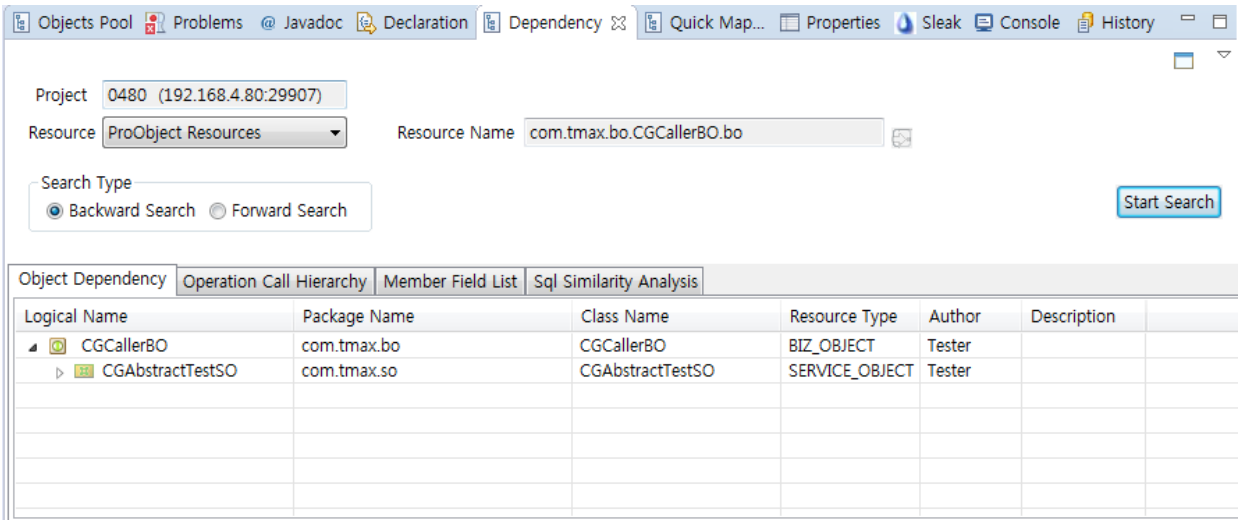
- **[Dependency] 탭**

ProObject에서 생성된 리소스는 통합서버 DB 정보에 존재하며 의존 관계를 개발자들이 사용할 수 있도록 Dependency 기능을 제공한다. 리소스를 수정한 후 해당 리소스를 호출하는 다른 리소스에 수정된 리소스 모듈을 반영하지 않으면 런타임에 오류가 발생할 수 있다. 모듈을 수정하기 전 반드시 해당 리소스의 호출 관계를 파악하여 영향이 있으면 담당자에게 알려 동시에 수정될 수 있도록한다.

**[Dependency]** 탭에 해당 리소스를 기준으로 Backward, Forward를 검색해서 영향도 파악할 수 있다.

  - **Backward 검색**

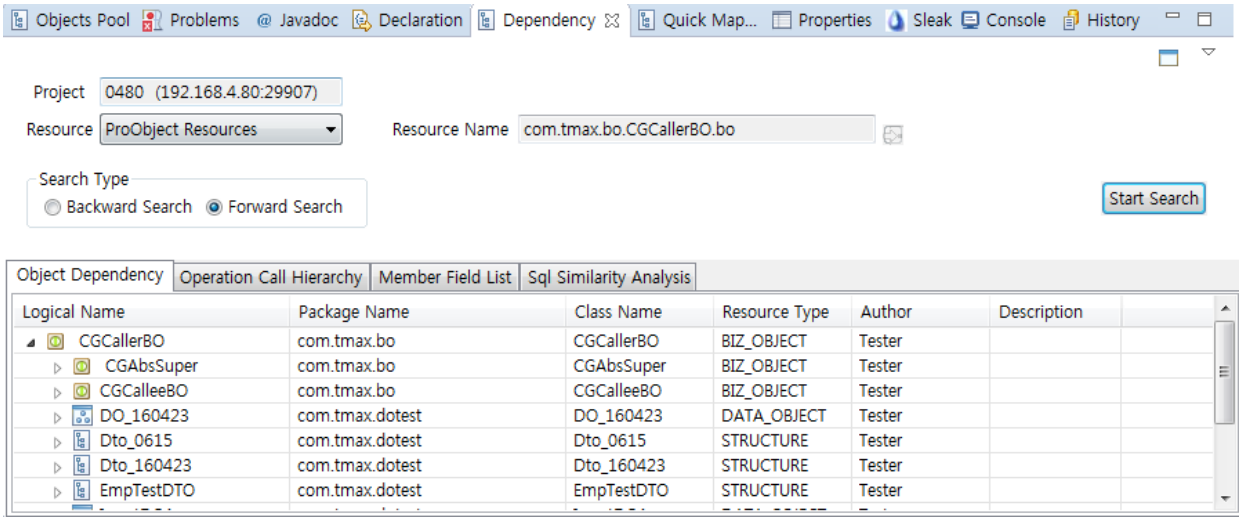
    의존 검색한 리소스를 호출하는 리소스 모듈을 조회한다.




[Dependency] 탭 - Backward

◦ **Forward 검색**

의존 검색한 리소스가 호출하는 모듈 정보를 조회한다.



[Dependency] 탭 - Forward

# 2.4. 프로젝트 생성

ProObject에서 리소스를 관리하는 단위는 **애플리케이션**, **서비스 그룹**이다.

애플리케이션은 여러 서비스 그룹으로 구성된 하나의 프로그램으로, 논리적인 관리 단위이다. 서비스 그룹은 애플리케이션을 구성하는 업무 단위 프로그램으로 ProStudio에서 하나의 프로젝트를 의미한다.

ProStudio에서 작업하는 모든 리소스는 서비스 그룹(IDE project)에 속해 있어야 한다. 서비스 그룹에 속한 서비스는 서비스 오브젝트(Service Object, SO)를 의미하며 한 서비스 그룹 안에 여러 개의 서비스 오브젝트 및 서비스 오브젝트를 구성하는 비즈니스 오브젝트(Biz Object), 데이터 오브젝트 팩토리(Data Object Factory), 데이터 오브젝트(Data Object)가 존재한다.

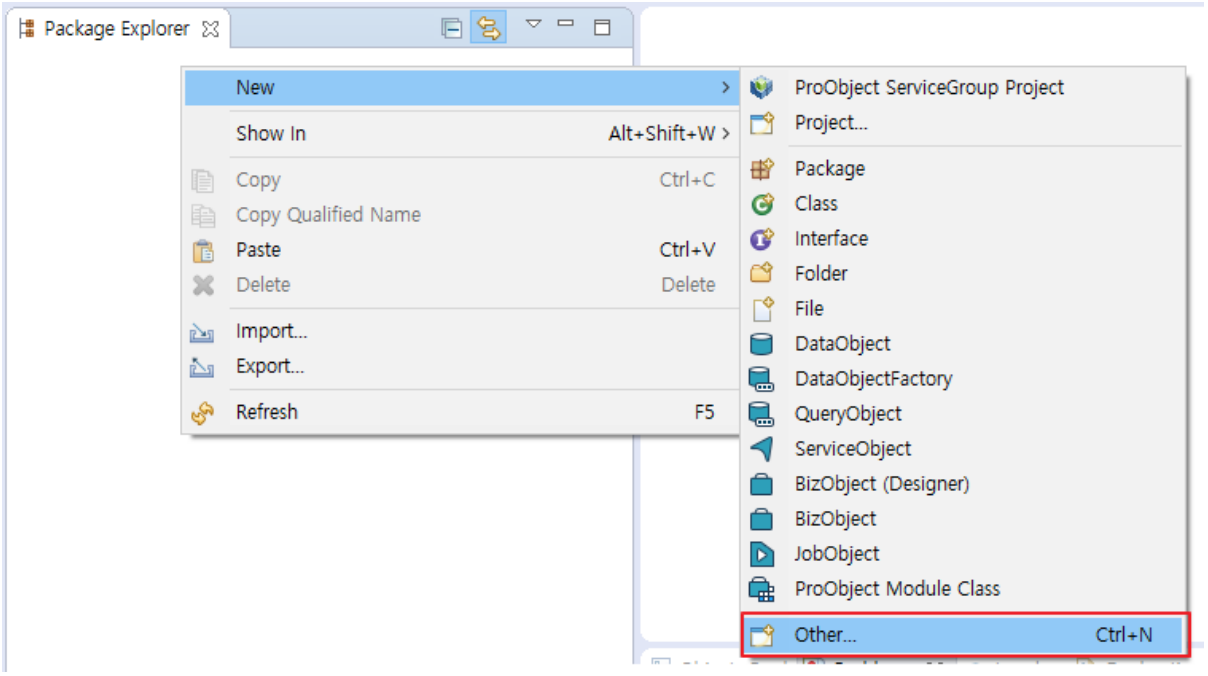
ProStudio에서 구현한 ProObject 프로그램을 배포한다는 것은 해당 애플리케이션 디렉터리 하위에 서비스 그룹별로 디렉터리가 나누어져 리소스 바이너리가 배포된다는 의미이다.

개발자는 관리자가 ProObject Manager(이하 ProManager)에서 생성한 애플리케이션 및 서비스 그룹을 선택하여 프로젝트를 생성한다. 생성된 애플리케이션 프로젝트 및 서비스 그룹 프로젝트은 공동 작업을 위해 Git에 연동하는 작업이 필요하다. 자세한 내용은 로컬에 Remote Git Repository 복사와 프로젝트 Git 서버에 복사를 참고한다.

## 2.4.1. 애플리케이션 프로젝트 생성

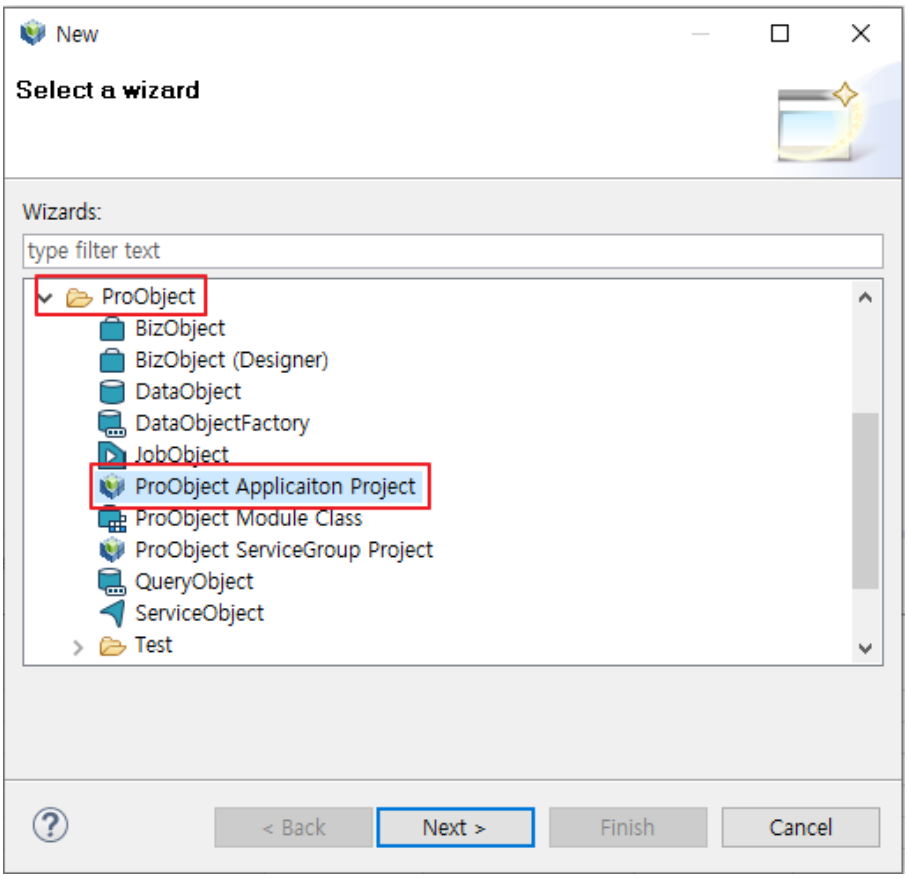
다음은 **ProStudio**에서 애플리케이션 프로젝트를 생성하는 과정에 대한 설명이다.

1. ProManager에서 애플리케이션을 생성하고, 애플리케이션을 구성할 서비스 그룹을 정의한다. 자세한 내용은 "ProObject Manager 사용자 안내서"를 참고한다.
2. 프로젝트를 생성하기 위해 **[파일]** 메뉴 또는 **Package Explorer**의 컨텍스트 메뉴에서 **[New] > [Other]**를 선택한다.



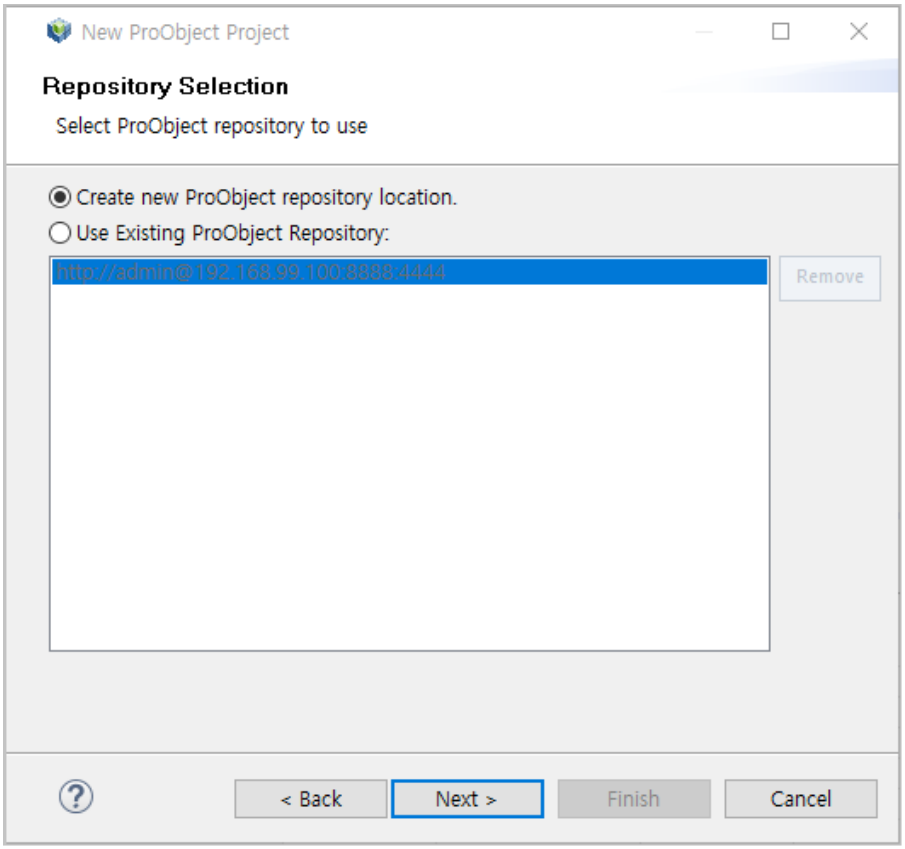

애플리케이션 생성 메뉴

3. **NEW Wizard 화면**의 업무 트리에서 **[ProObject] > [ProObject Application Project]**를 선택한다.



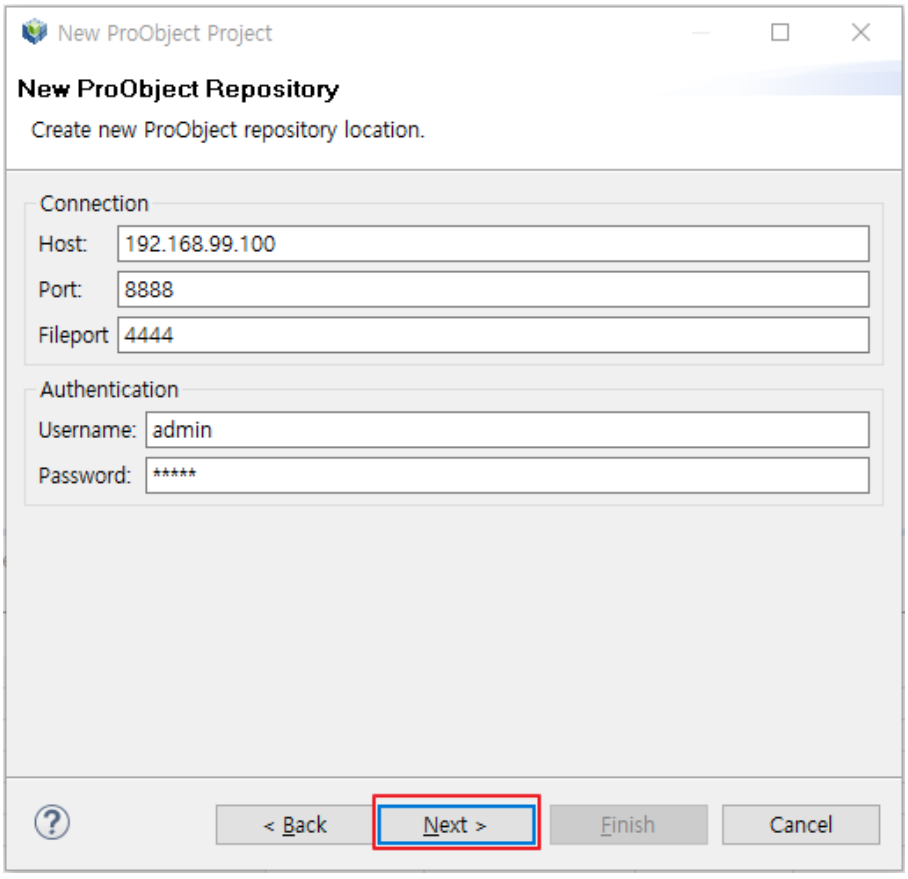

NEW Wizard 화면

4. 프로젝트의 옵션을 선택한 후 **[Next]** 버튼을 클릭한다. 예제에서는 신규로 Repository를 생성하는 옵션을 선택한다.




프로젝트 저장소 위치 생성

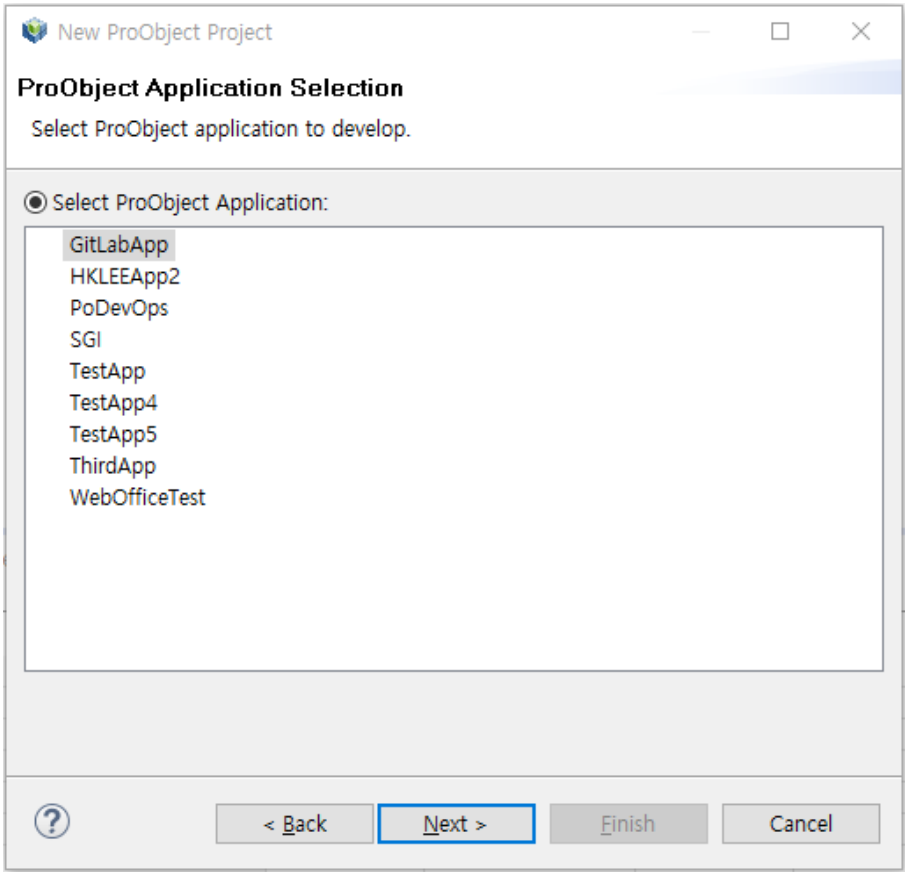
5. **New ProObject Repositoy 화면**에 새로운 프로젝트의 Repository 정보를 입력하고 **[Next]** 버튼을 클릭한다.




프로젝트 저장소 연결

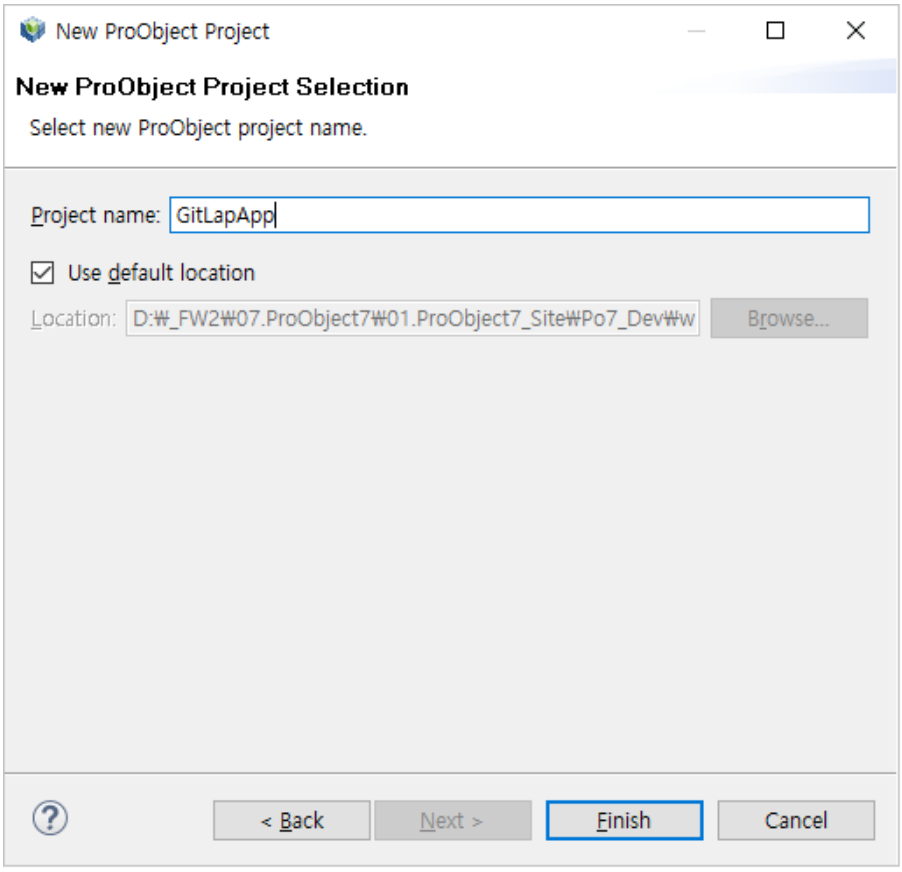
| 항목 | 설명 |
|---|---|
| Host | ProObject 개발 서버 IP 주소를 설정한다. |
| Port | ProObject 개발 서버 HTTP Listen Port를 설정한다. |
| Fileport | 파일을 업로드하고 다운로드할 때 사용할 TCP 포트를 설정한다. (기본값: 4444) |
| Username | ProManager에서 등록한 사용자 ID를 입력한다. |
| Password | ProManager에서 등록한 사용자 ID의 암호를 입력한다. |

6. Repository 선택이 완료되면 **ProObject Application Selection 화면**에 ProManager에서 생성한 애플리케이션과 하위 서비스 그룹들이 조회된다. 생성할 애플리케이션을 선택한 후 **[Next]** 버튼을 클릭한다.



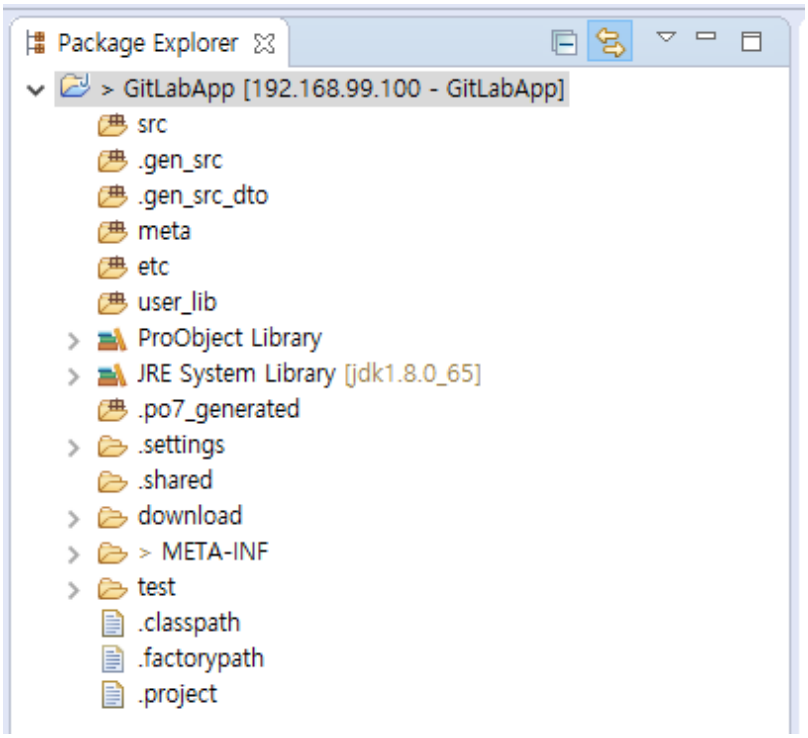

프로오브젝트 애플리케이션 그룹 리스트

7. 개발자 PC에서 사용할 프로젝트 이름을 입력한 후 **[Finish]** 버튼을 클릭한다. **'Project name'**은 선택된 애플리케이션명과 동일해야 한다.




프로젝트 이름 생성

8. 생성된 프로젝트는 **Package Explorer**에서 확인할 수 있다.

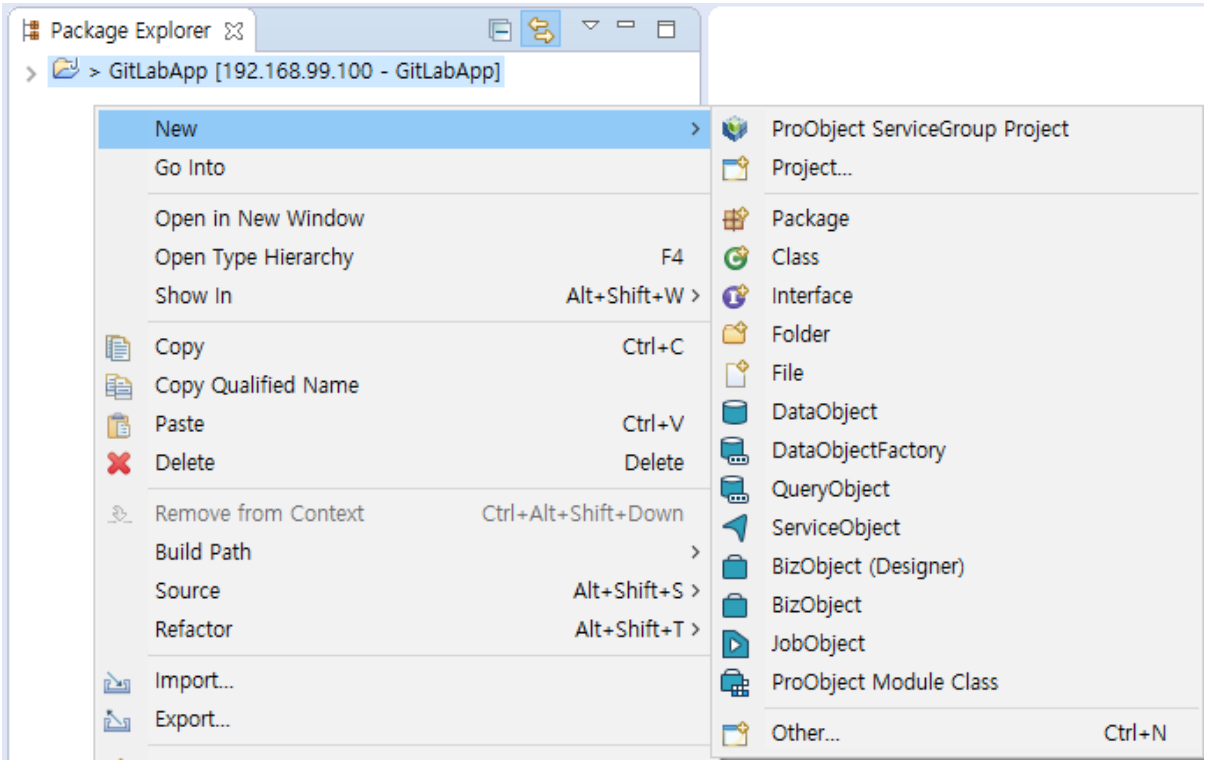



프로젝트 이름 확인

## 2.4.2. 서비스 그룹 프로젝트 생성

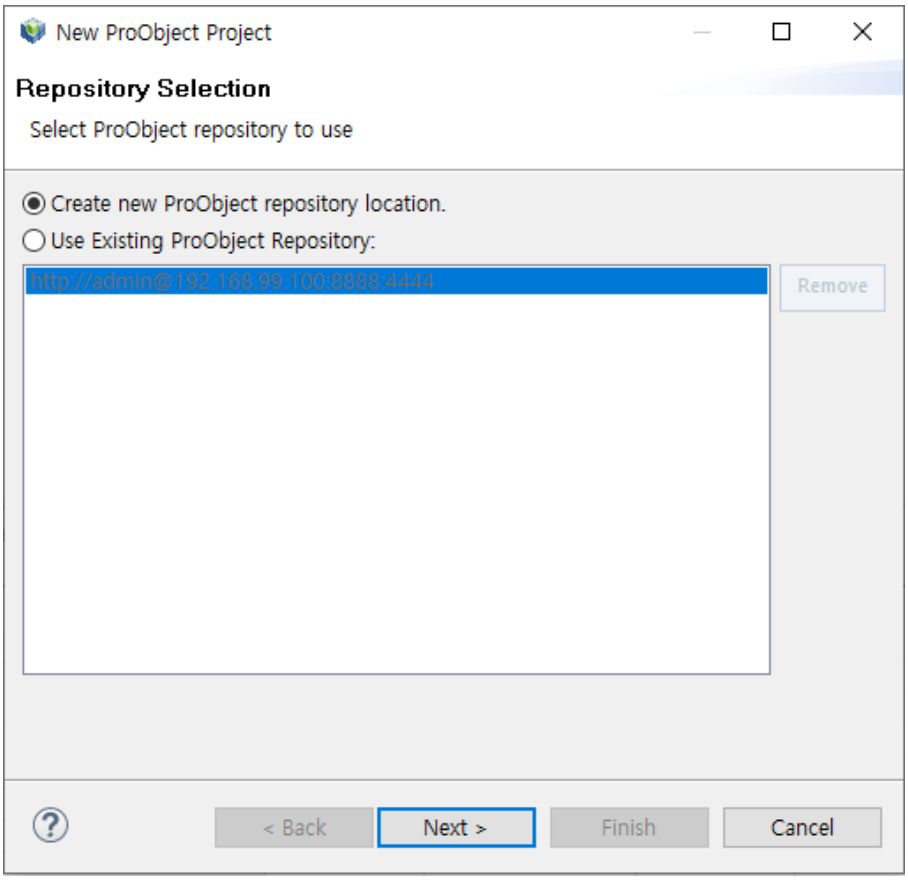
다음은 ProStudio에서 서비스 그룹 프로젝트를 생성하는 과정에 대한 설명이다.

1. ProManager에서 애플리케이션을 생성하고, 애플리케이션을 구성할 서비스 그룹을 정의한다.
2. 프로젝트를 생성하기 위해 **[파일]** 메뉴 또는 **Package Explorer**의 컨텍스트 메뉴에서 **[New] > [ProObjcet ServiceGroup Project]**를 선택한다.



프로젝트 신규 생성

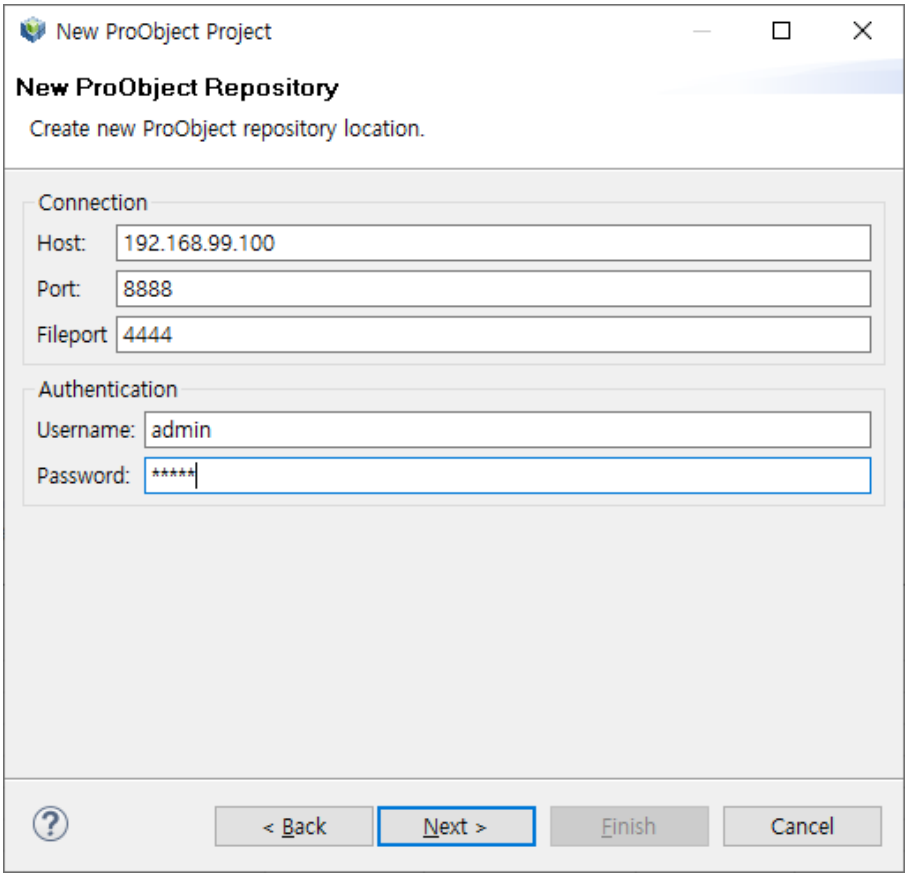
3. 프로젝트의 옵션을 선택한 후 **[Next]** 버튼을 클릭한다.



프로젝트 저장소 위치 생성

| 항목 | 설명 |
| --- | --- |
| Caeate new Project repository location | Service Group, Application을 생성한 이력이 없는 경우 선택한다. |
| Use Existing Project Repository | 이전에 Service Group, Application을 생성한 경우 기존 설정내역이 저장된 내역에 설정 내역이 표시된다. 원하는 설정 내역을 선택한다. |

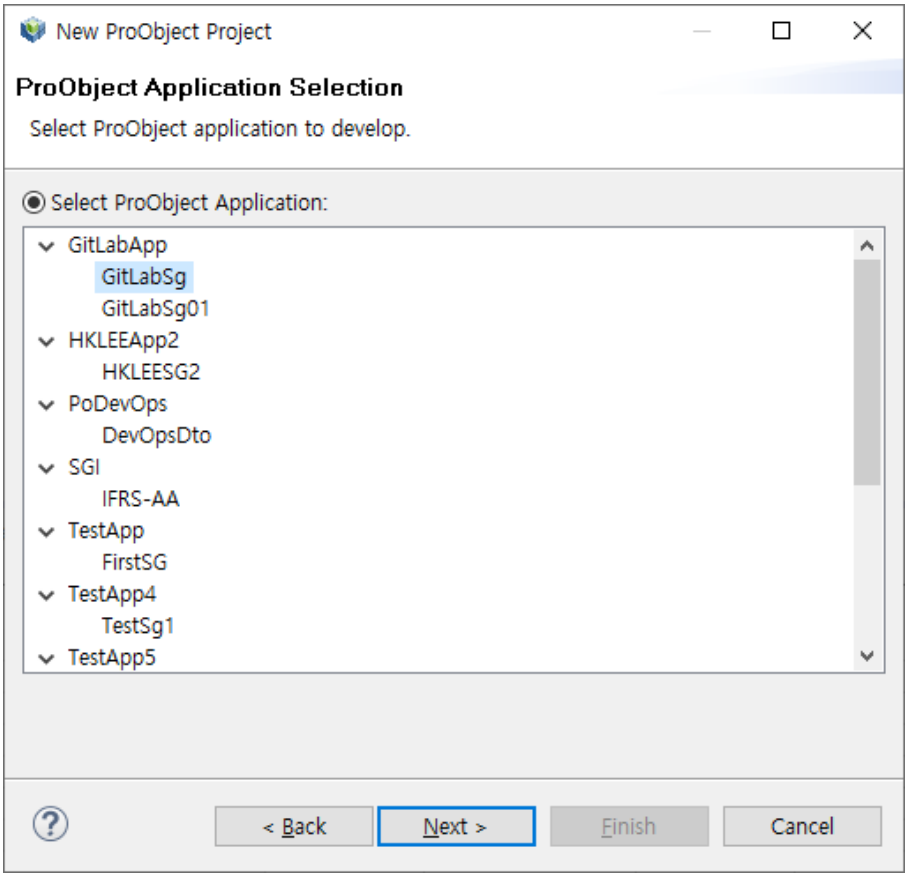
4. **Repository Selection 화면**에서 **'Caeate new Project repository location'**를 선택하면 **New ProObject Repositoy 화면**이 나타난다. 해당 화면에서 새로운 프로젝트의 정보를 입력하고 **[Next]** 버튼을 클릭한다.




프로젝트 저장소 연결

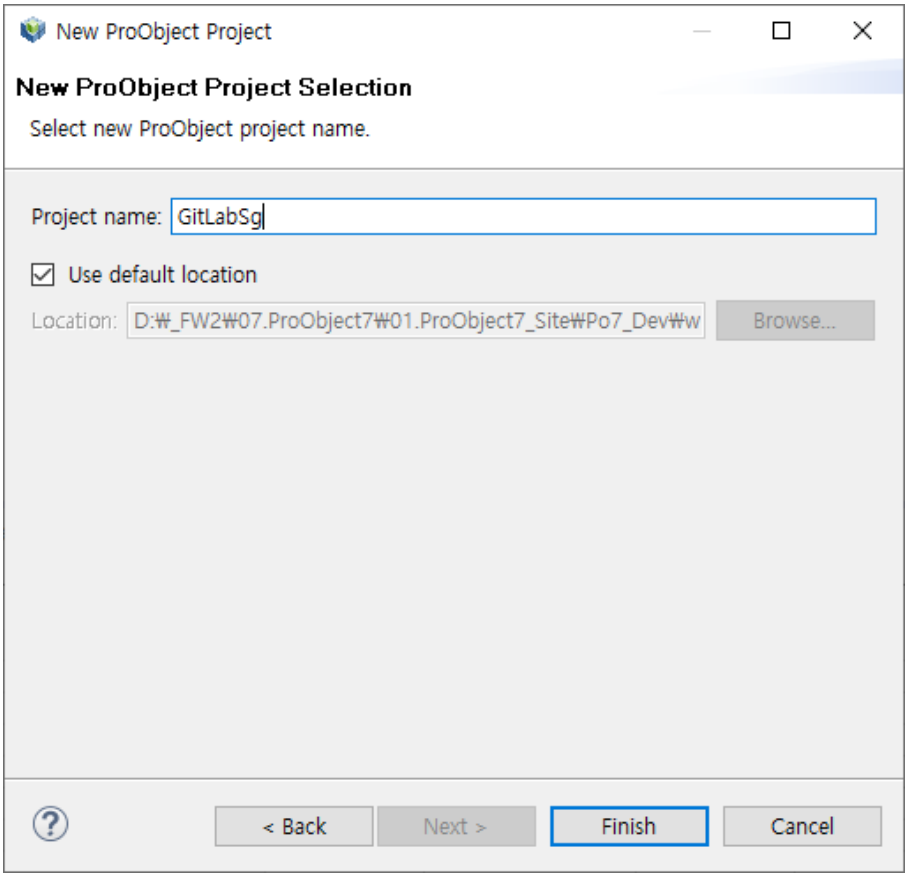
| 항목 | 설명 |
| --- | --- |
| Host | ProObject 개발 서버 IP 주소를 설정한다. |
| Port | ProObject 개발 서버 HTTP Listen Port를 설정한다. |
| Fileport | 파일을 업로드하고 다운로드할 때 사용할 TCP 포트를 설정한다. (기본값: 4444) |
| Username | ProManager에서 등록한 사용자 ID를 입력한다. |
| Password | ProManager에서 등록한 사용자 ID의 암호를 입력한다. |

5. Repository 선택이 완료되면 **ProObject Application Selection 화면**에 ProManager에서 생성해 놓은 애플리케이션과 하위 서비스 그룹들이 조회된다. 생성할 서비스 그룹을 선택한 후 **[Next]** 버튼을 클릭한다.



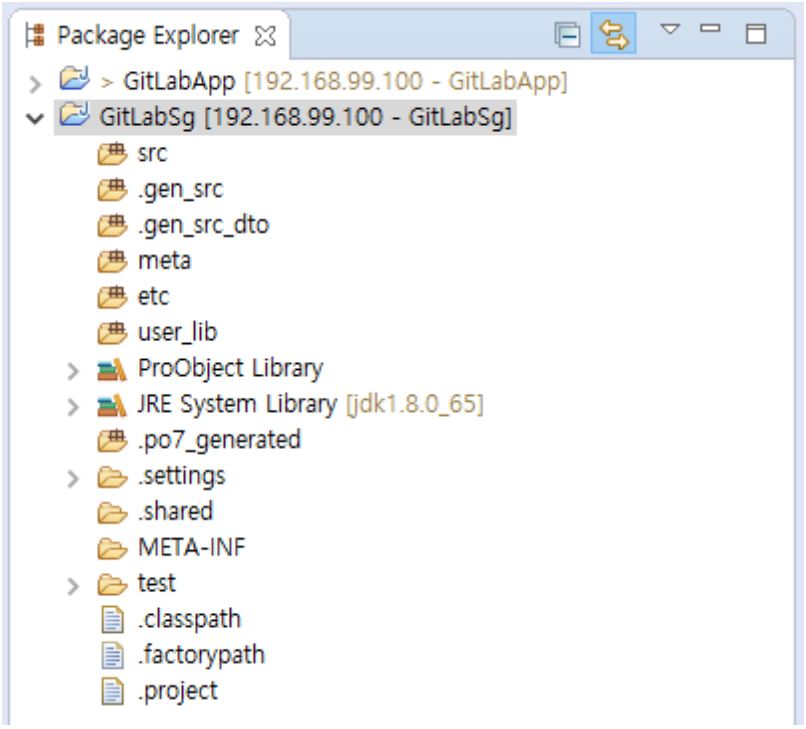

프로오브젝트 서비스 그룹 리스트

6. 개발자 PC에서 사용하는 프로젝트 이름을 입력한 후 **[Finish]** 버튼을 클릭한다.




프로젝트 이름 생성

7. 생성된 프로젝트는 **Package Explorer**에서 확인할 수 있다.




프로젝트 이름 확인

## 2.5. 프로그램 개발

다음은 ProStudio를 사용해서 프로그램을 개발하는 과정에 대한 설명이다.

1. **서비스 플로우 설계서 작성**

   경우에 따라서 모델링 툴을 통해 업무를 설계한다.

2. **메타 정보 ProManager에 등록**

   사용할 메타 정보를 ProObject Manager에 등록한다. 데이터 오브젝트를 생성할 때 메타에 등록된 프러퍼티 정보만 사용 가능하다. 메타 등록은 "ProObject Manager 사용자 안내서"를 참고한다.

3. **서비스의 입출력 전문 데이터 오브젝트 생성**

   서비스의 입력, 출력 데이터가 담기는 Object인 DO를 생성한다.

4. **데이터 오브젝트 팩토리 생성**

   데이터 오브젝트를 이용해 데이터베이스에 대한 표준적인 방법으로 데이터베이스를 제어하는 데이터 오브젝트 팩토리를 생성한다.

5. **데이터 오브젝트 팩토리 호출 및 서비스 오브젝트 입출력 데이터 오브젝트에 값 매핑**

   비즈니스 로직이 작성된 비즈니스 오브젝트로 데이터 오브젝트 팩토리 호출 및 서비스 오브젝트의 입출력 데이터 오브젝트에 값을 매핑한다. (General, Abstract, Interface을 지원)

6. **서비스 오브젝트 생성, 플로우 작성**

서비스 오브젝트를 생성하고 플로우를 작성한다. Flow Editor를 통해 비즈니스 오브젝트 메서드 조립, 매핑을 지원하며 Flow Statement를 이용하여 플로우 상태를 제어할 수 있다.

7. **서비스 디플로이**

   DOF, DO, SO, servicegroup.xml를 **Git Staging 화면**에서 **Staged Changs** 내역에 추가한 후 **[Commit and Push]** 버튼을 클릭한다(내부적으로 Jankins에서 Hooker에 의해 자동 배포된다. 자동 배포가 설정되지 않은 경우는 **Jankins 화면**에서 해당 프로젝트를 선택한 후 **[Build with Parameters]**을 수행한다).

8. **서비스 테스트**

   ProManager의 **[Test] > [Test]**에서 서비스를 테스트한다. 자세한 내용은 ProObject Manager 사용자 안내서의 "Test"를 참고한다.

# 3. 데이터 오브젝트/데이터 오브젝트 팩토리

본 장에서는 데이터 오브젝트 개요, 특징, 구성요소 및 데이터 오브젝트 팩토리 기능에 대해 설명한다.

## 3.1. 개요

데이터 오브젝트(Data Object, DO)는 모듈 간 데이터 전달이 가능하며 데이터 오브젝트 Optimizer를 통한 성능 최적화 기능을 제공한다. ProObject는 온라인 및 배치 애플리케이션 성능을 향상시키는 아키텍처 구조로 개발되어 있다.

다음은 구성 요소에 대한 설명이다.

- **Service Object(SO)**

  In/Out 구조체, DB 혹은 File에 대한 In/Out의 기능을 수행한다.

- **Data Object(DO) / DO Factory (DOF)**

  프로그램을 처리하기 위한 I/O 데이터로 데이터 오브젝트/데이터 오브젝트 팩토리를 통해서만 DB, File에 접근한다. 개발자는 오직 데이터 오브젝트/데이터 오브젝트 팩토리만 바라본다. 데이터 오브젝트 팩토리는 DB, File 두 가지 타입이 존재한다.

| Type | 설명 |
|---|---|
| DO | 여러 형태의 데이터를 담아 모듈 간 데이터 전달에 사용하는 객체로 Data Transfer Object의 기능을 수행한다. 자세한 내용은 데이터 오브젝트를 참고한다. |
| DB DOF | DB를 사용해서 데이터를 조회하거나 조작하는 객체로 DAO(Data Access Object)의 기능을 수행한다. 자세한 내용은 데이터 오브젝트 팩토리(DB)를 참고한다. |
| File DOF | 파일을 읽거나 쓰는데 사용하는 객체로 여러 형태의 파일을 가공할 수 있다. 자세한 내용은 데이터 오브젝트 팩토리(FILE)를 참고한다. |

- **Query Object(QO)**

  데이터베이스에 접근하여 데이터를 조회하거나 조작하는데 사용하는 객체이다. 자세한 내용은 쿼리 오브젝트를 참고한다.

데이터 오브젝트와 파일 데이터 오브젝트 팩토리를 작성할 경우에는 사용할 메타 정보를 반드시 사전에 ProManager에서 등록한다.

## 3.2. 데이터 오브젝트

데이터 오브젝트는 데이터 오브젝트 객체를 의미한다. 데이터 오브젝트 객체는 서비스 오브젝트 입출력 전문의 데이터를 담을 수 있는 객체로 모듈 간 데이터 전달에 사용한다. 객체 내 데이터 구성은 메타 정보에 등록된 필드를 사용하여 구성한다. 사용자가 정의한 필드에 대해 getter와 setter가 자동으로 생성되며 필드 관리에 필요한 여러

가지 메소드가 제공된다. 또한 Excel 파일을 이용한 데이터 오브젝트 생성이 가능하다. 본 절에서는 데이터 오브젝트의 기본 개념과 시작하는 방법, 데이터 오브젝트를 사용한 예제를 설명한다.

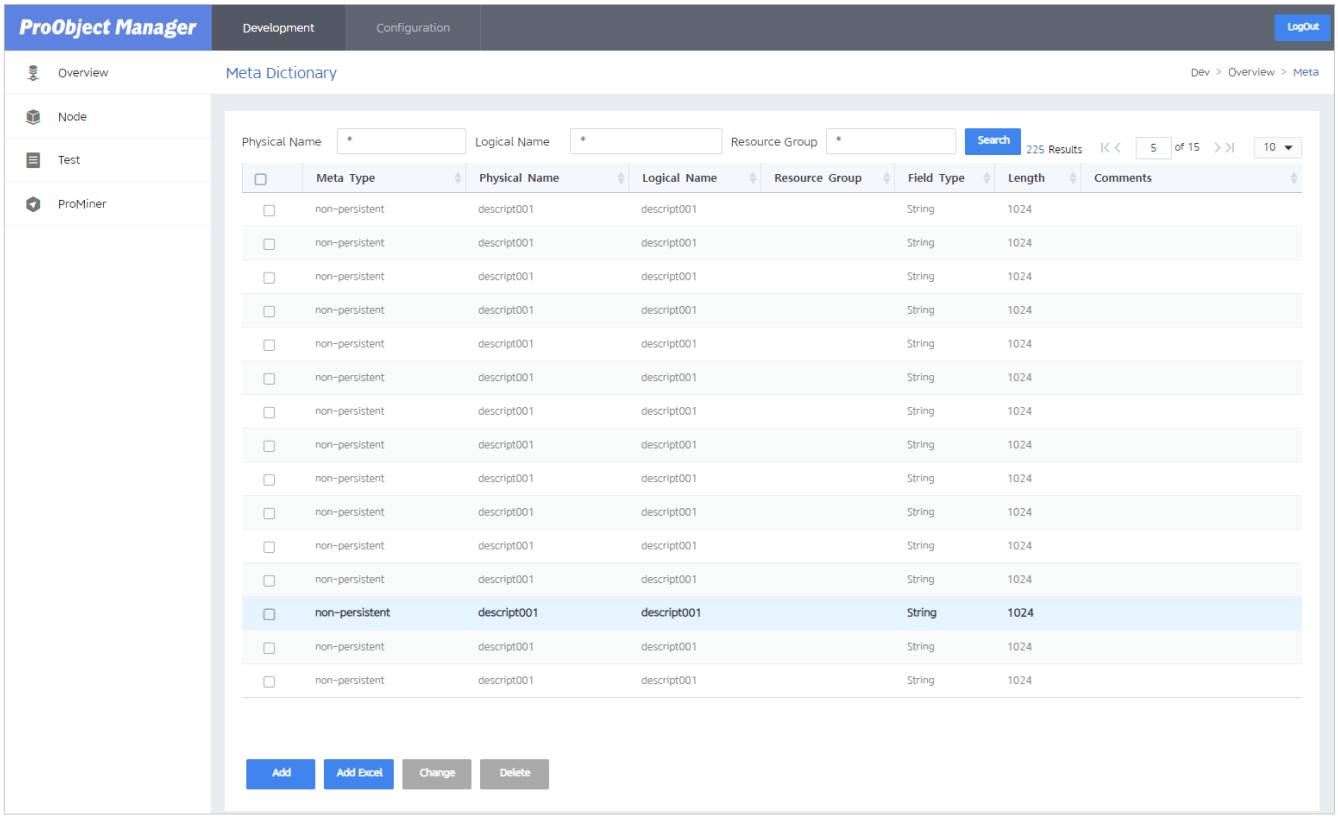
## 3.2.1. 환경설정

데이터 오브젝트를 사용하기 위해서는 Property 등록 및 검색할 수 있는 화면이 필요하다. Property 등록은 ProManager에서 가능하며, Property 검색은 ProStudio의 **[MetaData View]** 메뉴에서 조회한다.

### 3.2.1.1. Property 등록

Property는 메타 시스템에서 등록할 수 있다. 메타 시스템은 서비스 간의 input, output을 전달해야 하는 데이터 오브젝트의 경우 통일된 형식(이름, 길이, 속성)을 갖어야 하기 때문에 미리 등록된 데이터들을 이용한다.

**ProManager**의 **[Overview] > [Meta]** 메뉴를 선택해서 등록된 메타를 이용하여 데이터 오브젝트의 필드를 추가한다. **메타 관리 화면**에서 **[Search]**, **[Add]**, **[Change]**, **[Delete]** 버튼을 클릭해서 메타를 생성 및 관리할 수 있다.




ProManager 메타 관리 화면

메타 관리에 대한 자세한 내용은 "ProObject Manager 사용자 안내서"를 참고한다.

### 3.2.1.2. Property 검색

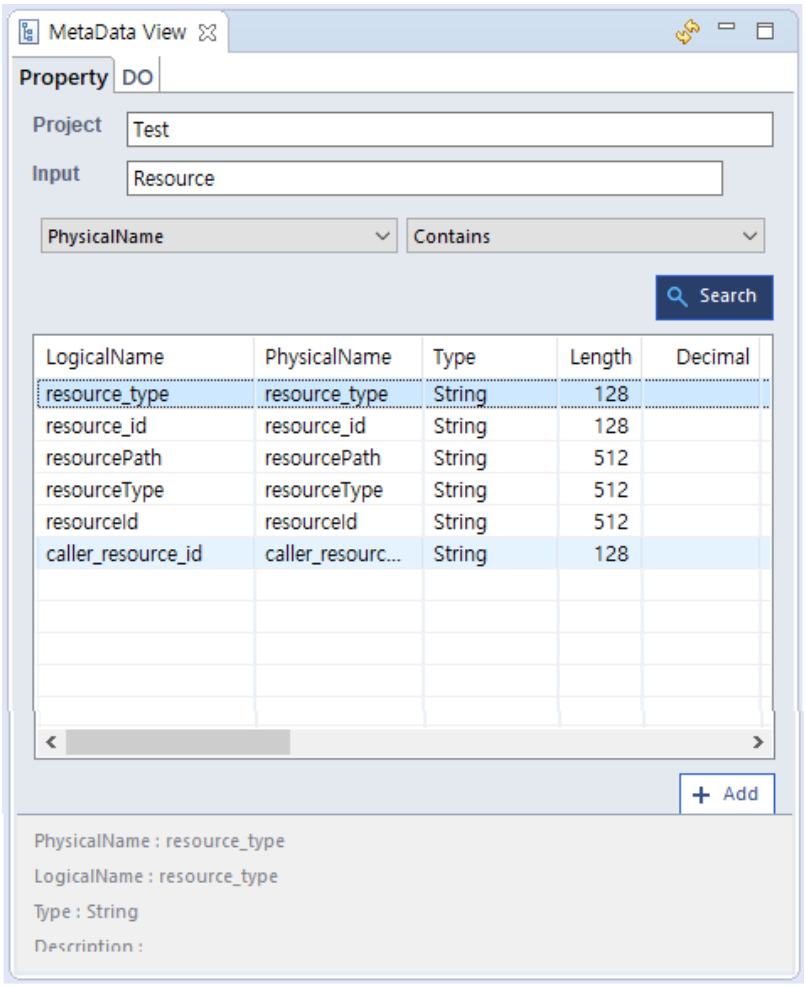
데이터 오브젝트 생성하려면 **MetaData View 화면**에서 등록된 메타를 조회해야 한다.

**MetaData View 화면**은 **ProStudio** 오른쪽 상단에 위치하며 **[Refresh Property Information]** 아이콘을 클릭해서 메타 데이터의 타입이나 길이가 변경되었을 때 로컬 Property를 등록된 메타 데이터로 동기화한다. 사용 가능한 메타 타입은 **[Property]**와 **[DO]** 탭에서 조회한다.

기본적으로 ProManager에서 등록한 메타만 조회가 가능 하지만, 커스터마이징 서비스로 제공하는 조회 옵션도 제공한다. 자세한 사용 방법은 메타 조회 기능을 참고한다.

- **[Property] 탭**

  메타 시스템에 등록된 변수를 검색하여 에디터에 추가할 수 있다.



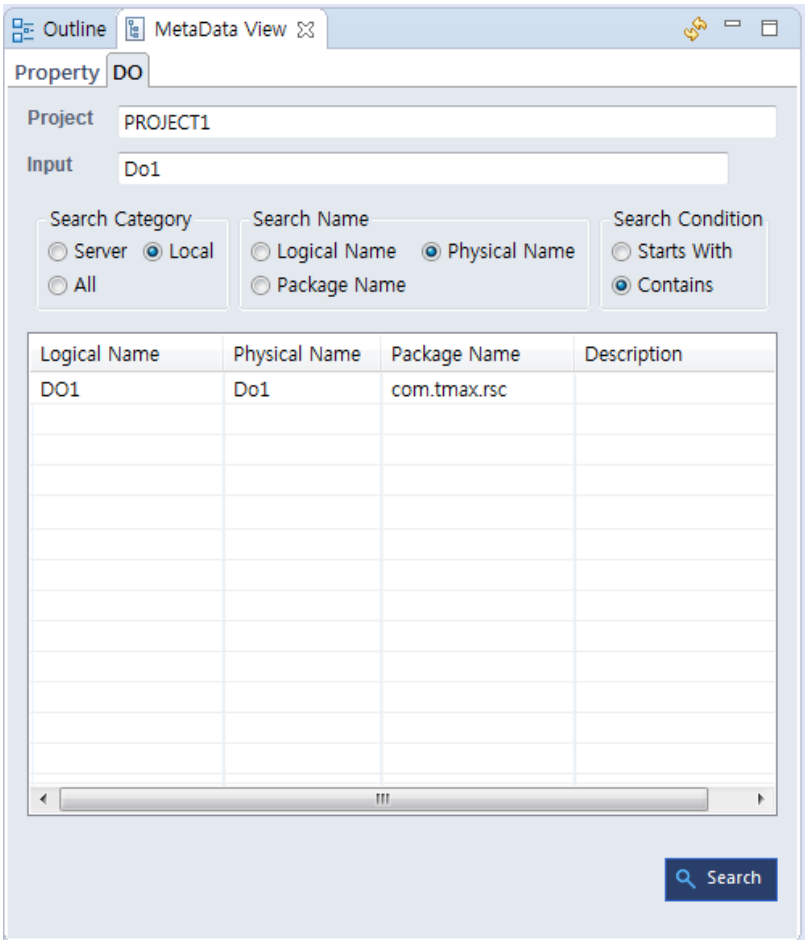

Property 정보 갱신 - [Property] 탭

| 항목 | 설명 |
|---|---|
| Project | Property를 추가할 데이터 오브젝트의 프로젝트를 나타낸다. |
| Input | 메타를 입력하여 검색할 수 있다. **[Search]** 버튼을 클릭하면 **'Input'** 항목에 입력된 값을 조회한다. 검색하여 리스팅된 메타 값을 지정한 후 **[Add]** 버튼을 클릭하면 에디터 화면에 필드를 추가할 수 있다. |
| 검색 명칭 | 메타에 등록된 Property를 검색어 명칭으로 검색한다.<br>• LogicalName : 논리명 기준으로 조회<br>• Group : 그룹 기준으로 조회<br>• PhysiclaName : 물리명 기준으로 조회 |

| 항목 | 설명 |
| --- | --- |
| 검색어 위치 | 메타에 등록된 Property를 검색어 위치를 선택해서 검색한다.<br>• Contains<br>• Starts With |
| Property 목록 | **'Input'** 항목의 값을 기준으로 조회된 Property가 나열된다. |

- **[DO] 탭**

  기존에 생성된 데이터 오브젝트를 검색하여 에디터에 추가할 수 있다.



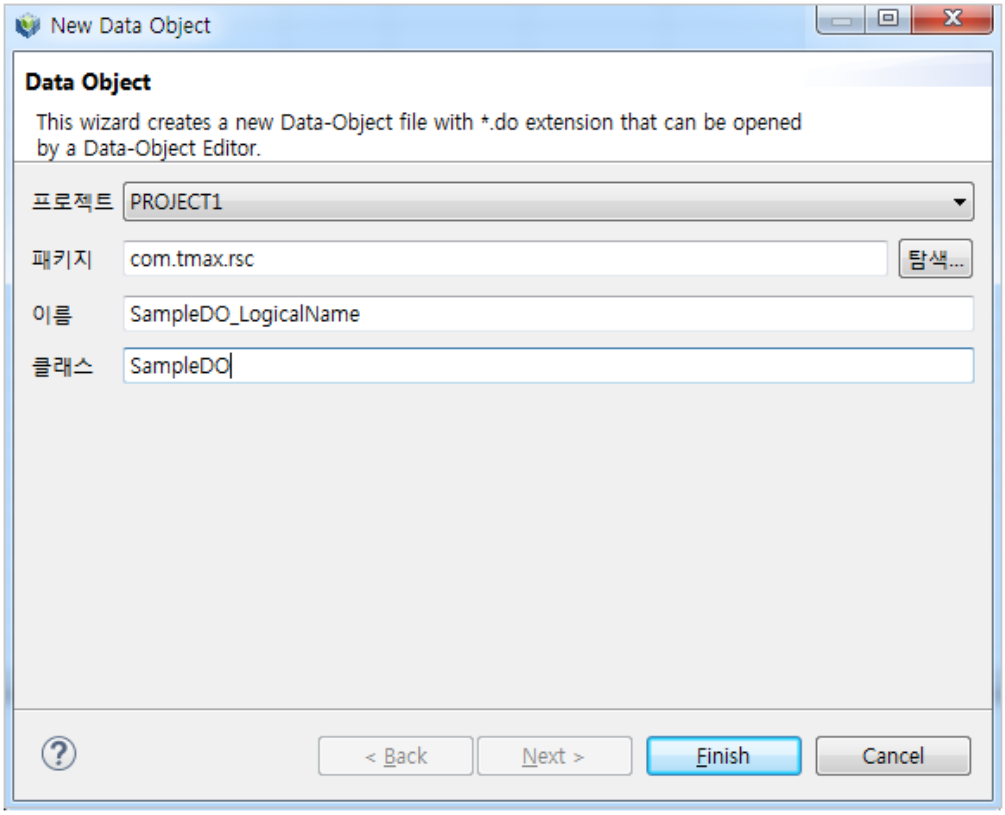

Property 정보 갱신 - [DO] 탭

| 항목 | 설명 |
| --- | --- |
| Project | 데이터 오브젝트를 포함할 데이터 오브젝트의 프로젝트를 나타낸다. |
| Input | 데이터 오브젝트를 입력하여 검색할 수 있다.<br><br>**[Search]** 버튼을 클릭하면 **'Input'** 항목에 입력된 값을 조회한다. 목록에서 메타 값을 더블클릭하면 에디터 화면에 필드를 추가할 수 있다. |

| 항목 | 설명 |
| --- | --- |
| Search Category | 기존에 생성된 데이터 오브젝트 검색 위치를 설정해서 검색한다.<br>• Server : DB의 리소스정보(DEV_RSC)기준으로 조회한다.<br>• Local : Local의 리소스정보를 기준으로 조회한다.<br>• All : Server와 Local을 모두 조회한다. |
| Search Name | 기존에 생성된 데이터 오브젝트를 어떤 검색 명칭으로 검색할지 정한다.<br>• Logical Name : 논리명 기준으로 조회한다.<br>• Physical Name : 물리명 기준으로 조회한다.<br>• Package Name : Package Name 기준으로 조회한다. |
| Search Condition | 'Input' 항목에 입력된 검색어 위치를 설정해서 데이터 오브젝트를 검색한다.<br>• Starts With : Input 값으로 시작하는 내역을 조회한다.<br>• Contains : Input 값을 포함하는 내역을 조회한다. |
| DO | **'Input'** 항목의 값을 기준으로 조회된 데이터 오브젝트가 리스트에 나열된다. |

## 3.2.2. 데이터 오브젝트 생성

**Package Explorer**의 컨텍스트 메뉴에서 **[New] > [DataObject]**를 선택하면 **New Data Object 화면**에서 데이터 오브젝트 생성할 수 있다.

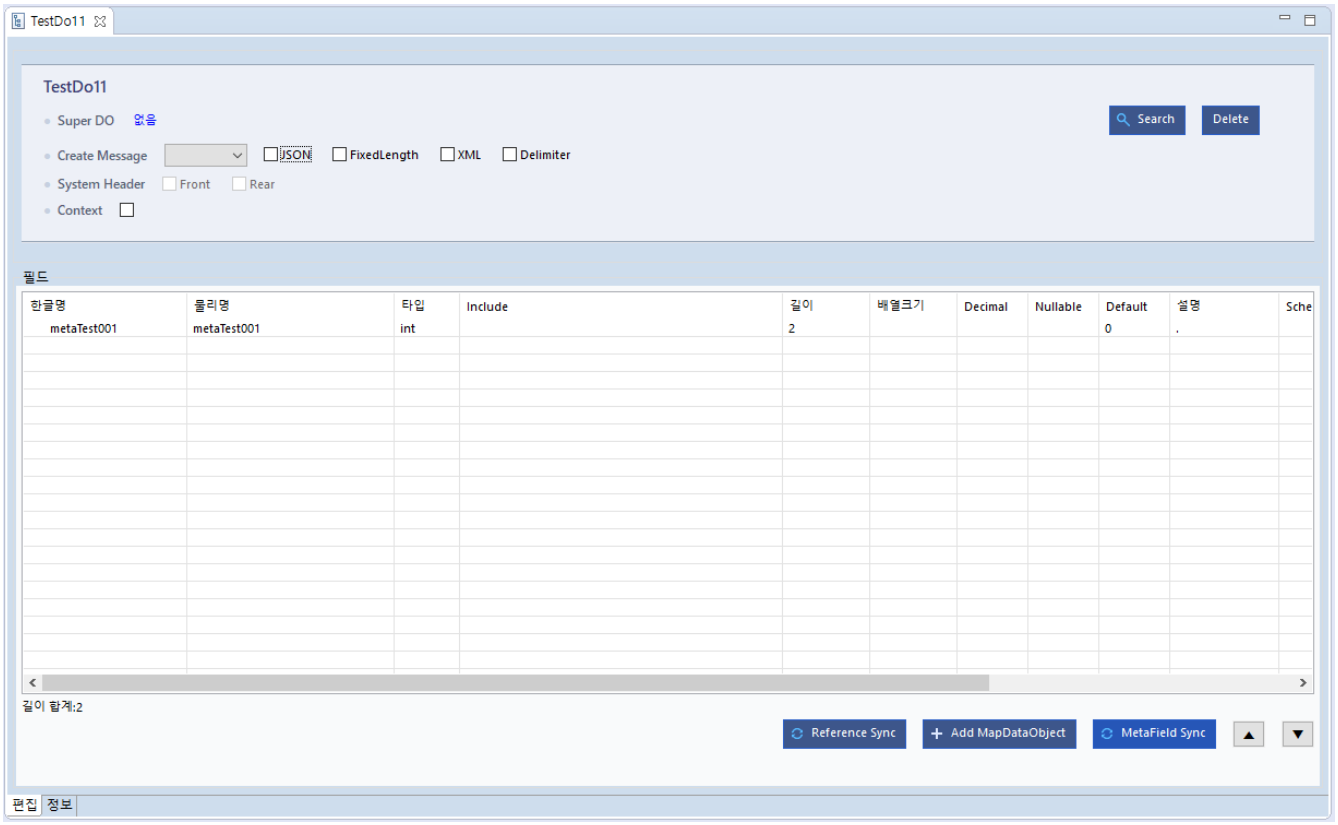



New Data Object 화면

| 항목 | 설명 |
| --- | --- |
| 프로젝트 | 생성할 데이터 오브젝트가 위치할 실제 프로젝트명을 선택한다. |
| 패키지 | 실제 사용하는 패키지 네이밍 규칙에 따라 패키지명을 정의한다.<br><br>리소스를 생성하는 경우 Fix된 Nameing Rule Check 기능을 제공하며, 자세한 내용은 Class Naming Rule Check를 참고한다. |
| 이름 | 데이터 오브젝트 객체의 논리명을 입력한다. |
| 클래스 | 데이터 오브젝트의 실제 생성되는 클래스 파일명을 입력한다. |

## 3.2.3. 데이터 오브젝트 에디터

**데이터 오브젝트 에디터**는 Super DO 검색 및 등록, 메지지 생성, 필드 편집 영역으로 나뉜다. 메타 및 데이터 오브젝트를 추가, 편집할 수 있으며 필드 편집 내용과 동일하게 FixedLength, XML, Delimiter, JSON 형태로 메시지 생성이 가능하다.




데이터 오브젝트 에디터

- **Super DO**

  Super DO를 생성한 후 Super DO를 상속받아 사용할 수 있다.

- **Create Message**

  Message Type을 체크하면 메시지 파일이 자동 생성되며 생성된 파일은 **Package Explorer**에서 확인할 수 있다.

| Message Type | 설명 |
|---|---|
| JSON | 입출력 전문이 JSON 방식으로 구성될 때 적용한다. |
| FixedLength | 입출력 전문의 기준이 길이에 따라 구분될 때 적용한다. |
| XML | 입출력 전문이 xml 방식으로 구성될 때 적용한다. |
| Delimiter | 입출력 전문이 구분자에 의해서 구성될 때 적용한다. |

- **Context**

  맵 데이터 오브젝트(MapDataObject)는 메타를 통한 멤버변수 선언 없이 사용하기 위한 데이터 오브젝트 이다.

- **필드**

  **MetaData View**에서 필드 편집 영역으로 컨텍스트 메뉴와 버튼을 이용해서 추가한 필드를 이동, 편집, 삭제할 수 있다.

  ◦ 필드 항목

| 항목 | 설명 |
|---|---|
| 한글명 | 메타의 Logical Name(논리명)이다. |
| 물리명 | 메타의 Physical Name이며, Java로 SourceGen될 때 변수의 기준이 된다. |
| 타입 | 메타의 Field Type이다. |
| 길이 | 메시지를 생성하는 경우 기준이 되는 길이이다. |
| 배열크기 | Array로 변수를 생성하는 경우 1 이상의 값을 입력한다.<br><br>JSON Type의 경우 unbound를 입력하면 가변처리가 가능하다. |
| Decimal | 실수형 Type의 경우 유효 소수점 자리수이다. |
| Nullable | Null 값이 입력 가능 여부 이다. |
| Default | 메타의 Default Value이며, 변수가 생성될 때의 기본값이다. |
| 설명 | 메타에 대한 상세 내용이다. |

  ◦ 컨텍스트 메뉴

| 메뉴 | 설명 |
|---|---|
| **[삭제]** | 선택된 필드를 삭제한다. |
| **[복사]** | 선택된 필드를 복사하여 다른 데이터 오브젝트 에디터 필드 영역에 붙여넣기를 할 수 있다. |
| **[붙여넣기]** | 복사된 필드를 필드 에디터에 붙여 넣는다. |

  ◦ 버튼

| 버튼 | 설명 |
|---|---|
| **[Reference Sync]** | 참조하고 있는 DO(target Do), SO, BO(assign Modules) 정보를 현행화한다. |

| 버튼 | 설명 |
| --- | --- |
| **[Add MapDataObject]** | MapDataObject를 추가한다.<br><br>MapDataObjectMsgJson를 DO에서 사용하기 위해 제공하는 기능으로 Json Type만 지원한다. |
| **[MetaField Sync Sync]** | ProManager에서 변경한 메타 정보를 현행화한다. |
| **[▲]** | 여러 개의 필드가 존재할 경우 선택된 필드를 위로 이동할 수 있다. |
| **[▼]** | 여러 개의 필드가 존재할 경우 선택된 필드를 아래로 이동할 수 있다. |

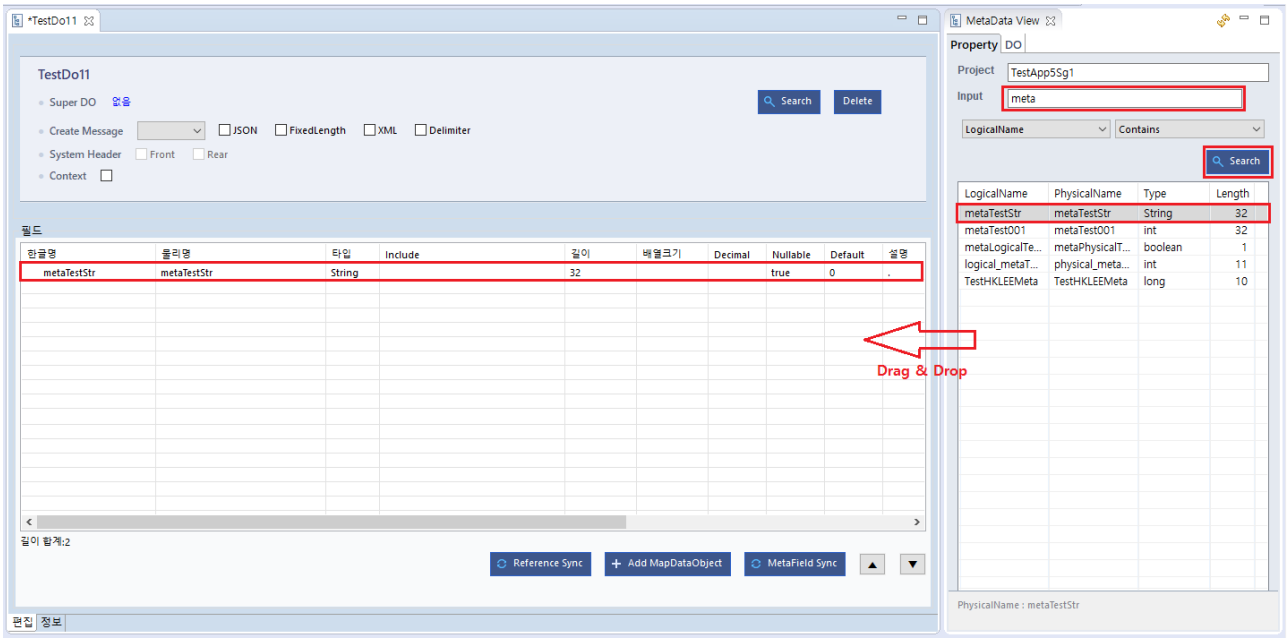
## 3.2.4. 개발 방법

본 절에서는 데이터 오브젝트를 개발하는 방법을 **단일 데이터 오브젝트**, **Include 데이터 오브젝트**, **부모 데이터 오브젝트**로 나눠서 설명한다.

### 3.2.4.1. 단일 데이터 오브젝트

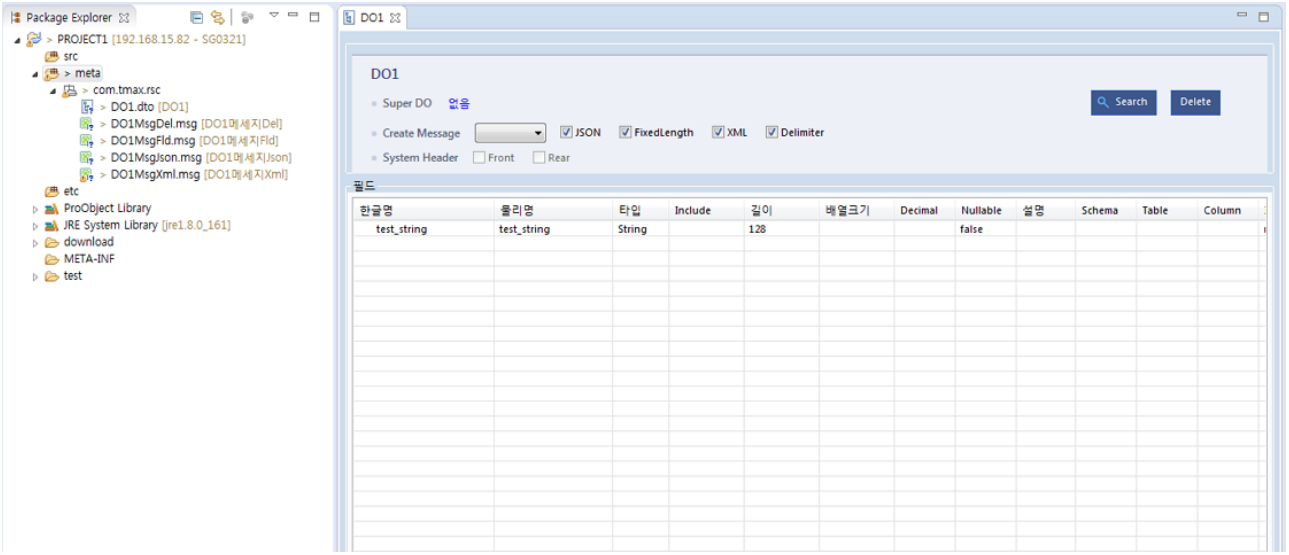
단일 데이터 오브젝트는 가장 기본적인 데이터 오브젝트 방식으로 다음의 방법으로 개발한다.

1. 데이터 오브젝트 생성을 참고해서 데이터 오브젝트를 생성한다.
2. 기존에 생성된 데이터 오브젝트를 오른쪽 **MetaData View 화면**의 **[Property]** 탭(Property 정보 갱신 - [Property] 탭)에서 메타를 검색한다. 추가할 항목을 선택한 후 에디터(데이터 오브젝트 에디터)의 **필드** 영역에 드래그 앤드 드롭해서 추가한다.



단일 데이터 오브젝트 개발 (1)

3. **'Create Message'** 항목에서 원하는 타입의 메시지를 체크한다.




단일 데이터 오브젝트 개발 (2)

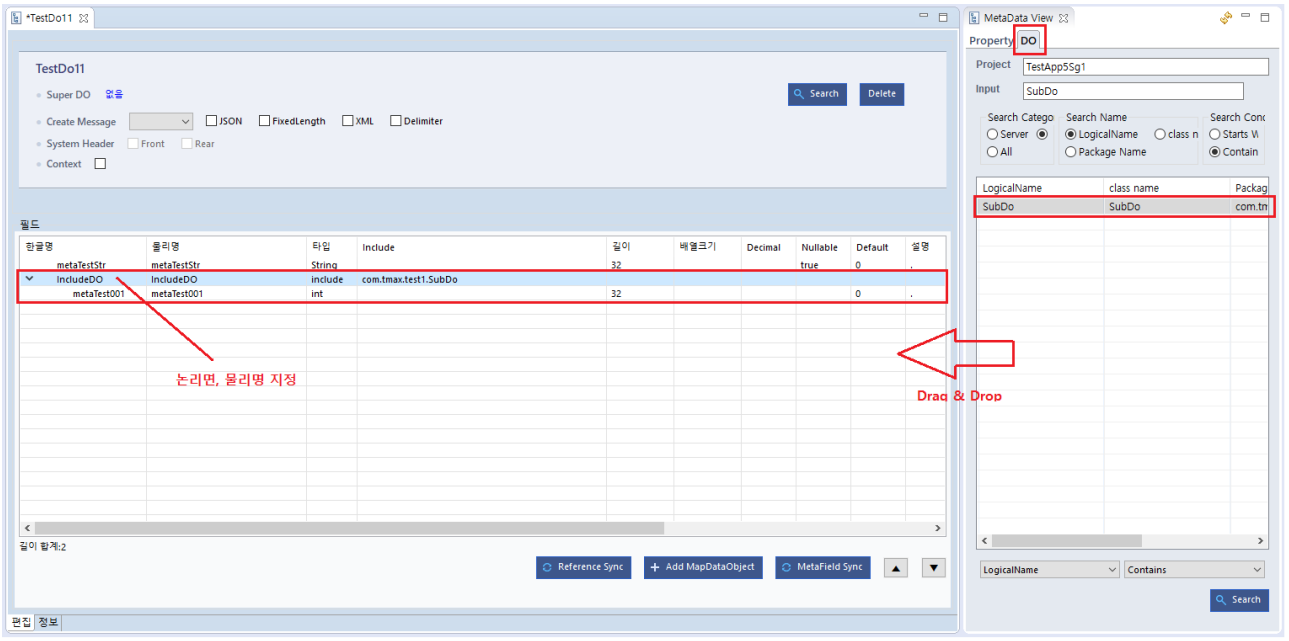
4. 단축키 <Ctrl>+S 또는 툴바의 **[Save]** 버튼을 클릭하여 데이터 오브젝트를 저장한다.

### 3.2.4.2. Include 데이터 오브젝트

데이터 오브젝트를 생성할 때 기존에 만들어진 데이터 오브젝트를 포함하여 생성할 수 있다. 주로 다건(List) 데이터 오브젝트를 생성할 때 사용한다.

다음은 Include 데이터 오브젝트를 개발하는 방법에 대한 설명이다.

1. 데이터 오브젝트 생성을 참고해서 데이터 오브젝트를 생성한다.
2. 기존에 생성된 데이터 오브젝트를 오른쪽 **MetaData View 화면**의 **[Property]** 탭(Property 정보 갱신 - [DO] 탭)에서 검색한다. 추가할 항목을 선택한 후 에디터의 **필드** 영역에 드래그 앤드 드롭해서 추가한다.




Include 데이터 오브젝트 개발

논리명, 물리명을 수정한다(논리명, 물리명 지정은 필수이다). Include DO는 자기 자신을 include할 수 없으며 다른 필드 타입과 마찬가지로 배열로 사용하려면 fixed된 배열 크기를 지정해야 된다.

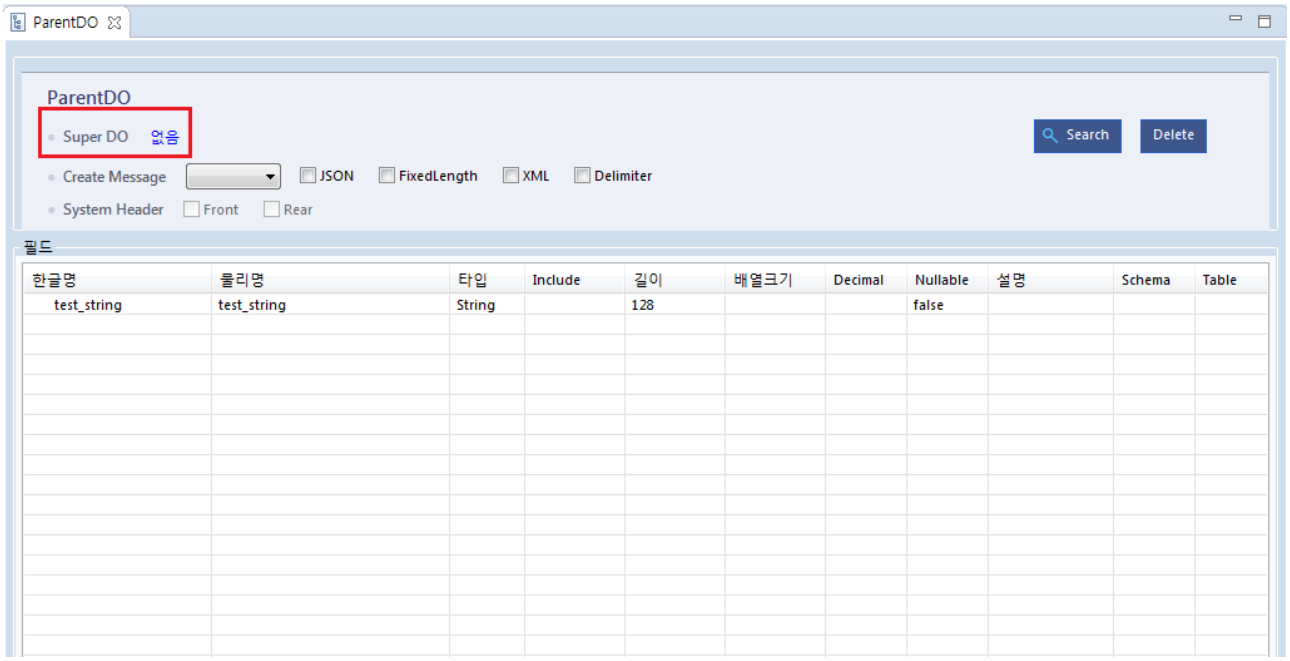
3. **'Create Message'** 항목에서 메시지 타입을 체크한다. 생성된 메시지는 **Package Explorer**에서 [.msg] 파일로 확인할 수 있다.
4. 단축키 <Ctrl>+S 또는 툴바의 **[Save]** 버튼을 클릭하여 데이터 오브젝트를 저장한다.

### 3.2.4.3. 부모 데이터 오브젝트

공통적으로 사용되는 필드들은 부모 데이터 오브젝트에 추가하고 이후 생성되는 데이터 오브젝트에 부모 데이터 오브젝트를 지정하면 공통 필드들을 따로 추가 없이 상속받아 사용할 수 있다.

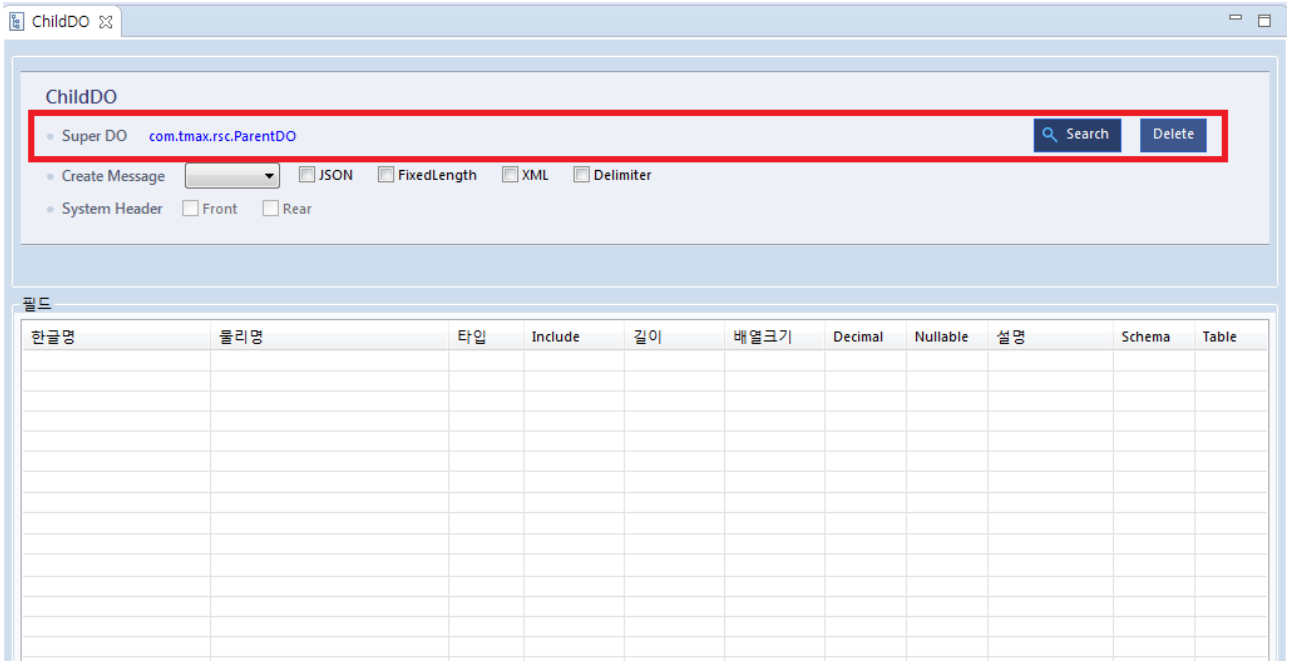
다음은 부모 데이터 오브젝트(Super DO) 사용 방법에 대한 설명이다.

1. 데이터 오브젝트 생성을 참고해서 데이터 오브젝트를 생성한다.
2. 부모 데이터 오브젝트로부터 상속받을 데이터 오브젝트를 생성한다(데이터 오브젝트 생성 참고).
3. 다음은 부모 데이터 오브젝트의 필드 화면이다.



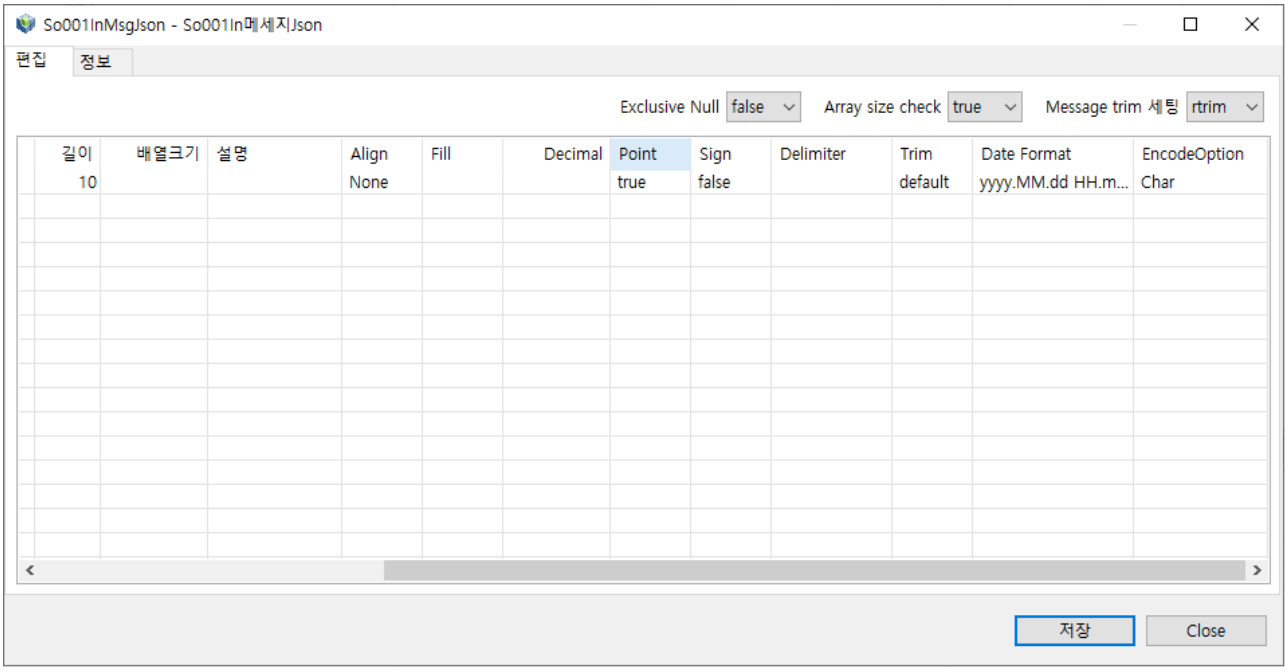
부모 데이터 오브젝트 개발 (1)

4. **[Search]** 버튼을 클릭한 후 부모로 지정할 데이터 오브젝트를 찾아서 선택한 후 필수 필드로 구성한 데이터 오브젝트를 부모로 선언한다.




부모 데이터 오브젝트 개발 (2) - Super DO 지정 후

5. **'Create Message'** 항목에서 메시지 타입을 체크한다. 항목의 콤보박스를 클릭하면 다음과 같이 상세 메시지를 설정할 수 있다.




Create Message - 상세 Message 설정 화면(Json 선택 시)

◦ **검색 조건**

| 항목 | 설명 |
|---|---|
| Exclusive Null<br><br>(Json Type) | Value값이 Null일 경우 marshal하지 않는다.<br>• true : Null일 경우 제외한다.<br>• false : Null일 경우 Json Node를 추가한다. |

| 항목 | 설명 |
| --- | --- |
| Array size check<br><br>(Json Type) | Array Size를 Check하는 옵션이다. |
| Message Trim 세팅 | 메시지를 Trim하는 옵션이다. |

◦ **설정 항목**

| 항목 | 설명 |
| --- | --- |
| 배열크기 | Json type일 경우 가변적인 메시지를 처리하는 경우 **DO 편집기**의 **'배열크기'**에 ubound를 입력하여 설정 가능하다. |
| Date Format | Marshal할 때 기준이 되는 Date Format를 정의한다.<br><br>해당 설정은 ProbuilderConfig.xml의 dateTypeFormatList Node에 다양한 Date Format을 설정할 수 있다.<br><br>`<dateTypeFormatList>yyyy.MM.dd HH.mm.ss,yyyy,MM, long</dateTypeFormatList>` |

6. 단축키 <Ctrl>+S 또는 툴바의 **[Save]** 버튼을 클릭하여 데이터 오브젝트를 저장한다.

# 3.3. 데이터 오브젝트 팩토리(DB)

DB Type 데이터 오브젝트 팩토리(DataObjectFactory, DOF)는 데이터베이스에 접근하여 데이터를 조회하거나 조작하는데 사용하는 객체이다.

데이터베이스를 처리할 때 JDBC를 이용할 경우 사용자가 직접 구현해야 했던 여러 가지 반복적인 작업들을 대신해줌으로써 개발의 절차를 간소화한다. 또한 사용자가 직접 작성한 SQL 문을 미리 검증하거나 실행계획을 조회할 수 있는 기능을 제공한다.

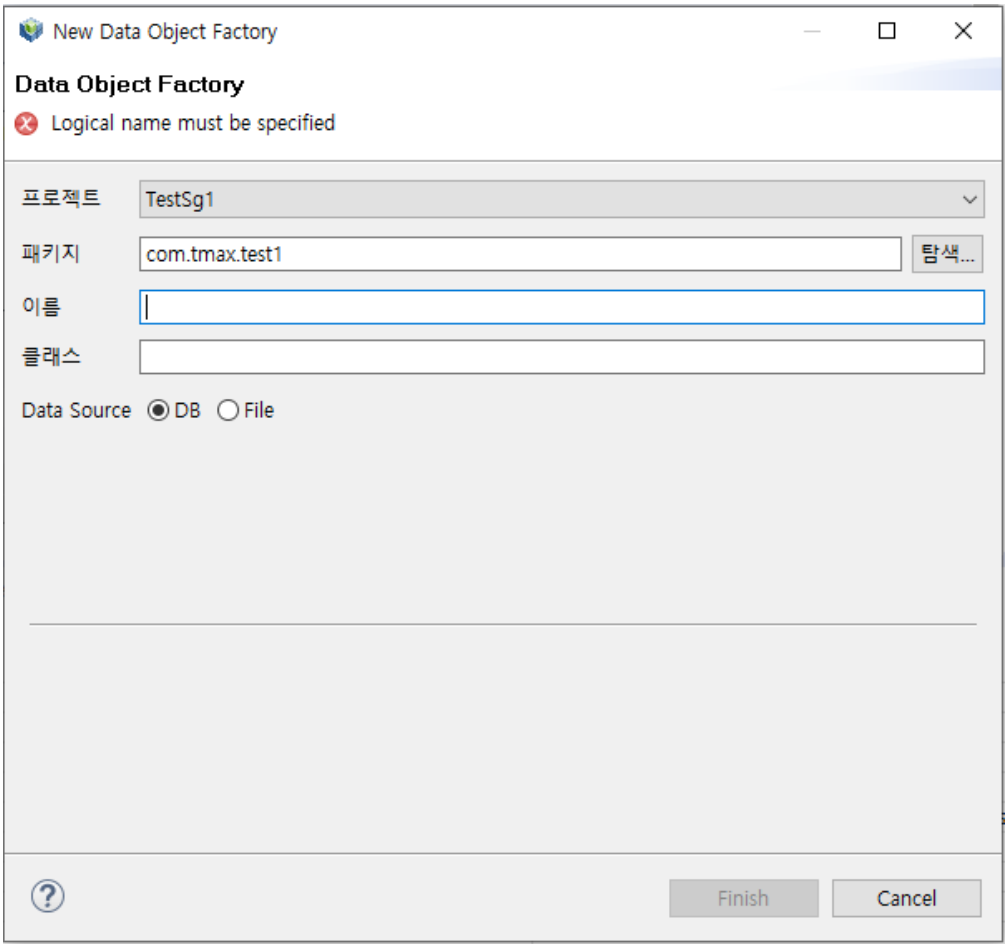
본 절은 데이터 오브젝트 팩토리의 기본 개념과 시작하는 방법, 예제를 설명한다.

## 3.3.1. 환경설정

ProStudio 환경설정에서 JDBC 설정이 되어 있어야 한다. 자세한 내용은 사용 환경 확인을 참고한다.

## 3.3.2. 데이터 오브젝트 팩토리 생성

**Package Explorer**의 컨텍스트 메뉴에서 **[New] > [DataObjectFactory]**를 선택한 후 **New Data Object Factory 화면**에 데이터 오브젝트 팩토리 정보를 입력한다.

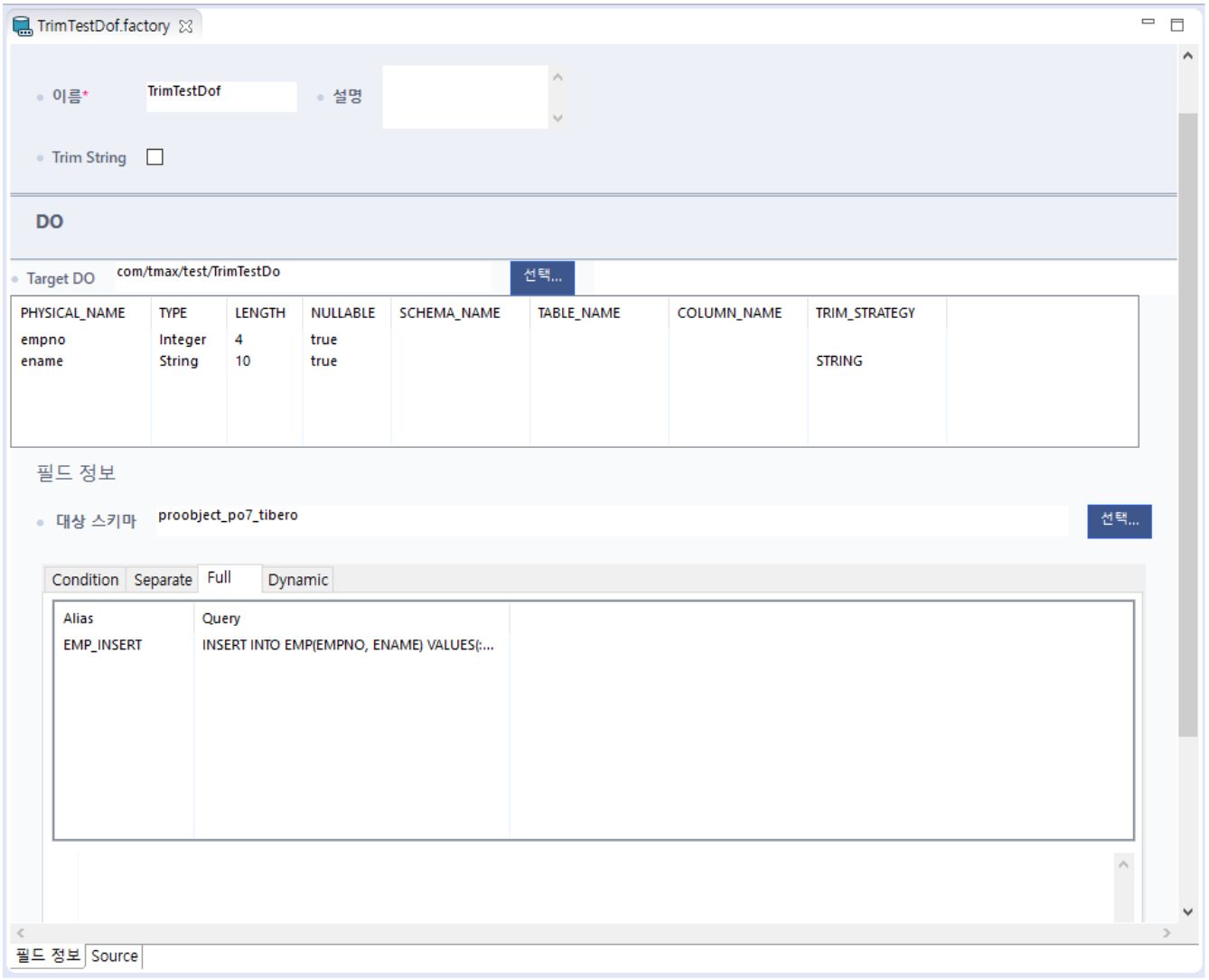


New Data Object Factory 화면

| 항목 | 설명 |
|---|---|
| 프로젝트 | 생성할 데이터 오브젝트 팩토리가 위치할 실제 프로젝트명을 선택한다. |
| 패키지 | 실제 사용하는 패키지 네이밍 규칙에 따라 패키지명을 정의한다.<br><br>Fix된 Nameing Rule Check기능을 제공하며, 자세한 내용은 Class Naming Rule Check를 참고한다. |
| 이름 | 데이터 오브젝트 팩토리 객체의 논리명을 입력한다. |
| 클래스 | 데이터 오브젝트 팩토리의 실제 생성되는 클래스 파일명을 입력한다. |
| Data Source | 데이터소스 유형을 선택한다.<br><br>◦ DB : SQL을 처리할 때 선택한다.<br>◦ File : File을 읽고 쓸 수 있을 때 선택한다. |

## 3.3.3. 데이터 오브젝트 팩토리 에디터

**DB 데이터 오브젝트 팩토리 에디터**는 기본 정보, Target DO 정보, 스키마 정보, 쿼리 영역으로 구성된다.



DB 데이터 오브젝트 팩토리 에디터

- **기본 사항**

| 항목명 | 설명 |
|---|---|
| 이름 | Target DO는 쿼리의 결과 값을 받는다. |
| 설명 | 해당 DOF 논리명 및 해당 리소스 설명이다. |
| Trim String | Do의 변수 Type이 String일 때 등록된 등록된 길이만큼만 설정한다.<br><br>체크를 하는 경우 Time String 처리를 하고, 체크하지 않으면 Trime하지 않는다. |

- **Target DO 정보**

데이터 오브젝트 팩토리 에디터는 하나의 데이터 오브젝트를 필수로 Target DO로 지정해야 된다. Target DO는 실행되는 쿼리의 타입에 따라 쿼리에 입력 또는 결과값을 담당하게 된다.

| 쿼리 타입 | 설명 |
|---|---|
| SELECT | Target DO는 쿼리의 결과 값을 받는다. |
| UPDATE | Target DO는 쿼리에서 SET 절에 들어가는 입력 값들을 가지고 있다. |
| INSERT | Target DO는 쿼리에서 INTO 절에 들어가는 입력 값들을 가지고 있다. |

| 쿼리 타입 | 설명 |
| --- | --- |
| DELETE | Target DO는 사용되지 않는다. |

- **스키마 정보**

  **쿼리 영역**에서 쿼리 작성 및 작성된 SQL 쿼리가 정상 동작하는지 정합성을 체크할 수 있다.

  다음의 쿼리 유형의 동작을 확인할 수 있다.

| 탭 | 설명 |
| --- | --- |
| **[Condition]** 탭 | 팩토리에 자신이 사용할 쿼리를 지정한다. (조건은 Separate에서 입력) |
| **[Separate]** 탭 | Condition의 조건절을 바꿔가면서 사용한다. (Base SQL은 Condition 유형) |
| **[Full]** 탭 | WHERE 절이 고정적인 CRUD 유형의 SQL 입력 유형이다. (데이터 오브젝트 팩토리 생성(Full) 참고) |
| **[Dynamic]** 탭 | 업무 프로그램에서 입력되는 Sub SQL과 조합되어 수행되는 유형이다. (데이터 오브젝트 팩토리 생성(Dynamic) 참고) |

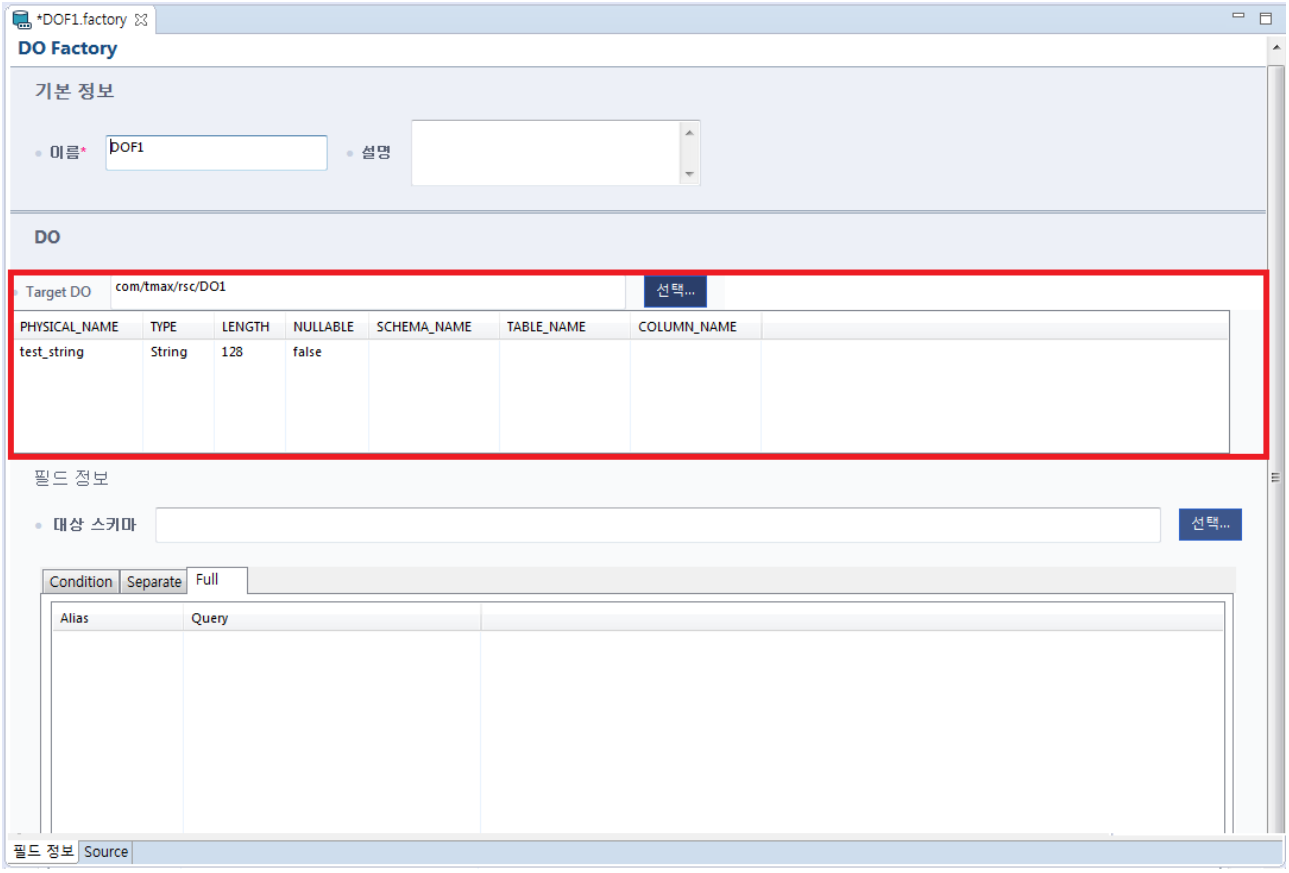
## 3.3.4. 개발 방법

본 절에서는 데이터 오브젝트 팩토리 생성 및 사용법을 설명한다. 데이터 오브젝트 팩토리는 Full, Dynamic으로 나눠서 생성한다.

### 3.3.4.1. 데이터 오브젝트 팩토리 생성(Full)

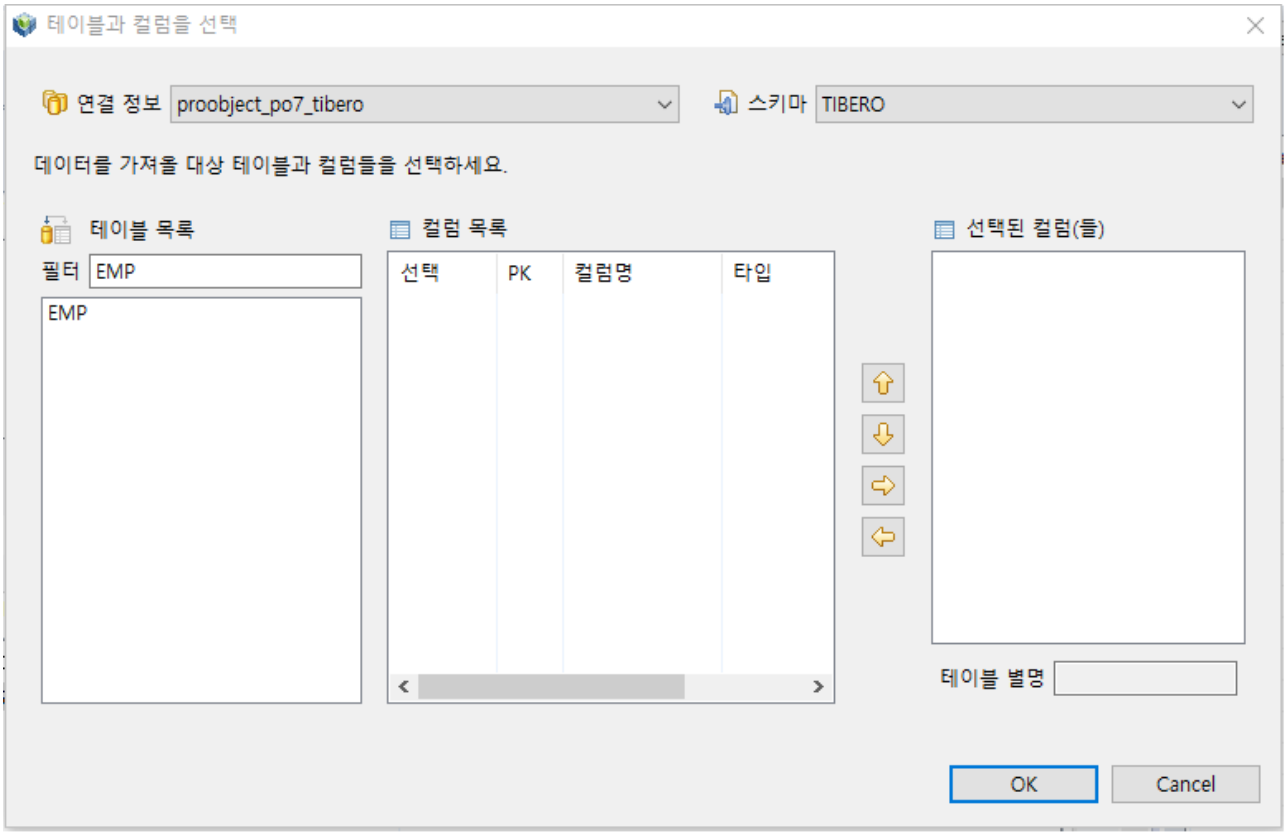
다음은 데이터 오브젝트 팩토리 에디터 객체를 생성해서 쿼리를 입력하는 과정에 대한 설명이다.

1. 데이터 오브젝트 팩토리 생성을 참고해서 데이터 오브젝트 팩토리를 생성한다.
2. Target DO의 **[선택..]** 버튼을 클릭해서 Target DO를 지정한다. Include 데이터 오브젝트 및 부모 데이터 오브젝트는 Target DO로 지정될 수 없다.



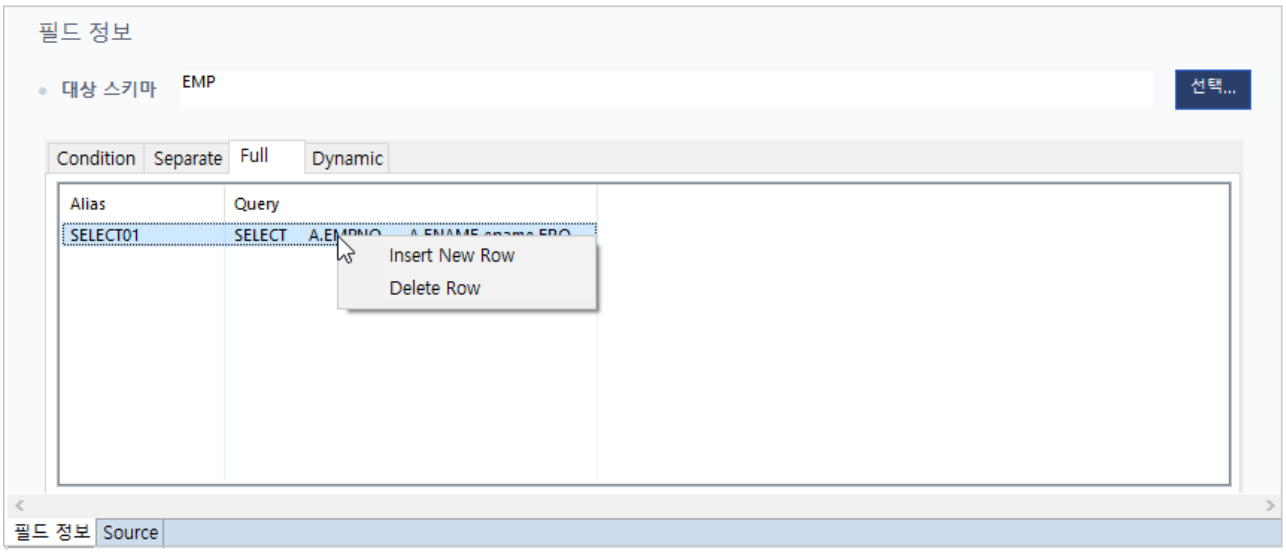

데이터 오브젝트 팩토리 생성 - Full (1)

3. **'대상 스키마'** 항목의 **[선택..]** 버튼을 클릭해서 데이터소스 및 DB 스키마를 지정한다. 테이블과 컬럼 선택 화면에서 스키마 정보를 선택한 후 **[OK]** 버튼을 클릭한다.



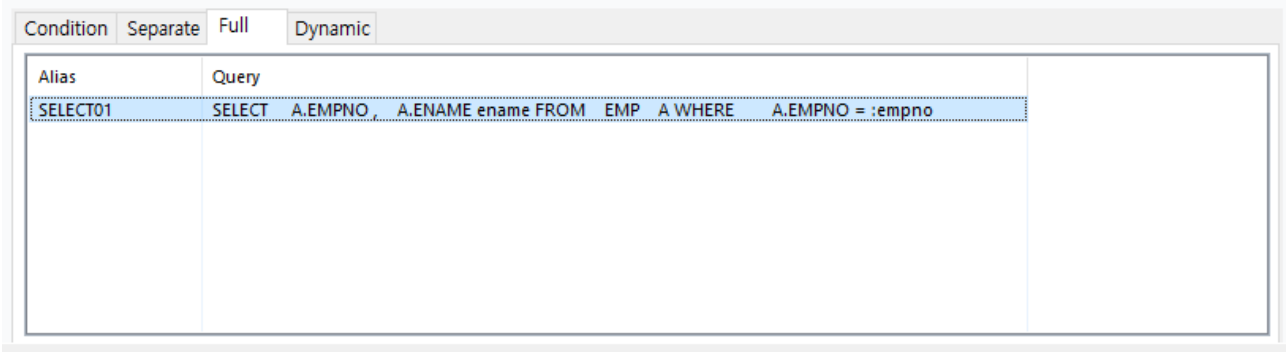

데이터 오브젝트 팩토리 생성 - Full (2)

4. **필드 정보**의 **[Full]** 탭을 선택한 후 컨텍스트 메뉴에서 **[Insert New Row]**를 선택해서 쿼리를 추가한다. 사용자는 다양한 타입의 여러 개의 쿼리를 추가할 수 있다.




데이터 오브젝트 팩토리 생성 - Full (3)

Dynamic keyword는 파라미터 앞에 콜론(:)을 붙여서 표시한다(Where 절 및 set 절에 dynamic keyword가 들어갈 수 있다).



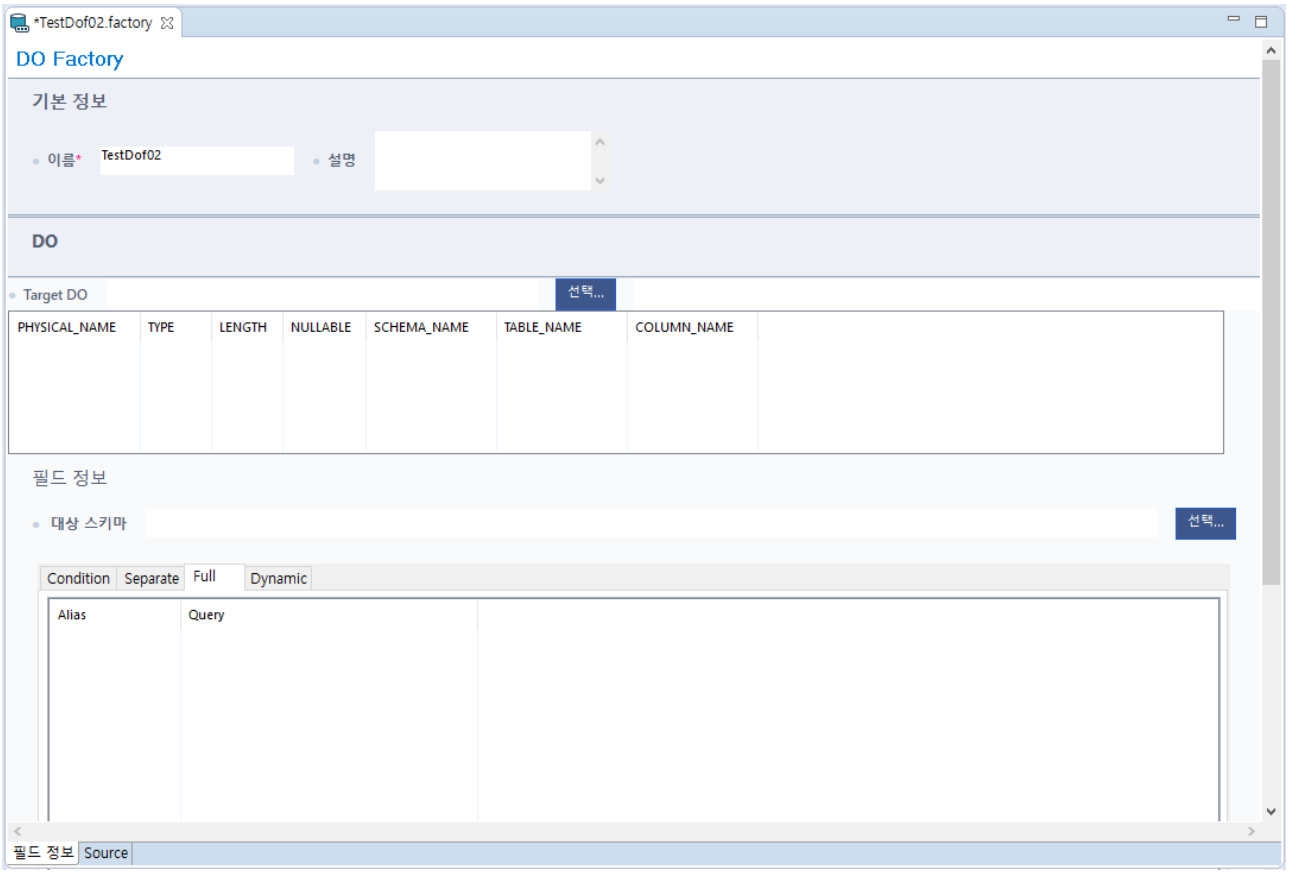

데이터 오브젝트 팩토리 생성 - Full (4) - 쿼리 추가

5. 데이터 오브젝트 팩토리 에디터를 저장한다.

### 3.3.4.2. 데이터 오브젝트 팩토리 생성(Dynamic)

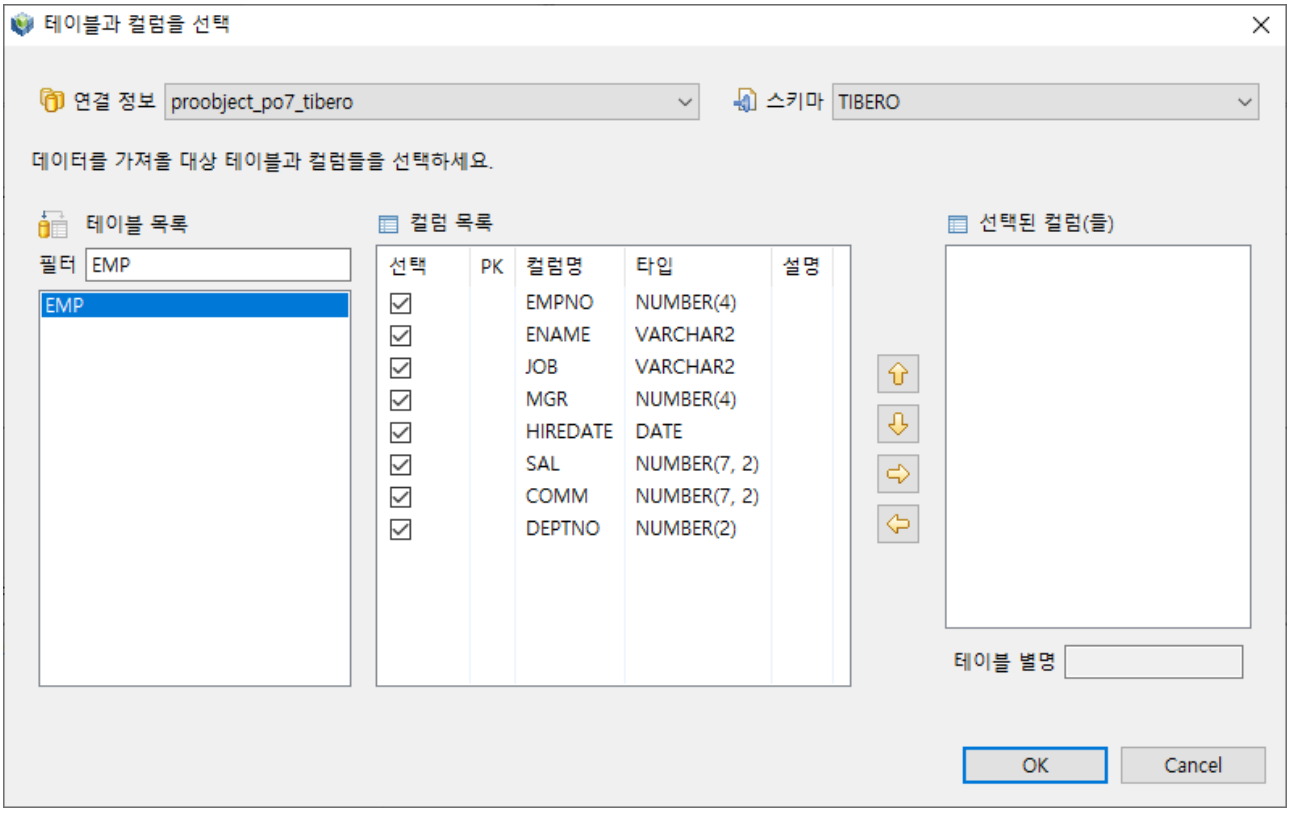
다음은 데이터 오브젝트 팩토리 에디터 객체를 Dynamic으로 생성해서 쿼리를 입력하는 과정에 대한 설명이다.

1. 데이터 오브젝트 팩토리 생성을 참고해서 데이터 오브젝트 팩토리를 생성한다.
2. Target DO의 **[선택..]** 버튼을 클릭해서 Target DO를 지정한다. Include 데이터 오브젝트 및 부모 데이터 오브젝트는 Target DO로 지정될 수 없다.



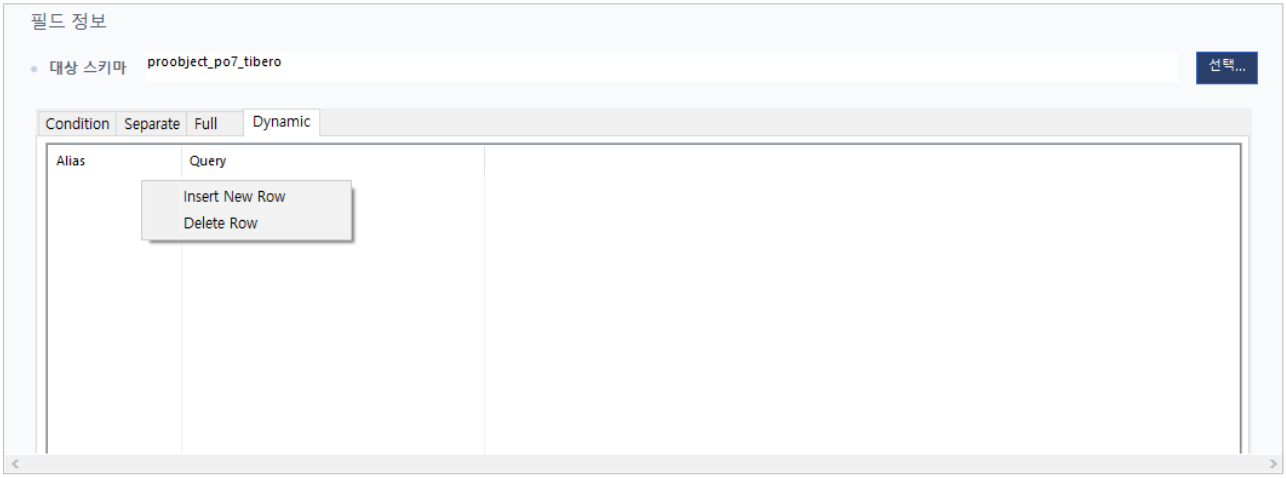

데이터 오브젝트 팩토리 생성 - Dynamic (1)

3. **'대상 스키마'** 항목의 **[선택..]** 버튼을 클릭해서 데이터소스 및 DB 스키마를 지정한다. 테이블과 컬럼 선택 화면에서 스키마 정보를 선택한 후 **[OK]** 버튼을 클릭한다.



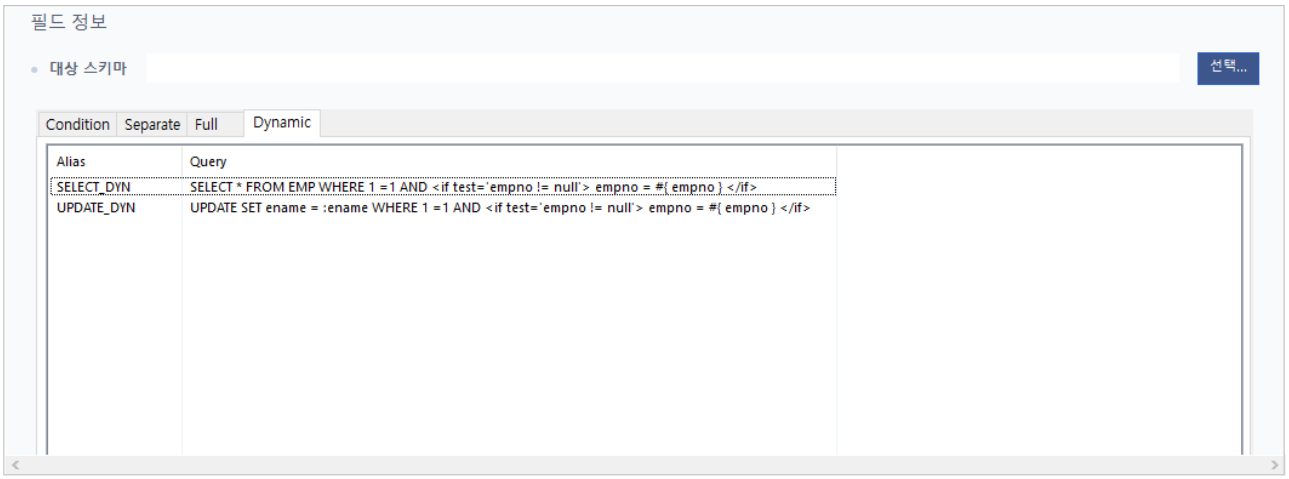

데이터 오브젝트 팩토리 생성 - Dynamic (2)

4. 필드 정보의 **[Dynamic]** 탭을 선택한 후 컨텍스트 메뉴에서 **[Insert New Row]**를 선택해서 쿼리를 추가한다.




데이터 오브젝트 팩토리 생성 - Dynamic (3)

사용자는 다양한 타입의 여러 개의 쿼리를 추가할 수 있다.




데이터 오브젝트 팩토리 생성 - Dynamic (4)

5. 데이터 오브젝트 팩토리 에디터를 저장하면 다음과 같이 생성된 소스를 조회할 수 있다.

```
SELECT_DYN("SELECT * FROM EMP WHERE 1 =1 AND <if test='empno != null'> empno = #{ empno } </if>"){
    @Override
    public String getDynamicQuery(TestDof02 factory) {
        Map<String,Object> varMap = new HashMap<>();
        for(MetaField meta: factory.getFactoryFields()){
        varMap.put( meta.getFieldName(), factory.getField(meta.getFieldName()));
        }
        ExpressionEvaluator evaluator = new ExpressionEvaluator();
        StringBuilder sqlSrcBuilder = new StringBuilder();
        {
          sqlSrcBuilder.append("SELECT * FROM EMP WHERE 1 =1 AND\n");
          if(evaluator.evaluateBoolean("empno != null",varMap)){
            sqlSrcBuilder.append("empno = #{ empno }\n");

          }
        }
        return sqlSrcBuilder.toString();
    }
},
```

데이터 오브젝트 팩토리 생성 - Dynamic (5) - SELECT

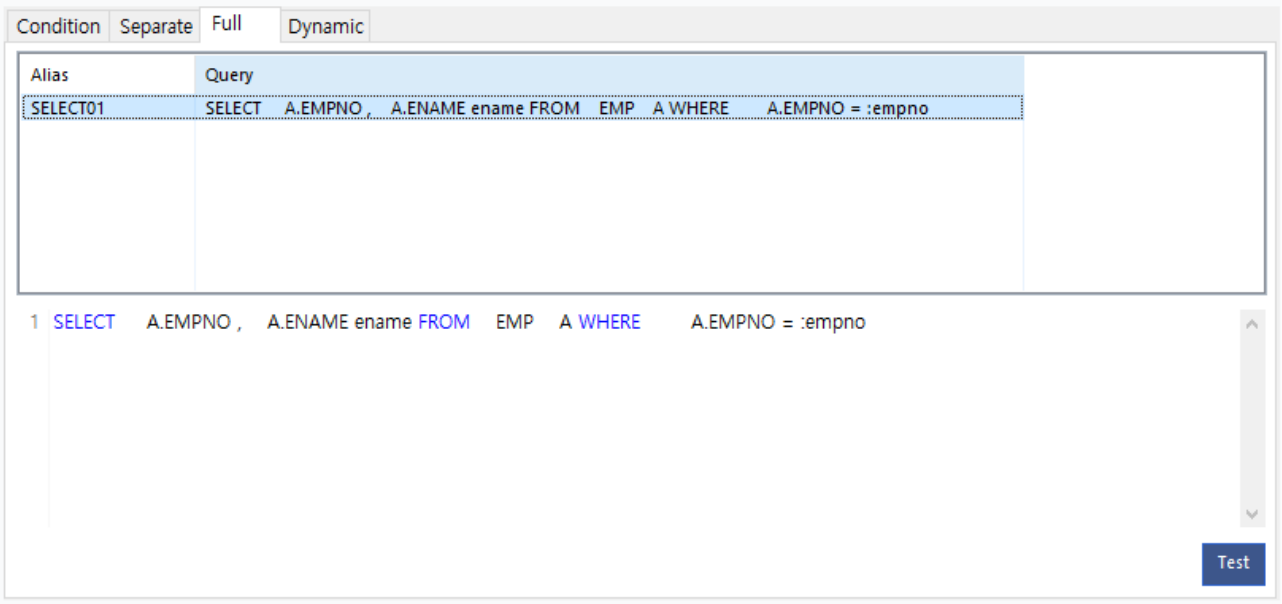
```
UPDATE_DYN("UPDATE SET ename = ? WHERE 1 =1 AND <if test='empno != null'> empno = #{ empno } </if>"){
    @Override
    public String getDynamicQuery(TestDof02 factory) {
        Map<String,Object> varMap = new HashMap<>();
        for(MetaField meta: factory.getFactoryFields()){
        varMap.put( meta.getFieldName(), factory.getField(meta.getFieldName()));
        }
        ExpressionEvaluator evaluator = new ExpressionEvaluator();
        StringBuilder sqlSrcBuilder = new StringBuilder();
        {
          sqlSrcBuilder.append("UPDATE SET ename = :ename WHERE 1 =1 AND\n");
          if(evaluator.evaluateBoolean("empno != null",varMap)){
            sqlSrcBuilder.append("empno = #{ empno }\n");

          }
        }
        return sqlSrcBuilder.toString();
    }
}
```

데이터 오브젝트 팩토리 생성 - Dynamic (6) - UPDATE

### 3.3.4.3. 데이터 오브젝트 팩토리 쿼리 테스트

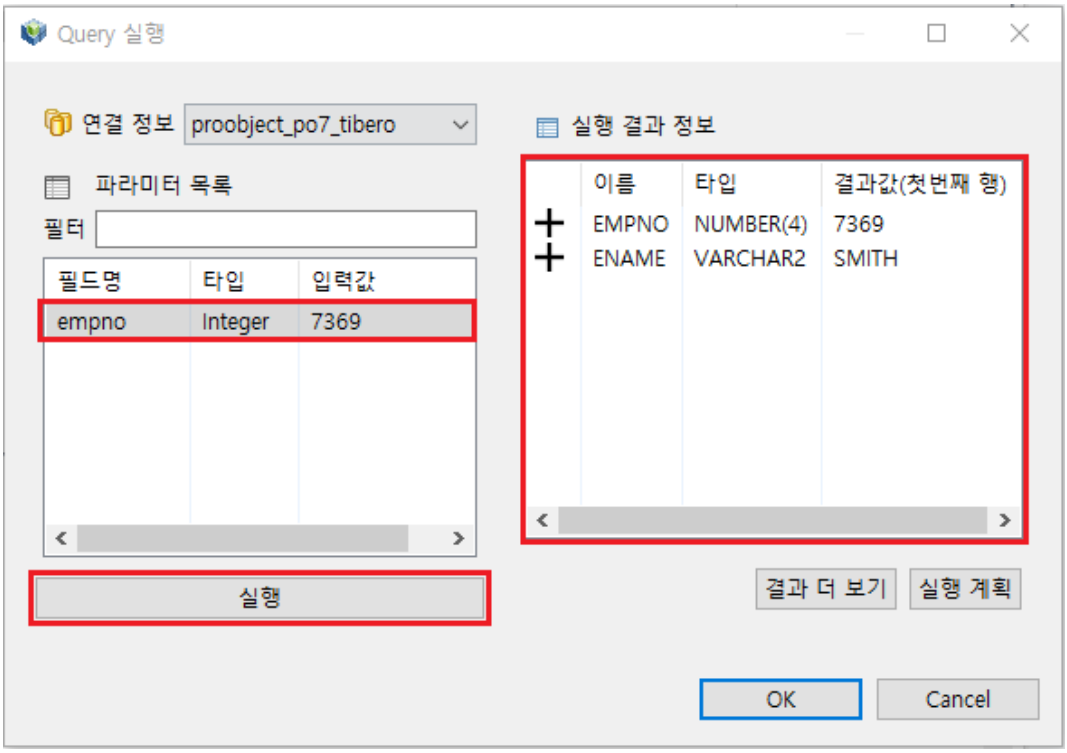
다음은 작성한 쿼리를 DB 직접 적용해 보고 미리 검증하고 실행계획을 조회하는 방법이다.

1. 쿼리 테이블에서 테스트할 쿼리를 선택한 후 **[Test]** 버튼을 클릭한다.



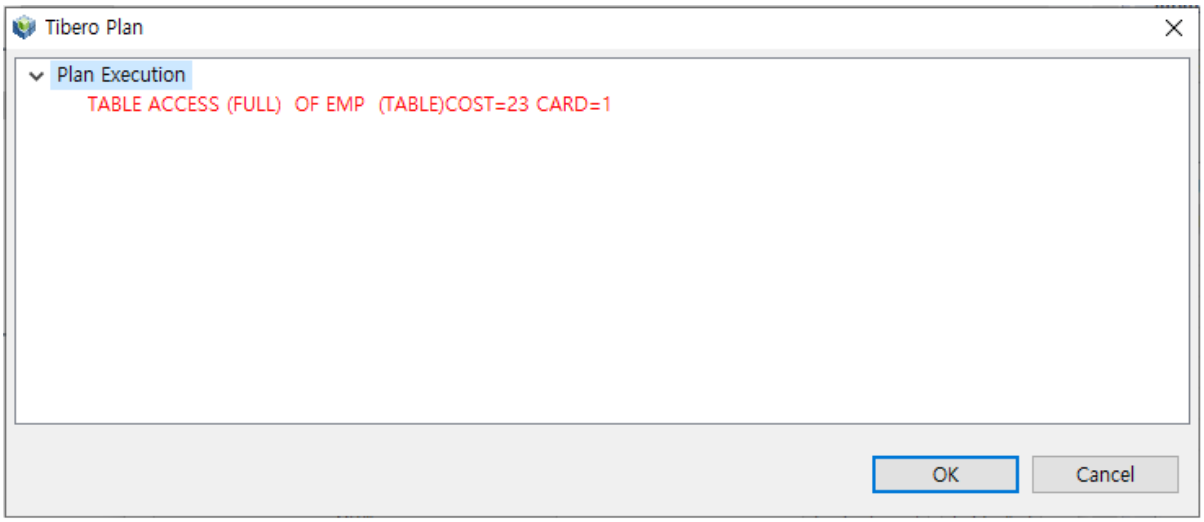

쿼리 선택

2. **Quey 실행 화면**에서 Dynamic keyword에 테스트 값을 넣은 후 **[실행]** 버튼을 클릭하면 SQL 문의 정합성을 확인한 후 실제 DB에 SQL 문을 수행하여 실행 결과 값이 **실행 결과 정보**에 조회된다. SELECT SQL 문의 경우 결과 값을 보여주고 INSERT/UPDATE/DELETE SQL 문은 변형된 행 개수를 보여준다. DB에 적용된 내용은 커밋되지 않는다.




테스트 쿼리 화면

3. **[실행 계획]** 버튼을 클릭하면 SQL 수행에 대한 Oracle/Tibero의 실행 계획(Execution plan)을 확인할 수 있다.

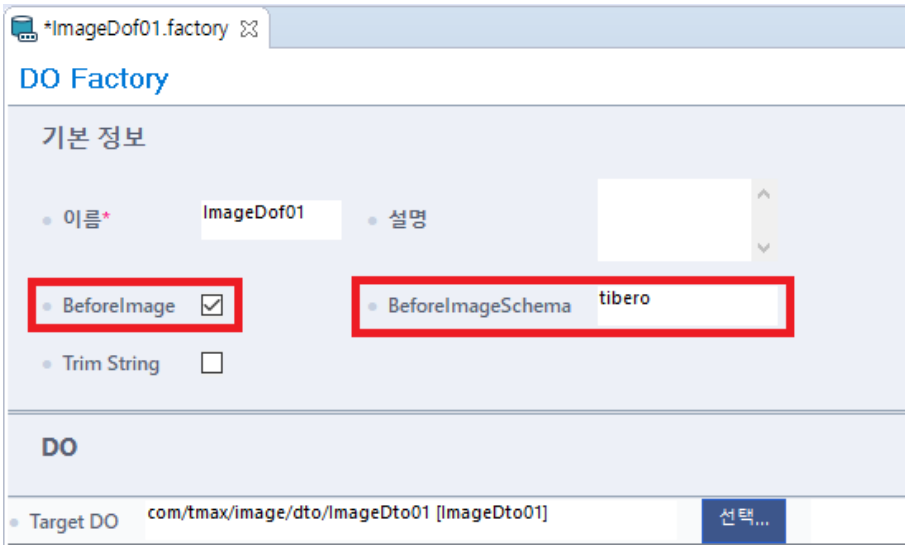



쿼리 실행 계획

### 3.3.4.4. 데이터 오브젝트 팩토리 비포이미지 설정

다음은 데이터 오브젝트 팩토리 비포이미지 설정 방법이다.

1. 비포이미지의 기본 정보(이름, 설명)을 설정한다.
2. **'BeforeImage'** 항목을 체크하고, **'BeforeImageSchema'** 항목을 설정한다.



비포이미지 선택 및 비포이미지 스키마 설정

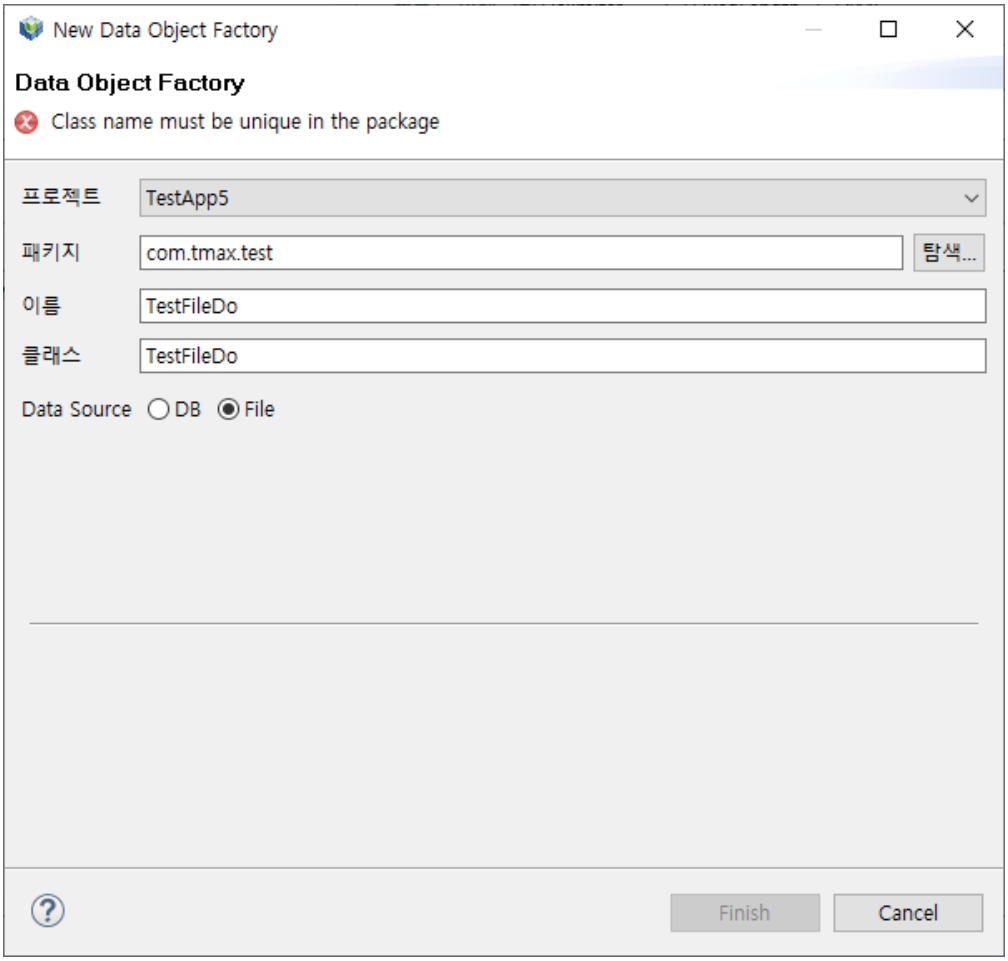
# 3.4. 데이터 오브젝트 팩토리(FILE)

File Type 데이터 오브젝트 팩토리(DataObjectFactory, DOF)는 Layout을 가지는 File을 생성하거나 읽을 때 사용하는 객체이다. Header, Body, Tail 영역이 구분되어 있으며, 지원하는 Message Type은 Delimiter, XML, FixedLength이다.

본 절에서는 File Type 데이터 오브젝트 팩토리의 기본 개념과 시작하는 방법, 예제를 설명한다.

## 3.4.1. 데이터 오브젝트 팩토리 생성

**Package Explorer**의 컨텍스트 메뉴에서 **[New] > [DataObjectFactory]**를 선택한 후 **New Data Object Factory 화면**에서 데이터 오브젝트 팩토리를 생성할 수 있다.

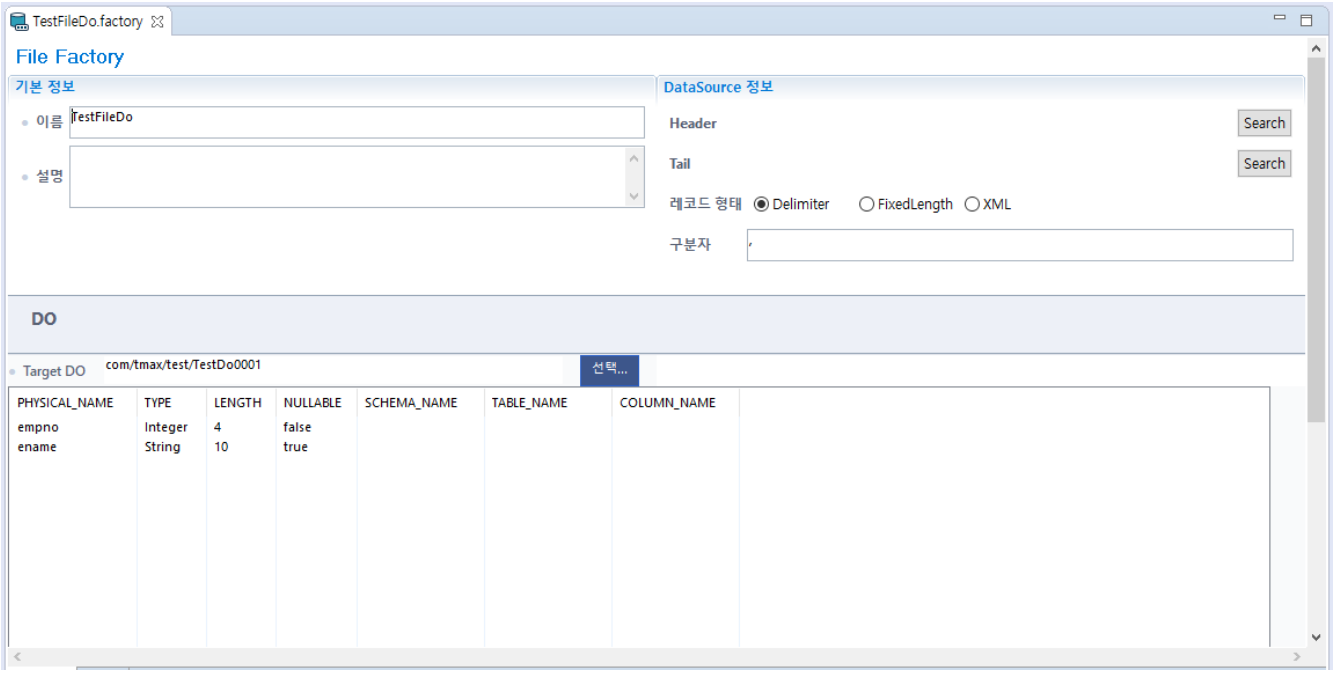



New Data Object Factory 화면

| 항목 | 설명 |
|---|---|
| 프로젝트 | 생성할 데이터 오브젝트 팩토리가 위치할 실제 프로젝트명을 선택한다. |
| 패키지 | 실제 사용하는 패키지 네이밍 규칙에 따라 패키지명을 정의한다.<br><br>Fix된 Nameing Rule Check기능을 제공하며, 자세한 내용은 Class Naming Rule Check를 참고한다. |
| 이름 | 데이터 오브젝트 팩토리 객체의 논리명을 입력한다. |
| 클래스 | 데이터 오브젝트 팩토리의 실제 생성되는 클래스 파일명을 입력한다. |
| Data Source | 데이터소스 유형을 선택한다.<br><br>◦ DB : SQL을 처리할 때 선택한다.<br>◦ File : File을 읽고 쓸 수 있을 때 선택한다. |

## 3.4.2. 데이터 오브젝트 팩토리 에디터

**File 데이터 오브젝트 팩토리 에디터**는 File을 Read/Write할 대상의 Header(DOF 유형), Body(DO 유형), Tail(DOF 유형) 및 Message Type 등을 설정하여 SO/BO에서 File IO를 처리할 수 있다.



File 데이터 오브젝트 팩토리 에디터

- **기본 사항**

| 항목명 | 설명 |
|---|---|
| 이름 | FileDO Class 이름이다. |
| 설명 | 해당 DOF 논리명 및 해당 리소스 설명이다. |
| Header | File의 Head, Body, Tail 중 Header를 나타낸다. Header에는 DOF를 설정한다. |
| Tail | File의 Head, Body, Tail 중 Tail을 나타낸다. Tail에는 DOF를 설정한다. |
| 레코드 형태 | Marshal/Unmarshal할 대상이 되는 Message Type을 나타낸다. (Delimiter, XML, FixedLength Type) |
| 구분자 | **'레코드 형태'** 항목이 'Delimiter'인 경우 Message 내의 구분자이고, **'레코드 형태'** 항목이 'FixedLength'인 경우 'Binary Type 여부'이다. |

- **Target DO**

  File의 Head, Body, Tail 중 Body에 해당하는 DO 이름을 표시하며 목록에 Layout이 조회된다.

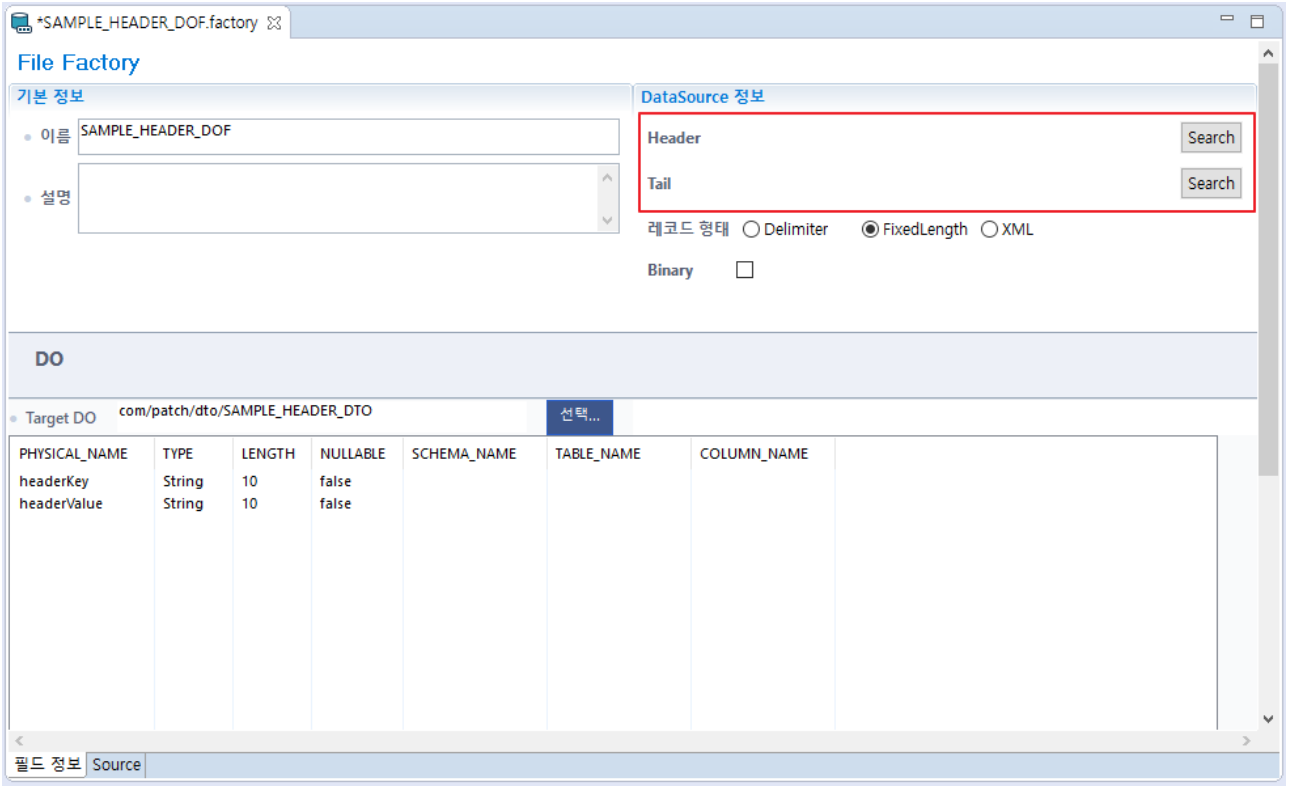
## 3.4.3. 개발 방법

본 절에서는 데이터 오브젝트 팩토리 File 유형의 생성 방법을 설명한다.

### 3.4.3.1. 데이터 오브젝트 팩토리 생성

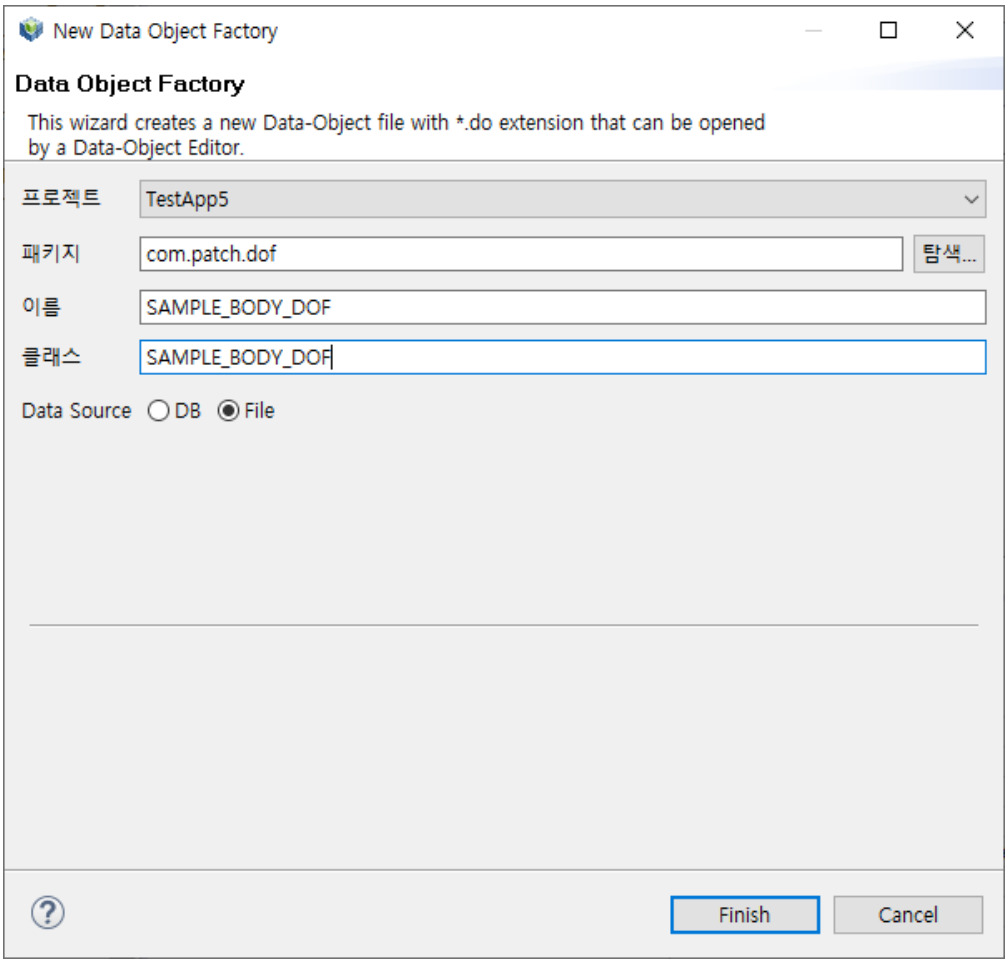
다음은 File 유형의 데이터 오브젝트 팩토리를 생성하는 과정에 대한 설명이다.

1. Header/Body를 생성한다. 이후 설명할 DOF 생성 방법과 동일 하지만 Header, Tail이 비어 놓는 부분이 차이점이다. Read/Write할 파일의 Layout이 Header, Body가 있을 경우에만 생성한다.



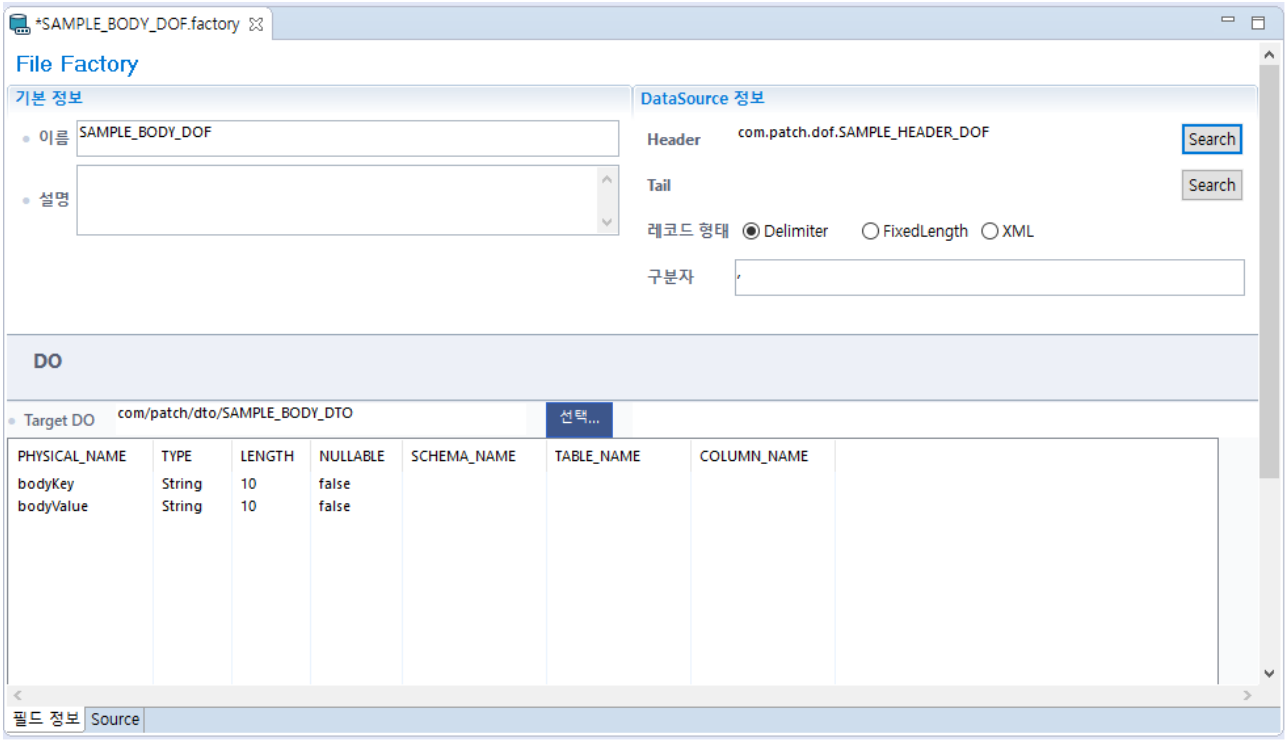

데이터 오브젝트 팩토리 에디터 - FILE (1)

2. 데이터 오브젝트 팩토리 생성을 참고해서 데이터 오브젝트 팩토리를 생성한다. File DOF 생성을 위해서는 **'Data Source'** 항목을 'File'로 선택한다.



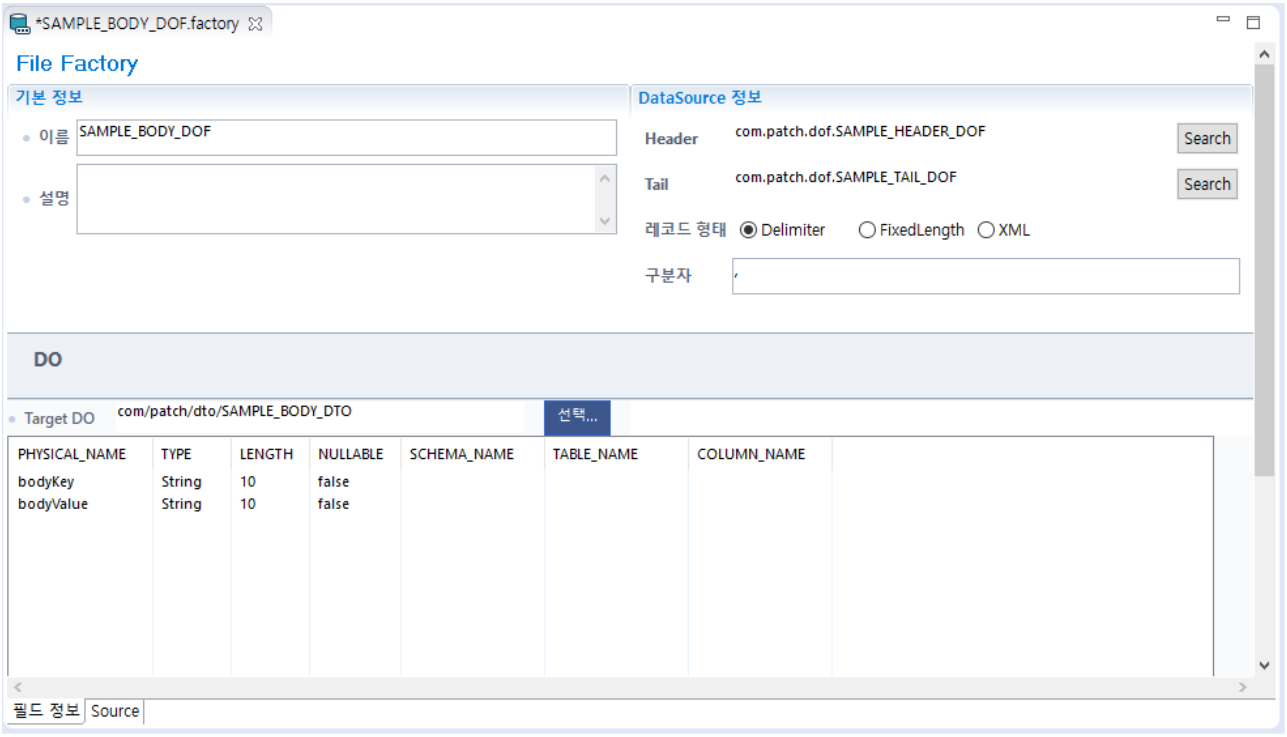

데이터 오브젝트 팩토리 에디터 - FILE (2)

3. File의 Body가 되는 Target DO를 선택한다. **[선택]** 버튼을 클릭하면 DO 내역을 조회/선택할 수 있다.




데이터 오브젝트 팩토리 에디터 - FILE (3)

4. Read/Write할 File에 Header/Tail이 있을 경우 각 항목의 **[Search]** 버튼을 클릭하여 해당 DOF를 선택한다. 선택할 Type은 DOF Type의 File Type이어야 한다.




데이터 오브젝트 팩토리 에디터 - FILE (4)

## 3.5. 쿼리 오브젝트

쿼리 오브젝트(Query Object, QO)는 데이터베이스에 접근하여 데이터를 조회하거나 조작하는데 사용하는

객체이다. DOF와의 차이는 각 SQL별 DO 매핑, 데이터소스를 DB만 적용할 수 있다. QO는 enum 형태의 Source Gen이다. 따라서 데이터 오브젝트의 멤버변수로 다른 데이터 오브젝트를 포함하는 경우는 허용하지 않으므로 주의한다.

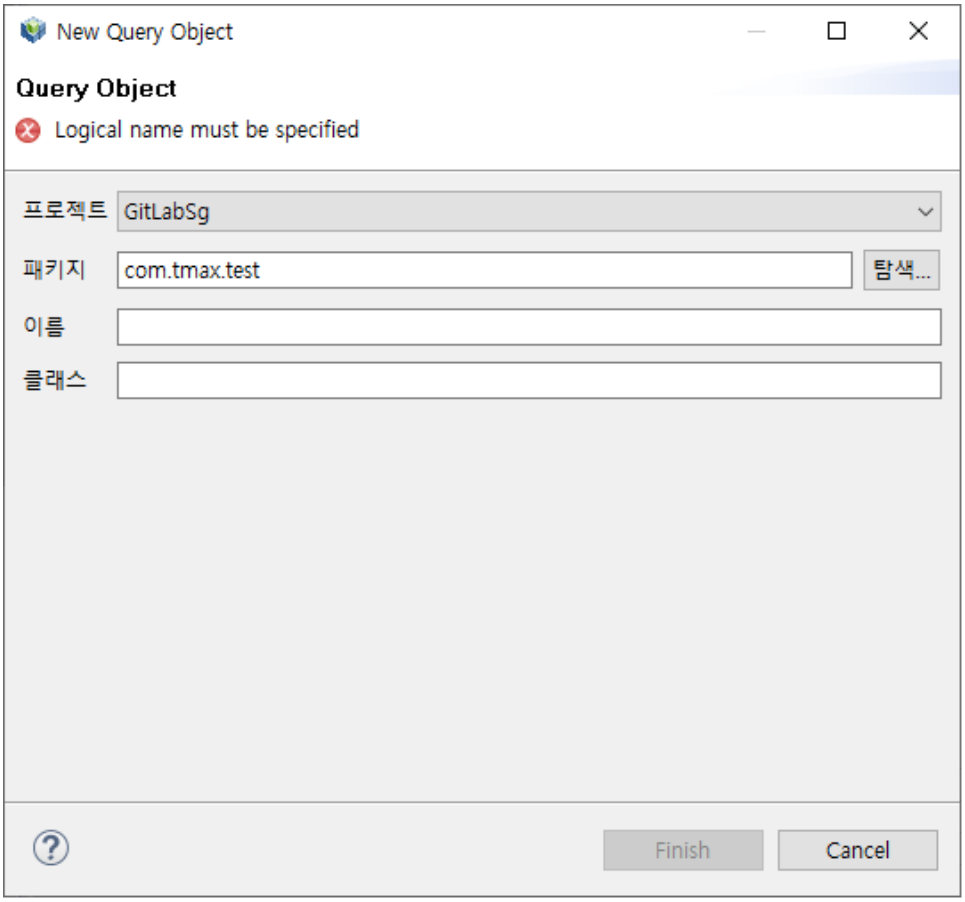
본 절은 데이터 쿼리 오브젝트 기본 개념과 시작하는 방법, 예제를 설명한다.

### 3.5.1. 환경설정

ProStudio의 환경설정에서 JDBC 설정되어 있어야 한다. 자세한 내용은 사용 환경 확인을 참고한다.

### 3.5.2. 쿼리 오브젝트 생성

**Package Explorer**의 컨텍스트 메뉴에서 **[New] > [QueryObject]**를 선택한 후 **New Query Object 화면**에서 쿼리 오브젝트를 생성한다.

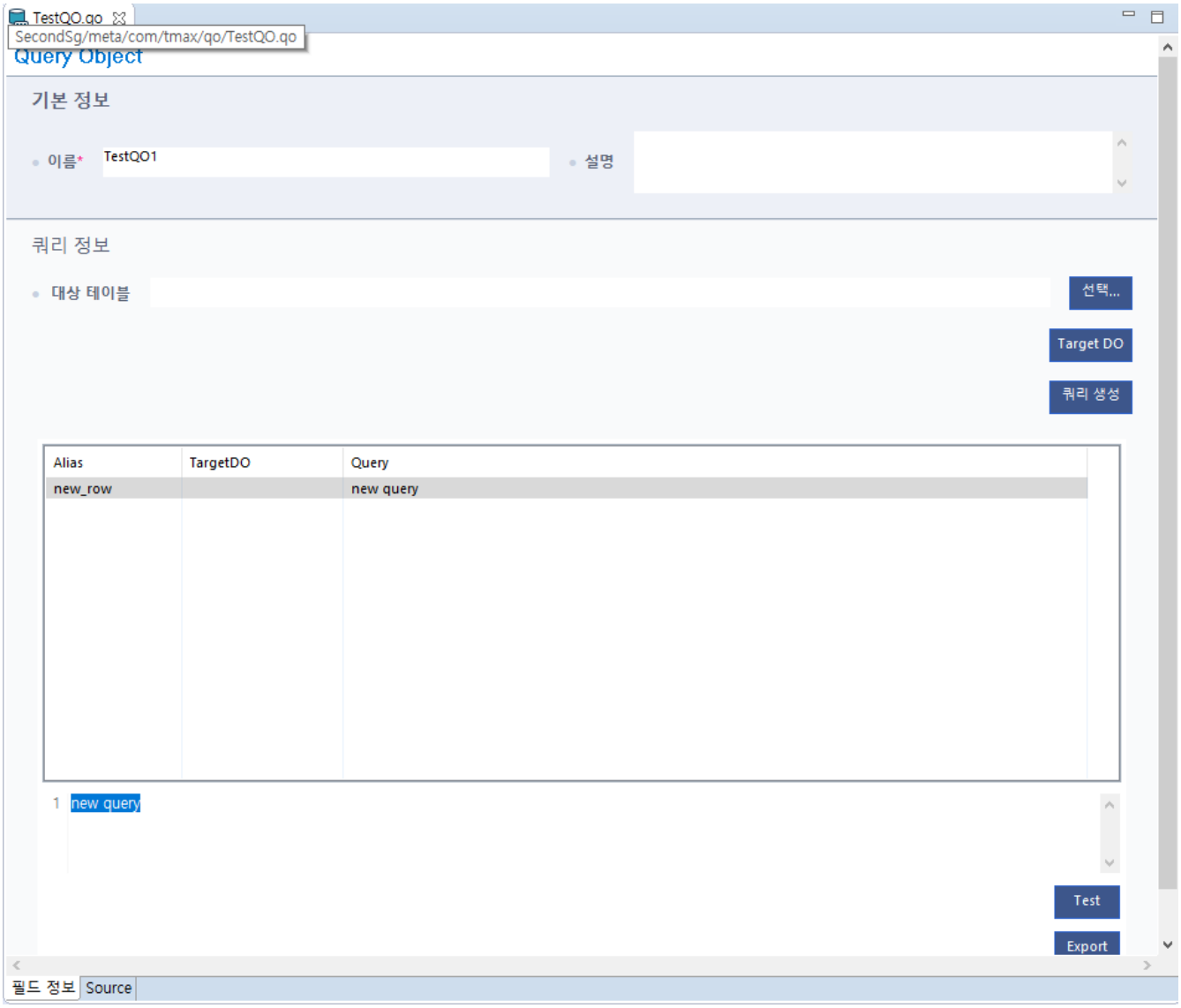



New Query Object 화면

| 항목 | 설명 |
|---|---|
| 프로젝트 | 생성할 쿼리 오브젝트가 위치할 실제 프로젝트명을 선택한다. |
| 패키지 | 실제 사용하는 패키지 네이밍 규칙에 따라 패키지명을 정의한다. Fix된 Naming Rule Check 기능을 제공하며, 자세한 내용은 Class Naming Rule Check를 참고한다. |
| 이름 | 쿼리 오브젝트 객체의 논리명을 입력한다. |
| 클래스 | 쿼리 오브젝트의 실제 생성되는 클래스 파일명을 입력한다. |

## 3.5.3. 쿼리 오브젝트 에디터

**쿼리 오브젝트 에디터**는 기본 정보, 쿼리 정보 영역으로 구성된다.




쿼리 오브젝트 에디터

- **기본 정보**

  해당 QO 논리명 및 해당 리소스의 설명을 나타낸다.

- **쿼리정보**

  쿼리 오브젝트 에디터는 각각의 쿼리에 대해 Allias, TargetDO, Query가 필수로 설정되어야 한다. 쿼리 타입은 TargetDO 지정을 각각 할 수 있다.

| 항목 | 설명 |
| --- | --- |
| Alias | QO의 Caller(SO, BO등)에서 사용할 QO의 Alias를 넣는다. |

| 항목 | 설명 |
| --- | --- |
| TargetDO | 각각의 Row(SQL)에 대해 필수로 Target DO를 지정해야 한다. Target DO는 실행되는 쿼리타입에 따라 쿼리 입력 또는 결과값을 담당한다.<br><br>다음은 쿼리 타입별 DO 매핑정보이다.<br><br>• SELECT : Target DO는 쿼리의 결과 값을 받는다.<br>• UPDATE : Target DO는 쿼리에서 SET 절에 들어가는 입력 값들을 갖는다.<br>• INSERT : Target DO는 쿼리에서 INTO 절에 들어가는 입력 값들을 갖는다.<br>• DELETE : Target DO는 사용되지 않는다. |
| Query | 등록할 쿼리를 입력한다. |

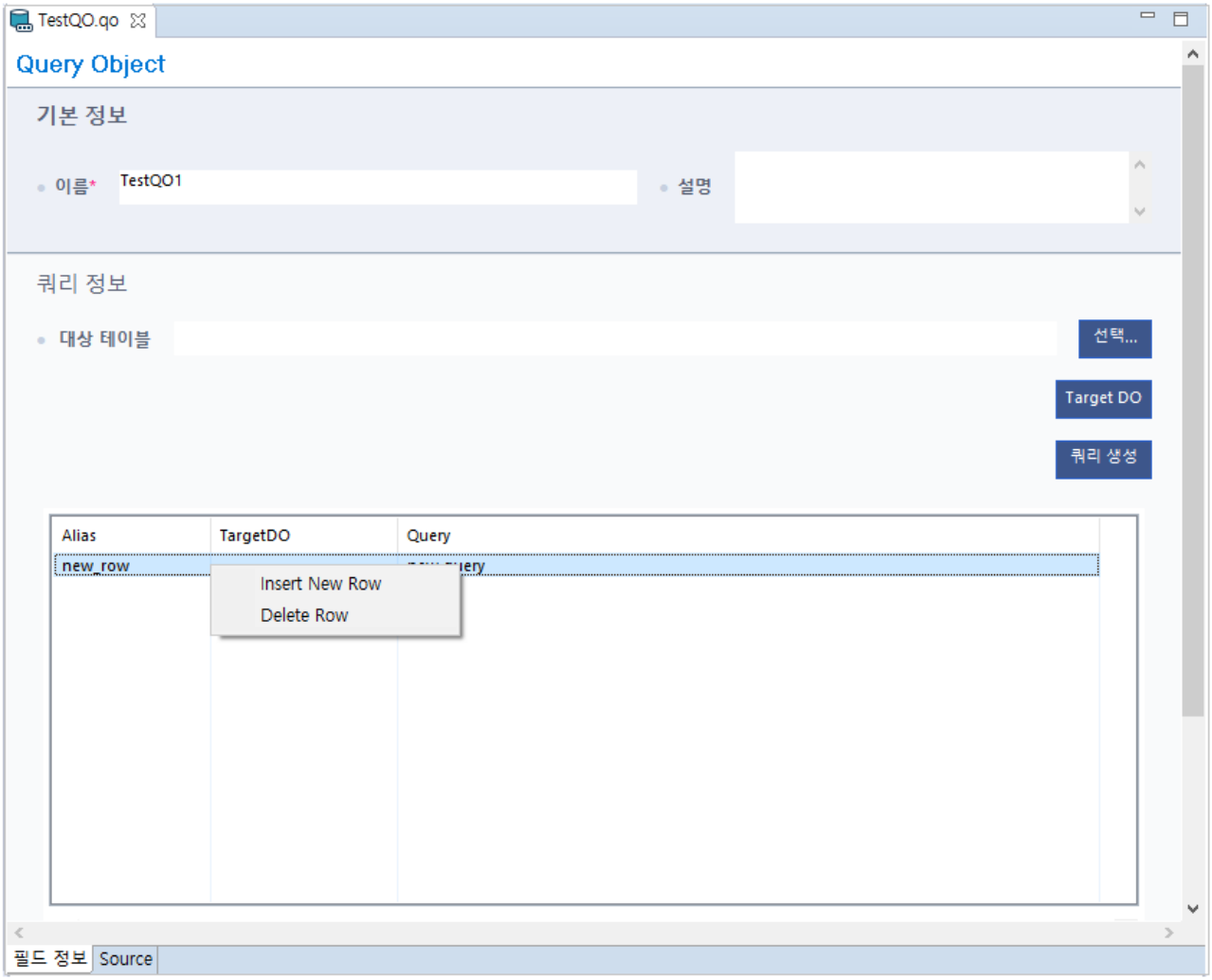
## 3.5.4. 개발 방법

본 절에서는 쿼리 오브젝트 생성 및 사용법을 설명한다.

### 3.5.4.1. 쿼리 오브젝트 생성

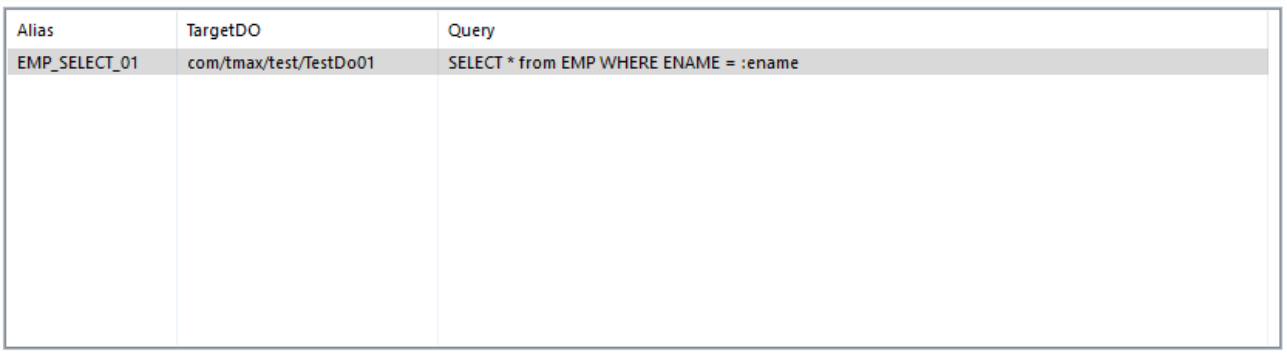
다음은 쿼리 오브젝트 에디터 객체를 생성해서 쿼리를 입력하는 과정에 대한 설명이다.

1. 쿼리 오브젝트 생성을 참고해서 쿼리 오브젝트를 생성한다.
2. 컨텍스트 메뉴에서 **[Insert New Row]**를 선택해서 Alias 및 Query를 추가한다. 사용자는 다양한 타입의 여러 개의 쿼리를 추가할 수 있다.



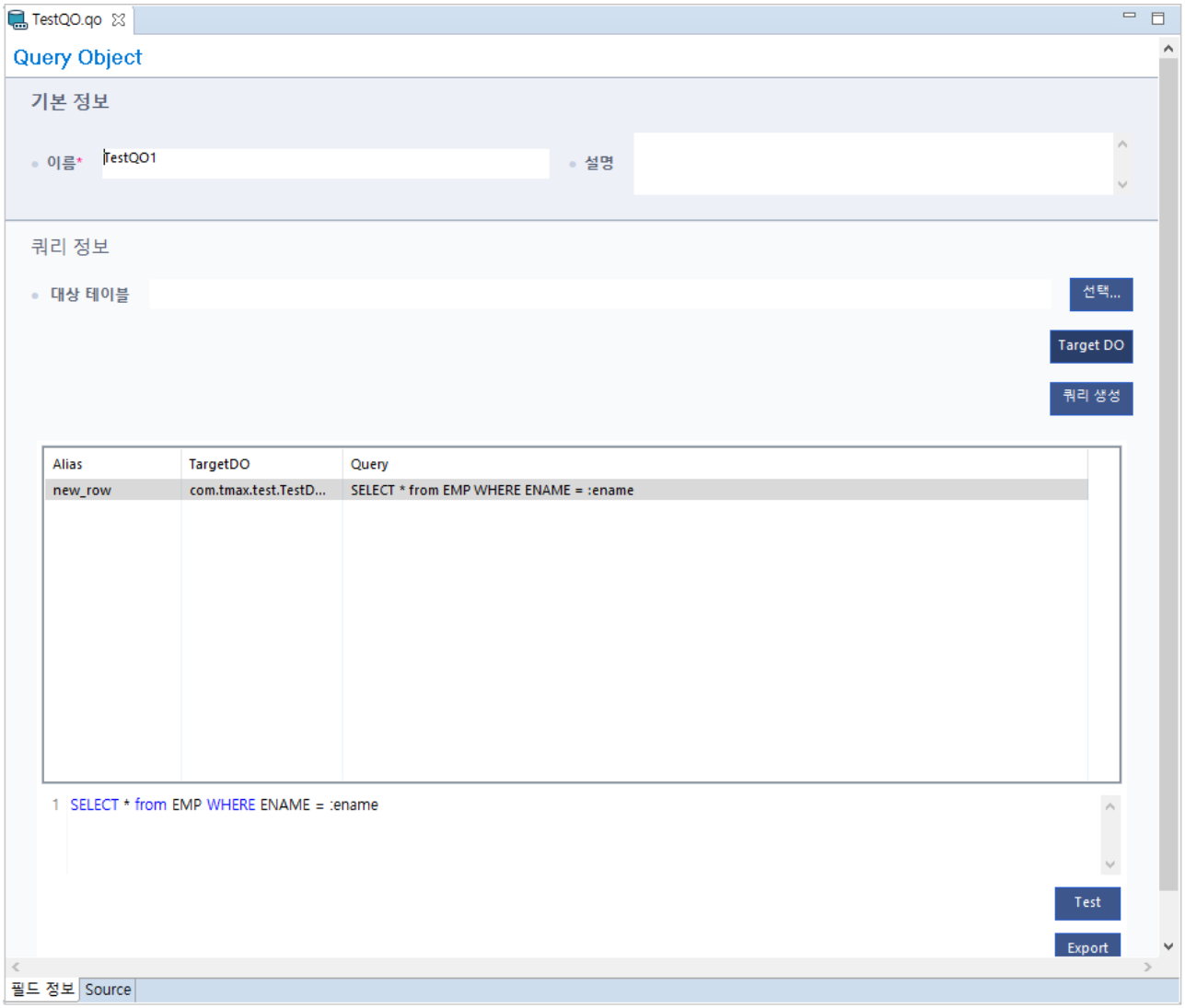

쿼리 오브젝트 생성(1)

파라미터 앞에 콜론(:)을 붙여서 표시한다(Where 절 및 set 절에 dynamic keyword가 들어갈 수 있다).




쿼리 오브젝트 생성 (2) - 쿼리 추가

3. Target DO를 지정할 Row에 Focus를 두고 **[Target DO 선택]** 버튼을 클릭해서 Target DO를 지정한다. Include 데이터 오브젝트 및 부모 데이터 오브젝트는 Target DO로 지정될 수 없다.




쿼리 오브젝트 생성 (3)

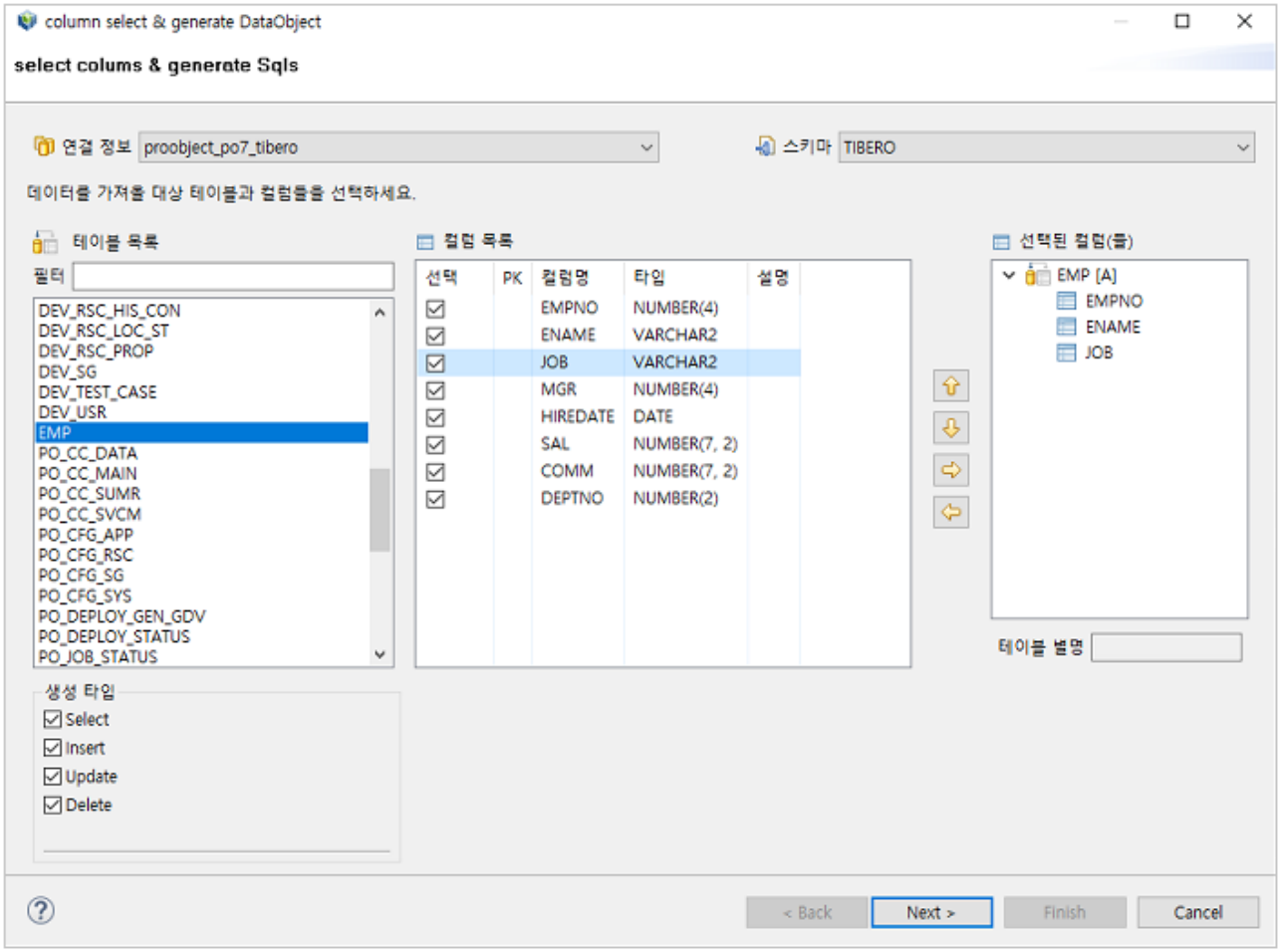
4. 쿼리 오브젝트 에디터를 저장한다.

### 3.5.4.2. 쿼리 및 데이터 오브젝트 생성

다음은 선택된 테이블 및 컬럼 정보로부터 쿼리 및 데이터 오브젝트를 생성하는 기능에 대한 설명이다.

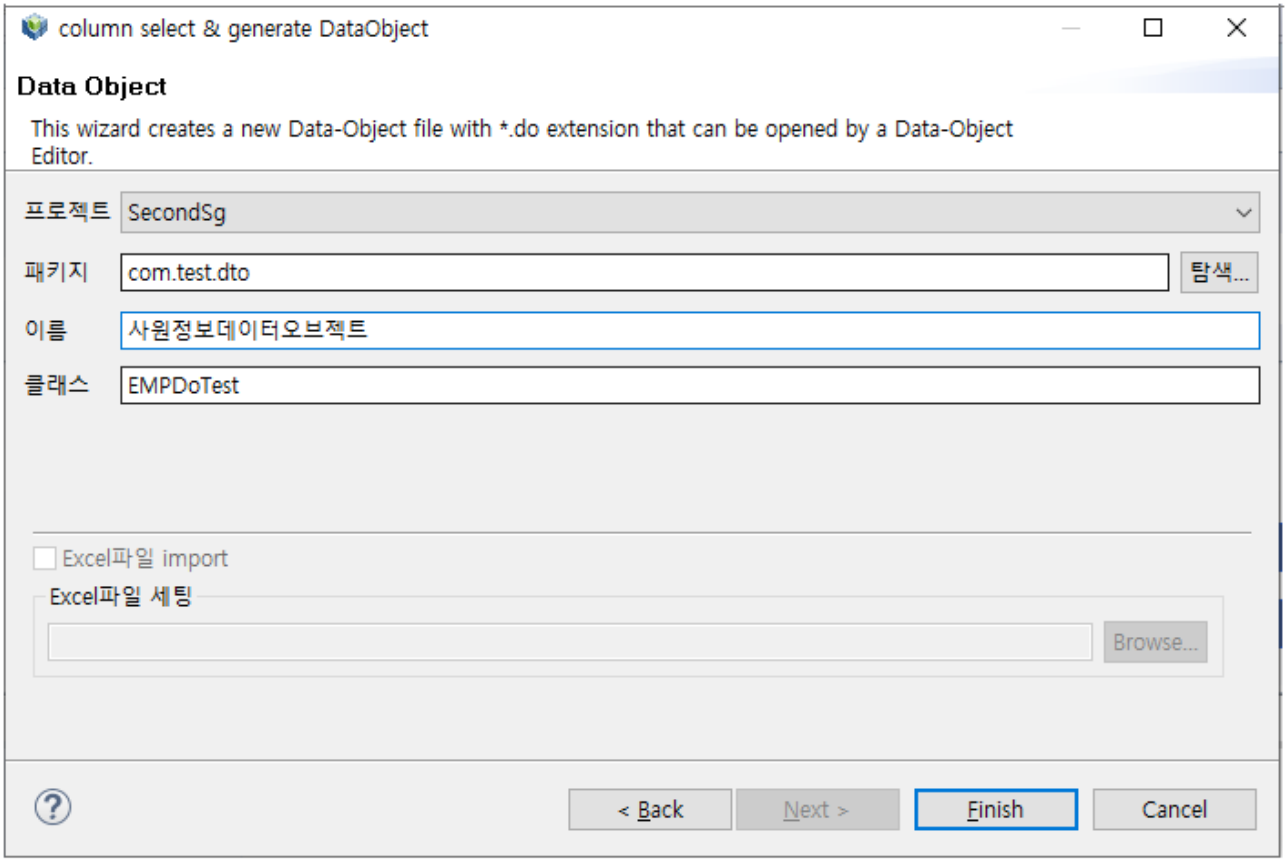
1. 쿼리 오브젝트 에디터에서 **[선택]** 버튼을 클릭하면 테이블 및 컬럼을 선택할 수 있는 다이얼로그가 나타난다.

   쿼리를 생성할 테이블을 선택하고, **생성 타입**에서 쿼리 타입을 선택한다. **컬럼 목록**에서 사용할 컬럼들을 선택한 후에 **[Next]** 버튼을 클릭한다.



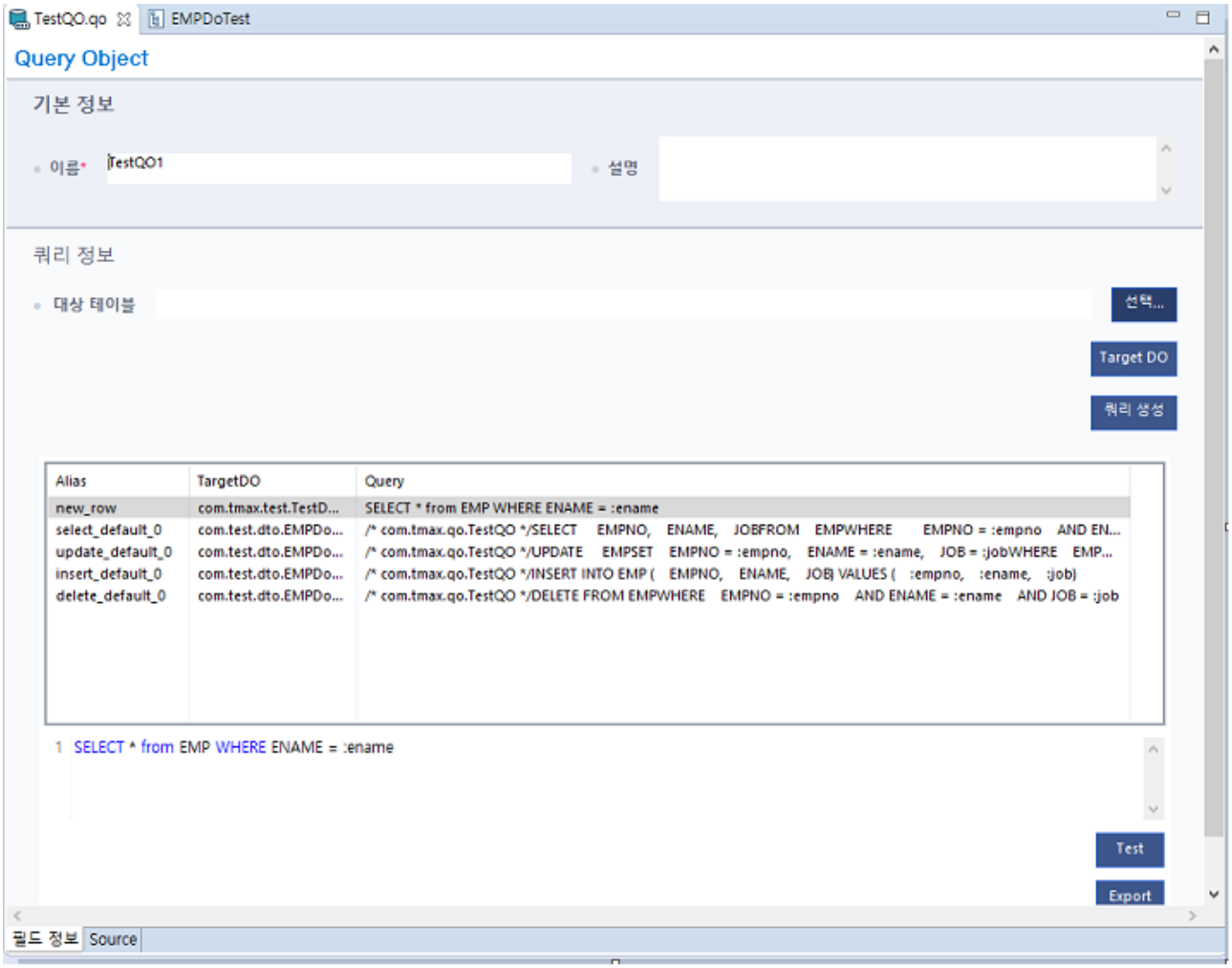
쿼리 생성 다이얼로그

2. 선택된 컬럼들을 필드로 가지는 데이터 오브젝트를 생성하기 위한 정보를 입력한다.

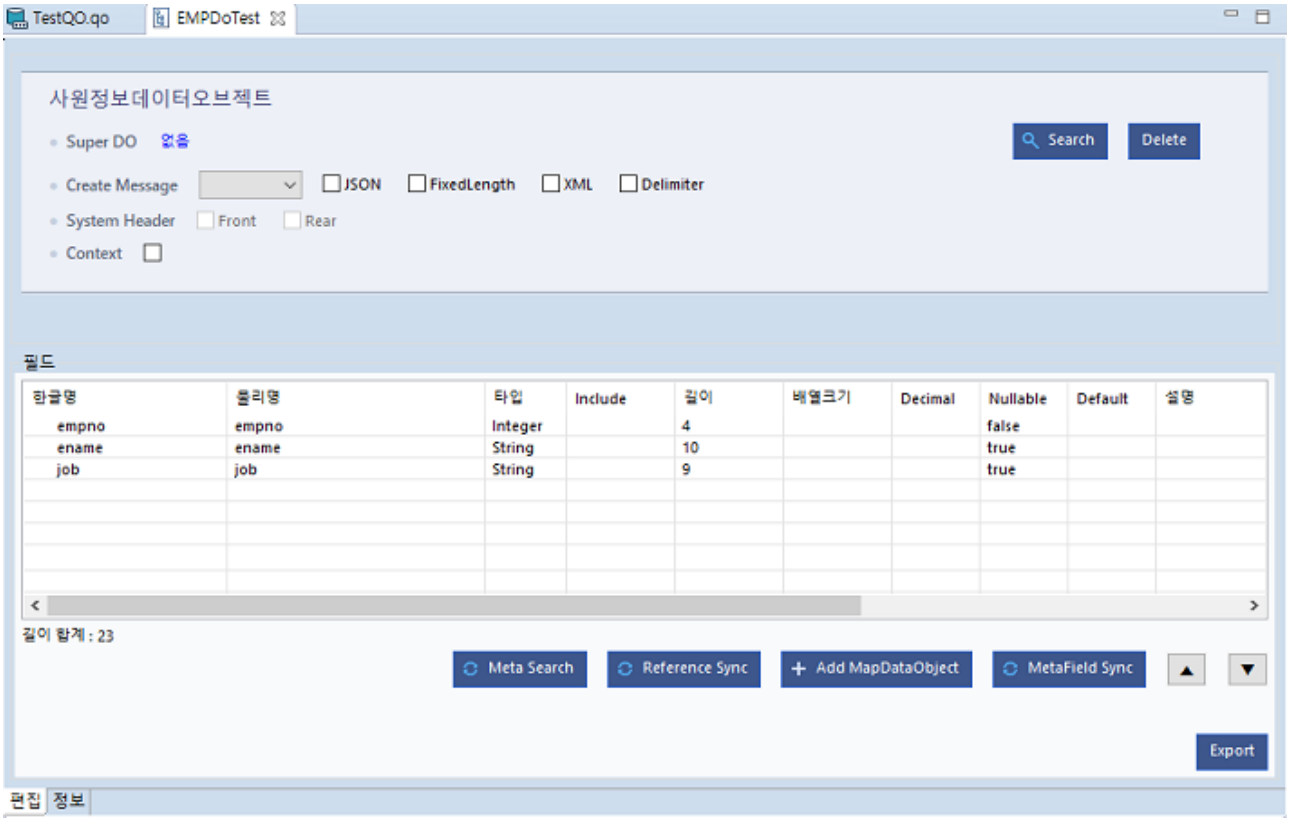



데이터 오브젝트 생성 다이얼로그

3. **[FInish]** 버튼을 클릭하면 다음과 같이 쿼리들과 데이터 오브젝트가 생성된다.




쿼리 오브젝트에 생성된 쿼리들

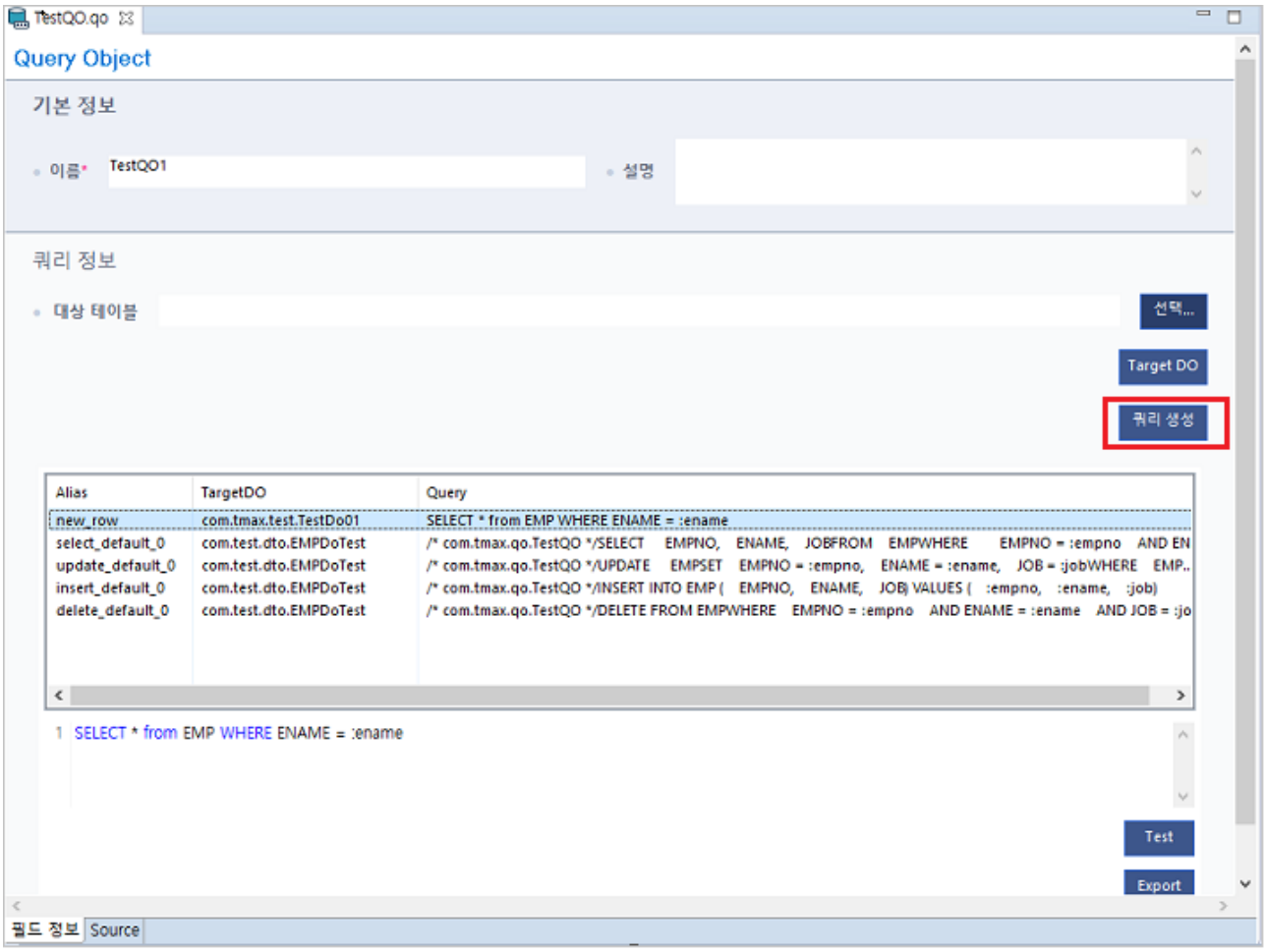



생성된 데이터 오브젝트

### 3.5.4.3. 쿼리 생성

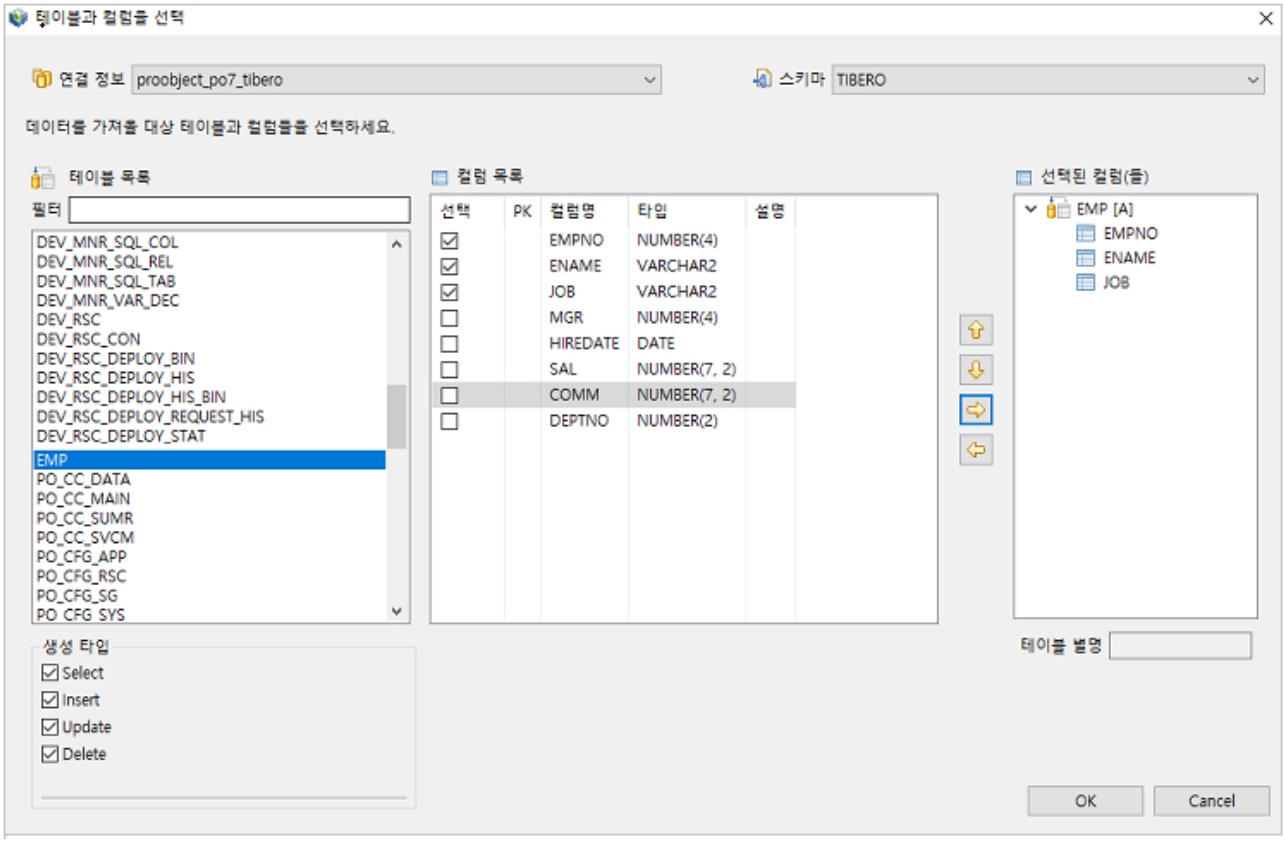
다음은 target DO가 선택된 상태에서 쿼리를 생성하는 방법을 소개한다.

1. 에디터에서 target DO가 선택된 행을 선택하고 **[쿼리 생성]** 버튼을 클릭한다.



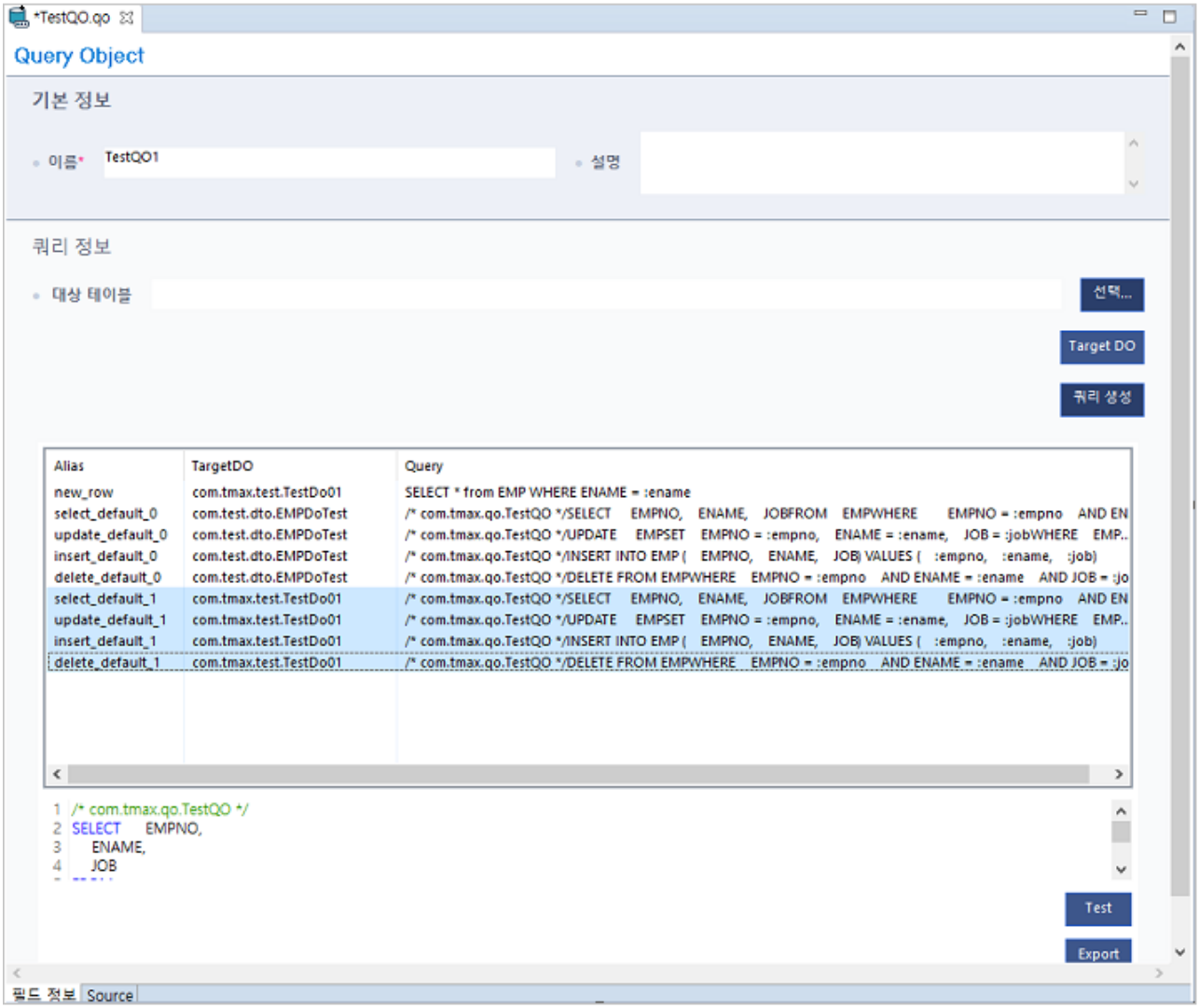
쿼리 생성 기능

2. 테이블 및 컬럼들을 선택하고 **[OK]** 버튼을 클릭한다.



쿼리 생성 다이얼로그

3. 선택된 CRUD 타입들의 쿼리들이 생성된다.



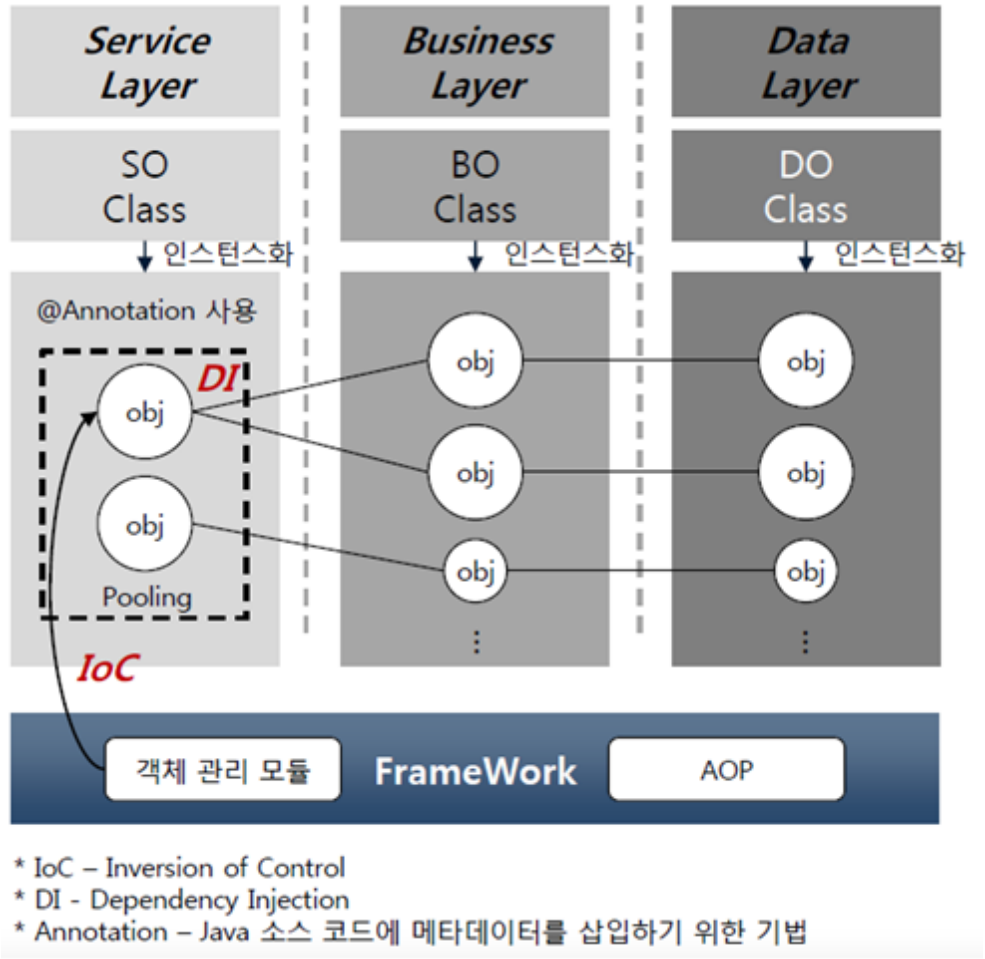

쿼리 생성 기능으로 생성된 쿼리들

# 4. 오브젝트 플로우 에디터

본 장에서는 오브젝트 플로우 에디터(Object Flow Editor)에서 플로우 디자인을 생성, 편집하는 절차와 방법에 대해 설명한다.

## 4.1. 개요

오브젝트 플로우 에디터는 각각의 오브젝트 모듈을 만들고 그 모듈들을 조합하여 새로운 오브젝트를 다양하게 만들 수 있는 오브젝트 기반 아키텍처이다. 오브젝트 플로우 에디터를 통해 서비스의 수행 흐름을 설계할 수 있으며 재사용 가능한 비즈니스 오브젝트(Biz Object, BO)의 조립을 통해 서비스 오브젝트(Service Object, SO) 및 잡 오브젝트(Job Object, JO)를 개발할 수 있다.




Layerd Object Model

업무 Layer에서 객체 생성을 지양하고 Annotation을 통해 메타 데이터를 삽입하고 프레임워크에서 객체의 생성, 운용, 소멸 등의 객체 라이프 사이클을 담당한다. Dependency Injection을 사용하여 Business Layer에서 자신의 의존적인 리소스나 Collaborator 객체에 대한 Lookup의 책임을 가지지 않게 하고 프레임워크에서 객체간의 관계를 설정하고 서비스를 수행하므로 서비스별 객체 관리가 가능하도록 되어 있다. ProObject에서는 Layer별 역할 정의가 정확하게 표현되어 있고 오브젝트 플로우 에디터에서 Layer별 오브젝트를 쉽고 편리하게 조합, 설계할 수 있다.

오브젝트 플로우 에디터를 이용하여 오브젝트를 생성할 수 있는 모듈은 다음과 같다.

- Biz Object(이하 BO)

비즈니스 기능을 수행하는 재사용성의 단위로 SO나 JO에서 레퍼런스 호출되며 정보 처리 및 계산, DO 호출 후 데이터 가공 등을 수행한다.

- Service Object(이하 SO)

  서비스를 수행 및 관리하는 오브젝트로 비즈니스 오브젝트를 Orchestration하는 플로우 중심 애플리케이션이다.

- Job Object(이하 JO)

  배치를 수행할 때 Task를 정의하고 데이터 가공할 BO의 수행 순서 및 런타임에 스레드 및 DB 세션관리를 제어할 수 있다.

**약어**

다음은 본 장에서 사용하는 약어에 대한 설명이다.

| 약어 | 설명 |
|---|---|
| DO | Data Object |
| DOF | Data Object Factory |
| DB DOF | DB DataObjectFactory |
| File DOF | File DataObjectFactory |
| JO | Job Object |
| BO | Biz Object |
| SO | Service Object |
| QO | Query Object |

# 4.2. 오브젝트 모듈 개발 절차

다음은 오브젝트 모듈을 개발하는 과정에 대한 설명이다.

1. **오브젝트 생성**

   ProStudio에서 오브젝트를 생성하기 위해 **New Object 화면**을 통해 기본 정보를 등록한다. 각 오브젝트별 생성 방법은 해당 절의 설명을 참고한다.

2. **서비스 또는 업무 플로우 다이어그램 작성**

   생성된 오브젝트의 기본 정보를 기준으로 오브젝트 플로우 에디터에서 서비스 또는 업무 플로우 다이어그램을 작성한다.

3. **플로우 내용 편집**

   오브젝트 플로우 에디터의 서비스 또는 업무 플로우 다이어그램의 프러퍼티 및 편집 화면에서 플로우 내용을 편집한다.

4. **모듈 커밋**

모듈 작성 후 커밋을 수행한다.

5. **서비스 및 Job 등록**

   모듈 커밋 완료 후 서비스 및 Job 등록을 수행한다.

6. **테스트 및 실행**

   BO 모듈은 단위테스트 수행, SO는 ProStudio 및 ProManager에서 테스트 및 실행할 수 있다.

# 4.3. Biz Object(BO)

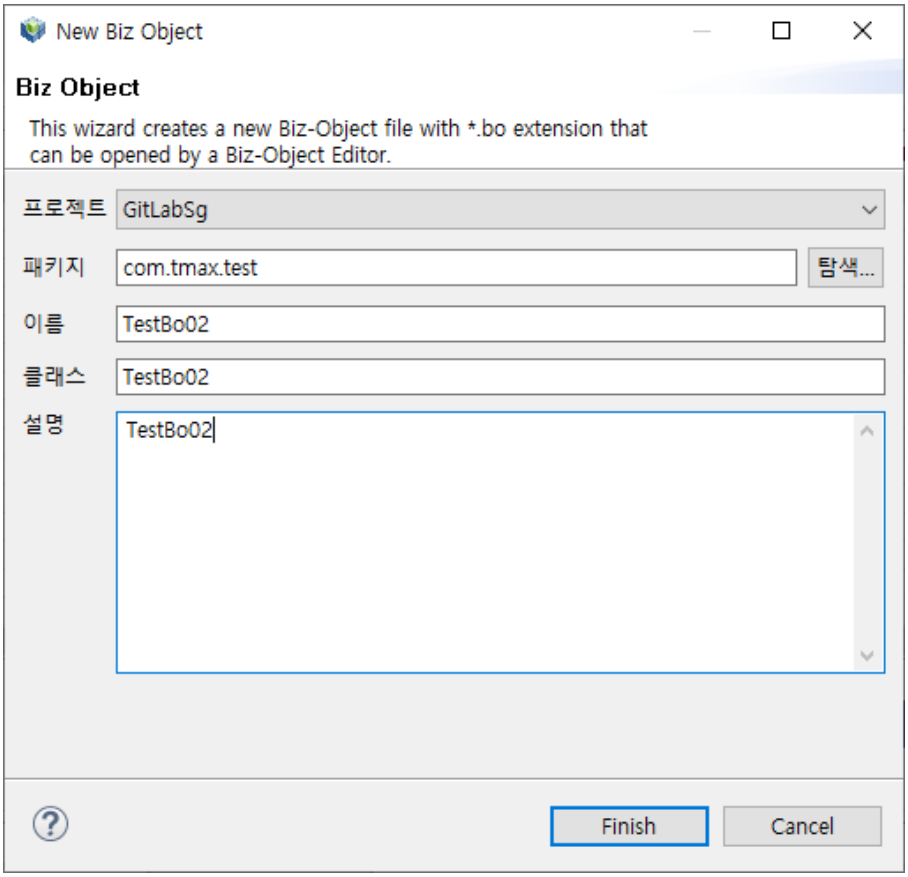
비즈니스 기능을 수행하는 재사용성의 단위로서 SO에서 레퍼런스가 호출되며, 정보 처리 및 계산 DOF를 호출한 후 데이터 가공 등을 수행한다. BO에는 Pojo java를 지원하는 BO 및 Design을 지원하는 BO(Design)이다.

## 4.3.1. BO 생성

기본 BO의 경우는 개발자가 소스 로직 전체를 편집할 수 있다. 앞서 설명한 BO(Design)의 문법을 모두 작업 가능한 영역이다. 관리적인 측면에서는 가독성을 위한 EMB를 지원하지 않기 때문에 제한적인 부분에서만 사용하는것을 권장한다.

다음은 BO를 생성하는 과정에 대한 설명이다.

1. **Package Explorer**의 컨텍스트 메뉴에서 **[New] > [BizObject]**를 선택한다.
2. **New Biz Object 화면**에서 BO를 생성하기 위해 기본 정보를 설정하고 **[Finish]** 버튼을 클릭한다.



New Biz Object 화면

| 항목 | 설명 |
|---|---|
| 프로젝트 | 생성할 BO가 위치할 실제 프로젝트명을 입력한다. |
| 패키지 | 실제 사용하는 패키지 네이밍 규칙에 따라 패키지명을 정의한다. |
| 이름 | 생성할 BO의 논리명을 입력한다. |
| 클래스 | 생성할 BO의 물리명을 입력한다. 메타에 등록되는 이름으로 동일 패키지 내에서 유일해야 한다. |
| 설명 | 생성할 BO에 대한 부가 설명을 입력한다. |

## 4.3.2. BO 생성(Design)

다음은 BO를 생성하는 과정에 대한 설명이다.

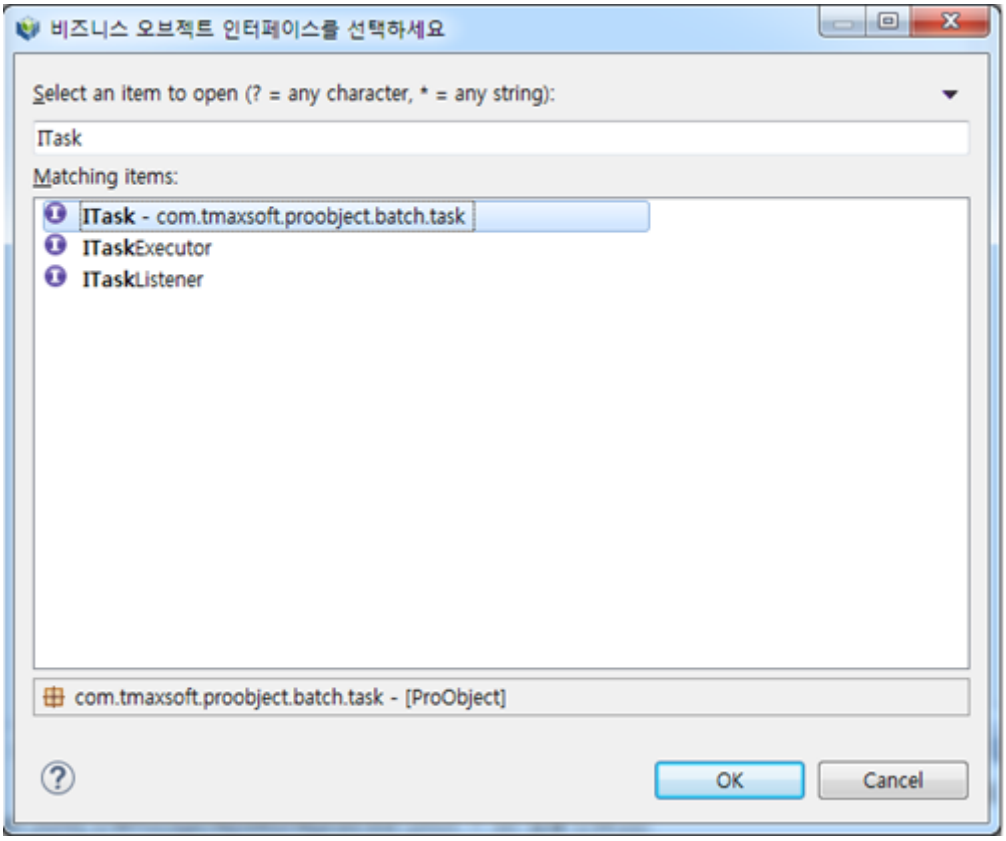
1. **Package Explorer**의 컨텍스트 메뉴에서 **[New] > [BizObject(Designer)]**를 선택한다.
2. **New Biz Object 화면**에서 BO를 생성하기 위해 기본 정보를 설정하고 **[Finish]** 버튼을 클릭한다.


New Biz Object 화면

| 항목 | 설명 |
|---|---|
| 프로젝트 | 생성할 BO가 위치할 실제 프로젝트명을 입력한다. |
| 패키지명 | 실제 사용하는 패키지 네이밍 규칙에 따라 패키지명을 정의한다.<br><br>리소스를 생성하는 경우 Fix된 Nameing Rule Check 기능을 제공하며, 자세한 내용은 Class Naming Rule Check를 참고한다. |
| 클래스명 | 생성할 BO의 물리명을 입력한다. 메타에 등록되는 이름으로 동일 패키지 내에서 유일해야 한다. |
| 상위 클래스 | 상속 받을 오브젝트를 선택할 수 있다.<br><br>**[탐색]** 버튼을 클릭해서 상위 클래스를 등록할 수 있다. **[삭제]** 버튼을 클릭하면 상위 클래스 등록이 삭제된다. |
| 인터페이스 클래스 | 구현할 인터페이스 오브젝트를 선택할 수 있다.<br><br>**[추가]** 버튼을 클릭하면 인터페이스 클래스 등록 화면이 나타난다.<br><br>추가된 인터페이스를 지정한 후 인터페이스 **[삭제]** 버튼을 클릭할 때 해당 인터페이스는 적용되지 않는다. |
| 한글명 | 생성할 BO의 논리명을 입력한다. |

| 항목 | 설명 |
| --- | --- |
| 설명 | 생성할 BO에 대한 부가 설명을 입력한다. |

**'인터페이스 클래스'** 항목에서 **[추가]** 버튼을 클릭하면 **인터페이스 클래스 등록**이 나타난다. 검색할 인터페이스명을 입력하면 검색 내용이 조회된다. 적용할 인터페이스를 선택한 후 **[OK]** 버튼을 클릭한다.



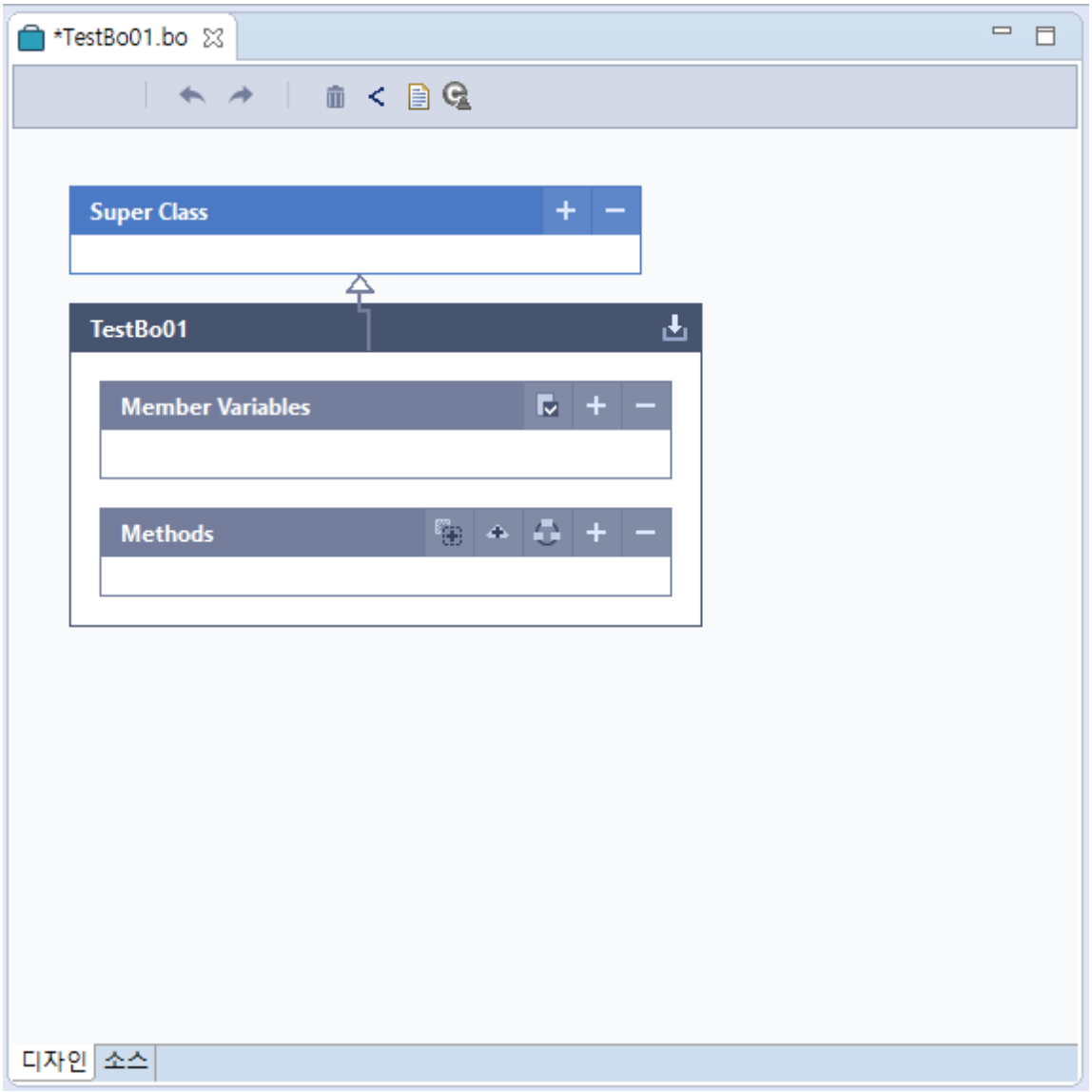
인터페이스 클래스 등록 화면

3. BO 생성이 완료되면 디자인 화면(BO 디자인 에디터)으로 이동한다.

## 4.3.3. Design Editor

기본 디자인 에디터는 기본 정보를 등록할 때 입력했던 물리명으로 클래스를 생성하며, 논리명은 최상위 박스의 이름으로 보인다.

기본 디자인 에디터는 Super Class(부모 클래스)와 해당 BO로 이루어져 있다. 에디터에서 Super Class 및 BO의 멤버 변수와 메소드를 정의할 수 있다. 생성된 메소드와 멤버 변수의 내용은 Super Class 박스, 메소드 박스, 멤버 변수 박스에서 확인할 수 있다. 클래스와 변수, 메소드를 정의하면 각 박스에 내용이 삽입된다. 자세한 내용은 EMB Designer를 참고한다.



BO 디자인 에디터

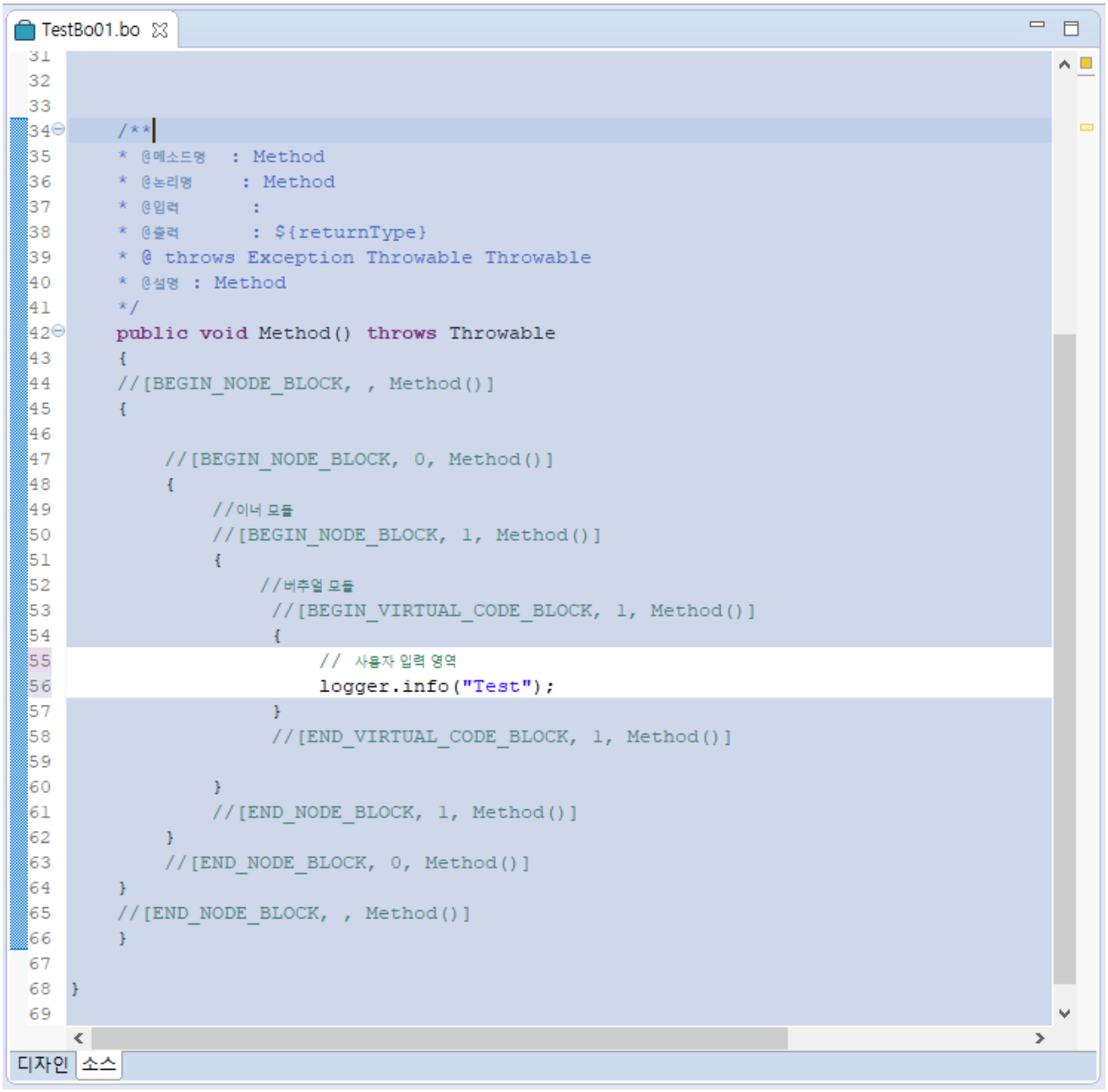
- **[디자인]** 탭

  오브젝트의 기본 정보를 등록한 후 나타나는 초기 화면으로 BO/SO의 메소드 내역과 선언된 변수 리스트를 조회할 수 있다. 기본 디자인 에디터의 메소드를 더블클릭하면 **EMB Designer**으로 이동한다.

- **[소스]** 탭

  **[소스]** 탭의 **소스 편집기**에서 적용된 모듈의 로직을 확인할 수 있으며 소스를 직접 추가, 편집, 삭제할 수 있다. 소스 편집기는 편집 가능한 영역과 자동 생성된 Source 영역(소스젠)으로 구분된다.

  다음은 소스 편집기에서 자동 생성된 "이너 모듈" 아래에 편집 가능 영역에 "버추얼 모듈"을 추가한 예제이다.



소스 편집기 화면 예

#### 4.3.3.1. 툴바

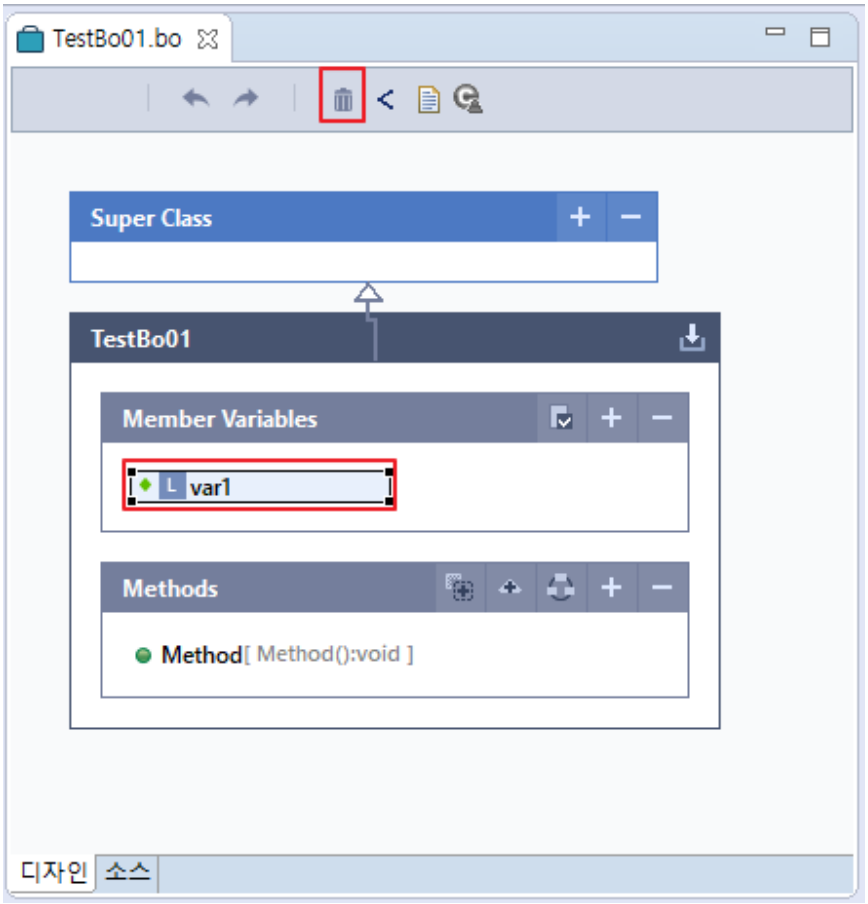
디자인 화면은 디자인 툴바를 통해 멤버 변수 선언, 메소드 정의, 삭제, 비구현 메소드 정의 등 로직을 추가 및 편집, 삭제할 수 있다.

| 아이콘 | 설명 |
|---|---|
| (화살표 이동) | 메소드 플로우 에디터에서 BO 기본 디자인 화면으로 이동한다. |
| (화살표 이동) | BO 기본 디자인 화면에서 이동하고자 하는 메소드를 선택한 후 플로우 에디터로 이동한다. |
| (삭제) | 정의한 메소드 및 변수를 삭제할 수 있다. 자세한 내용은 "삭제"를 참고한다. |
| (XOR 처리) | 이너 모듈의 XOR 처리하는 아이콘으로 플로우 Editor에서만 사용 가능하다. 자세한 내용은 "XOR 처리"를 참고한다. |

| 아이콘 | 설명 |
| --- | --- |
| (산출물 내보내기) | 사이트 커스터마이징 영역으로 산출물 템플릿을 등록하여 산출물 내보내기를 할 수 있다. 자세한 내용은 "산출물 내보내기"를 참고한다. |
| (Constructor) | 개발자가 Constructor를 정의할 때 사용한다. 자세한 내용은 "Constructor"를 참고한다. |

### (삭제)

디자인 에디터에서 메소드와 변수를 지정한 후 **[삭제]** 아이콘을 클릭하면 메소드와 변수를 삭제할 수 있다. **[-]** 버튼을 이용한 삭제는 Design Editor에서 멤버 변수 및 메소드 선언 하위에 설명되어 있다.



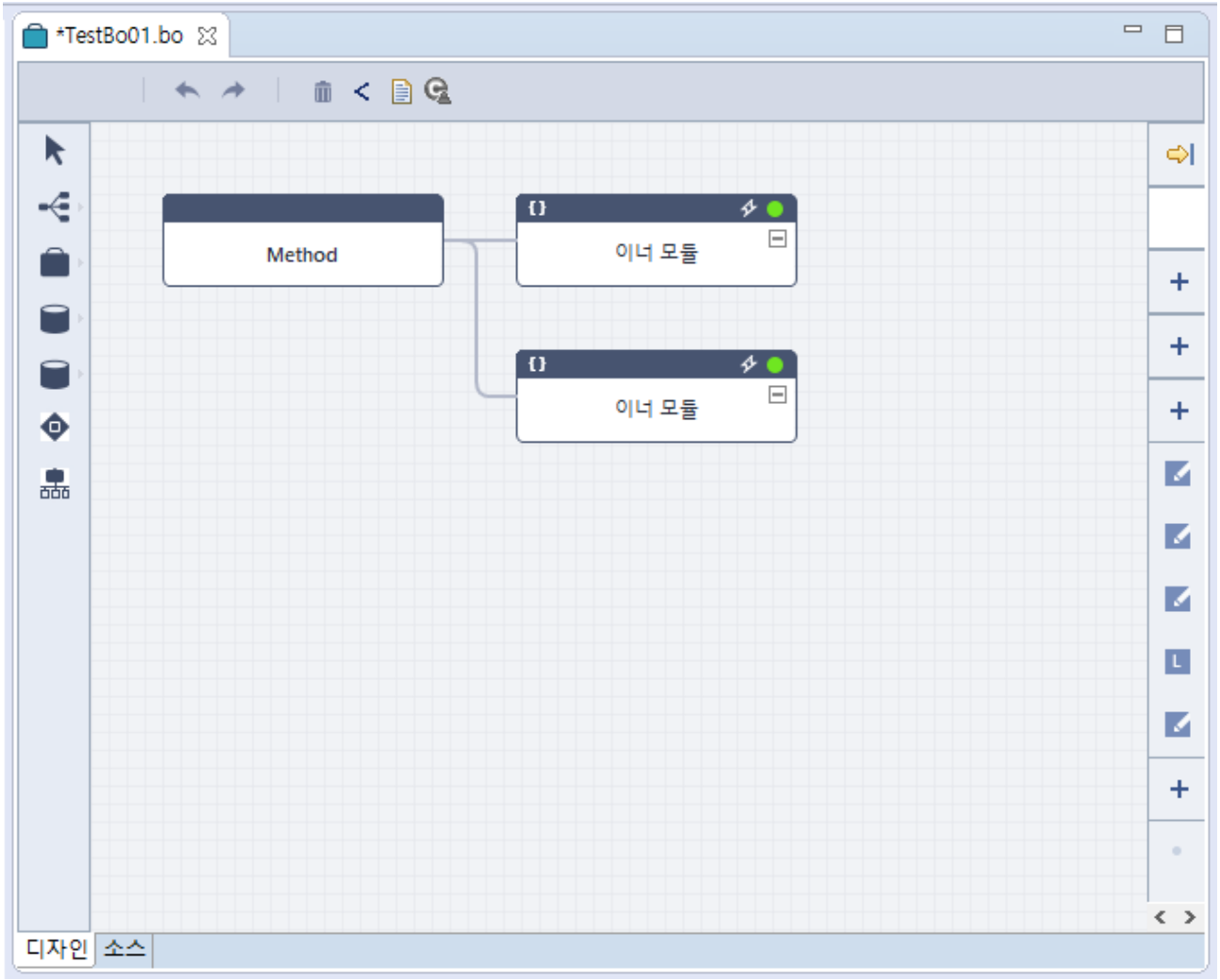
디자인 에디터 - 툴바 - 삭제

### (XOR 처리)

**[XOR 처리]** 아이콘은 EMB Designer의 플로우 에디터에서 사용된다. IF 타입의 이너 모듈을 동시에 여러 개 지정한 후 **[XOR 처리]** 아이콘을 클릭하면 디자인 에디터에서 XOR 처리된 이너 모듈을 확인할 수 있다.

- **XOR 처리를 하지 않은 경우**

  다음은 XOR 처리를 하지 않은 경우 디자인 편집화면과 소스젠 화면이다.



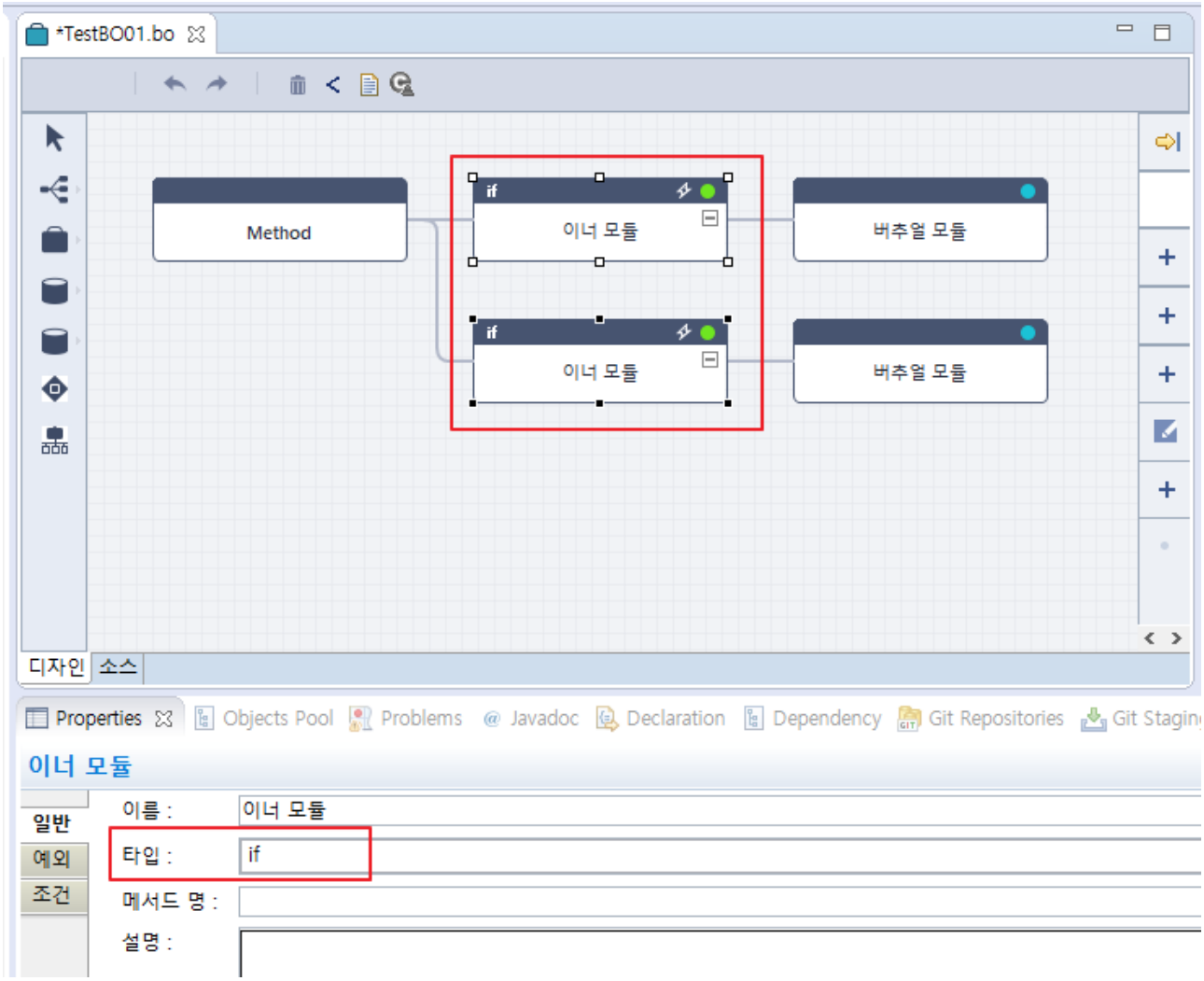

디자인 에디터 - 툴바 - XOR 미처리 - 디자인 영역

```
//[BEGIN_CONDITION_EXPRESSION, 1, selectEmpInfo(int_String)]
    if(true)
//[END_CONDITION_EXPRESSION, 1, selectEmpInfo(int_String)]
//[BEGIN_NODE_BLOCK, 1, selectEmpInfo(int_String)]
{
    //이너 모듈
}
//[END_NODE_BLOCK, 1, selectEmpInfo(int_String)]
//[BEGIN_CONDITION_EXPRESSION, 0, selectEmpInfo(int_String)]
    if(true)
//[END_CONDITION_EXPRESSION, 0, selectEmpInfo(int_String)]
//[BEGIN_NODE_BLOCK, 0, selectEmpInfo(int_String)]
{
    //이너 모듈
}
//[END_NODE_BLOCK, 0, selectEmpInfo(int_String)]
```

디자인 에디터 - 툴바 - XOR 미처리 - 소스젠

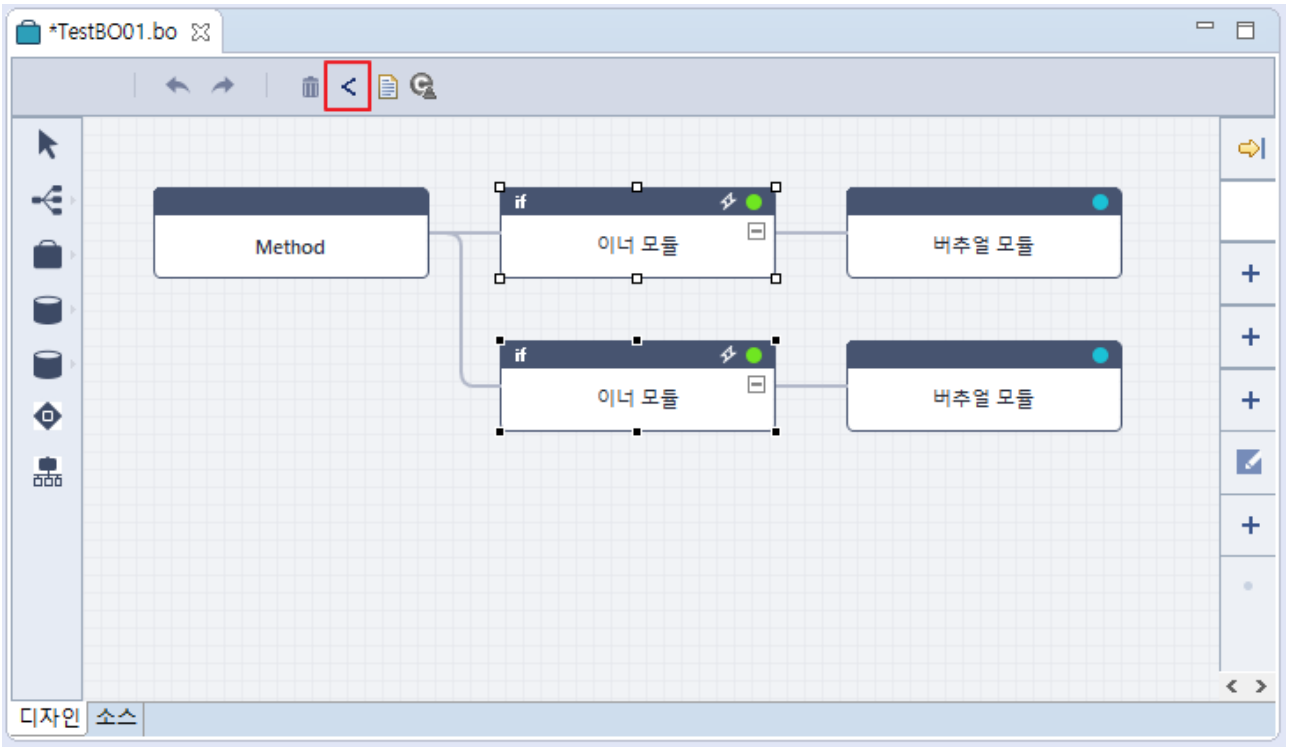
- **XOR 처리를 한 경우**

  XOR 처리를 위해서는 먼저 **'타입'**을 'if'로 변경해야 한다.

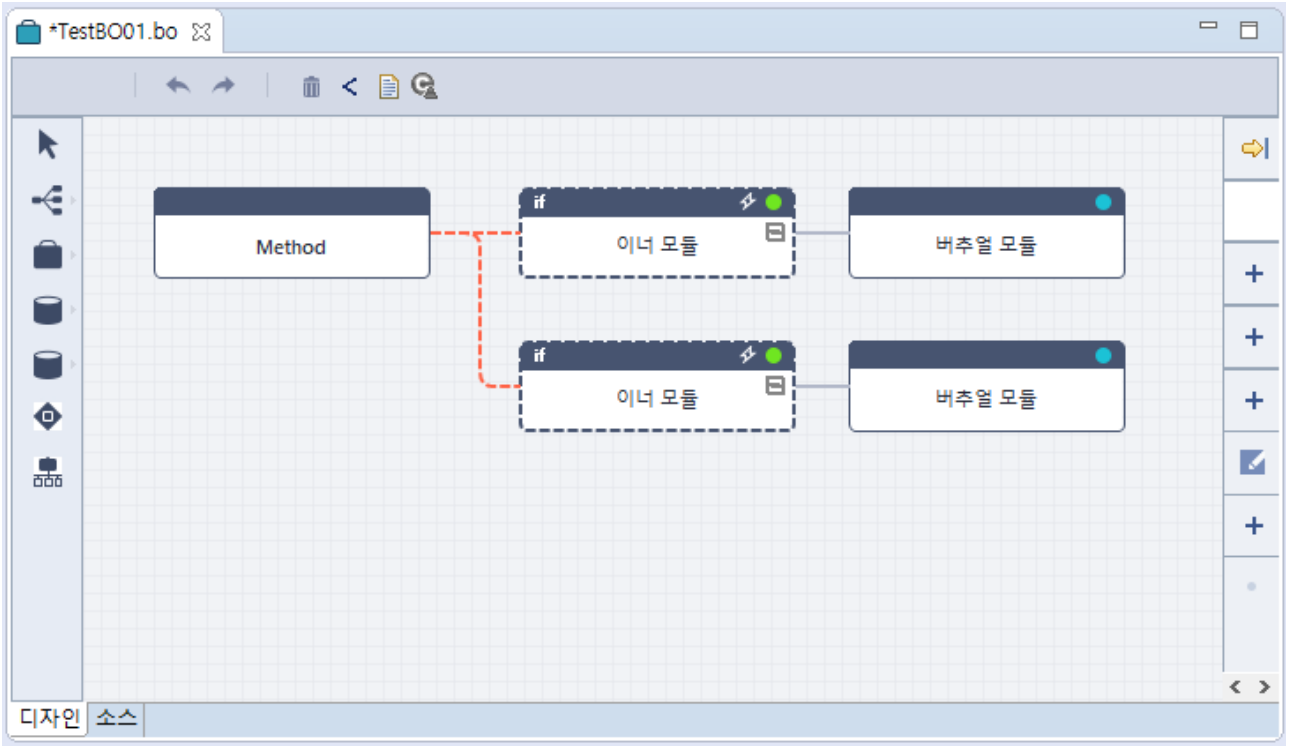



디자인 에디터 - 툴바 - XOR 처리 - 디자인 영역 (1)

다음은 XOR 처리를 하지 않은 경우 디자인 편집화면과 소스젠 화면이다.




디자인 에디터 - 툴바 - XOR 처리- 디자인 영역 (2)




디자인 에디터 - 툴바 - XOR 처리 - 디자인 영역 (3)

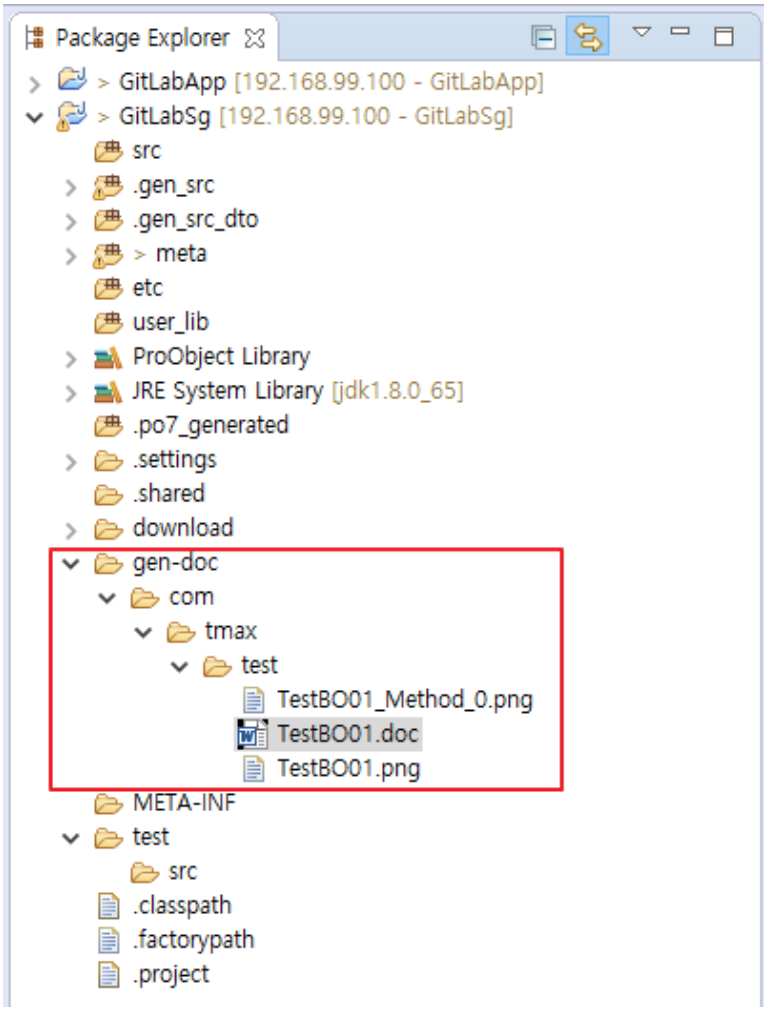
```
        if(true)
    //[END_CONDITION_EXPRESSION, 0, Method()]
    //[BEGIN_NODE_BLOCK, 0, Method()]
    {
        //이너 모듈
        //[BEGIN_NODE_BLOCK, 1, Method()]
        {
            //버추얼 모듈
            //[BEGIN_VIRTUAL_CODE_BLOCK, 1, Method()]
            {

            }
            //[END_VIRTUAL_CODE_BLOCK, 1, Method()]

        }
        //[END_NODE_BLOCK, 1, Method()]
    }
    //[END_NODE_BLOCK, 0, Method()]
    //[BEGIN_CONDITION_EXPRESSION, 2, Method()]
        else
    //[END_CONDITION_EXPRESSION, 2, Method()]
    //[BEGIN_NODE_BLOCK, 2, Method()]
    {
        //이너 모듈
        //[BEGIN_NODE_BLOCK, 3, Method()]
        {
            //버추얼 모듈
            //[BEGIN_VIRTUAL_CODE_BLOCK, 3, Method()]
            {

            }
            //[END_VIRTUAL_CODE_BLOCK, 3, Method()]

        }
        //[END_NODE_BLOCK, 3, Method()]
    }
```

디자인 에디터 - 툴바 - XOR 처리 - 소스젠

### (산출물 내보내기)

작성한 BO에 대한 정보를 **[산출물 내보내기]** 아이콘을 클릭하면 Word 문서로 Export한다. 해당 문서는 프로젝트의 gen-doc 폴더 하위에 생성된다. 산출물 내보내기 기능에 대한 자세한 내용은 산출물 내보내기를 참고한다.

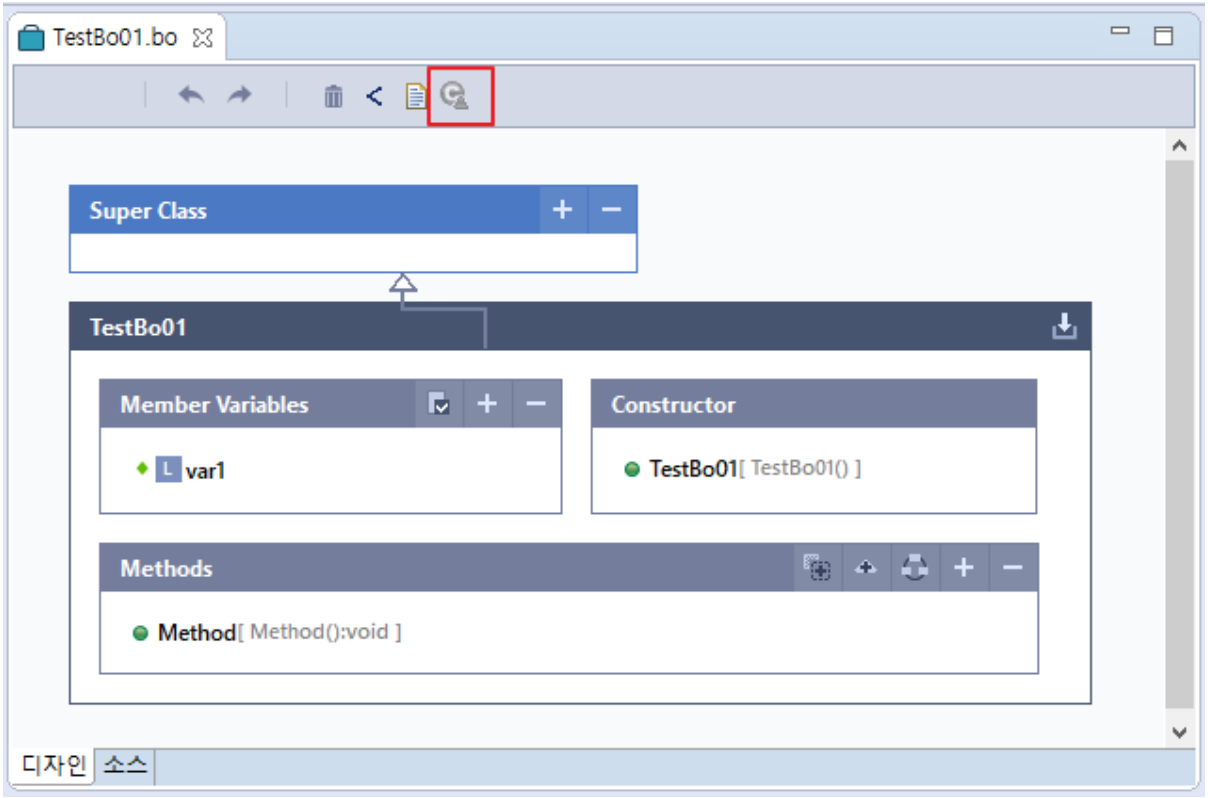
문서에 대한 내용을 추가 또는 변경하려면 확장점 플러그인 설치 및 구현해야 한다. 플러그인 설치에 대한 자세한 내용은 확정점 플러그인 설치 및 구현을 참고한다.




디자인 에디터 - 툴바 - 산출물 내보내기

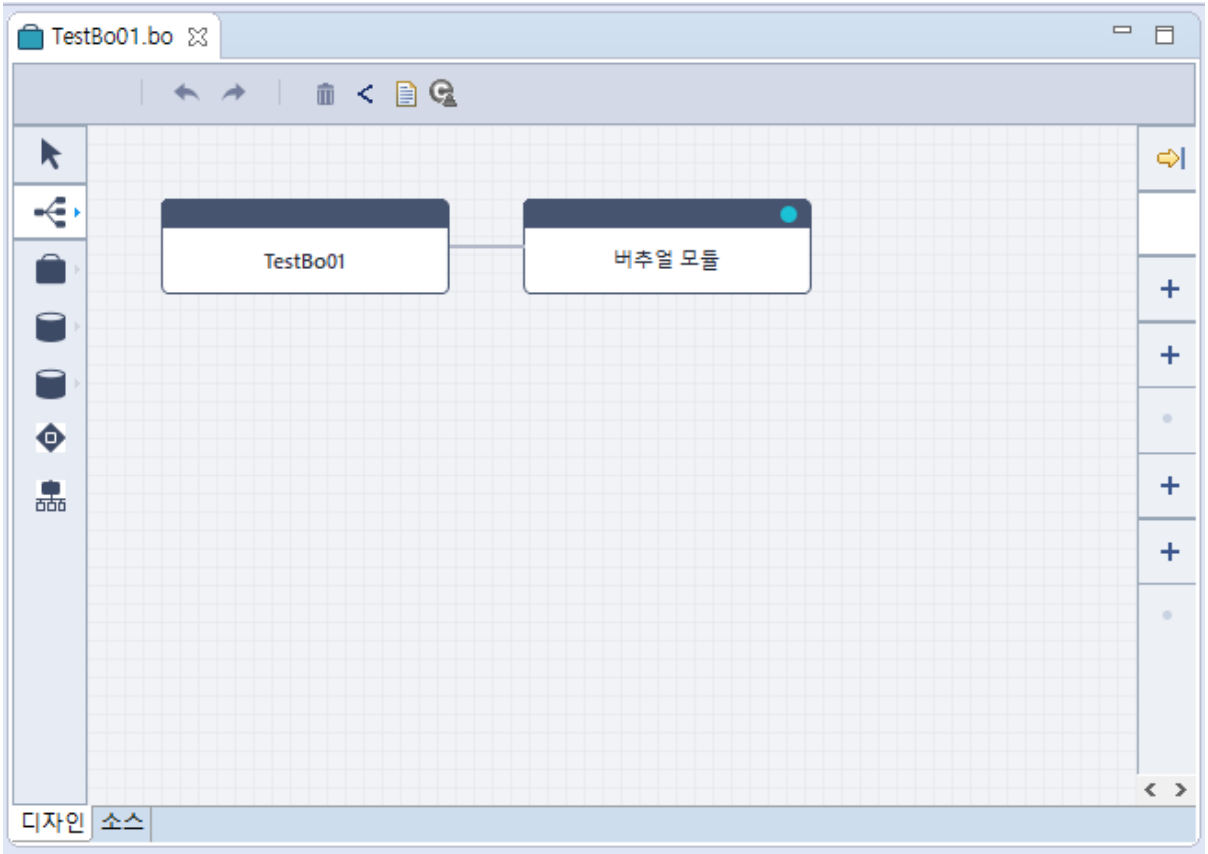
### (Constructor)

BO 개발자가 직접 Constructor를 정의하는 경우 **[Constructor]** 아이콘을 클릭해서 Constructor Method를 추가한다.




디자인 에디터 - 툴바 - Constructor - 디자인 영역 (1)

추가한 Class Constructor를 더블클릭하면 **EMB Designer 화면**에서 모듈을 추가할 수 있다. 다음은 버추얼 모듈을 추가한 화면이다.




디자인 에디터 - 툴바 - Constructor - 디자인 영역 (2)

**EMB Designer 화면**에서 추가한 버추얼 모듈을 더블클릭하면 소스 코딩을 할 수 있다.

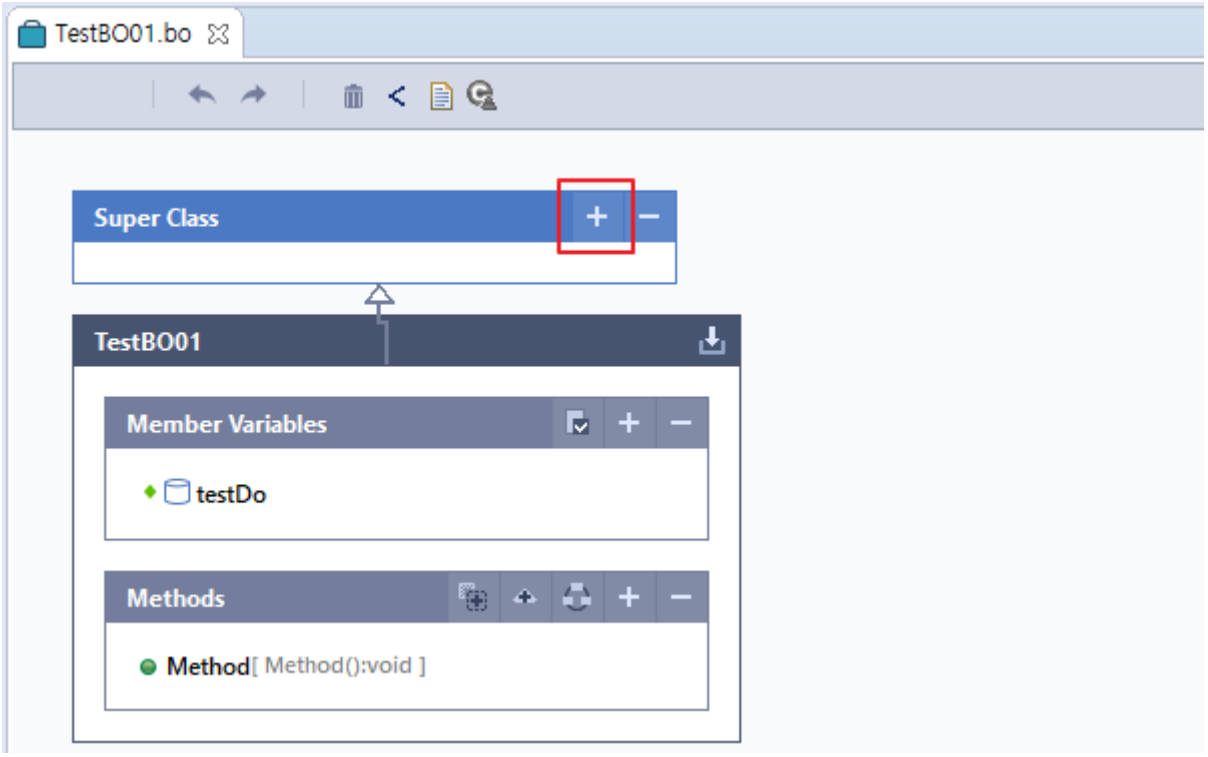

디자인 에디터 - 툴바 - Constructor - 소스젠

### 4.3.3.2. Super Class

Super Class 박스의 **[+]**, **[-]** 아이콘을 클릭하면 Super Class를 선언하거나 삭제할 수 있다.

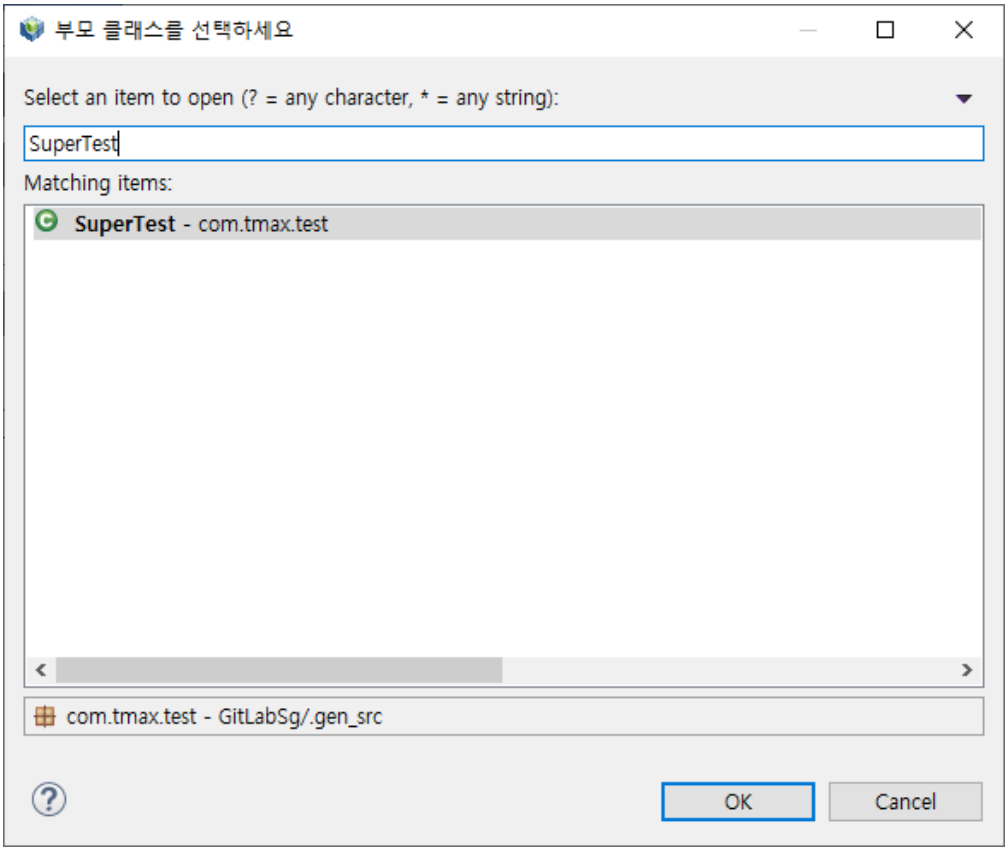
다음은 디자인 에디터에서 Super Class를 선언하는 과정에 대한 설명이다.

1. Super Class 박스에서 **[+]** 버튼을 클릭한다.



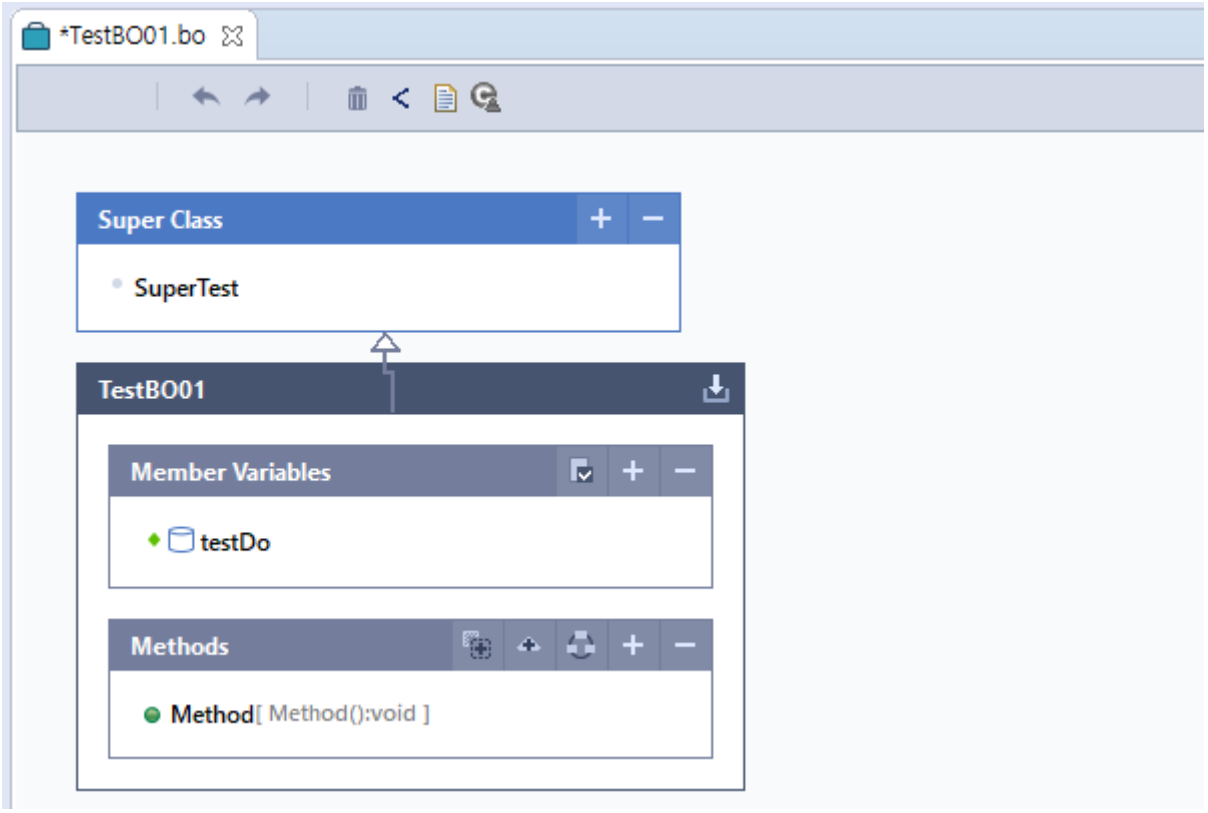

디자인 에디터 - Super Class 선언 (1)

2. Super Class를 선택하고 **[OK]** 버튼을 클릭한다.




디자인 에디터 - Super Class 선언 (2)

3. Super Class 박스에 선택한 클래스 정보가 등록된다.



디자인 에디터 - Super Class 선언 (3)

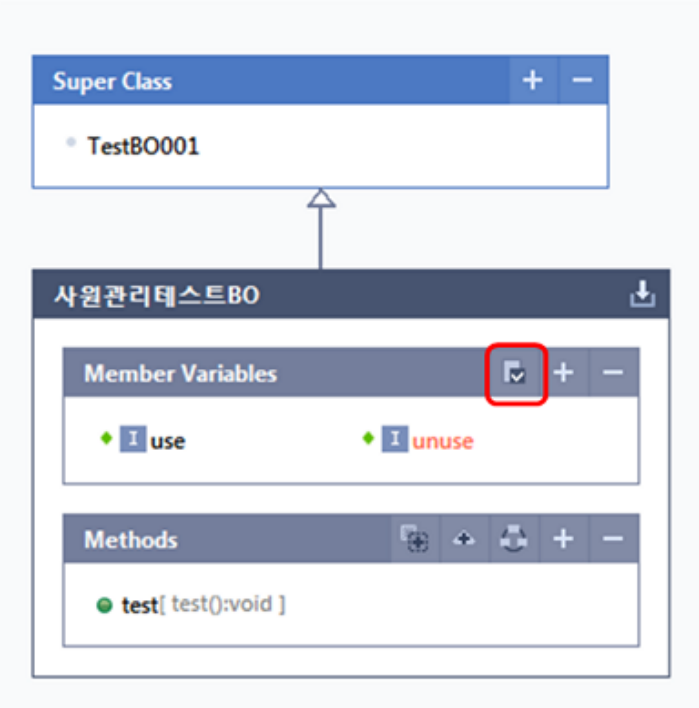
#### 4.3.3.3. 멤버 변수

멤버 변수 정의 박스에서 변수를 선언하거나 미사용 필드 분석, 변수 삭제 등의 작업을 할 수 있다.

| 아이콘 | 설명 |
|---|---|
| (미사용 필드분석) | 멤버 필드로 선언된 필드 중 BO에서 단 한번도 사용되지 않은 필드를 사용자에게 보여준다. 자세한 내용은 "미사용 필드분석"을 참고한다. |
| (변수 선언) | 멤버 변수 박스 안의 **[+]** 버튼을 클릭하면 **변수 선언 화면**(디자인 에디터 - 변수 선언 화면)이 나타나며 DO/DOF Type, Primitive Type, Object Type별로 변수 등록이 가능하다. 자세한 내용은 "변수 선언"을 참고한다. |
| (변수 삭제) | 정의한 변수를 삭제할 수 있다. 자세한 내용은 "변수 삭제"를 참고한다. |

#### (미사용 필드분석)

멤버 필드에 선언되어 있는 변수임에도 불구하고 BO 로직에서 사용되지 않은 필드들은 **[미사용 필드분석]** 아이콘을 클릭하면 찾아준다. BO에서 사용되지 않은 변수의 경우는 빨강색으로 표시된다.

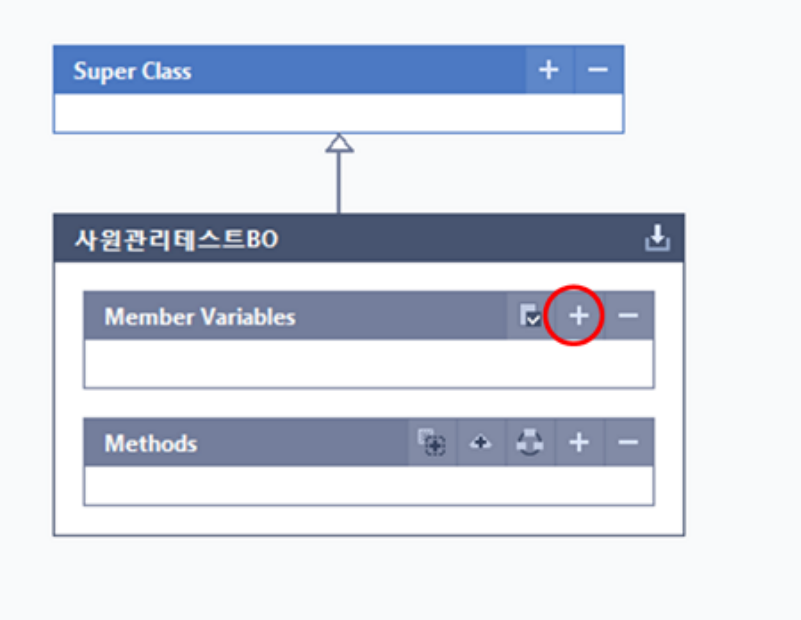



디자인 에디터 - 미사용 필드분석

### (변수 선언)

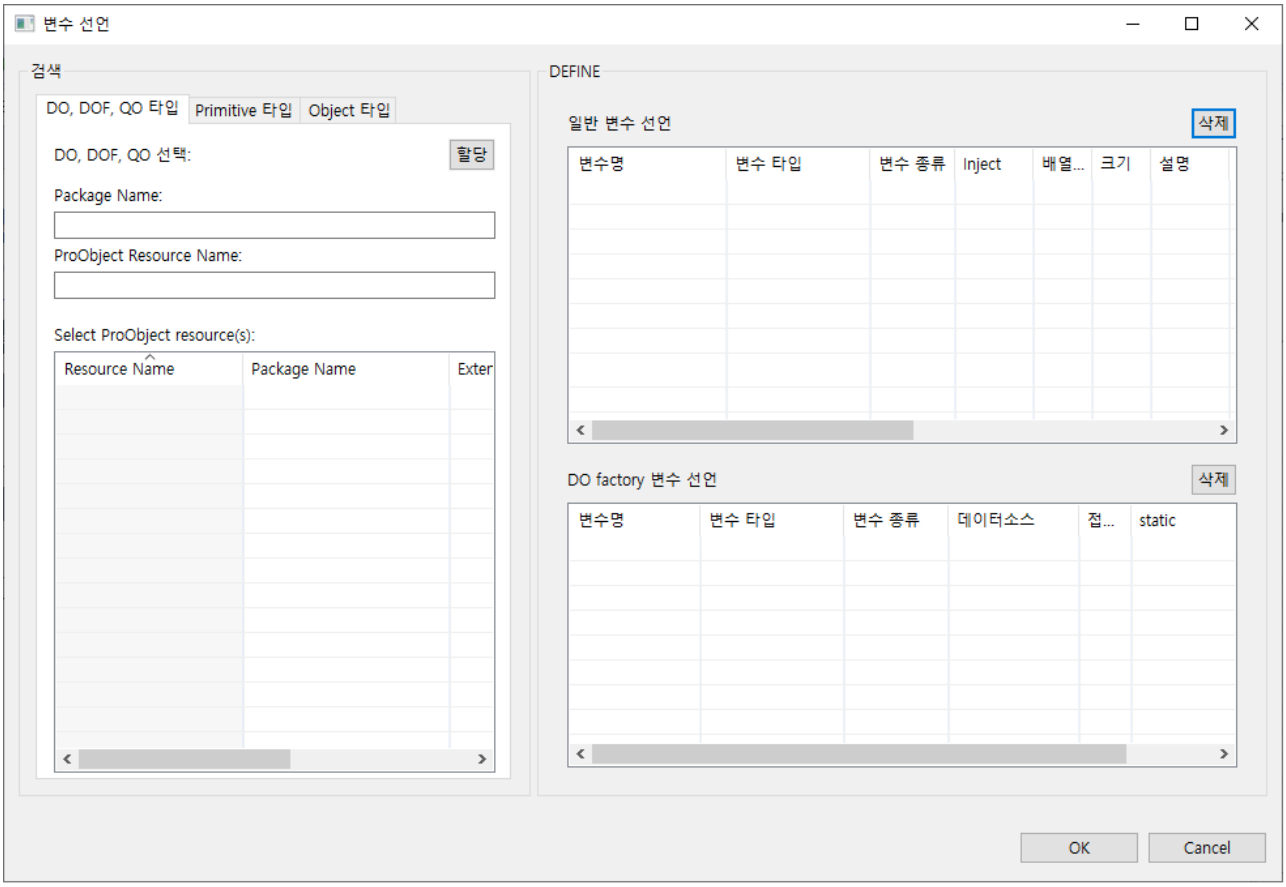
다음은 변수 선언하는 과정에 대한 설명이다.

1. 멤버 변수를 선언하기 위해 멤버 변수 박스 안의 **[변수 선언]** 아이콘을 클릭한다.




디자인 에디터 - 변수 선언 아이콘

2. 변수의 기본 정보를 등록할 수 있는 **변수 선언 화면**이 나타난다. 변수 타입은 DO, DOF, QO, Primitive, Object 5가지 타입을 지원한다.




디자인 에디터 - 변수 선언 화면

다음은 검색 영역의 **[DO, DOF, QO 타입]**, **[Primitive 타입]**, **[Object 타입]** 탭에 대한 설명이다.

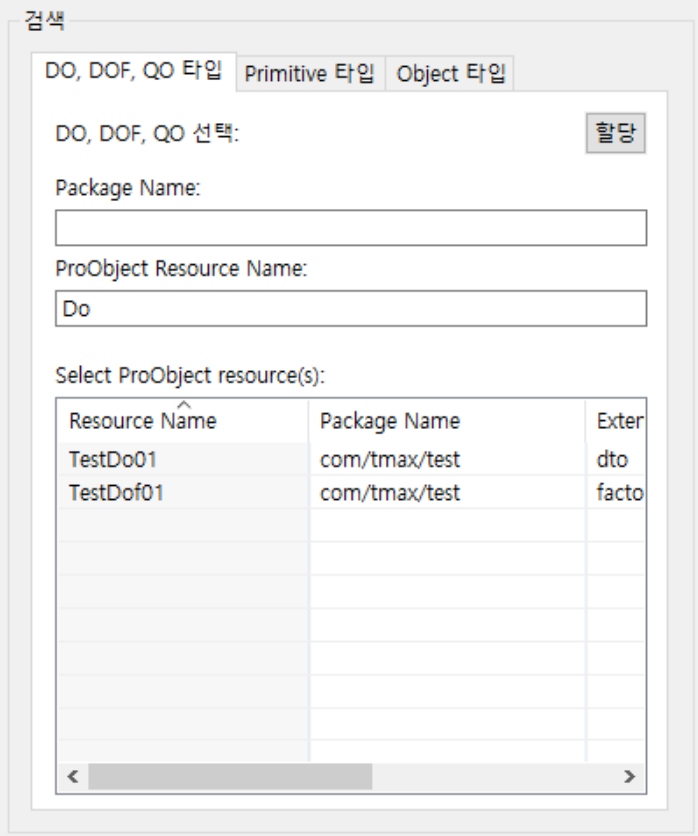
- **[DO, DOF, QO 타입]** 탭

  **[DO, DOF, QO 타입]** 탭은 메타에 등록된 DO, DOF, QO 리소스의 변수를 선언할 때 사용한다.

| 구분 | 설명 |
| --- | --- |
| DO | 일반 변수 선언 영역에 변수 정보가 나타난다.<br>◦ 변수 타입 : FIXED, VARIABLE 두 가지 종류의 배열로 선언할 수 있다.<br>◦ FIXED 배열 : **'크기'** 항목에 설정된 크기의 배열로 선언된다.<br>◦ VARIABLE 배열 : ArrayList로 선언된다.<br>◦ Inject : 해당 리소스의 injection 범위를 'NONE', 'Inject' 중 하나를 지정할 수 있다. (기본값: NONE(사용하지 않음))<br>◦ 접근 제어자 : private, public, protected 중 하나를 지정한다.<br>◦ Getter, Setter : 체크박스를 이용해 DOF의 getter와 setter를 자동 생성할 수 있다. |

| 구분 | 설명 |
| --- | --- |
| DOF | DOF 변수 선언 영역에 변수 정보가 나타난다.<br>◦ 데이터소스 : DOF가 사용할 데이터소스를 선택한다.<br>◦ 접근 제어자 : private, public, protected 중 하나를 지정한다.<br>◦ Getter, Setter : 체크박스를 이용해 DOF의 getter와 setter를 자동 생성할 수 있다. |
| QO | QO 변수 선언 영역에 변수 정보가 나타난다.<br>◦ 데이터소스 : QO가 사용할 데이터소스를 선택한다.<br>◦ 접근 제어자 : private, public, protected 중 하나를 지정한다.<br>◦ Getter, Setter : 체크박스를 이용해 DOF의 getter와 setter를 자동 생성할 수 있다. |

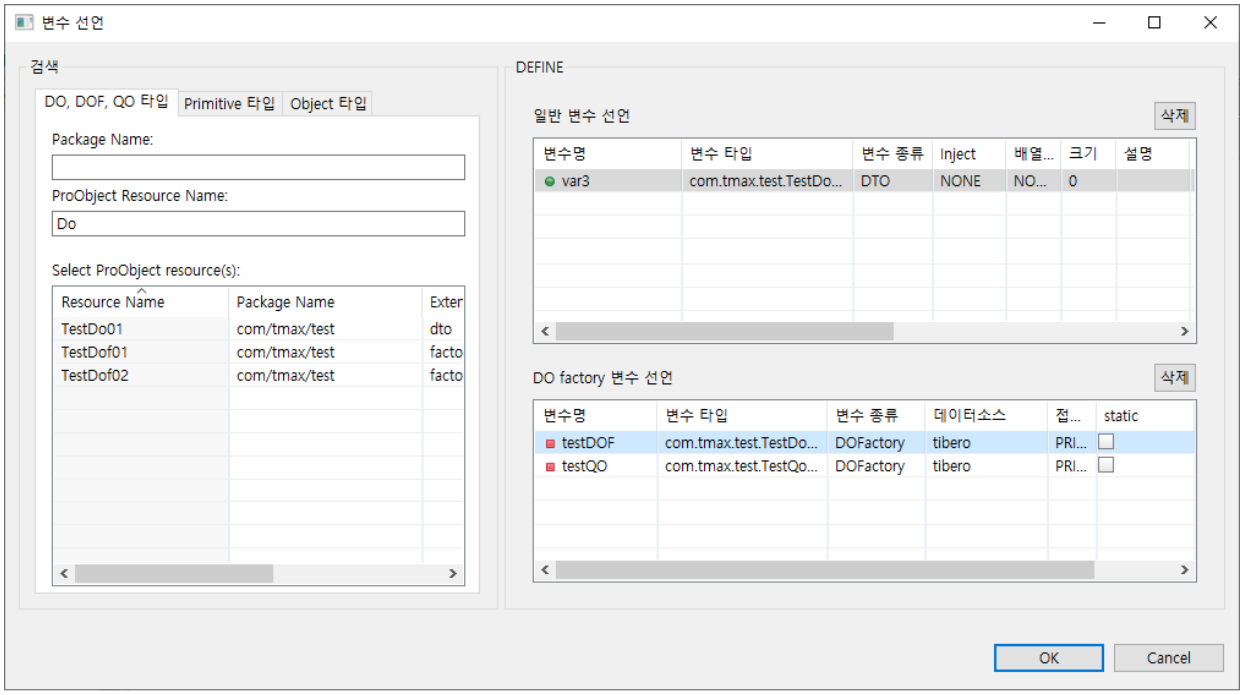
변수 선언을 위해 먼저 DO, DOF, QO 타입을 검색한다.



디자인 에디터 - 변수 선언 - DO, DOF, QO 타입 검색

DO와 DOF, QO 타입 검색한 후 등록할 리소스를 선택한 후 더블클릭하거나 **[할당]** 버튼을 클릭하면 **DEFINE 영역**의 **'일반 변수 선언'**과 **'DO factory 변수 선언'** 테이블에 추가된 변수가 각각 나타난다.

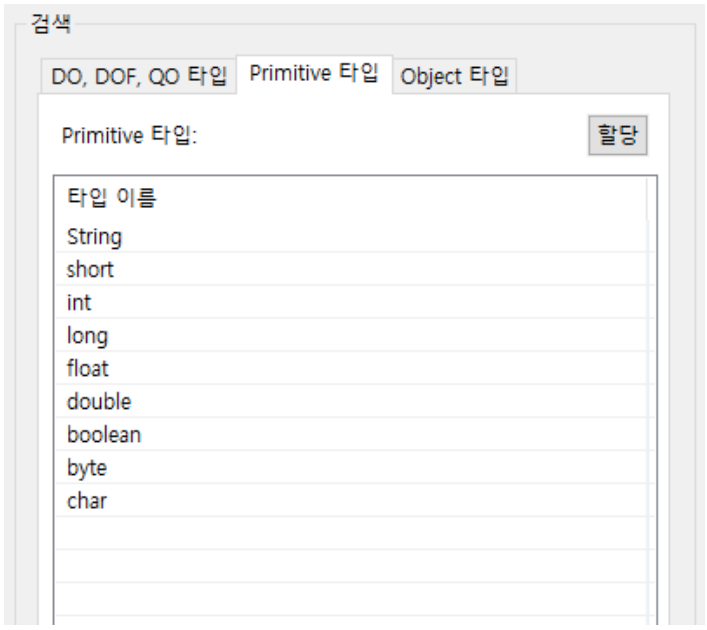
**[OK]** 버튼을 클릭하면 디자인 화면에서 선언된 해당 변수를 멤버 변수 박스에서 확인할 수 있다.




디자인 에디터 - 변수 선언 - DO, DOF 타입 추가

◦ **[Primitive 타입]** 탭

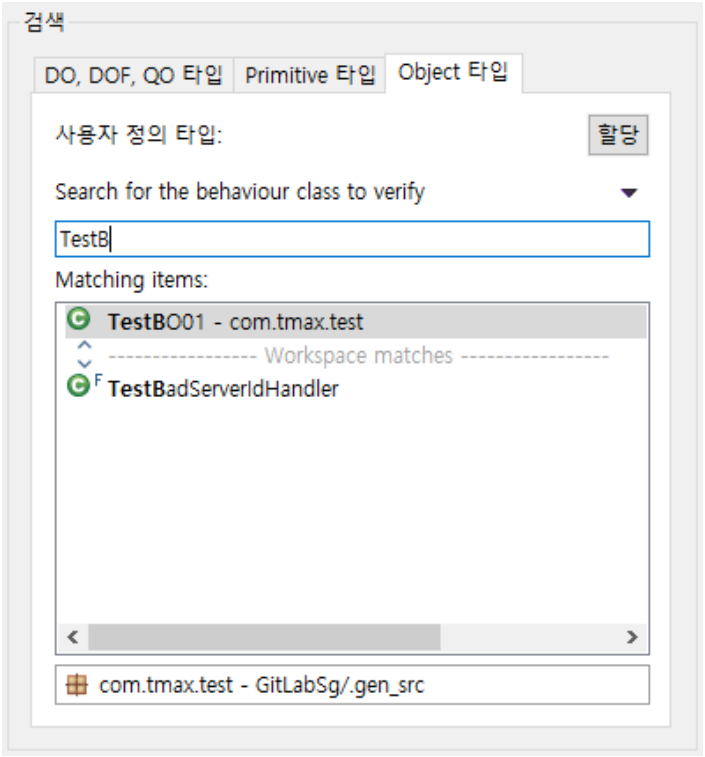
**[Primitive 타입]** 탭은 기본 변수 타입의 변수를 선언할 때 사용한다. Primitive로 생성된 변수 역시 DO와 마찬가지로 **'배열 종류'** 항목을 통해 변수를 배열로 선언할 수 있다.




디자인 에디터 - 변수 선언 - Primitive 타입 검색

◦ **[Object 타입]** 탭

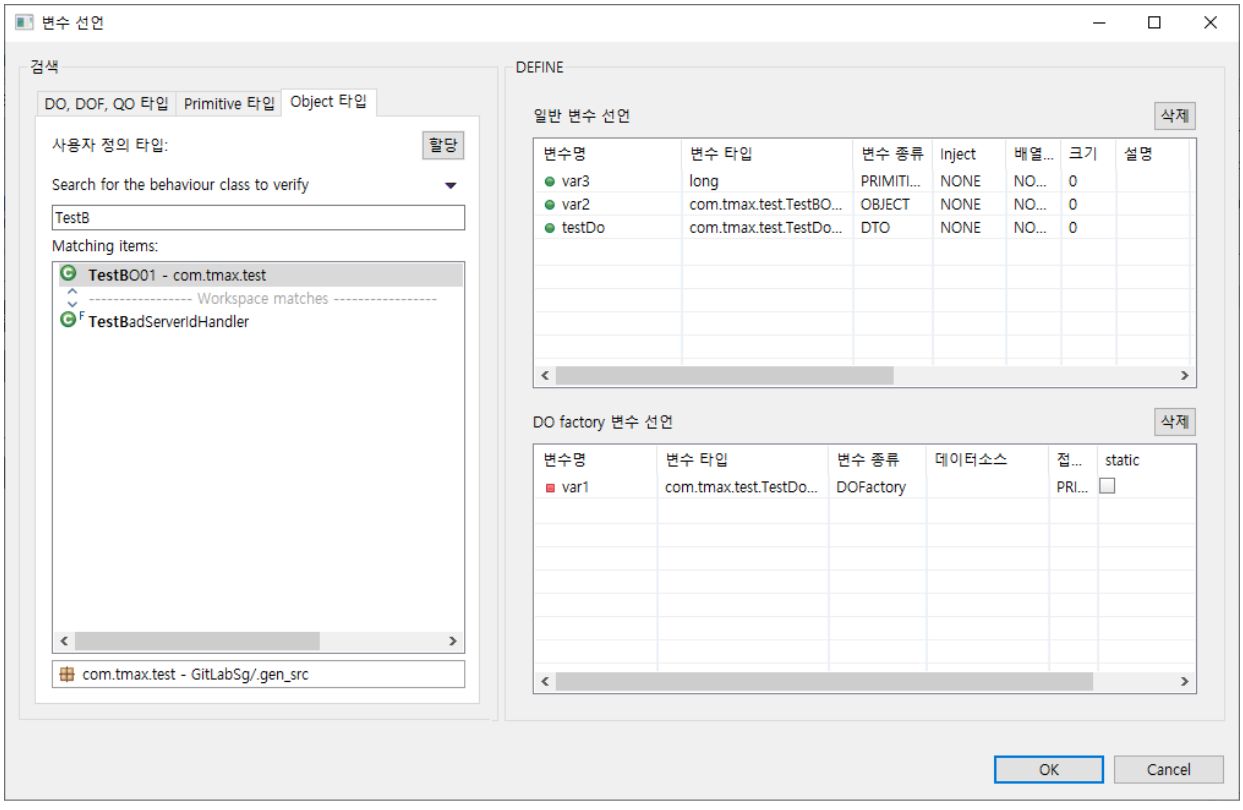
**[Object 타입]** 탭은 Build Path와 프로젝트에 등록된 오브젝트 객체의 변수를 선언할 때 사용한다.




디자인 에디터 - 변수 선언 - Object 타입 검색

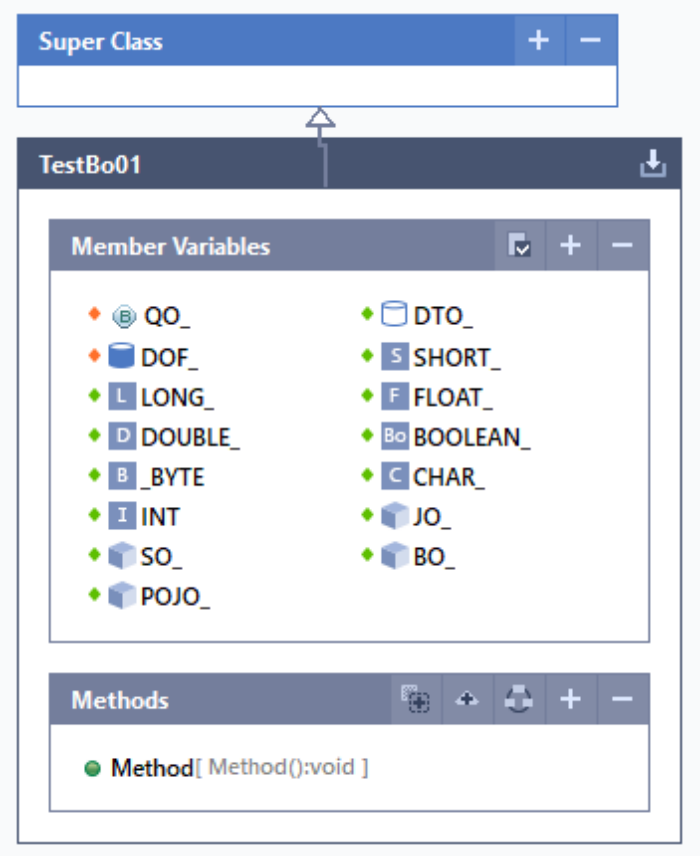
선언할 변수의 타입을 확인한 뒤 해당 변수 타입의 리소스명을 검색하면 목록에 리소스가 조회된다.

검색된 리소스 이름을 선택한 후 리소스를 선택한 후 더블클릭하거나 **[할당]** 버튼을 클릭하면 **DEFINE 영역**에 자동으로 변수명, 변수 타입, 변수 종류, 배열크기, 설명 내용이 삽입된다. 변수명을 확인한 후 작성할 변수명으로 변경한다. 변수명을 변경하지 않으면 자동으로 입력된 값으로 변수가 선언된다. **[OK]** 버튼을 클릭하면 디자인 화면에서 선언된 해당 변수를 멤버 변수 박스에서 확인할 수 있다.




디자인 에디터 - 변수 선언 - Object 타입 추가

3. 다음은 멤버 변수가 추가된 디자인 에디터이다. **[소스]** 탭에서 변수 선언 내용을 확인할 수 있다.




디자인 에디터 - 변수 추가 (2)

PO에서는 아래와 같이 DTO, PRIMITIVE, OBJECT, DOFactory Type을 지원한다.

◦ DTO

| 변수명 | 설명 |
|---|---|
| DTO_ | DO 리소스의 변수 종류이다. |

◦ PRIMITIVE

| 변수명 | 설명 |
|---|---|
| SHORT_ | Java의 short type과 동일하다. |
| LONG_ | Java의 long type과 동일하다. |
| FLOAT_ | Java의 float type과 동일하다. |
| DOUBLE_ | Java의 double type과 동일하다. |
| BOOLEAN_ | Java의 boolean type과 동일하다. |
| BYTE_ | Java의 byte type과 동일하다. |
| CHAR_ | Java의 char type과 동일하다. |
| INT_ | Java의 int type과 동일하다. |

◦ OBJECT

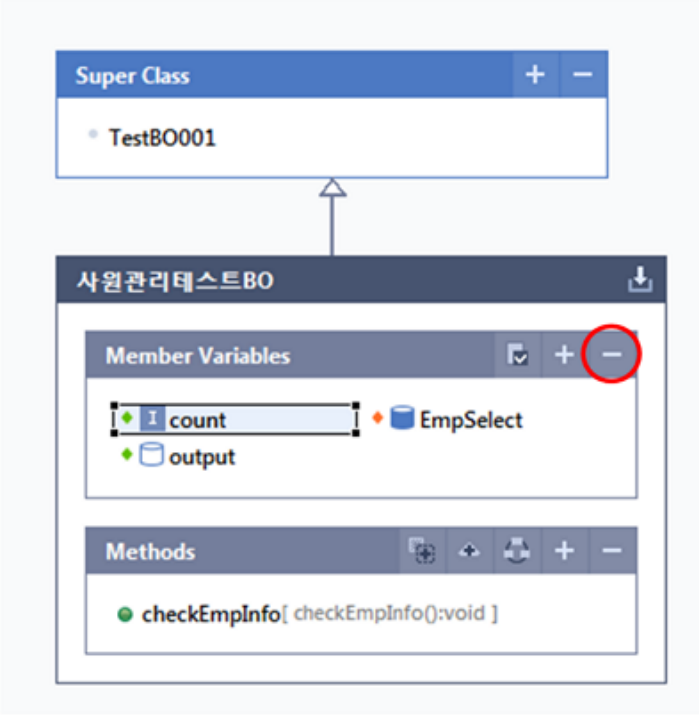
| 변수명 | 설명 |
| --- | --- |
| JO_ | JO Type은 Object로 생성된다. |
| SO_ | SO Type은 Object로 생성된다. |
| BO_ | BO(Designer) Type은 Object로 생성된다. |
| POJO_ | BO(POJO) Type은 Object로 생성된다. |

◦ DOFactory

| 변수명 | 설명 |
| --- | --- |
| QO_ | QO는 DOFactory로 생성된다. |
| DOF_ | DOF는 DOFactory로 생성된다. |

### — (변수 삭제)

삭제할 변수를 클릭한 후 [-] 버튼을 클릭하거나 <Delete> 키를 누르면 클릭한 변수가 삭제된다.




디자인 에디터 - 변수 삭제

#### 4.3.3.4. 메소드

메소드 정의 박스에서 메소드를 정의, 삭제, 비구현 메소드 추가, 오버라이드 메소드 생성 작업을 할 수 있다.

| 아이콘 | 설명 |
| --- | --- |
| (비구현 메소드 추가) | 인터페이스 및 추상 클래스를 해당 BO 수퍼로 등록하는 경우 반드시 구현해야 하는 메소드에 대해 자동으로 메소드를 추가한다. 자세한 내용은 "비구현 메소드 추가"를 참고한다. |

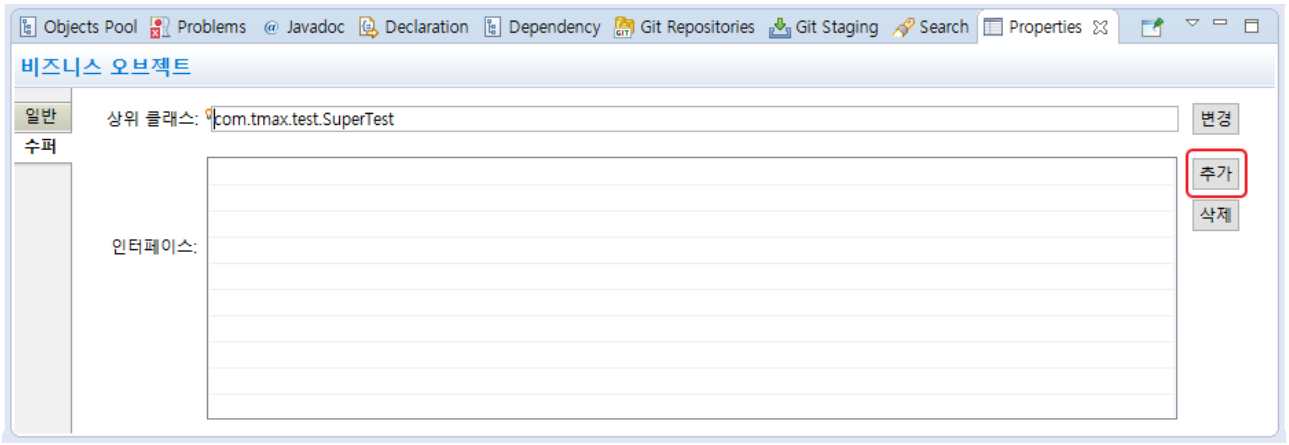
| 아이콘 | 설명 |
| --- | --- |
| (오버라이드 메소드 생성) | Override 또는 implement할 메소드를 선택하여 생성한다. 자세한 내용은 "오버라이드 메소드 생성"을 참고한다. |
| (메서드명 수정) | 메소드명을 수정할 수 있다. 자세한 내용은 "메소드명 수정"을 참고한다. |
| (메소드 정의) | 메소드 박스 안의 **[+]** 버튼을 클릭하면 메소드 기본 정보를 등록할 수 있다. 자세한 내용은 "메소드 정의"를 참고한다. |
| (메소드 삭제) | 메소드 박스 안의 **[-]** 버튼을 클릭하면 정의한 메소드를 삭제할 수 있다. 자세한 내용은 "메소드 삭제"를 참고한다. |

### (비구현 메소드 추가)

상위 인터페이스나 추상 클래스를 상속 또는 구현하는 경우 **[비구현 메소드 추가]** 아이콘을 클릭하면 상위 클래스의 메소드를 자동으로 추가할 수 있다.

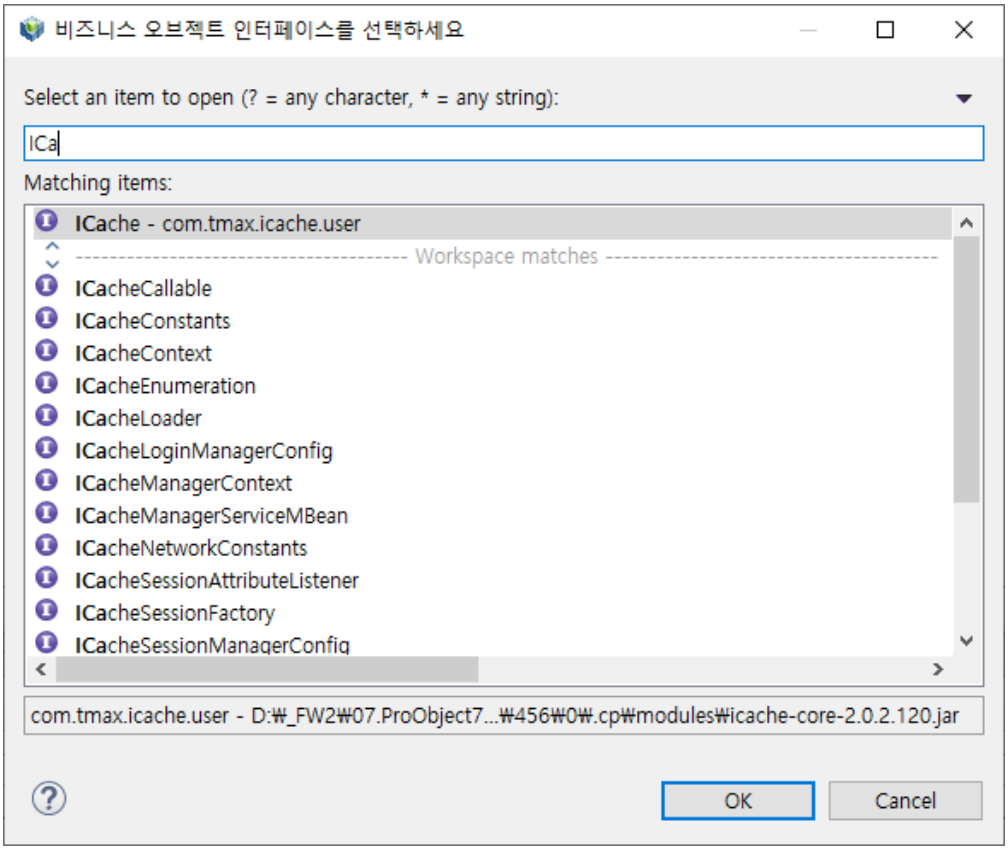
다음은 비구현된 메소드를 추가하는 과정에 대한 설명이다.

1. BO 인터페이스 추가하는 방법은 BO를 생성(BO 생성 참고)할 때 또는 **[Properties] > [수퍼]** 탭에서 상위 클래스나 인터페이스를 추가한다.



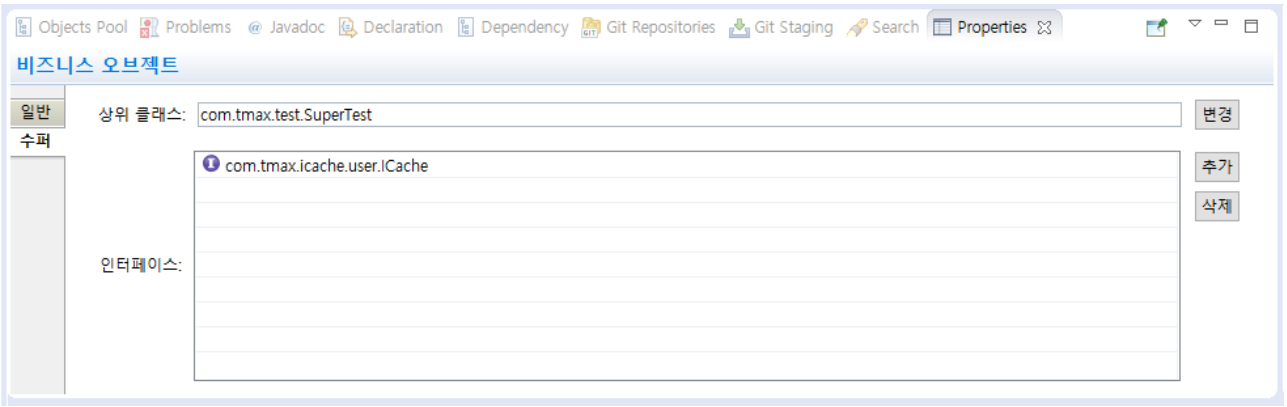

디자인 에디터 - 비구현 메소드 추가 (1)

**'인터페이스'** 항목에서 **[추가]** 버튼을 클릭해서 BO 속성으로 추가할 인터페이스를 선택한 후 **[OK]** 버튼을 클릭한다.



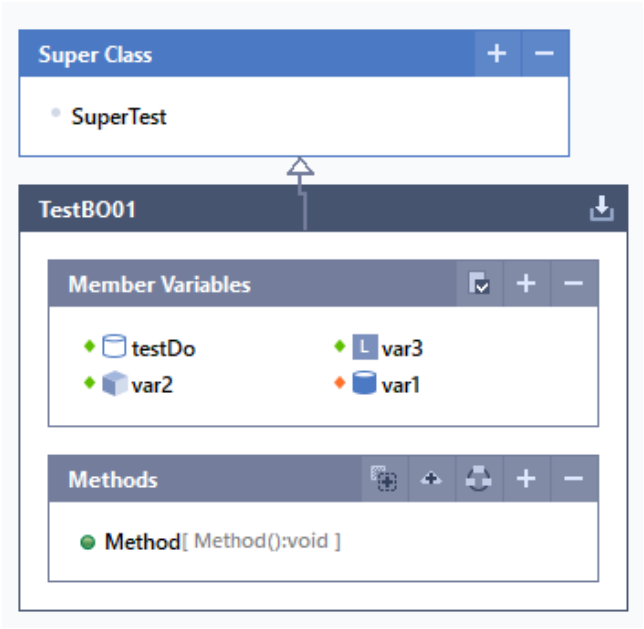
디자인 에디터 - 비구현 메소드 추가 (2)

**'인터페이스'** 항목에서 추가된 인터페이스를 확인할 수 있다.

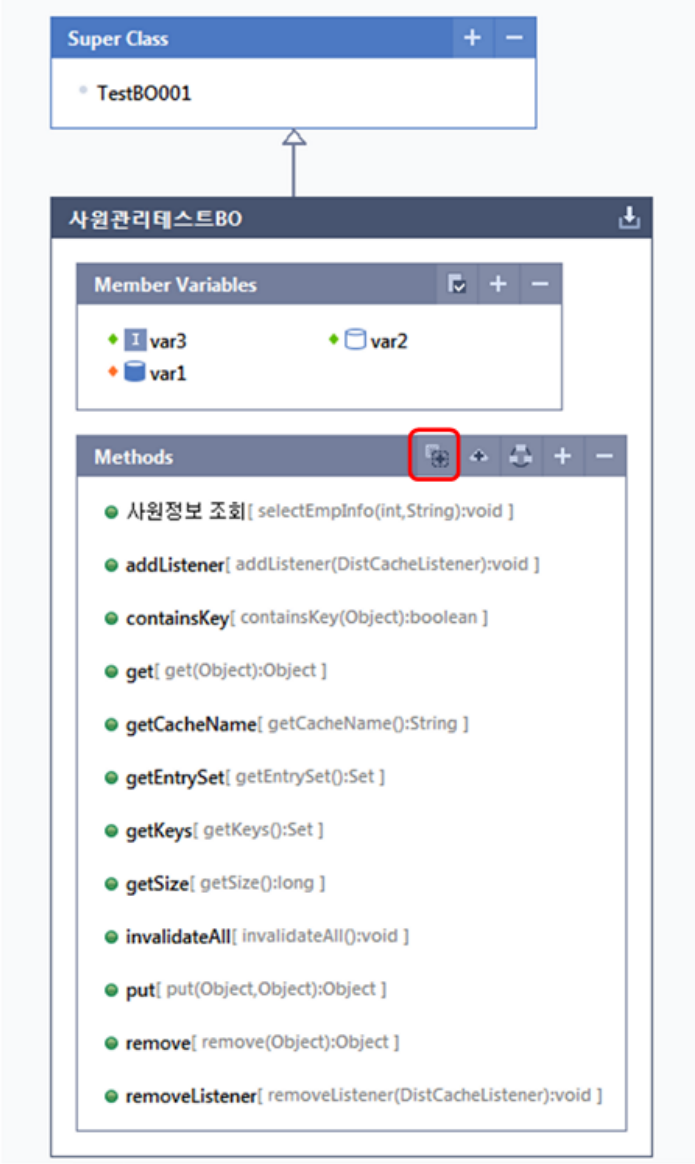


디자인 에디터 - 비구현 메소드 추가(3)

2. 기본 디자인 에디터에서 **[비구현 메소드 추가]** 아이콘을 클릭하면 상위 인터페이스에 선택한 메소드를 메소드 박스에서 확인할 수 있다.




디자인 에디터 - 비구현 메소드 추가 전 BO 화면



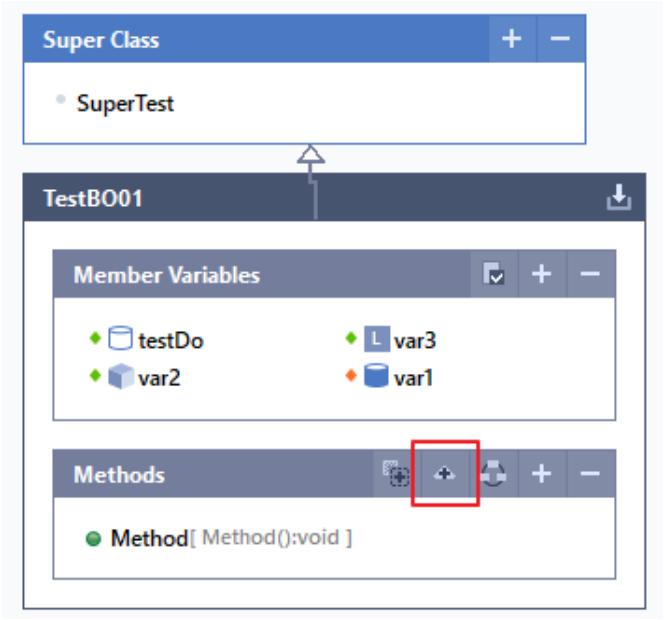

디자인 에디터 - 비구현 메소드 추가 버튼 클릭 후

**(오버라이드 메소드 생성)**

메소드 박스에서 **[오버라이드 메소드 생성]** 아이콘을 클릭하면 override 또는 implement할 메소드를 선택하여 생성할 수 있다.

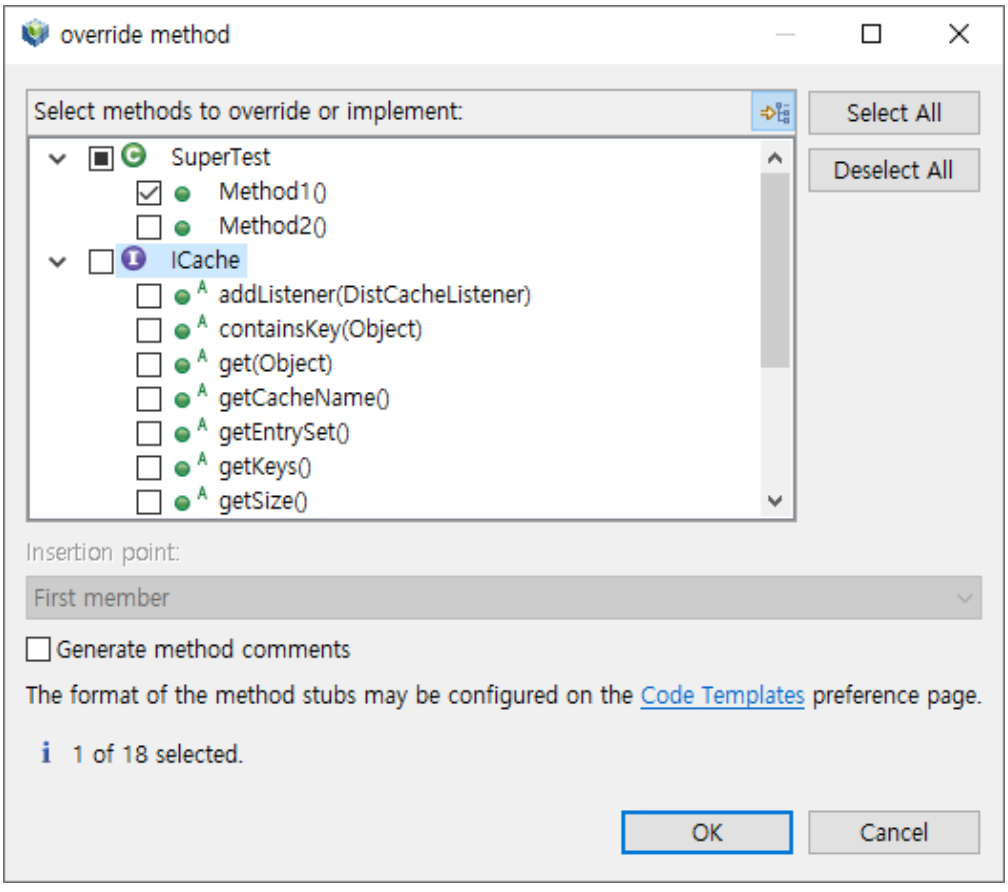
다음은 오버라이드 메소드를 생성하는 과정에 대한 설명이다.

1. 메소드 박스에서 **[오버라이드 메소드 생성]** 아이콘을 클릭한다.



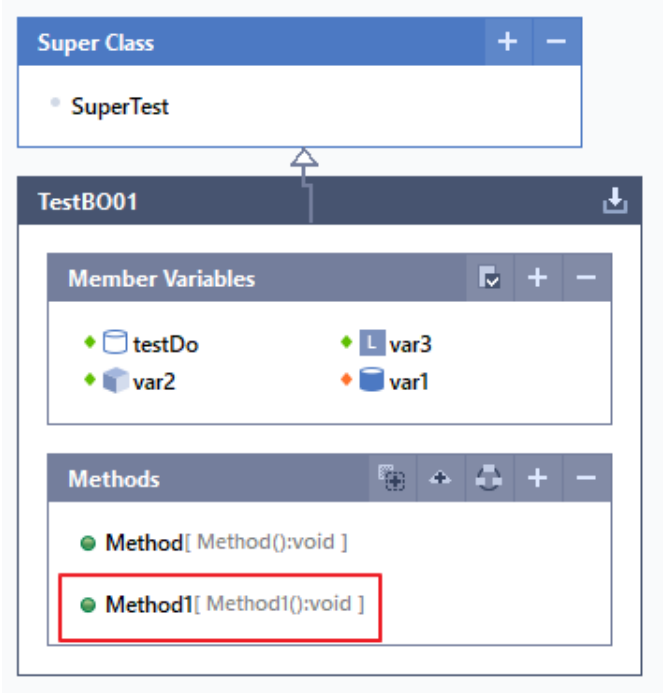

디자인 에디터 - 오버라이드 메소드 생성 (1)

2. Super Class에 정의된 오브젝트의 메소드를 오버라이드하는 경우 다음과 같이 **override method 화면**에서 메소드 목록에서 override 또는 implement할 메소드를 선택한 후 **[OK]** 버튼을 클릭한다.




디자인 에디터 - 오버라이드 메소드 생성 (2)

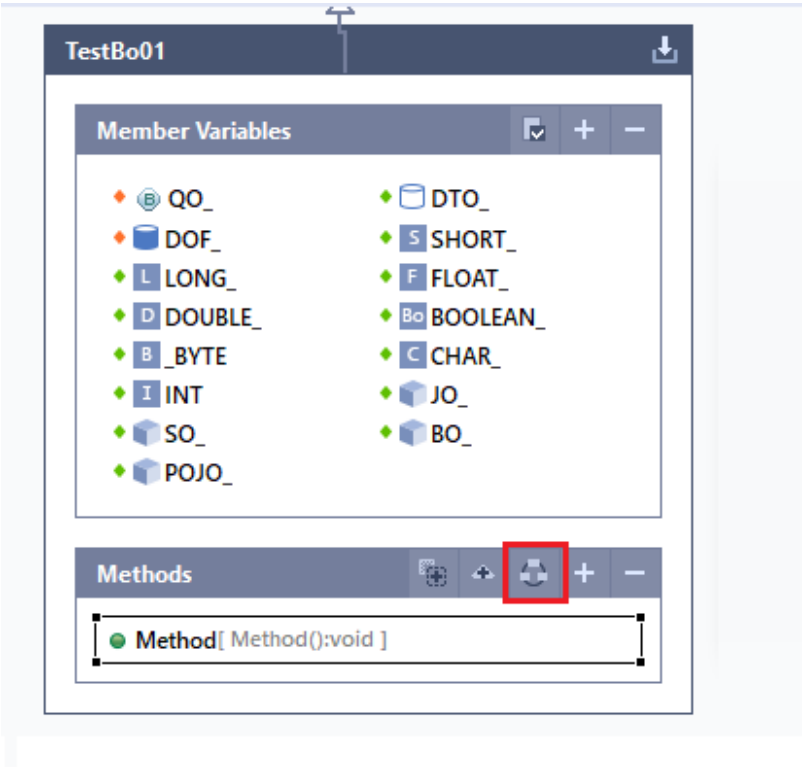
3. 선택한 메소드가 메소드 박스에 추가된 것을 확인할 수 있다.




디자인 에디터 - 오버라이드 메소드 생성 (3)

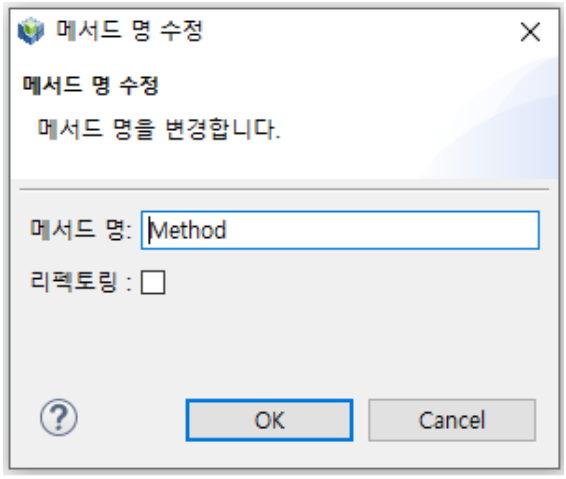
### (메소드명 수정)

수정할 메소드를 선택한 후 **[메소드명 수정]** 아이콘을 클릭하면 메소드 이름을 수정할 수 있다.




디자인 에디터 - 메소드명 수정 (1)

**메소드명 수정 화면**에서 수정할 이름을 입력한 후 **[OK]** 버튼을 클릭한다.



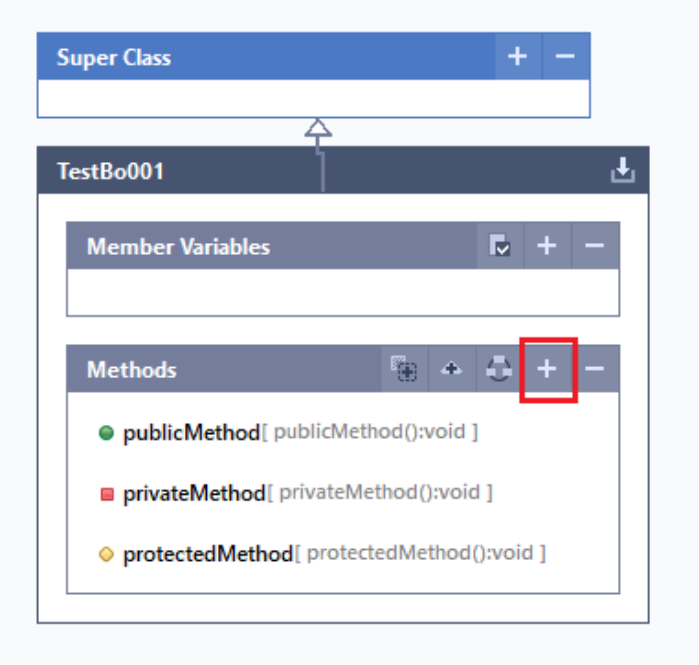
디자인 에디터 - 메소드명 수정 (2)

| 메뉴 | 설명 |
|---|---|
| 메소드 명 | 변경할 새로운 메소드명을 기입한다. |
| 리펙토링 | 체크하면 참조하는 리소스의 메소드 소스를 리펙토링한다. |

## + (메소드 정의)

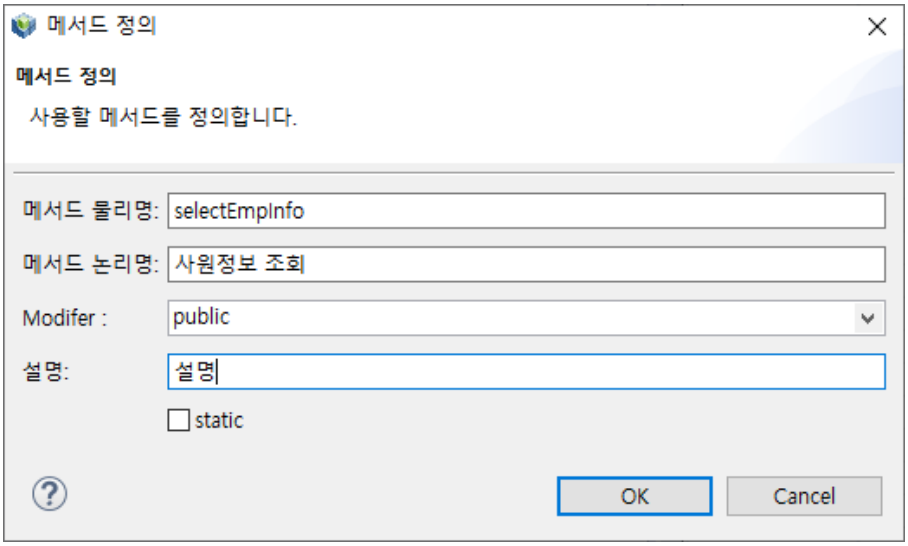
다음은 메소드를 정의하는 과정에 대한 설명이다.

1. 메소드를 정의하기 위해서 **[메소드 정의]** 아이콘을 클릭한다.



디자인 에디터 - 메소드 정의 아이콘

2. **메소드 정의 화면**에서 메소드 물리명, 논리명, 설명을 입력한 후 **[OK]** 버튼을 클릭한다.

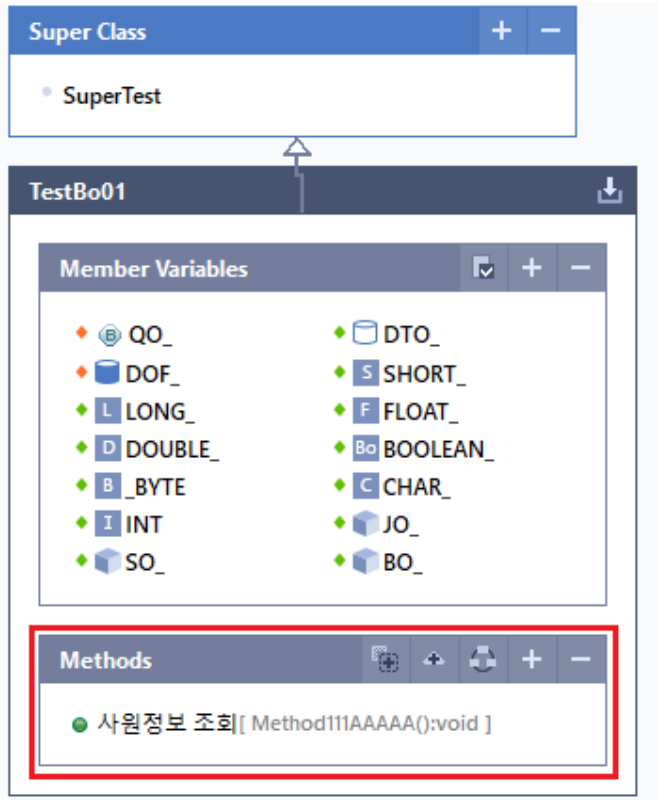


디자인 에디터 - 메소드 정의 화면

| 메뉴 | 설명 |
|---|---|
| 메소드 물리명 | 메소드의 물리명을 입력한다. |
| 메소드 논리명 | 개발자의 가독성이 있는 이름을 입력한다. |
| Modifer | public, protected, private 여부를 선택한다.<br><br>선택한 접근 제어자별로 디자인 편집화면에 메소드명에 다음의 색상이 표시된다 (메소드 정의 완료 후 디자인 편집화면 참고).<br><br>• public : 초록<br>• protected : 노랑<br>• private : 빨강 |
| 설명 | 메소드에 대한 설명을 입력한다. |
| static | static 여부를 선택한다. |

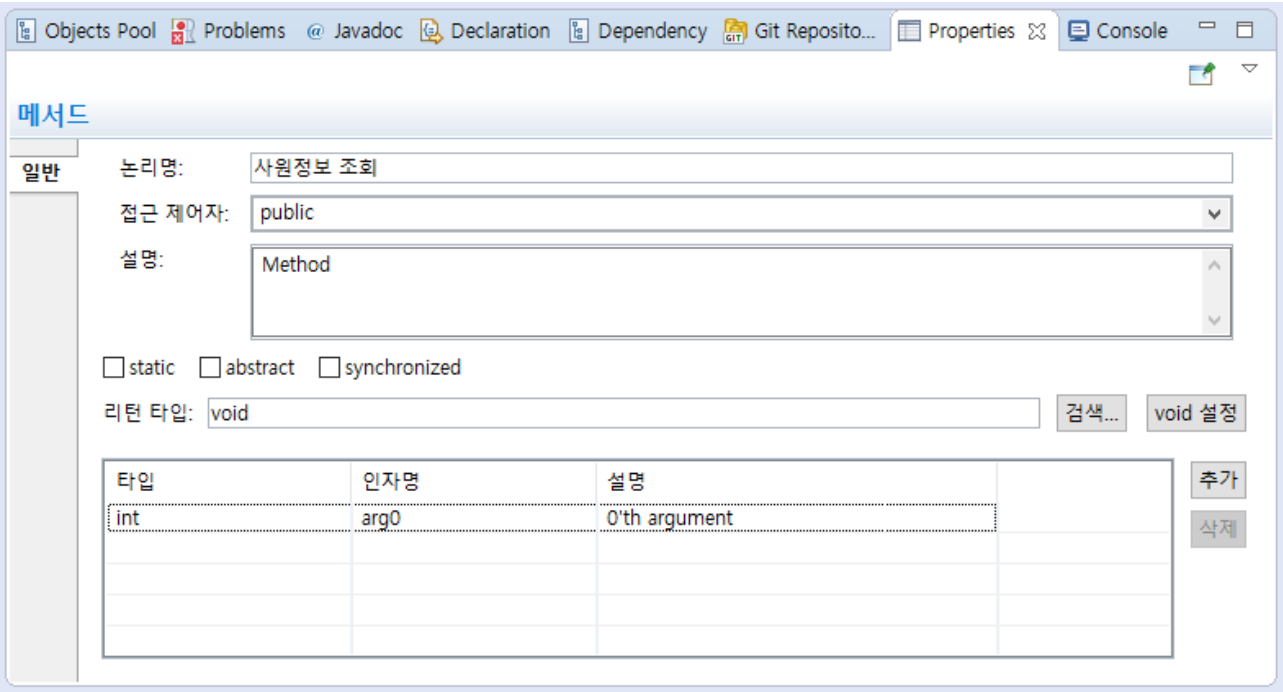
3. 메소드 정의가 완료되면 **디자인 에디터**에 생성한 메소드 내용을 조회할 수 있다. 생성된 메소드를 더블클릭하면 소스 개발을 위한 해당 메소드 플로우 디자인 화면에 조회된다. 자세한 내용은 EMB Designer에서 설명한다.

   **[소스]** 탭에서 추가된 메소드 정의 내용을 확인할 수 있다.




메소드 정의 완료 후 디자인 편집화면

4. 생성된 메소드의 설정 정보는 Properties의 **[일반]** 탭에서 수정할 수 있다.



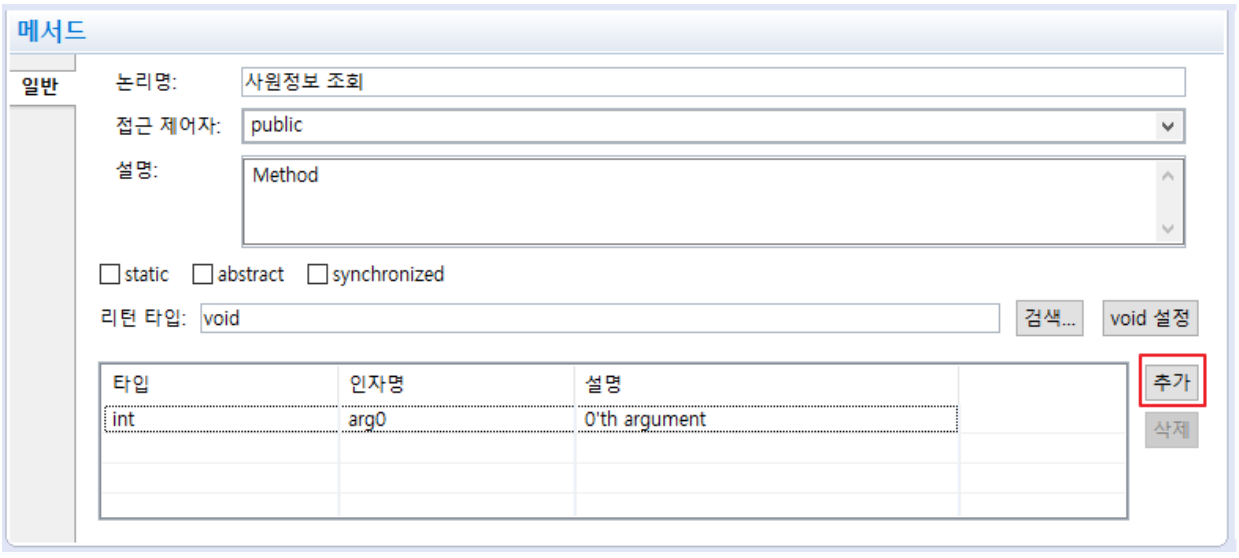

디자인 에디터 - [Properties] - [일반] 탭 - 메소드 설정

| 항목 | 설명 |
|---|---|
| 논리명 | 메소드 정의를 할 때 메소드 논리명이 기본 정보로 설정된다. |
| 접근제어자 | 메소드의 접근제어자를 public, private, protected 중 하나로 선언한다. |
| 설명 | 메소드의 기본 정보를 등록할 때 기입된 정보가 설정되며 설명을 수정할 수 있다. |

| 항목 | 설명 |
| --- | --- |
| static | static 메소드 여부를 선택한다. |
| abstract | abstract 메소드 여부를 선택한다. |
| synchronized | synchronized 메소드 여부를 선택한다. |
| 리턴 타입 | 기본적으로 void형으로 설정되어 있으며 객체 리턴 등 메소드 리턴 타입을 설정할 수 있다. **[검색]** 버튼을 클릭해서 DataObject, Primitive, 사용자 정의 오브젝트들을 리턴 타입으로 지정할 수 있다. (디자인 에디터 - [Properties] - [일반] 탭 - 파라미터 설정 (2) 참고) |
| 파라미터 리스트 | 메소드의 목록을 조회한다.<br>• 타입 : 파라미터의 데이터 타입(타입을 선택한 후 **[...]** 버튼을 클릭해서 디자인 에디터 - [Properties] - [일반] 탭 - 파라미터 설정 (1)에서 변수 타입을 설정 가능)<br>• 인자명 : 파라미터명(인자명을 선택한 후 사용할 이름이나 설명을 수정)<br>• 설명 : 파라미터 변수에 대한 내용 |

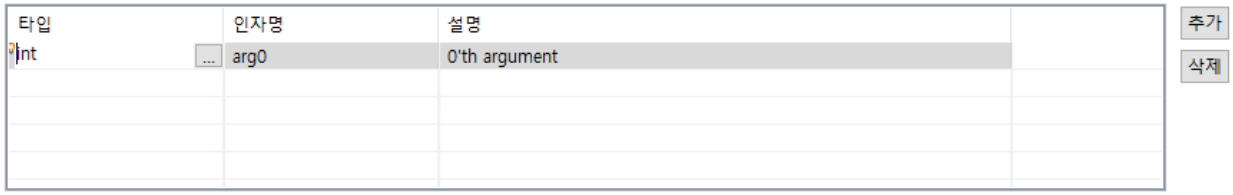
**[추가]**, **[삭제]** 버튼을 클릭해서 파라미터를 추가하거나 삭제할 수 있다. 다음은 메소드의 변수를 설정하는 방법에 대한 설명이다.

a. 파라미터 리스트의 **[추가]** 버튼을 클릭한다.



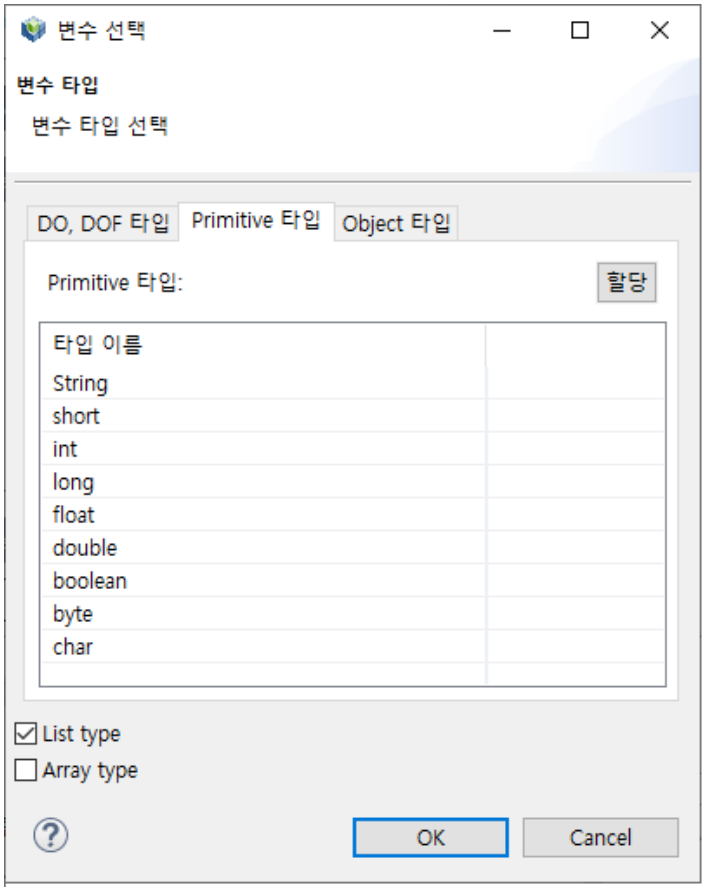

디자인 에디터 - [Properties] - [일반] 탭 - 변수 타입 설정

b. 파라미터 리스트의 **'타입'** 항목의 **[...]** 버튼을 클릭해서 타입을 설정할 수 있다.



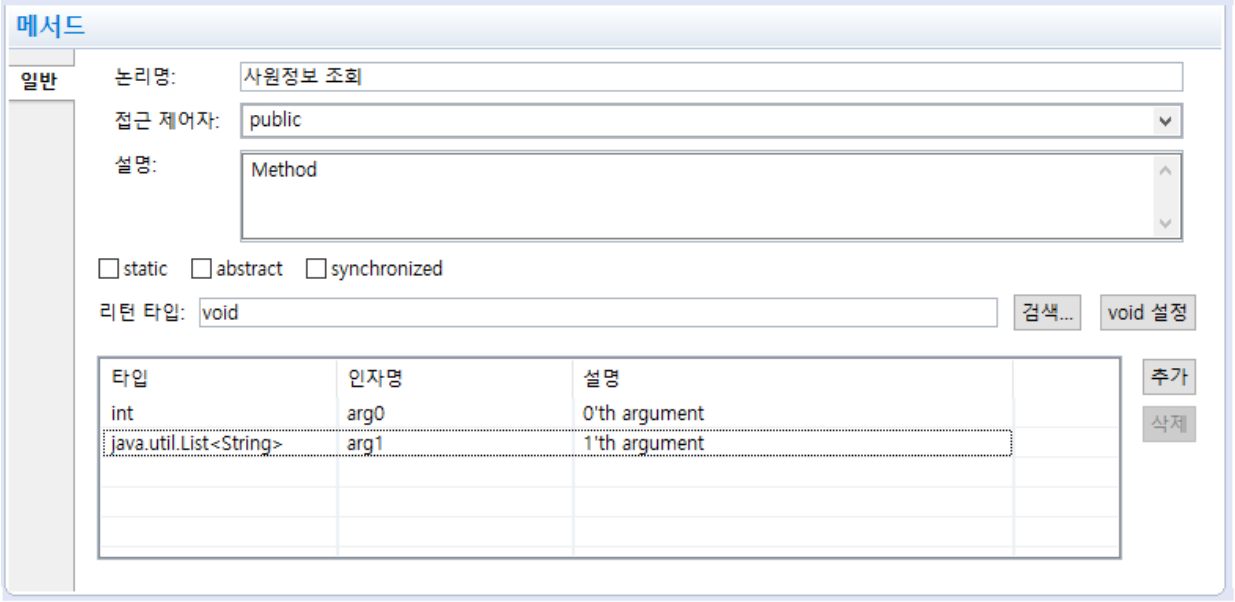

디자인 에디터 - [Properties] - [일반] 탭 - 파라미터 설정 (1)

c. **변수 선택 화면**에서 **'List type'**, **'Array type'**을 체크하면 변수 타입을 List와 Array로 설정할 수 있다.




디자인 에디터 - [Properties] - [일반] 탭 - 파라미터 설정 (2)

d. 다음은 타입 설정을 완료한 후의 파라미터 목록이다. Sting 타입은 'List type'으로 Int 타입은 'Array type'으로 설정되었다.




디자인 에디터 - [Properties] - [일반] 탭 - 파라미터 설정 (3)

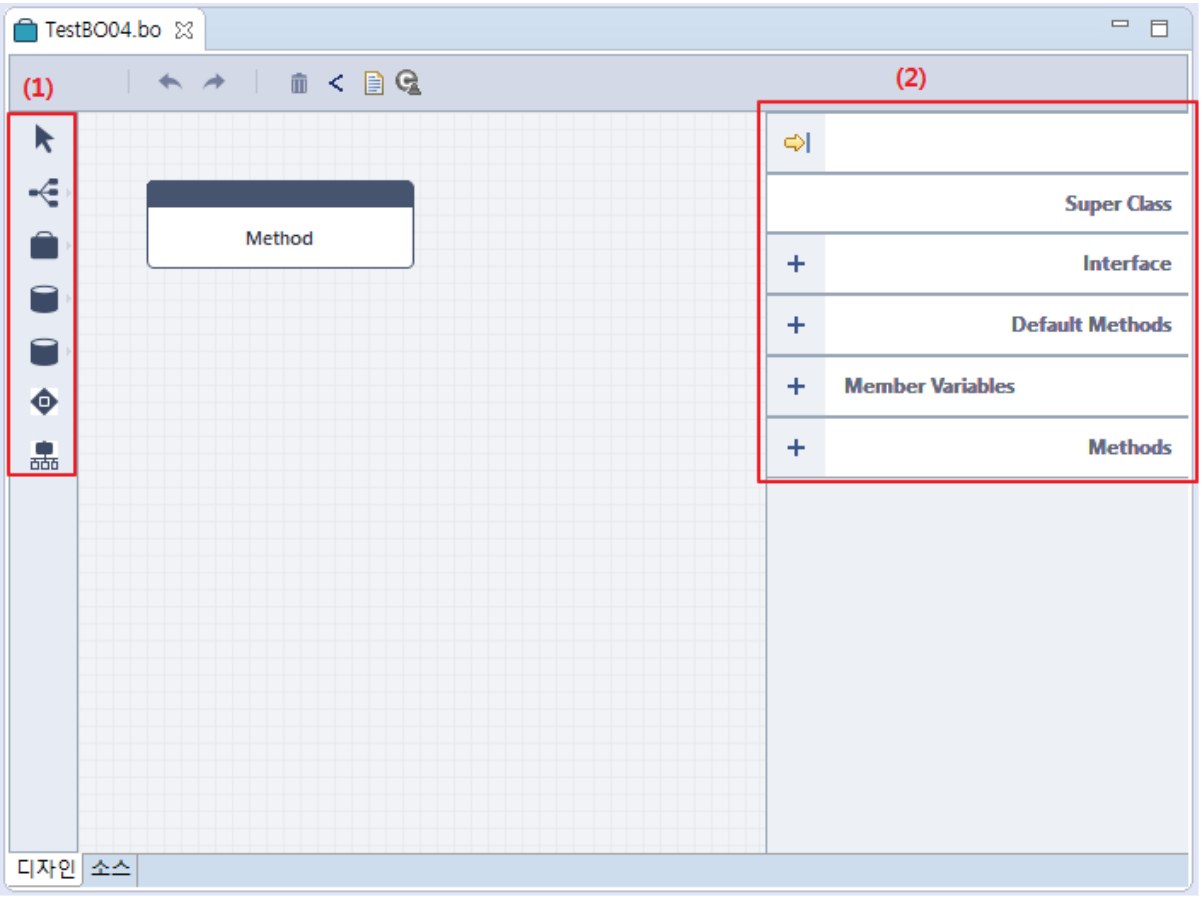
### — (메소드 삭제)

삭제할 메소드를 클릭한 후 [-] 버튼을 클릭하거나 <Delete> 키를 누르면 클릭했던 변수가 삭제된다.


디자인 에디터 - 메소드 삭제

## 4.3.4. EMB Designer

**EMB Designer**는 메소드 내용을 플로우 설계, 플로우 편집, 소스 편집, 플로우 삭제 등 개발자가 실질적으로 개발할 수 있는 영역으로 EMB 모듈의 플로우를 가시적으로 나타낸다. 사용자는 **Palette**에서 다양한 모듈을 가져다 드래그 앤드 드롭으로 추가할 수 있다.




EMB Designer

- **(1) 왼쪽 Palette**

  해당 Task 모듈을 선택하여 디자인 에디터에 드래그 앤드 드롭하여 모듈을 등록하고 구현한다.

| 메뉴 | 설명 |
|---|---|
| (Nodes) | • **[이너 모듈]** : 여러 모듈들이 동시에 사용하는 변수를 지정하거나 모듈들의 진입조건, 예외처리 등을 설정하기 위해 사용하는 모듈이다. 자세한 내용은 이너 모듈을 참고한다.<br>• **[버추얼 모듈]** : 개발자가 소스 코딩을 플로우에 삽입하는 경우 사용하는 모듈로 자식 노드를 가질 수 없다. 자세한 내용은 버추얼 모듈을 참고한다.<br>• **[루프 모듈]** : 반복문을 구현할 때 사용하는 모듈로 하위에 자식 노드를 가질 수 있다. 자세한 내용은 루프 모듈을 참고한다.<br>• **[Assign 모듈]** : 입출력 매핑 또는 DO를 매핑할 때 사용하는 모듈로 자식 노드를 가질 수 있으며 모듈을 더블클릭하면 매핑화면이 나타난다. 자세한 내용은 Assign 모듈을 참고한다.<br>• **[static method 모듈]** : Static Type의 메소드를 호출하기 위한 모듈이다. 자세한 내용은 static method 모듈을 참고한다.<br>• **[로컬 트랜잭션]** : EMB Designer에서 Assign 모듈은 로직 중간에 Commit, Rollback 등의 트랜잭션 작업을 개발자가 할 수 있는 기능이다. 자세한 내용은 로컬 트랜잭션을 참고한다.<br>• **[DDL 모듈]** : SQL의 DDL(Data Definition Language)의 지원을 위한 기능이다.자세한 내용은 DDL 모듈을 참고한다.<br>• **[getReply 모듈]** : 다른 메소드의 응답을 기다릴 때 사용된다. 자세한 내용은 getReply 모듈을 참고한다. |
| (BizObject 변수) | BO 객체를 레퍼런스 변수로 정의하여 사용할 수 있는 모듈로 변수를 정의할 때 메타에 등록된 BO를 Object type으로 선언하면 나타난다. 자세한 내용은 BizObject 변수를 참고한다. |
| (DO Factory 변수) | DOF 객체를 레퍼런스 변수로 정의하여 사용할 수 있는 모듈로 변수를 정의할 때 메타에 등록된 DOF를 DOF type으로 선언하면 나타난다. 자세한 내용은 DO Factory 변수 (DB)와 DO Factory 변수 (FILE)을 참고한다. |
| (Query Object 변수) | QO 객체를 레퍼런스 변수로 정의하여 사용할 수 있는 모듈로 변수를 정의할 때 메타에 등록된 QO를 QO type으로 선언하면 나타난다. 자세한 내용은 Query Object 변수를 참고한다. |
| (로컬 메소드) | BO에 정의된 메소드 내용을 볼 수 있으며 EMB Designer에 추가할 경우 로컬 메소드를 호출할 수 있다. 자세한 내용은 로컬 메소드를 참고한다. |
| (상위 메소드) | BO에 정의된 상위 클래스의 메소드 내용을 볼 수 있으며 EMB Designer에 추가할 경우 상위 클래스의 메소드를 호출할 수 있다. 자세한 내용은 상위 메소드를 참고한다. |

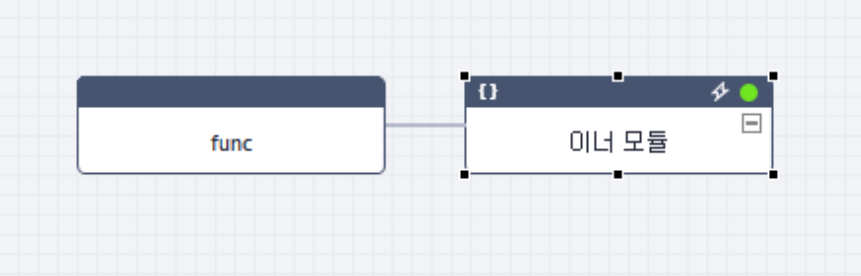
- **(2) 오른쪽 Palette**

  정의된 변수내역을 바탕으로 빠른 확인/수정할 수 있는 Short Cut을 제공한다.

| 메뉴 | 설명 |
| --- | --- |
| Super Class | Super Class로 선언한 클래스가 표시된다. |
| Interface | BO를 생성하는 경우 선택한 Interface Class가 표시되며, **[+]** 버튼을 클릭하면 추가도 가능하다. |
| Default Methods | 현재 BO의 클래스 이름이 표시된다. |
| Member Variables | 선언된 멤버 변수 내역이 표시되며, **[+]** 버튼을 클릭하면 추가도 가능하다. |
| Methods | 선언된 메소드 내역이 표시되며, **[+]** 버튼을 클릭하면 추가도 가능하다. |

#### 4.3.4.1. 이너 모듈

EMB Designer에서 이너 모듈은 분기점의 역할을 수행하고 변수를 정의하여 다양한 서비스를 작성할 수 있다. **Palette**에서 **[Node] > [이너 모듈]**을 선택한 후 디자인 영역을 클릭하면 이너 모듈이 삽입된다. 이너 모듈을 클릭하면 원하는 위치로 이동할 수 있다.




EMB Designer - 이너 모듈 - 디자인 영역

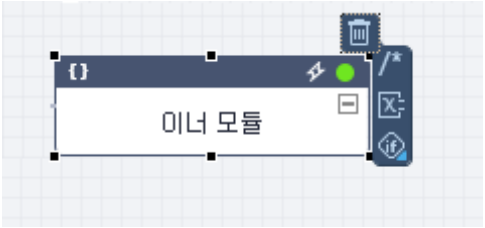
다음은 이너 모듈 디자인 에디터에 생성한 소스이다.

```
//[BEGIN_NODE_BLOCK, 0, func()]
{
    //이너 모듈
}
//[END_NODE_BLOCK, 0, func()]
```

EMB Designer - 이너 모듈 - 소스젠

- **이너 모듈 노드 속성**

  이너 모듈을 클릭하면 아래와 같이 아이콘이 보이며 각각의 아이콘은 기능을 갖고 있다.



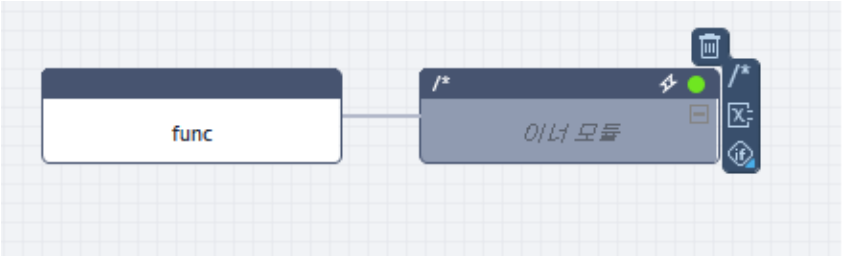

EMB Designer - 이너 모듈 - 속성

  -  (삭제)

    이너 모듈을 삭제한다.

◦ (주석 처리)

이너 모듈 이하는 주석 처리한다. 주석 처리 노드 속성을 클릭하며 다음과 같이 해당 이너 모듈이 주석 처리된다.




EMB Designer - 이너 모듈 - 주석 처리 - 디자인

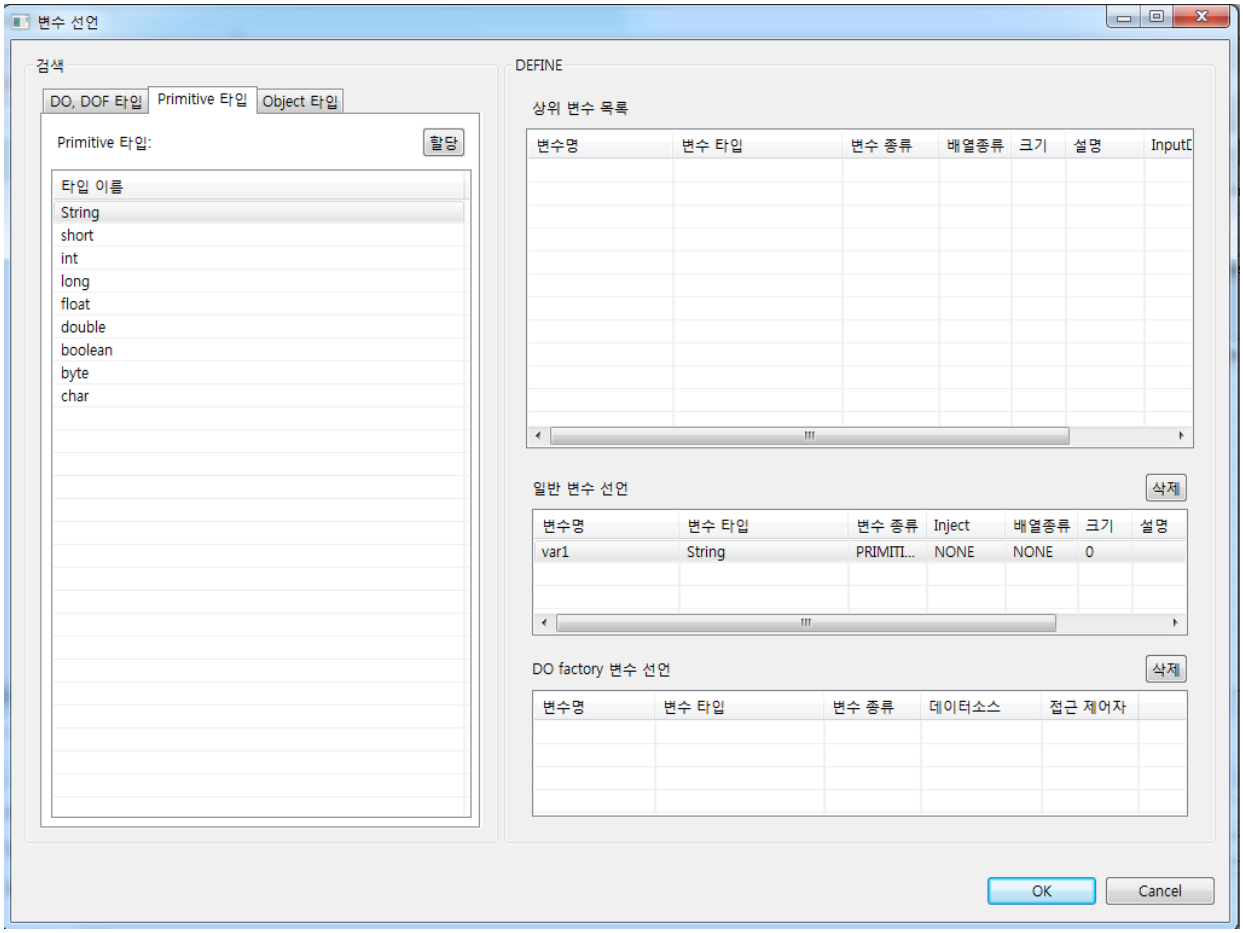
아래 그림은 주석 처리한 이너 모듈의 소스젠 내용이다. 주석 처리를 해제하려면 **[주석 처리]** 노드 속성 아이콘을 다시 한 번 클릭한다.

```
//[BEGIN_NODE_BLOCK, 0, func()]
//{
    //이너 모듈
//}
//[END_NODE_BLOCK, 0, func()]
```

EMB Designer - 이너 모듈 - 주석 처리 - 소스젠

◦ (지역변수 선언)

이너 모듈 내에서 사용할 수 있는 지역변수를 선언한다. 지역변수 선언 노드 속성을 클릭하여 **지역변수 선언 화면**에서 필요한 변수를 선언한다.




EMB Designer - 이너 모듈 - 지역변수 선언 - 디자인

위와 같이 일반 변수 선언에 'var1'을 설정하고 **[OK]** 버튼을 클릭하면 다음과 같이 소스가 생성된다. 사용법은 메소드의 "변수 정의"와 동일하나 위와 같이 이너 모듈 안에 지역변수로 생성된다. 변수 정의의 상세 내용은 "변수 선언"을 참고한다.

```
//[BEGIN_NODE_BLOCK, 0, func()]
{
    //이너 모듈
    //DECLARE INNER VARIABLE
    String var1 = "";
}
//[END_NODE_BLOCK, 0, func()]
```

EMB Designer - 이너 모듈 - 지역변수 선언 - 소스젠

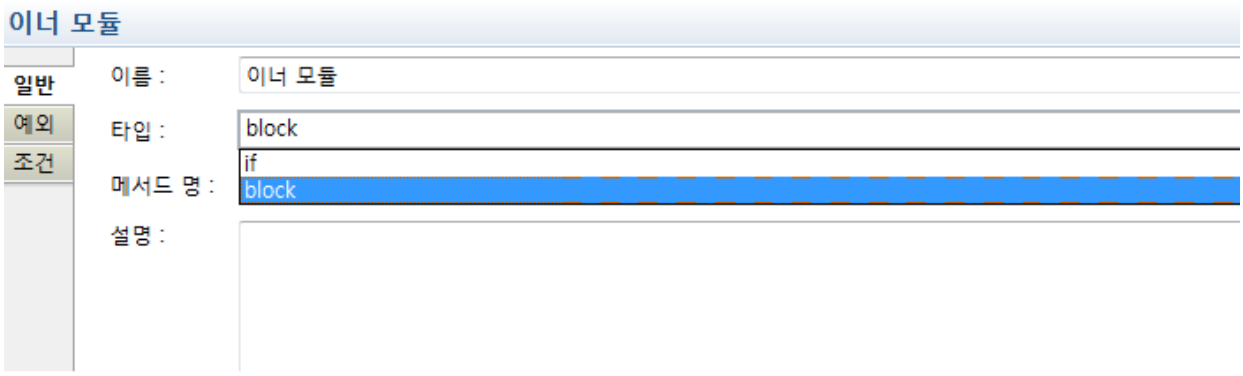
◦ (이너 모듈 타입 선택)

이너 모듈을 'IF' 타입으로 할지, 'BLOCK' 타입으로 할지 선택한다.

- **[Properties]** 탭

이너 모듈을 선택하면 나타나는 화면 하단의 **[Properties]** 탭에서 다음과 같은 특성을 설정할 수 있다.

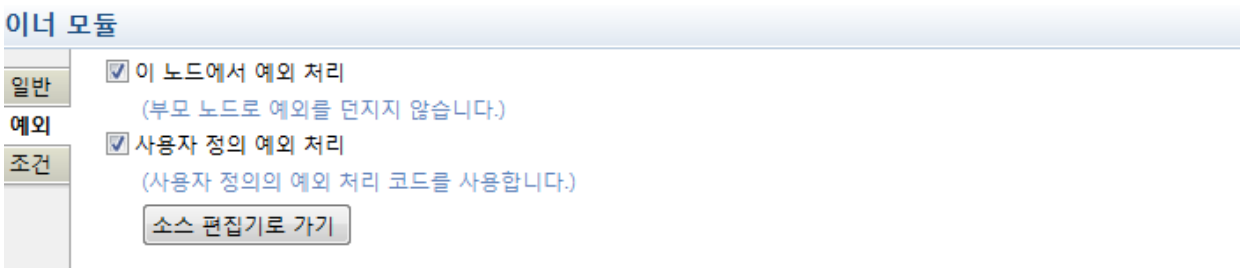
◦ **[일반]** 탭



EMB Designer - 이너 모듈 - [Properties] - [일반] 탭

| 항목 | 설명 |
|---|---|
| 이름 | 이너 모듈의 논리 이름을 부여한다. 기본적으로 '이너 모듈'로 설정된다. |
| 타입 | 이너 모듈의 타입을 지정한다.<br>◦ if : 이너 모듈을 조건 분기형태로 사용한다. 소스 코드에 if 구문이 생성되며 **[Properties] > [조건]** 탭에서 조건을 입력한다. 조건을 입력하지 않으면 기본으로 true 값이 입력된다.<br>◦ block : 이너 모듈을 생성할 때 기본 타입으로 지정된 형태이다. 소스 코드에 블록 영역으로 생성된다. |
| 설명 | 이너 모듈의 내용을 입력한다. |

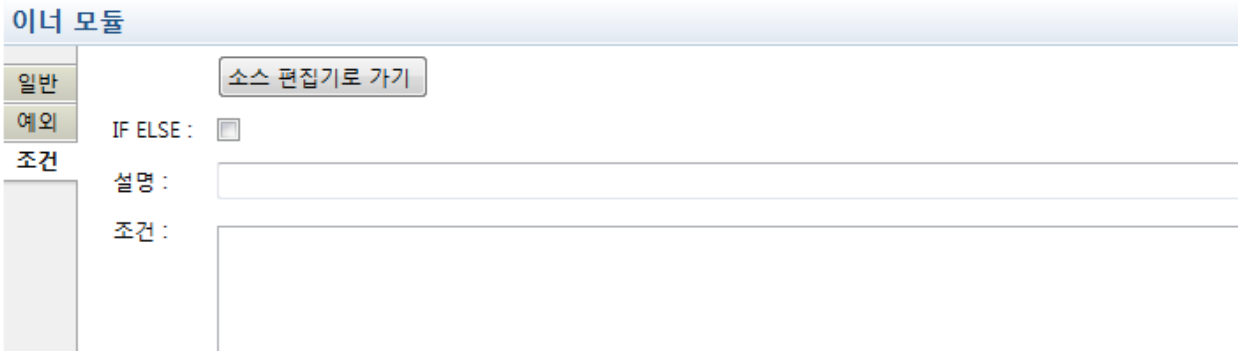
◦ **[예외]** 탭



EMB Designer - 이너 모듈 - [Properties] - [예외] 탭

| 항목 | 설명 |
|---|---|
| 이 노드에서 예외 처리 | try-catch 문이 소스젠에 자동으로 생성되여 부모 노드로 예외를 던지지 않는다. 해당 기능이 체크되어 있어야 **'사용자 정의 예외처리'** 항목이 활성화된다. |
| 사용자 정의 예외 처리 | 소스에서 개발자가 직접 catch 문을 추가할 수 있다. |
| **[소스 편집기로 가기]** | 이너 모듈의 try-catch 해당 소스 라인으로 이동한다. |

◦ **[조건]** 탭



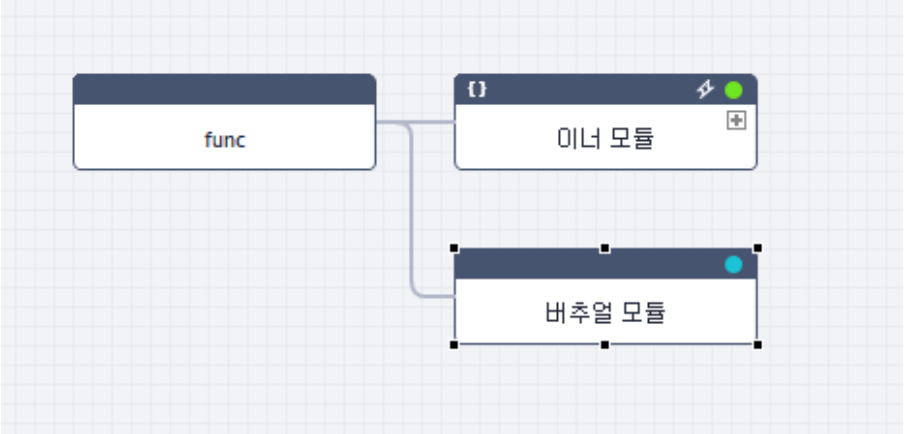
EMB Designer - 이너 모듈 - [Properties] - [조건] 탭

| 항목 | 설명 |
|---|---|
| **[소스 편집기로 가기]** | 이너 모듈의 "if(...)" 해당 소스 라인으로 이동한다. |
| IF ELSE | 여러 개의 if 타입 이너 모듈이 존재하는 경우 제일 마지막 이너 모듈을 else로 설정한다. |
| 설명 | 이너 모듈의 조건절에 대한 내용을 입력한다. |
| 조건 | 이너 모듈의 조건문을 설정할 수 있다. 조건 텍스트 화면에 내용을 입력하면 소스에 반영된다. |

#### 4.3.4.2. 버추얼 모듈

모듈은 개발자가 로직을 삽입하여 개발할 수 있는 영역으로 이너 모듈과 달리 하위 모듈을 가질 수 없다.

**Palette**에서 **[Node] > [버추얼 모듈]**을 선택한 후 디자인 영역을 클릭하면 버추얼 모듈이 삽입된다. 버추얼 모듈을 클릭하면 원하는 위치로 이동할 수 있다.



EMB Designer - 버추얼 모듈 - 디자인 영역

다음은 버추얼 모듈 디자인 에디터에 생성한 소스이다.

```
//[BEGIN_NODE_BLOCK, 1, func()]
{
    //버추얼 모듈
     //[BEGIN_VIRTUAL_CODE_BLOCK, 1, func()]
     {

     }
     //[END_VIRTUAL_CODE_BLOCK, 1, func()]

}
//[END_NODE_BLOCK, 1, func()]
```

EMB Designer - 버추얼 모듈- 소스젠

- **버추얼 모듈 속성**

  버추얼 모듈을 클릭하면 아래와 같이 아이콘이 보이며 각각의 아이콘은 기능을 갖고 있다.


EMB Designer - 버추얼 모듈 - 속성

  - (삭제)

    버추얼 모듈을 삭제한다.

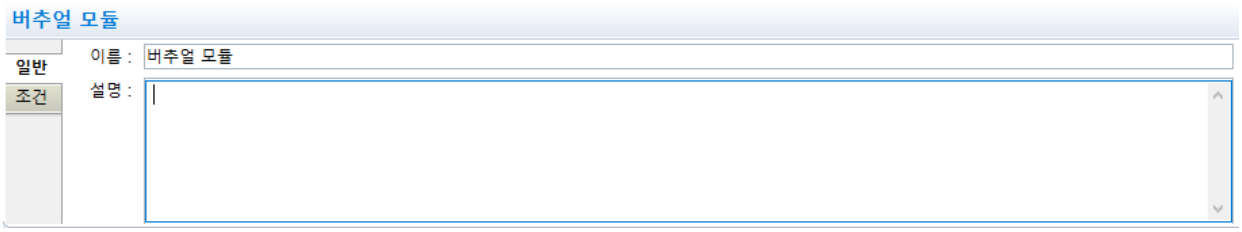
  - (주석 처리)

    버추얼 모듈에 작성된 로직을 주석 처리한다.

- **[Properties]** 탭

  버추얼 모듈을 선택하면 나타나는 화면 하단의 **[Properties]** 탭에서 다음과 같은 특성을 설정할 수 있다.

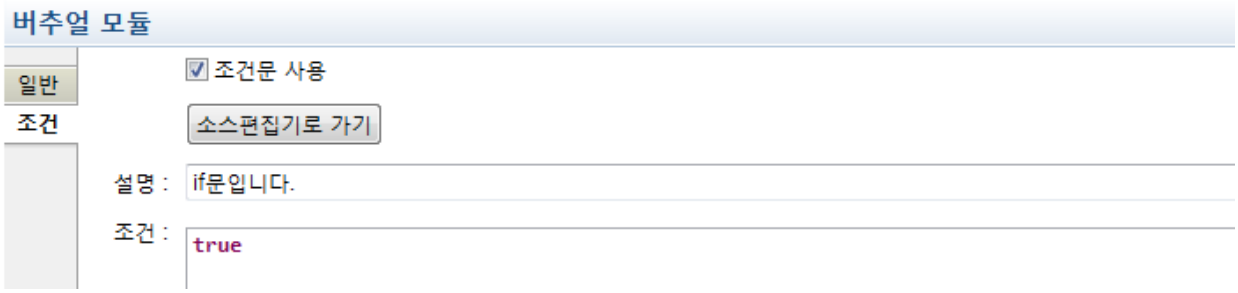
  - **[일반]** 탭




EMB Designer - 버추얼 모듈 - [Properties] - [일반] 탭

| 항목 | 설명 |
|---|---|
| 이름 | 버추얼 모듈의 논리 이름을 부여한다. 기본적으로 '버추얼 모듈'로 설정된다. |
| 설명 | 버추얼 모듈의 내용을 입력한다. |

  - **[조건]** 탭



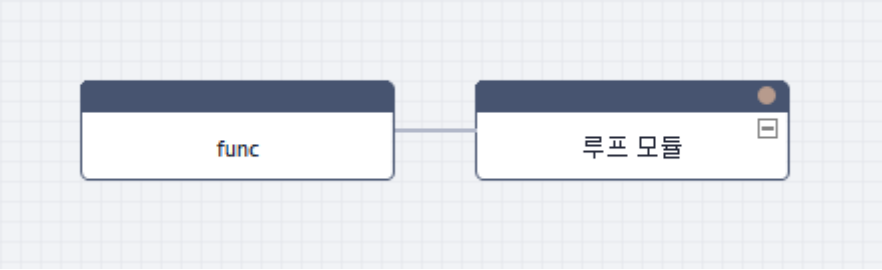
EMB Designer - 버추얼 모듈 - [Properties] - [조건] 탭

| 항목 | 설명 |
|---|---|
| 조건문 사용 | 버추얼 모듈에서 조건문 사용을 위한 설정으로 체크할 때에만 Properties 조건 특성이 적용된다. |
| **[소스 편집기로 가기]** | 버추얼 모듈의 "if(...)" 해당 소스 라인으로 이동한다. |
| 설명 | 버추얼 모듈의 조건절에 대한 설명을 입력할 수 있다. |
| 조건 | 버추얼 모듈의 조건문을 설정할 수 있다. 조건 텍스트 화면에 내용을 입력하면 소스에 반영된다. |

#### 4.3.4.3. 루프 모듈

EMB Designer에서 루프 모듈은 반복문 작성을 위한 모듈로 하위 모듈을 가질 수 있다.

**Palette**에서 **[Node] > [루프 모듈]**을 선택한 후 디자인 영역을 클릭한다. 루프 모듈 삽입한 후 개발 로직을 추가할 경우 다음과 같이 버추얼 모듈 등을 이용하여 개발 영역을 추가한다. 루프 모듈을 클릭한 후 원하는 위치로 이동할 수 있다.



EMB Designer - 루프 모듈 - 디자인 영역

다음은 루프 모듈 디자인 에디터에 생성한 소스이다.

```
//[BEGIN_NODE_BLOCK, 0, func()]
{
    //루프 모듈
    //[BEGIN_LOOP_MODULE_PREPROCESSING, 0, func()]

    //[END_LOOP_MODULE_PREPROCESSING, 0, func()]
    //[BEGIN_LOOP_MODULE_POSTPROCESSING, 0, func()]

    //[END_LOOP_MODULE_POSTPROCESSING, 0, func()]

}
//[END_NODE_BLOCK, 0, func()]
```
EMB Designer - 루프 모듈 - 소스젠

- **루프 모듈 노드 속성**

  루프 모듈을 클릭하면 아래와 같이 아이콘이 보이며 각각의 아이콘은 기능을 갖고 있다. 삭제 및 주석 처리 등 노드 속성은 타 노드 속성과 동일하다.


EMB Designer - 루프 노드 - 속성

  - (삭제)

    루프 모듈을 삭제한다.

  - (주석 처리)

    루프 모듈을 주석 처리한다.

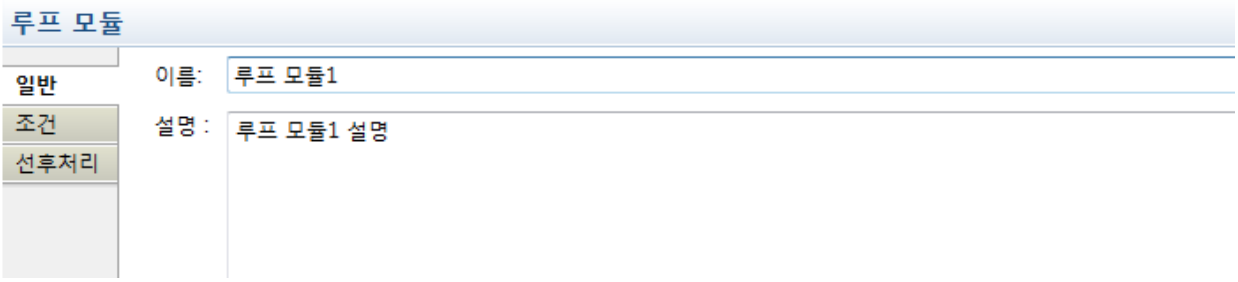
  - (선처리 이동)/ (후처리 이동)

    루프 모듈의 선/후처리 소스 영역으로 이동한다.

- **[Properties]** 탭

  루프 모듈을 선택하면 나타나는 화면 하단의 **[Properties]** 탭에서 다음과 같은 특성을 설정할 수 있다.

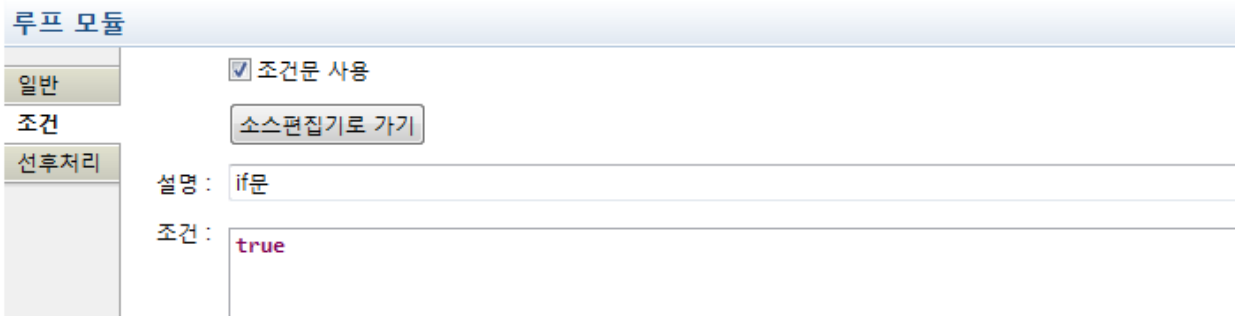
  - **[일반]** 탭




EMB Designer - 루프 모듈 - [Properties] - [일반] 탭

| 항목 | 설명 |
|---|---|
| 이름 | 루프 모듈의 논리 이름을 부여한다. 기본적으로 '루프 모듈'로 설정된다. |
| 설명 | 루프 모듈의 내용을 입력한다. |

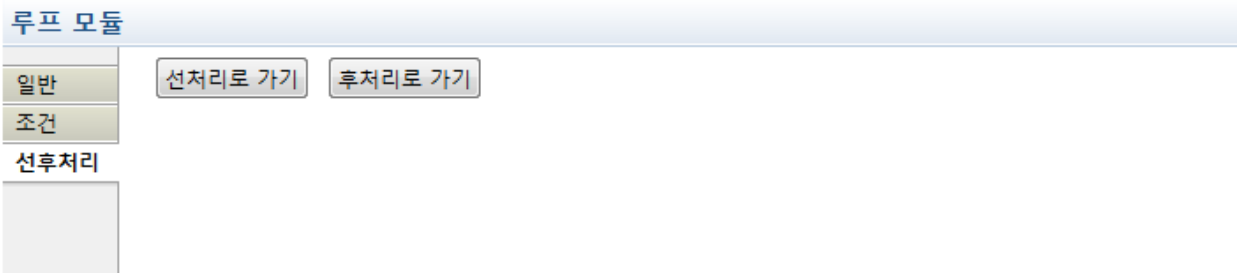
  - **[조건]** 탭



EMB Designer - 루프 모듈 - [Properties] - [조건] 탭

| 항목 | 설명 |
|---|---|
| 조건문 사용 | 루프 모듈에서 조건문 사용을 위한 설정으로 체크할 때에만 Properties 조건 특성이 적용된다. |
| **[소스 편집기로 가기]** | 루프 모듈의 "if(...)" 해당 소스 라인으로 이동한다. |
| 설명 | 루프 모듈의 조건절에 대한 설명을 입력할 수 있다. |
| 조건 | 루프 모듈의 조건문을 설정할 수 있다. 조건 텍스트 화면에 내용을 입력하면 소스에 반영된다. |

◦ **[선후처리]** 탭



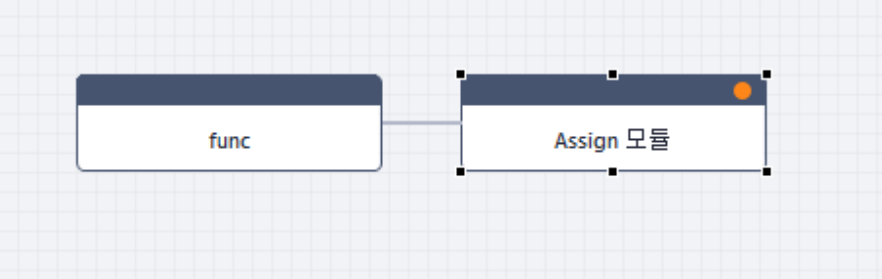
EMB Designer - 루프 모듈 - [Properties] - [선후처리] 탭

| 항목 | 설명 |
|---|---|
| **[선처리로 가기]** | 루프 모듈 "BEGIN_LOOP_MODULE_PREPROCESSING" 소스 영역으로 이동한다. |
| **[후처리로 가기]** | 루프 모듈 "BEGIN_LOOP_MODULE_POSTPROCESSING" 소스 영역으로 이동한다. |

#### 4.3.4.4. Assign 모듈

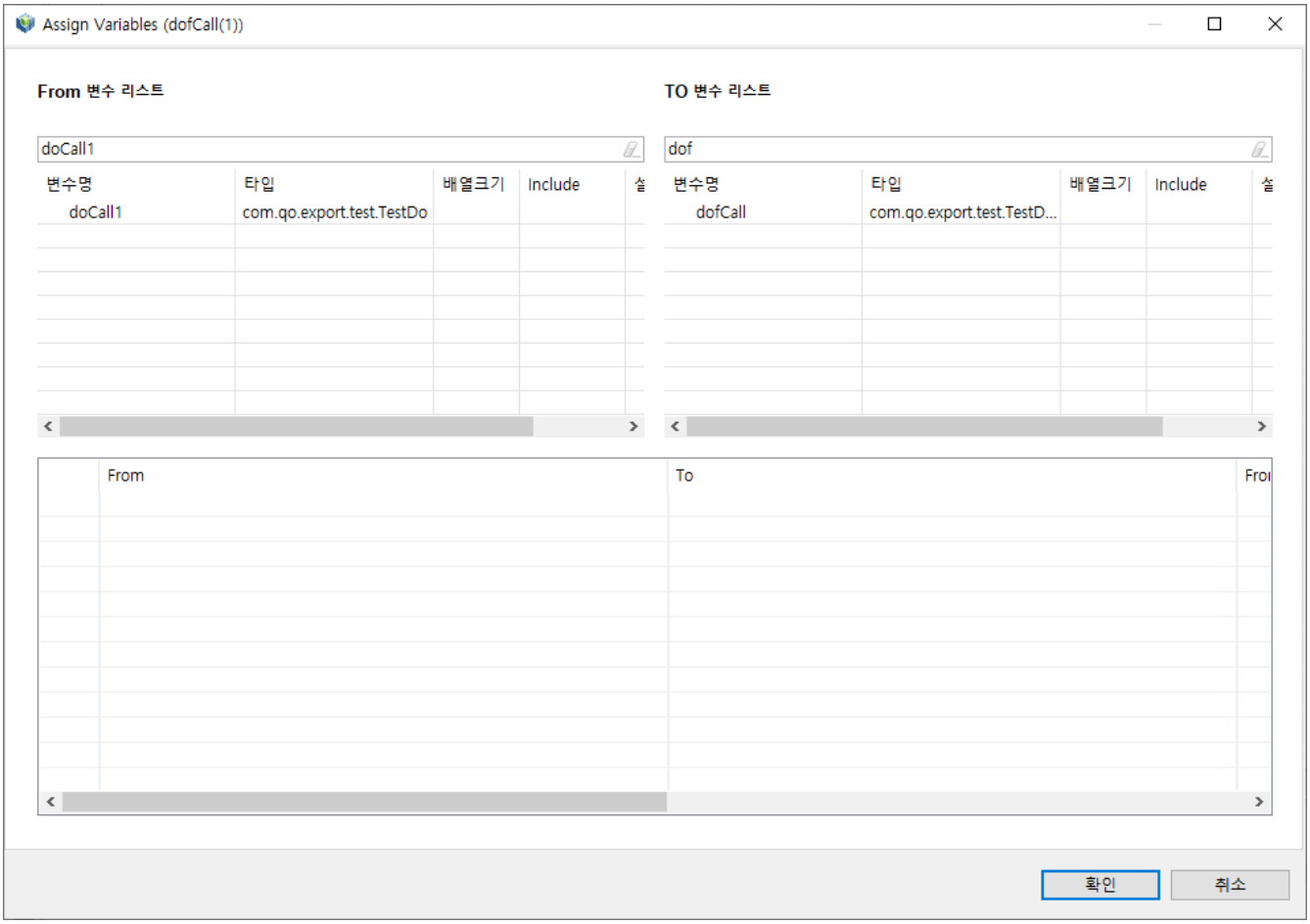
EMB Designer에서 Assign 모듈은 변수 간의 값을 할당할 수 있는 모듈이다. 변수 매핑 툴을 제공한다.

**Palette**에서 **[Node] > [Assign 모듈]**을 클릭해서 디자인 영역을 클릭해서 Assign 모듈을 삽입한다. Assign 모듈을 클릭한 후 원하는 위치로 이동할 수 있다.




EMB Designer - Assign 모듈

Assign 모듈을 더블클릭하면 나타나는 **변수 Assign 화면**에서 변수를 매핑한 후 저장한다. 매핑할 때 편의를 위해 From/To 변수 리스트 내역의 Filter 기능을 제공한다.




EMB Designer - 변수 Assign

- **Assign 모듈 노드 속성**

  Assign 모듈을 클릭하면 아래와 같이 아이콘이 보이며 각각의 아이콘은 기능을 갖고 있다.

  EMB Designer - Assign 모듈 - 노드 속성

  - (삭제)

Assign 모듈을 삭제한다.

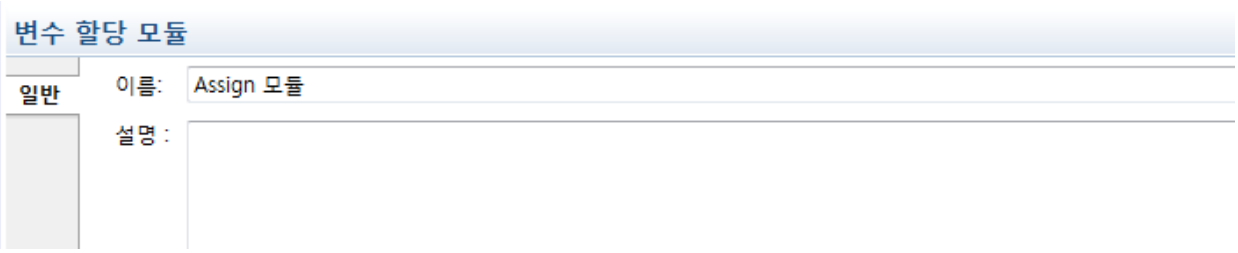
-  (주석 처리)

  Assign 모듈을 주석 처리한다.

- **[Properties]** 탭

Assign 모듈을 선택하면 나타나는 화면 하단의 **[Properties]** 탭에서 다음과 같은 특성을 설정할 수 있다.

- **[일반]** 탭

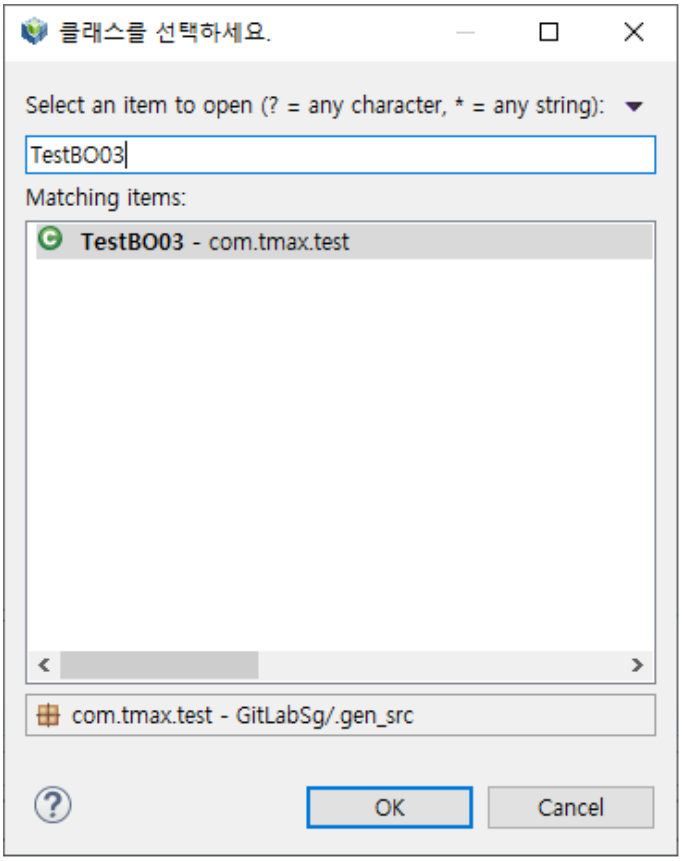



EMB Designer - Assign 모듈 - [Properties] - [일반] 탭

| 항목 | 설명 |
|---|---|
| 이름 | Assign 모듈의 논리 이름을 부여한다. 기본적으로 'Assgin 모듈'로 설정된다. |
| 설명 | Assign 모듈의 내용을 입력한다. |

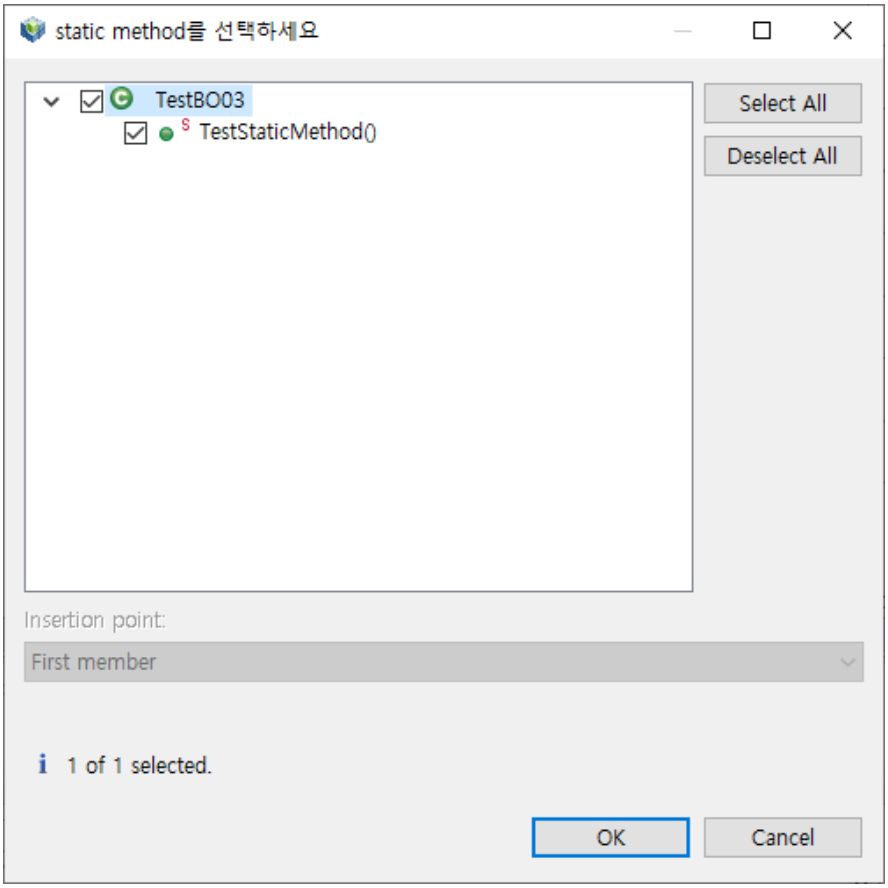
### 4.3.4.5. static method 모듈

**Palette**에서 **[Node] > [static method 모듈]**을 클릭하면 클래스를 선택할 수 있는 화면이 나타난다. Static Method를 포함하는 BO를 조회한 후 **[OK]** 버튼을 클릭한다.

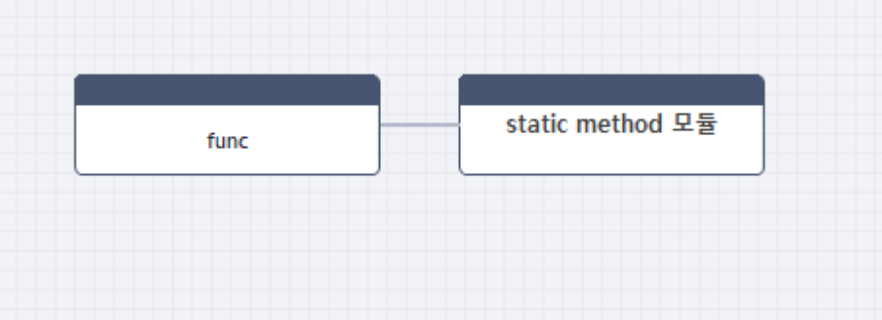



EMB Designer - Class 조회

선택한 BO에 포함된 static method 내역이 표시된다. 적용할 메소드 내역을 선택한 후 **[OK]** 버튼을 클릭하면 EMB Designer에 Static method 모듈이 추가된다.



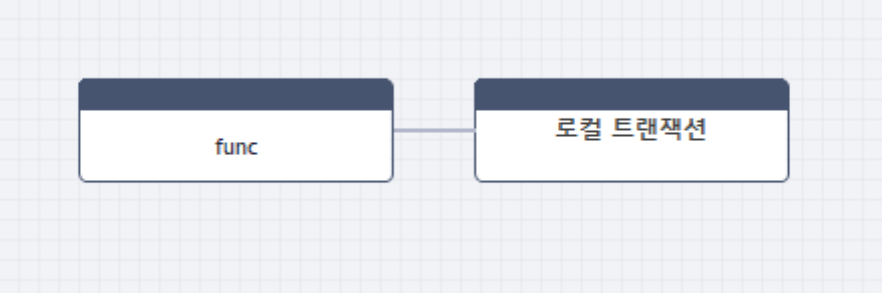

EMB Designer - static method 선택



EMB Designer - static method 모듈 추가

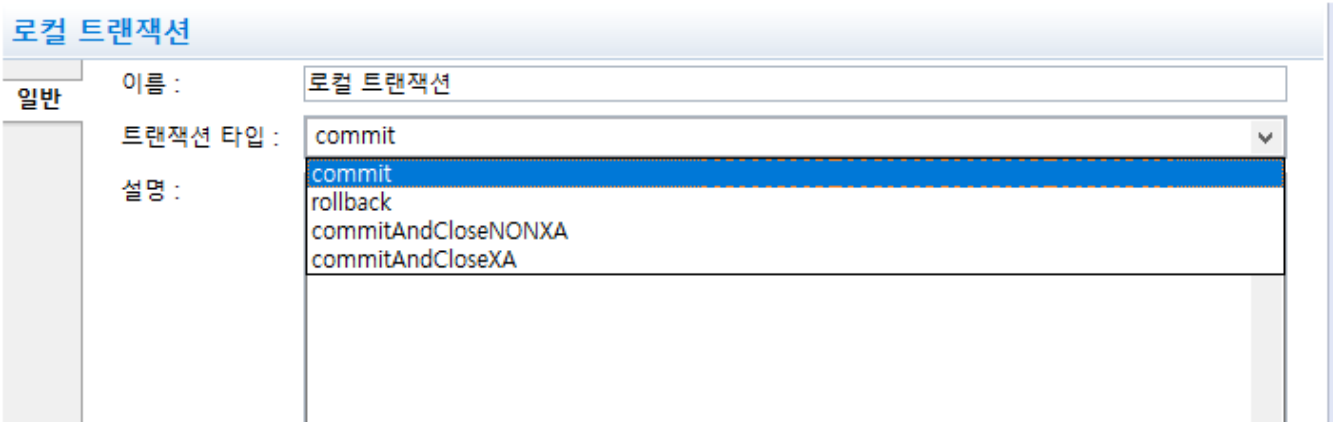
#### 4.3.4.6. 로컬 트랜잭션

EMB Designer에서 Assign 모듈은 로직 중간에 Commit, Rollback 등의 트랜잭션 작업을 개발자가 할 수 있는 기능이다. **Palette**에서 **[Node] > [로컬 트랜잭션]**을 클릭해서 디자인 영역을 클릭해서 로컬 트랜잭션을 삽입한다. 로컬 트랜잭션을 클릭한 후 원하는 위치로 이동할 수 있다.



EMB Designer - 로컬 트랜잭션

로컬 트랜잭션을 선택하면 나타나는 화면 하단의 **[Properties]** 탭에서 다음과 같은 특성을 설정할 수 있다.



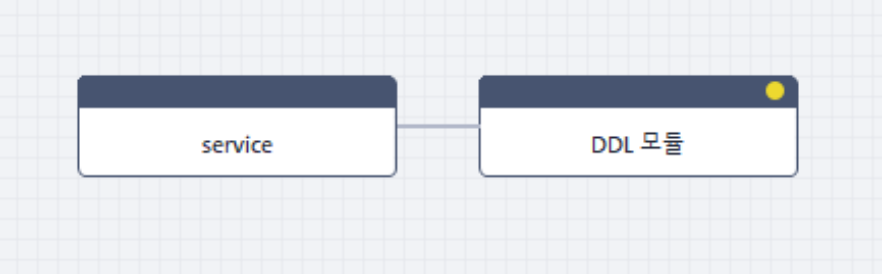
EMB Designer - 로컬 트랜잭션 - [Properties] - [일반] 탭

| 항목 | 설명 |
| --- | --- |
| 트랜잭션 타입 | 로컬 트랜잭션의 타입을 설정한다.<br>◦ commit : Commit 처리한다.<br>◦ rollback : Rollback 처리한다.<br>◦ commitANDCloseNONXA : NonXA 트랜잭션의 경우 Commit과 Close한다.<br>◦ commitANDCloseXA : XA 트랜잭션의 경우 Commit과 Close한다. |

#### 4.3.4.7. DDL 모듈

EMB Designer에서 DDL을 정의하여 테이블 생성 및 삭제를 할 수 있는 SQL을 수행할 수 있다.

**Palette**에서 **[Node] > [DDL 모듈]**을 클릭하고 디자인 영역을 클릭해서 DDL 모듈을 삽입한다. 디자인 화면에서 DDL 노드를 클릭한 후 원하는 위치로 이동할 수 있다.




EMB Designer - DDL 모듈 - 디자인 영역

다음은 DDL 모듈 디자인 에디터에 생성한 소스이다.

```
{
//DDL 모듈
/**
 * Test Table을 생성한다.
 */

//[BEGIN_DDL_MODULE_BLOCK, 2, service(So001In)]
 DataDefinitionExecutor.getInstance("tibero").execute("CREATE TALE TEST("
+ " TEST_COL CHAR(10)"
+ ")");
//[END_DDL_MODULE_BLOCK, 2, service(So001In)]
}
```

EMB Designer - DDL 모듈 - 소스젠

- **DDL 모듈 속성**

  getReply 노드를 클릭하면 아래와 같이 아이콘이 보이며 각각의 아이콘은 기능을 갖고 있다.

EMB Designer - DDL 모듈 - 속성

- (삭제)

  DDL 노드를 삭제한다.

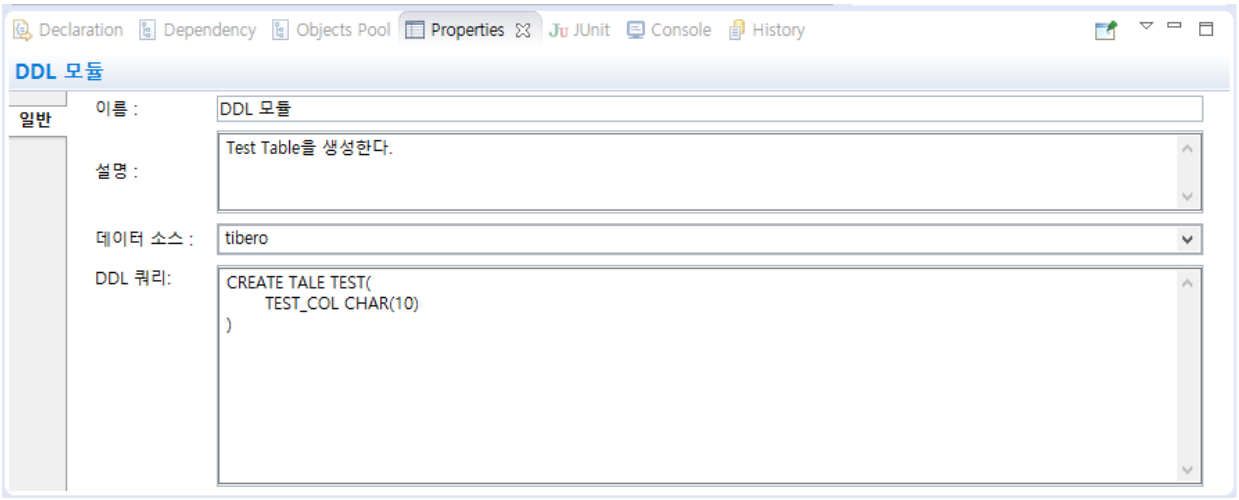
- (주석 처리)

  DDL 노드를 주석 처리한다.

- **[Properties]** 탭

  DDL 노드를 선택하면 나타나는 화면 하단의 **[Properties]** 탭에서 다음과 같은 특성을 설정할 수 있다.

  - **[일반]** 탭



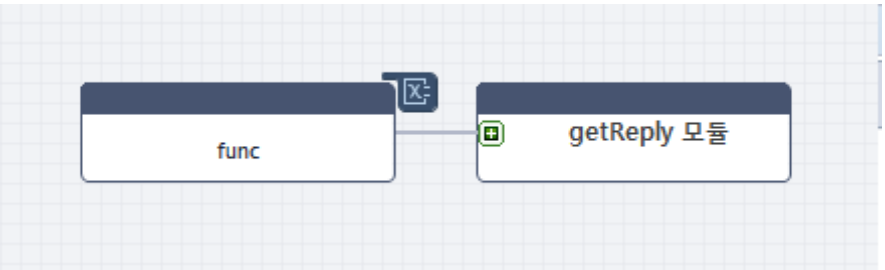
EMB Designer - DDL 모듈 - [Properties] - [일반] 탭

| 항목 | 설명 |
|---|---|
| 이름 | 모듈의 이름을 지정한다. |
| 설명 | 모듈의 설명을 넣는다. |
| 데이터 소스 | DDL이 수행된 데이터소스를 선택한다. |
| DDL 쿼리 | DDL SQL을 입력한다. 단일 DDL SQL만 입력 가능하다. |

### 4.3.4.8. getReply 모듈

EMB Designer에서 getReply 노드는 다른 메소드의 응답을 기다리는 모듈이다.

**Palette**에서 **[Node] > [getReply]**을 클릭하고 디자인 영역을 클릭해서 getReply 노드를 삽입한다. 디자인 화면에서 getReply 노드를 클릭한 후 원하는 위치로 이동할 수 있다.




EMB Designer - getReply 모듈 - 디자인 영역

다음은 getReply 모듈 디자인 에디터에 생성한 소스이다.

```
    //[BEGIN_GET_REPLY_BLOCK, 0, func()]
    {

     //[BEGIN_POST_ASSIGN_BLOCK, 0, func()]
    //[END_POST_ASSIGN_BLOCK, 0, func()]
    }
    //[END_GET_REPLY_BLOCK, 0, func()]
//[BEGIN_AFTER_CODE, 0, func()]

//[END_AFTER_CODE, 0, func()]
```

EMB Designer - getReply 모듈 - 소스젠

- **getReply 모듈 속성**

  getReply 노드를 클릭하면 아래와 같이 아이콘이 보이며 각각의 아이콘은 기능을 갖고 있다.

  EMB Designer - getReply 모듈 - 속성

  - (삭제)

    getReply 노드를 삭제한다.

  - (주석 처리)

    getReply 노드를 주석 처리한다.

  - (결과값 셋팅)

    결과값을 설정한다. Assign 모듈과 기능이 유사하다.

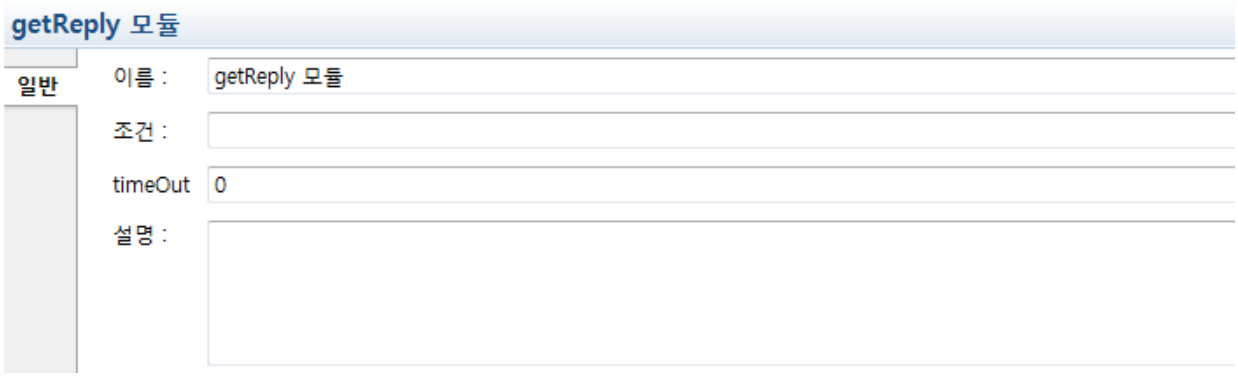
  - (호출 후 코드)

    getReply 모듈의 호출 후 코드 소스 영역으로 이동한다.

- **[Properties]** 탭

  getReply 노드를 선택하면 나타나는 화면 하단의 **[Properties]** 탭에서 다음과 같은 특성을 설정할 수 있다.

◦ **[일반]** 탭




EMB Designer - getReply 모듈 - [Properties] - [일반] 탭

| 항목 | 설명 |
|---|---|
| 이름 | getReply 노드의 논리 이름을 부여한다. 기본적으로 'getReply 모듈'로 설정된다. |
| 조건 | getReply 노드의 조건을 설정한다. |
| timeOut | 응답 시간 제한을 설정한다. (기본값: '0') |
| 설명 | getReply 노드의 내용을 입력한다. |

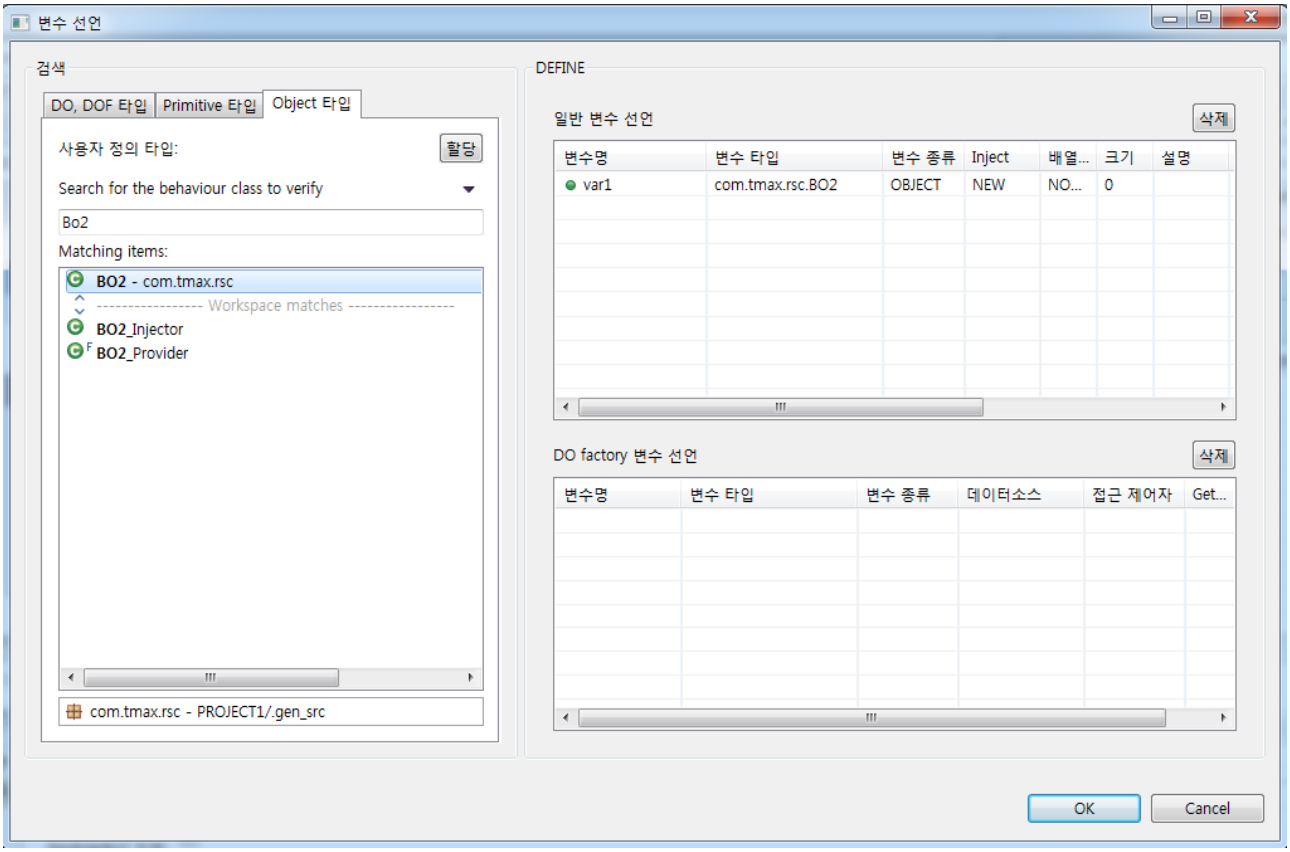
#### 4.3.4.9. BizObject 변수

BO 변수는 변수 선언에서 Object type으로 BO로 선언할 경우 **Palette**에서 BO 레퍼런스 변수를 확인할 수 있다.

다음은 BO 변수를 생성하는 과정에 대한 설명이다.

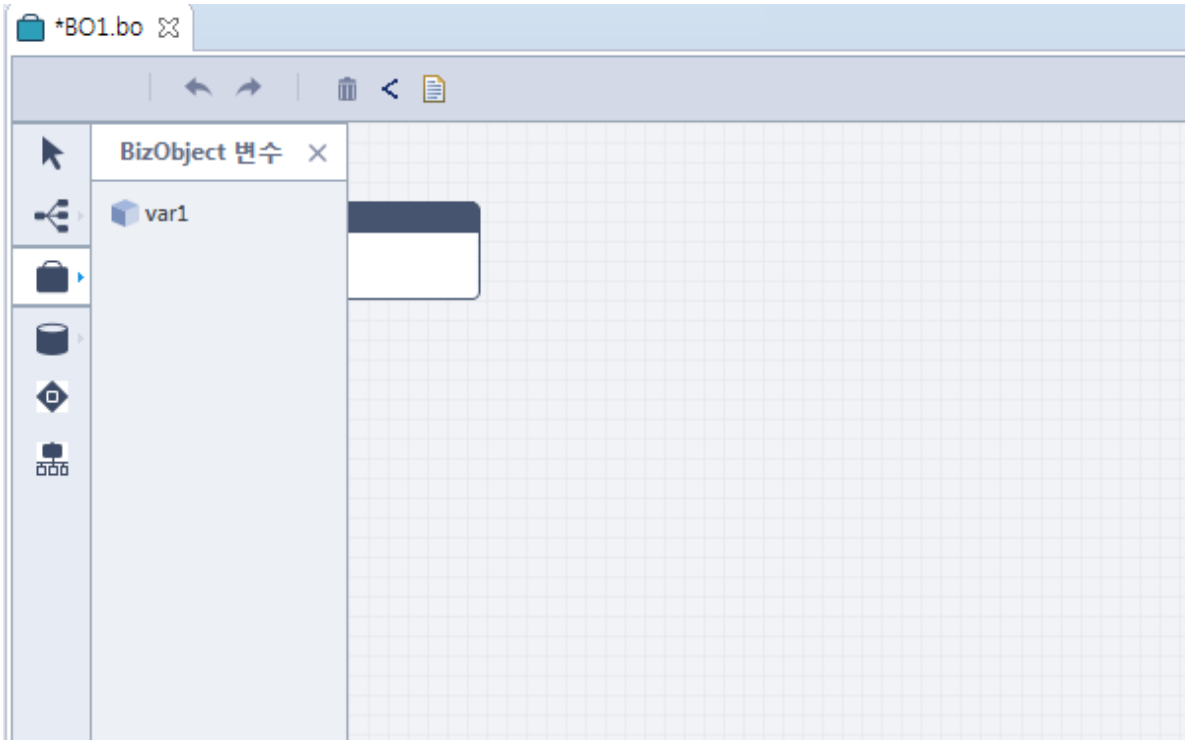
1. BO 디자인 에디터에서 + (**[멤버 변수 설정]**) 아이콘을 클릭하면 **변수 선언 화면**이 나타난다.

   해당 화면에서 Object Type을 설정한 후 생성한 Object Type을 검색한다. 검색된 Object Type을 더블클릭하면 아래와 같이 변수가 생성된다.



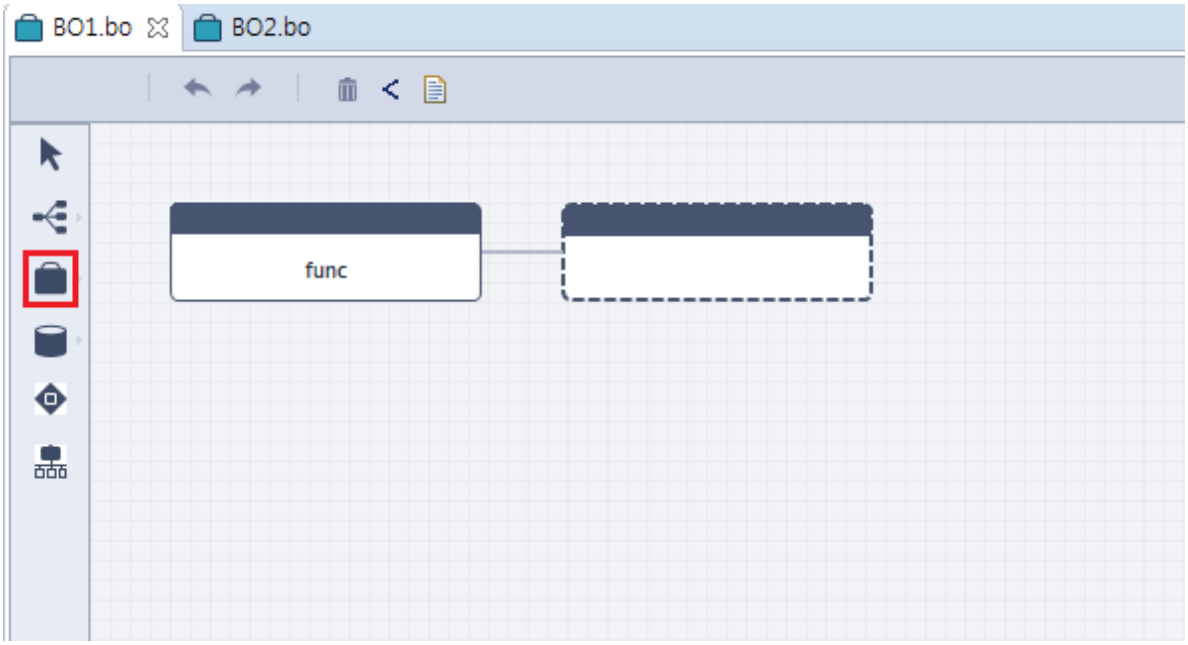

EMB Designer - BO 변수 선언

2. BO 변수를 선언한 후 Palette를 확인하면 BO 모듈이 생성된 것을 확인할 수 있다.



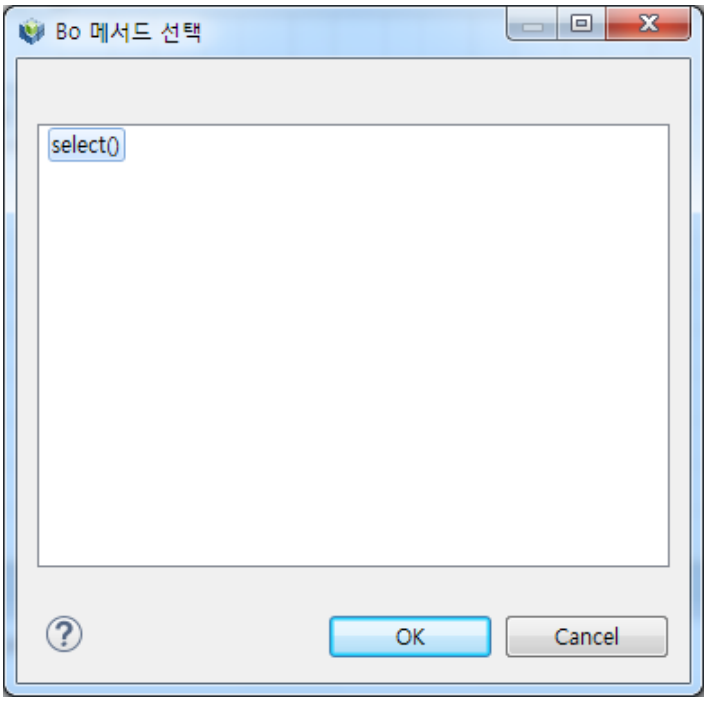

EMB Designer - BO 변수 모듈 - Palette

3. **Palette**에서  (**[BizObject 변수]**)를 클릭한 후 디자인 영역을 클릭한다.



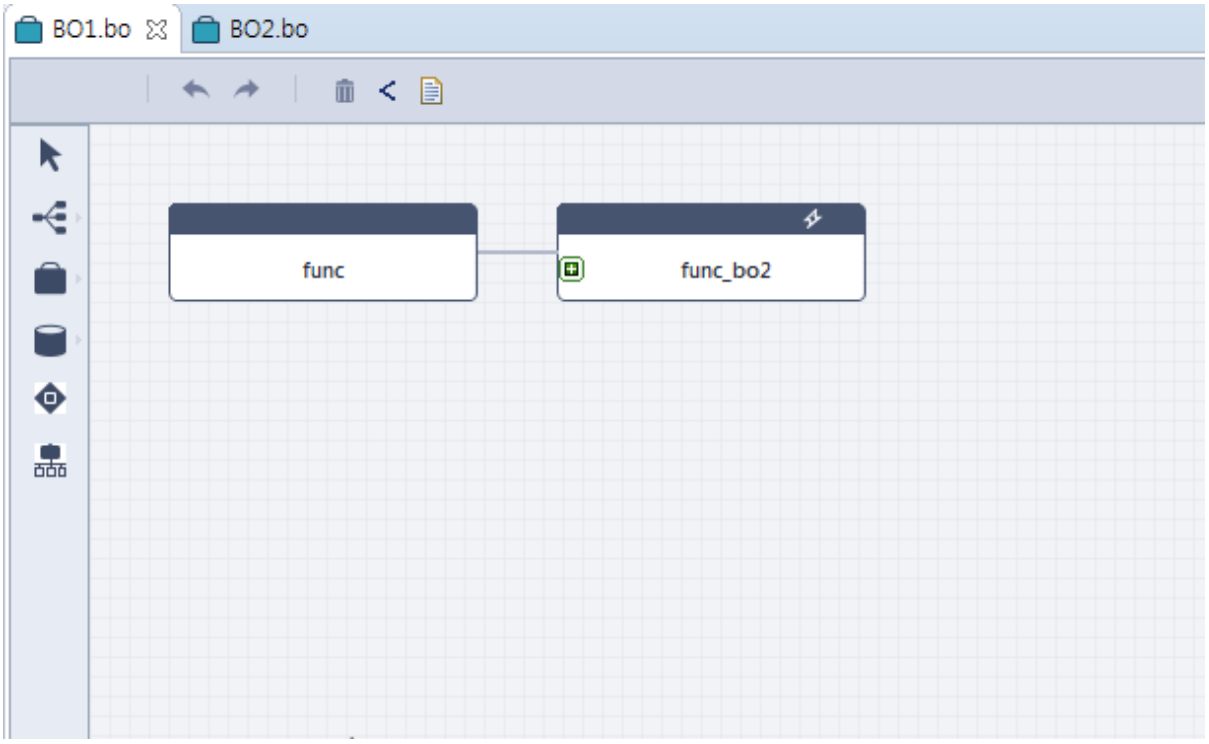

EMB Designer - BO 변수 모듈 - 디자인

4. 클릭하면 해당 Object Type의 **BO 메소드 선택 화면**이 나타난다.




EMB Designer - BO 메소드 선택 화면

5. BO 메소드를 선택하여 BO 변수를 디자인 에디터에 삽입한다. BO 변수 모듈을 클릭한 후 원하는 위치로 이동할 수 있다.




EMB Designer - BO 변수 - 디자인 영역

BO 변수를 디자인 에디터에 추가하면 아래와 같이 자동 소스가 생성된다.

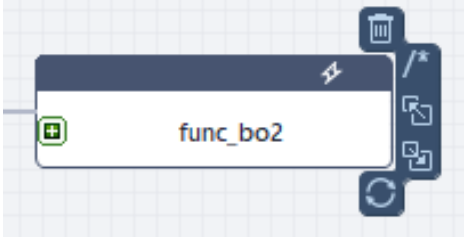
```
public void func() throws Throwable
{
//[BEGIN_NODE_BLOCK, , func()]
{

    //[BEGIN_NODE_BLOCK, 0, func()]
    {
        //[BEGIN_PRE_ASSIGN_BLOCK, 0, func()]

        //[END_PRE_ASSIGN_BLOCK, 0, func()]
        var1.get().func_bo2();
        //[BEGIN_POST_ASSIGN_BLOCK, 0, func()]

        //[END_POST_ASSIGN_BLOCK, 0, func()]
    }
    //[END_NODE_BLOCK, 0, func()]

}
//[END_NODE_BLOCK, , func()]
}
```

EMB Designer - BO 변수 - 소스젠

다음은 BO 변수 속성과 Properties에 대한 설명이다.

- **BO 변수 속성**

  BO 변수 모듈을 클릭하면 아래와 같이 아이콘이 보이며 각각의 아이콘은 기능을 갖고 있다.



EMB Designer - BO 변수 - 속성

- (삭제)

  BO 변수 모듈을 삭제한다.

- (주석 처리)

  BO 변수 모듈을 주석 처리한다.

- (선할당 처리)

  BO 메소드를 호출하기 전 set할 값을 설정한다. Assign 모듈과 기능이 유사하다.

- (후할당 처리)

  BO 메소드를 호출한 후 get할 값을 설정한다. Assign 모듈과 기능이 유사하다.

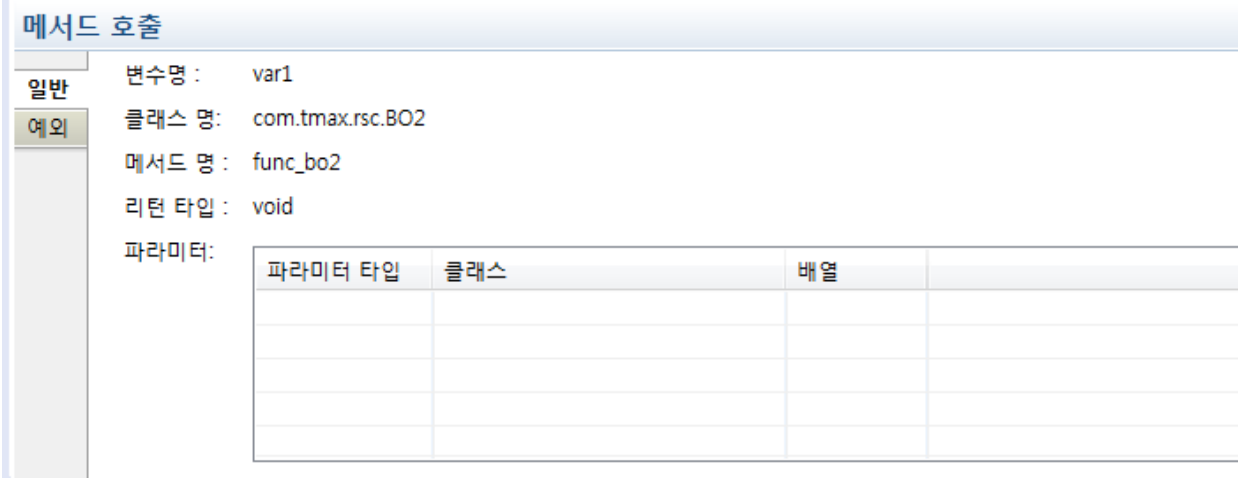
- (모듈 재구성)

  호출된 BO 메소드의 인자가 변경된 경우에 변경된 인자에 맞게 호출 모듈을 재구성한다.

- **[Properties]** 탭

BO 변수 모듈을 선택한 후 나타나는 화면 하단의 **[Properties]** 탭에서 다음과 같은 특성을 설정할 수 있다.

- **[일반]** 탭



EMB Designer - BO 변수 - [Properties] - [일반] 탭

| 항목 | 설명 |
|---|---|
| 변수명 | Object type을 선언할 때 정의한 변수명을 보여준다. |

| 항목 | 설명 |
|---|---|
| 클래스명 | 호출된 BO의 실제 리소스를 확인할 수 있다. |
| 메소드명 | 호출된 BO의 사용된 메소드를 확인할 수 있다. |
| 리턴 타입 | 호출된 BO 메소드의 리턴 타입을 확인할 수 있다. |
| 파라미터 | 호출된 BO 메소드의 agrs 내용을 확인할 수 있다. |

◦ **[예외]** 탭


EMB Designer - BO 변수 - [Properties] - [예외] 탭

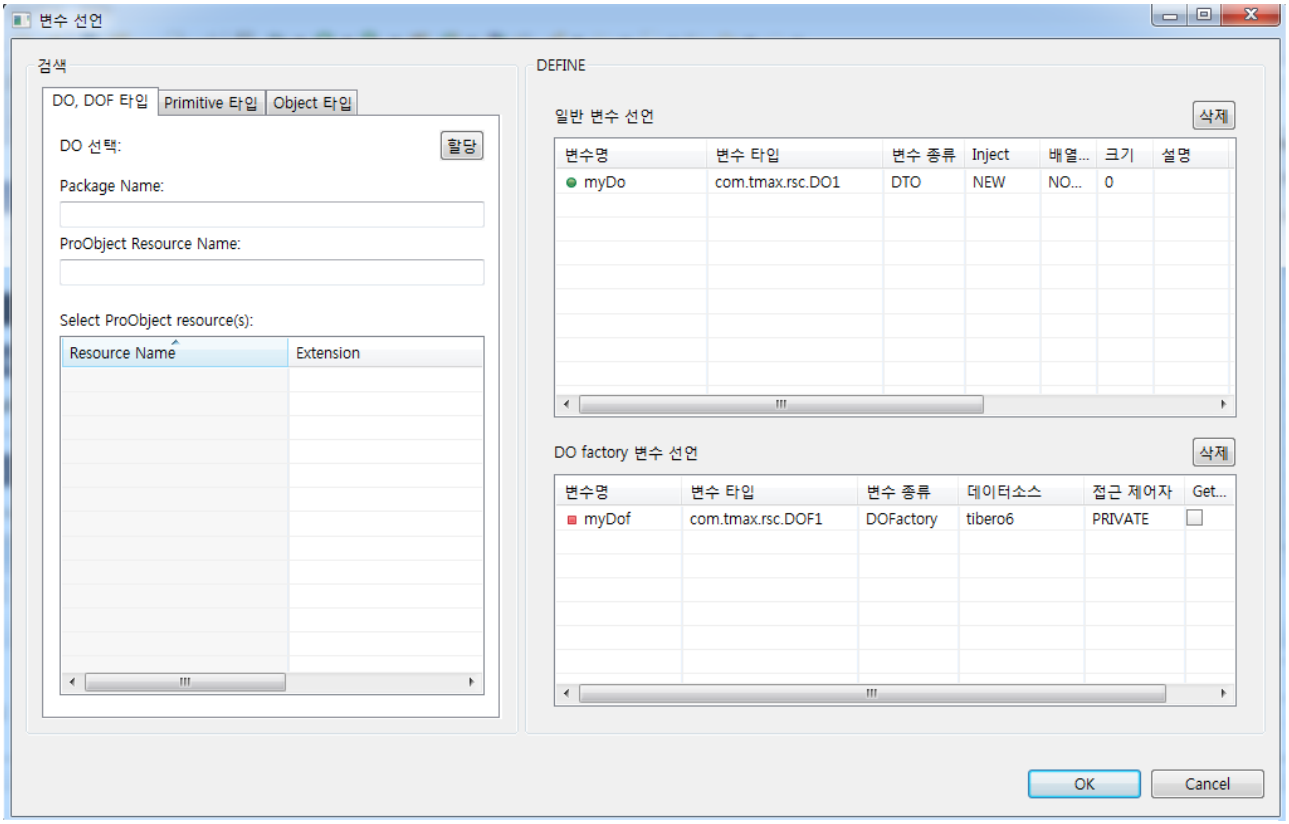
| 항목 | 설명 |
|---|---|
| 이 노드에서 예외 처리 | try-catch 문이 소스젠에 자동으로 생성되어 부모 노드로 예외를 발생하지 않는다. 해당 항목이 체크되어 있어야 **'사용자 정의 예외처리'** 항목이 활성화된다. |
| 사용자 정의 예외 처리 | 소스에서 개발자가 직접 catch 문을 추가할 수 있다. |
| **[소스 편집기로 가기]** | 모듈의 try-catch 해당 소스 라인으로 이동한다. |

### 4.3.4.10. DO Factory 변수 (DB)

DO Factory 변수는 변수 정의에서 DOF type으로 DOF를 변수 선언할 경우 **Palette**에서 DOF 레퍼런스 변수를 확인할 수 있다.

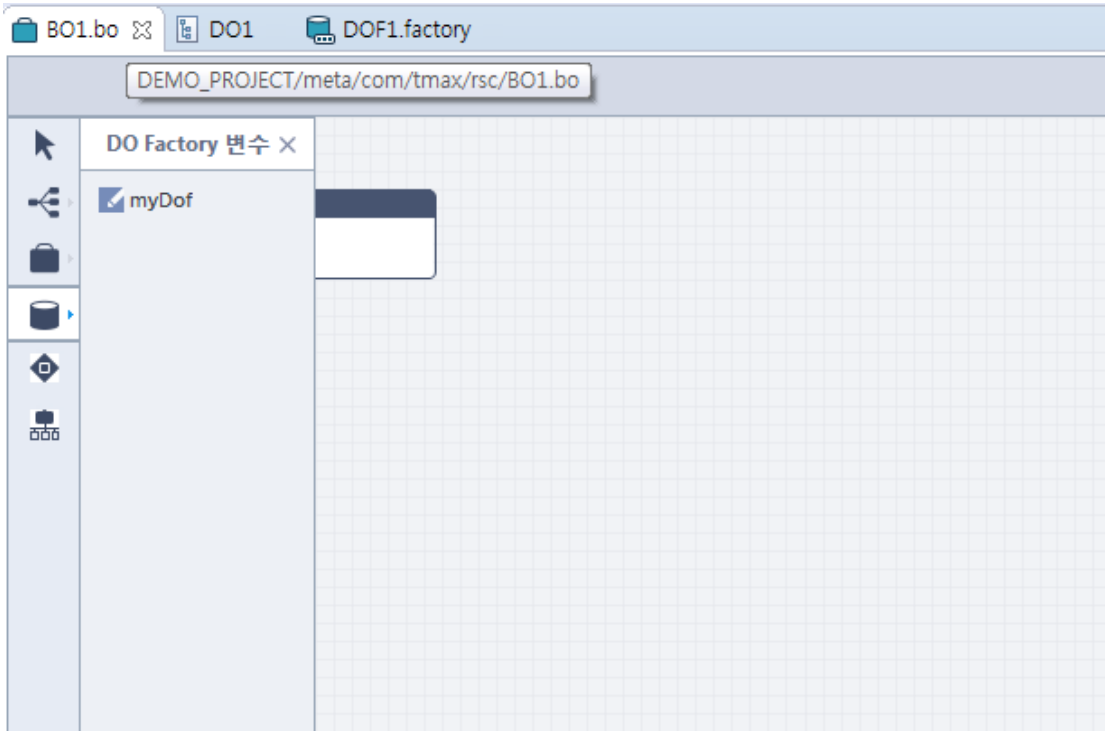
다음은 DB DO Factory Resource에 변수를 추가하는 과정에 대한 설명이다.

1. + (**[멤버 변수 설정]**) 아이콘을 클릭하면 **변수 정의 화면**이 나타난다. 해당 화면에서 DO Type을 설정한 후 생성한 DO Type을 검색한다. 검색된 DO Type을 더블클릭하면 아래와 같이 변수가 생성된다.



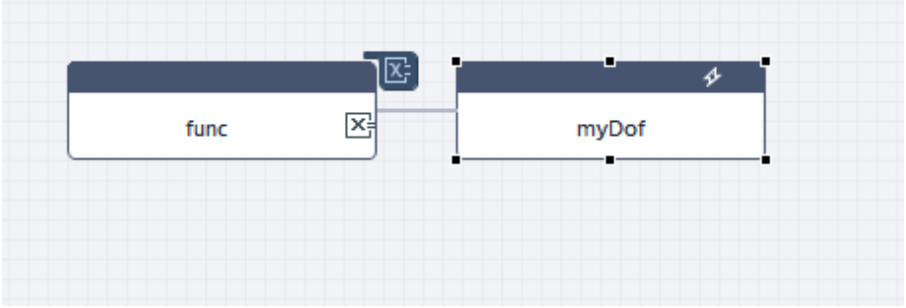

EMB Designer - DB DOF 변수 - 변수 정의

2. **변수 정의 화면**을 저장한 후 Palette에서 DOF 변수가 생성된 것을 확인할 수 있다.




EMB Designer - DB DOF 변수 - Palette

3. **Palette**에서  (**[DO Factory 변수]**)를 클릭한 후 디자인 영역을 클릭한다. DOF 변수 모듈을 클릭한 후 원하는 위치로 이동할 수 있다.




EMB Designer - DB DOF 변수 - 디자인 영역

DOF 변수를 디자인 에디터에 생성하면 다음과 같이 자동 소스가 생성된다.

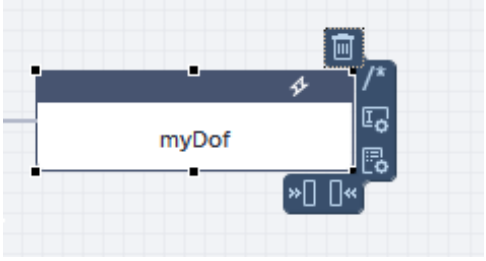
```
public void func() throws Throwable
{
//[BEGIN_NODE_BLOCK, , func()]
{
    //DECLARE ENTRY VARIABLES
    com.tmax.rsc.DO1 doCall1 = new com.tmax.rsc.DO1();
    int resultCnt1 = 0;

    //[BEGIN_NODE_BLOCK, 0, func()]
    {
        //myDof
        //[BEGIN_BEFORE_CODE, 0, func()]

        //[END_BEFORE_CODE, 0, func()]

        //[BEGIN_PRE_ASSIGN_BLOCK, 0, func()]
        //[END_PRE_ASSIGN_BLOCK, 0, func()]

        myDof.setAutoCommit(true);

        //[BEGIN_POST_ASSIGN_BLOCK, 0, func()]
        //[END_POST_ASSIGN_BLOCK, 0, func()]

        //[BEGIN_AFTER_CODE, 0, func()]

        //[END_AFTER_CODE, 0, func()]
    }
    //[END_NODE_BLOCK, 0, func()]
}
//[END_NODE_BLOCK, , func()]
}
```

EMB Designer - DB DOF 변수 - 소스젠

다음은 DOF 변수 속성과 Properties에 대한 설명이다.

- **DOF 변수 속성**

  DOF 변수 모듈을 클릭하면 아래와 같이 아이콘이 보이며 각각의 아이콘은 기능을 갖고 있다.




EMB Designer - DB DOF 변수 - 속성

◦ (삭제)

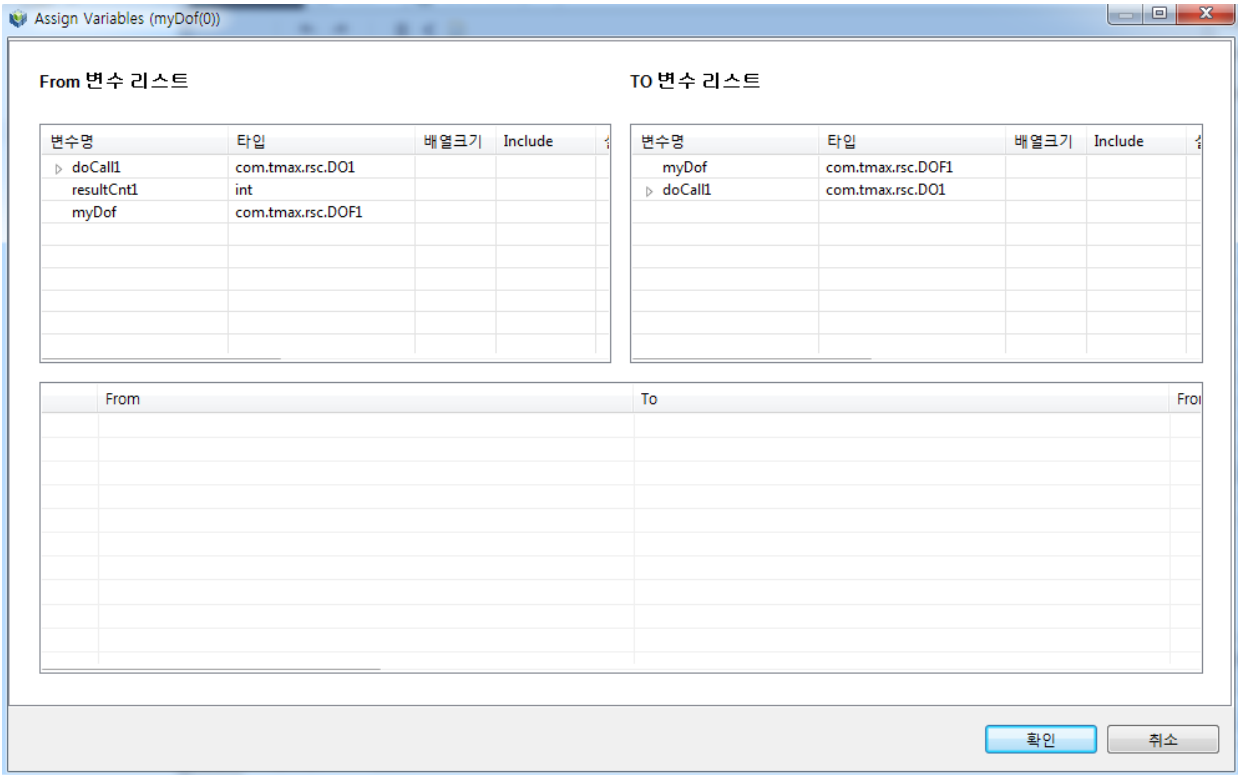
DOF 변수 모듈을 삭제한다.

◦ (주석 처리)

DOF 변수 모듈을 주석 처리한다.

◦ (조회 파라미터 셋팅)

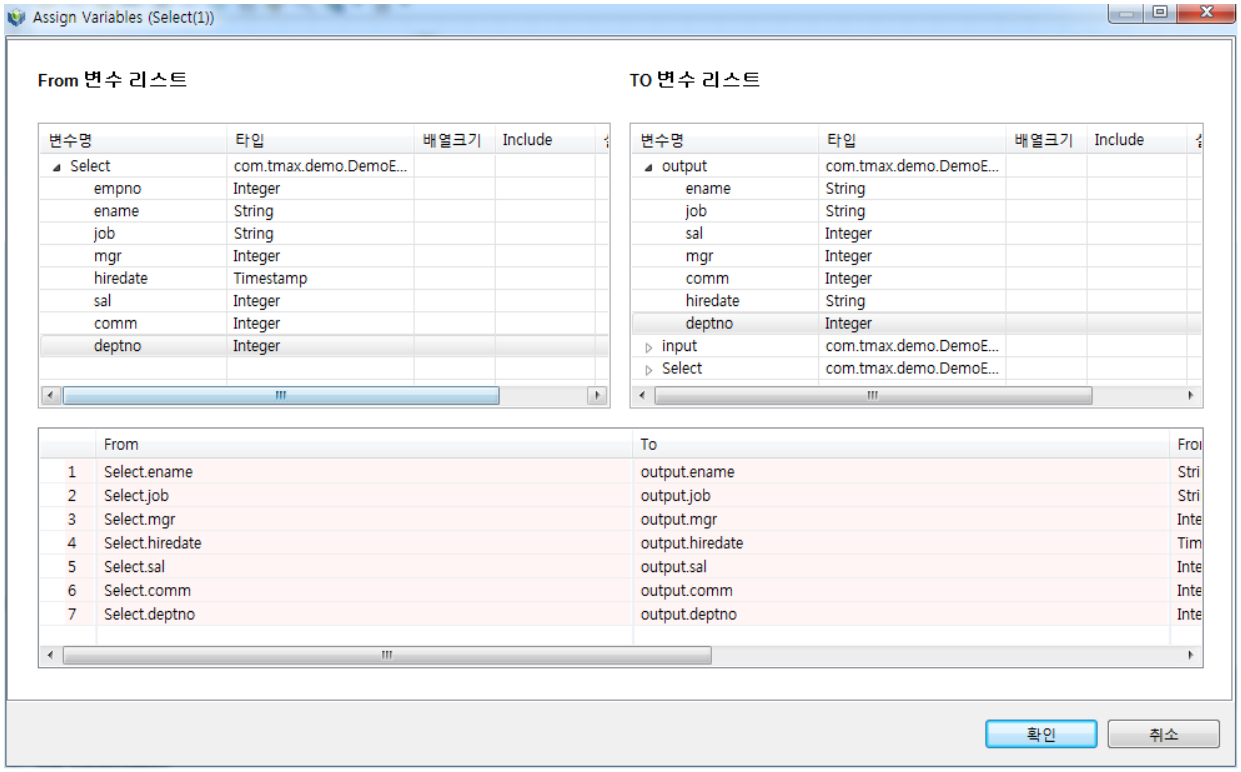
DOF 조회 파라미터에 값을 설정한다. Assign 모듈과 기능이 유사하다.




EMB Designer - DB DOF 변수 - 조회 파라미터 설정 화면

◦ (결과값 셋팅)

DOF를 통해 가져온 결과 값을 설정한다. Assign 모듈과 기능이 유사하다.




EMB Designer - DB DOF 변수 - 결과값 설정 화면

- (호출 전 코드)

  DOF가 호출되기 전 수행해야 하는 코드를 입력한다.

- (호출 후 코드)

  DOF가 호출된 후 수행해야 하는 코드를 입력한다.

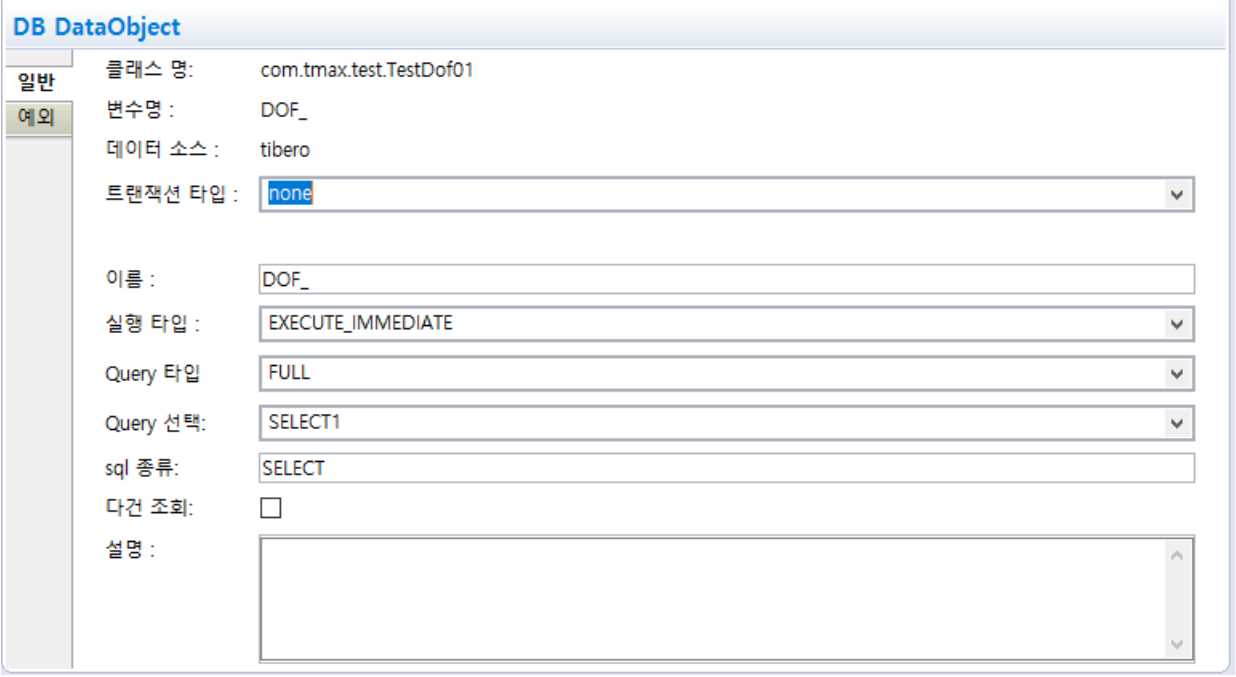
```
public void func() throws Throwable
{
//[BEGIN_NODE_BLOCK, , func()]
{
    //DECLARE ENTRY VARIABLES
    com.tmax.rsc.DO1 doCall1 = new com.tmax.rsc.DO1();
    int resultCnt1 = 0;

    //[BEGIN_NODE_BLOCK, 0, func()]
    {
        //myDof
        //[BEGIN_BEFORE_CODE, 0, func()]

        //[END_BEFORE_CODE, 0, func()]

        //[BEGIN_PRE_ASSIGN_BLOCK, 0, func()]
        {
            myDof.setEmpno(myString);
        }

        //[END_PRE_ASSIGN_BLOCK, 0, func()]

        myDof.setAutoCommit(false);
        myDof.setFullQuery(com.tmax.rsc.DOF1.FULLQUERY.SELECT_ALIAS);
        doCall1 = myDof.get();

        //[BEGIN_POST_ASSIGN_BLOCK, 0, func()]
        {
            myDo.setTest_string(doCall1.getTest_string());
        }

        //[END_POST_ASSIGN_BLOCK, 0, func()]

        //[BEGIN_AFTER_CODE, 0, func()]

        //[END_AFTER_CODE, 0, func()]
    }
    //[END_NODE_BLOCK, 0, func()]
}
//[END_NODE_BLOCK, , func()]
}
```

EMB Designer - DB DOF 변수 - 조회 파라미터, 결과값 셋팅 코드 생성

- **[Properties]** 탭

  DOF 변수 모듈을 선택한 후 나타나는 화면 하단의 **[Properties]** 탭에서 다음과 같은 특성을 설정할 수 있다.

  - **[일반]** 탭

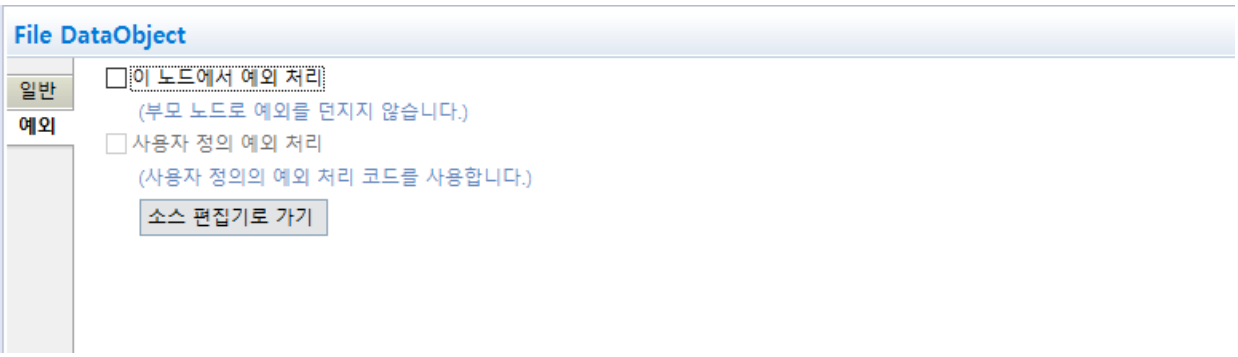



EMB Designer - DB DOF 변수 - [Properties] - [일반] 탭

| 항목 | 설명 |
|---|---|
| 클래스명 | 호출된 DOF의 실제 리소스를 확인할 수 있다. |
| 변수명 | DOF 타입을 선언할 때 정의한 변수명을 보여준다. |
| 데이터 소스 | DOF가 사용하는 데이터소스명이다. |
| 트랜잭션 타입 | 모듈 수행시 적용 되는 트랜잭션 타입이다.<br>◦ none : 해당 모듈에서 별도 트랜잭션 처리를 하지 않는다. 서비스 종료시 정상/오류 여부에 따라 트랜잭션이 Commit/Rollback된다.<br>◦ auto commit : 지정된 데이터소스는 같은 트랜잭션에 묶이지 않으며, 서비스의 성공/실패 여부와는 상관없이 항상 데이터가 반영된다. |
| 이름 | DOF 변수의 논리명이다. |
| 실행 타입 | DOF의 실행 타입을 설정한다.<br>◦ EXECUTE_IMMEDIATE : SQL을 즉시 수행한다.<br>◦ EXECUTE_BATCH : Jdbc의 addBatch/executeBatch 기능을 활용하여 모아서 SQL을 수행한다. |
| Query 타입 | DOF가 실행할 쿼리를 선택한다.<br>◦ FULL : FULL 타입으로 생성한 DOF의 SQL 내역을 선택할 수 있다. (데이터 오브젝트 팩토리 생성(Full) 참고)<br>◦ DYNAMIC : DYNAMIC 타입으로 생성한 DOF의 SQL 내역을 선택할 수 있다. BO/SO 내부에서 DOF에 정의된 Dynamic 변수로 추가적인 SQL 내용을 파라미터로 전달하여 완성된 SQL이 수행된다. (데이터 오브젝트 팩토리 생성(Dynamic) 참고) |
| Query 선택 | DOF가 실행할 쿼리를 선택한다. **'Query 타입'**에 해당하는 SQL Alias 내역이 선택 대상이다. |

| 항목 | 설명 |
| --- | --- |
| sql 종류 | DOF가 실행할 쿼리의 타입을 보여준다. |
| 다건 조회 | 조회된 결과가 다건일 경우 for 루프로 결과 값을 조회한다. **'sql 종류'** 항목이 SELECT일 경우에만 활성화된다. |
| 설명 | DOF 변수 모듈에 대한 내용을 입력한다. |

◦ **[예외]** 탭




EMB Designer - DB DOF 변수 - [Properties] - [예외] 탭

| 항목 | 설명 |
| --- | --- |
| 이 노드에서 예외 처리 | try-catch 문이 소스젠에 자동으로 생성되여 부모 노드로 예외를 발생하지 않는다. 해당 항목이 체크되어 있어야 **'사용자 정의 예외처리'** 항목이 활성화된다. |
| 사용자 정의 예외 처리 | 소스에서 개발자가 직접 catch 문을 추가할 수 있다. |
| **[소스 편집기로 가기]** | 모듈의 try-catch 해당 소스 라인으로 이동한다. |

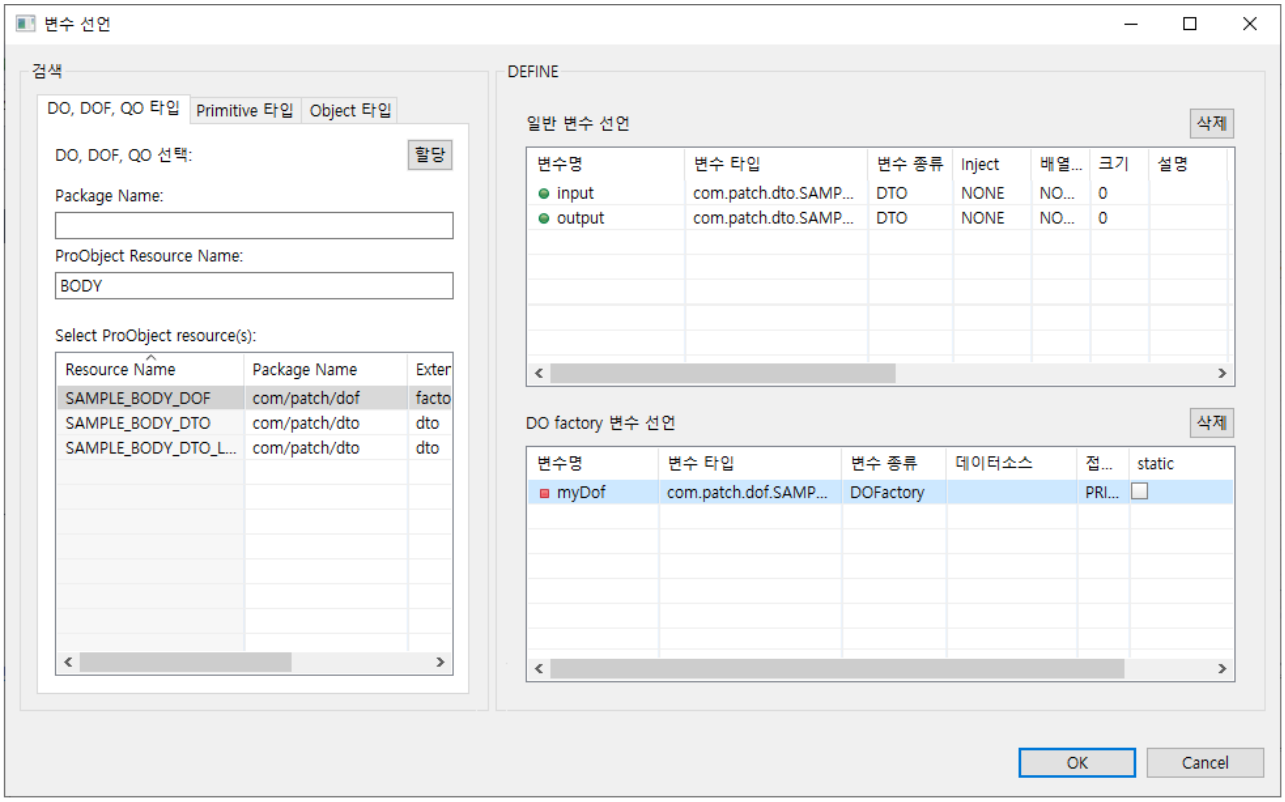
### 4.3.4.11. DO Factory 변수 (FILE)

DO Factory 변수는 변수 정의에서 DOF type으로 DOF를 변수 선언할 경우 **Palette**에서 DOF 레퍼런스 변수를 확인할 수 있다.

다음은 File DO Factory Resource에 변수를 추가하는 과정에 대한 설명이다. 개발할 때 필요한 API Level 설명은 ProObject 런타임 엔진 개발자 안내서의 "파일 데이터 오브젝트 팩토리"를 참고한다.

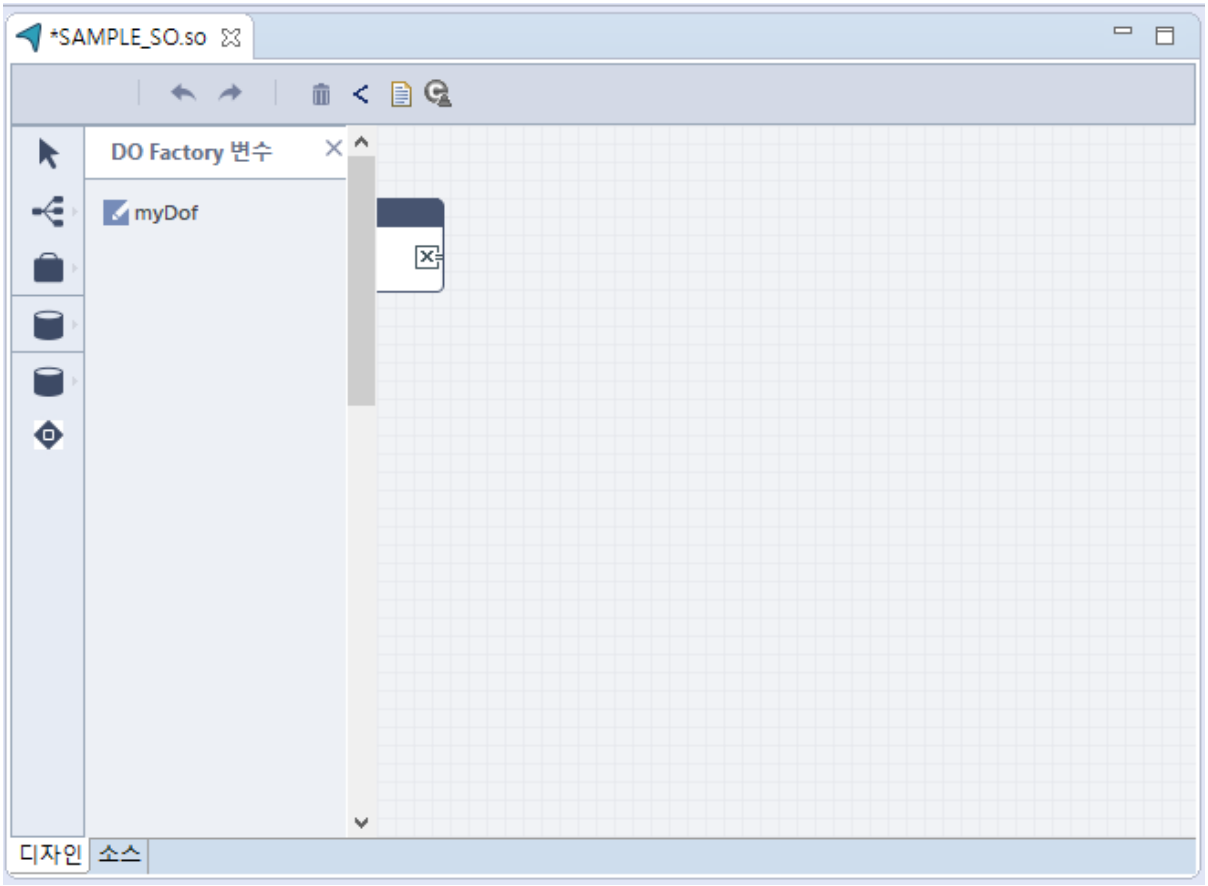
1. + (**[멤버 변수 설정]**) 아이콘을 클릭하면 **변수 정의 화면**이 나타난다.

   해당 화면에서 DO Type을 설정한 후 생성한 DO Type을 검색한다. 검색된 DO Type을 더블클릭하면 아래와 같이 변수가 생성된다.




EMB Designer - File DOF 변수 - 변수 정의

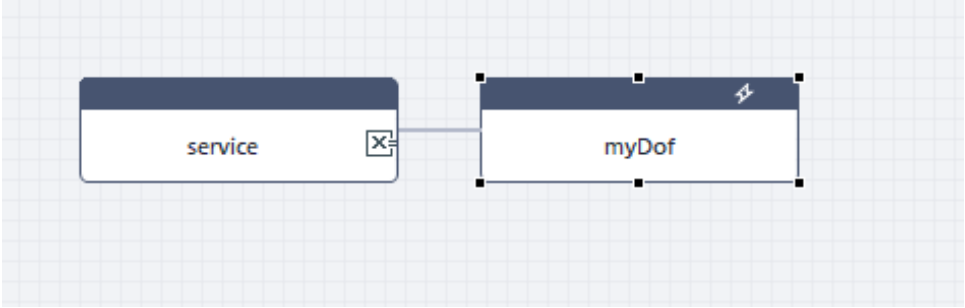
2. **변수 정의 화면**을 저장한 후 Palette에서 DOF 변수가 생성된 것을 확인할 수 있다.



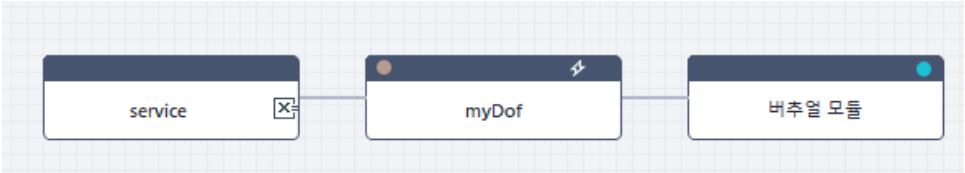

EMB Designer - File DOF 변수 - Palette

3. **Palette**에서 ([**DO Factory 변수**])를 클릭한 후 디자인 영역을 클릭한다. DOF 변수 모듈을 클릭한 후

원하는 위치로 이동할 수 있다.




EMB Designer - File DOF 변수 - 일반




EMB Designer - File DOF 변수 - 루프 모드

DOF 변수를 디자인 에디터에 생성하면 다음과 같이 자동 소스가 생성된다.

```
{
//이너 모듈
   //[BEGIN_NODE_BLOCK, 0, service(SAMPLE_IN)]
   {
   //myDof
   //[BEGIN_BEFORE_CODE, 0, service(SAMPLE_IN)]
```

```
   //[END_BEFORE_CODE, 0, service(SAMPLE_IN)]

   //[BEGIN_PRE_ASSIGN_BLOCK, 0, service(SAMPLE_IN)]
   //[END_PRE_ASSIGN_BLOCK, 0, service(SAMPLE_IN)]

   myDof.setFile(new File(""), false);

   //[BEGIN_POST_ASSIGN_BLOCK, 0, service(SAMPLE_IN)]
   //[END_POST_ASSIGN_BLOCK, 0, service(SAMPLE_IN)]

   //[BEGIN_AFTER_CODE, 0, service(SAMPLE_IN)]
```

```
   //[END_AFTER_CODE, 0, service(SAMPLE_IN)]
   }
   //[END_NODE_BLOCK, 0, service(SAMPLE_IN)]
}
```

EMB Designer - File DOF Source - 일반

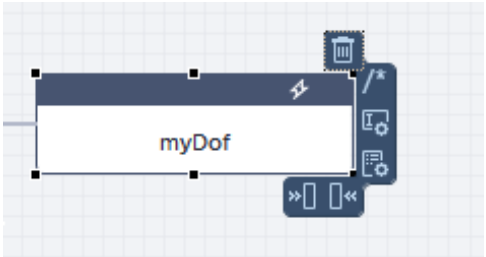
```
//이너 모듈
   //[BEGIN_NODE_BLOCK, 0, service(SAMPLE_IN)]
   {
   //myDof

   //[BEGIN_BEFORE_CODE, 0, service(SAMPLE_IN)]

   //[END_BEFORE_CODE, 0, service(SAMPLE_IN)]

   //[BEGIN_PRE_ASSIGN_BLOCK, 0, service(SAMPLE_IN)]
   //[END_PRE_ASSIGN_BLOCK, 0, service(SAMPLE_IN)]

   myDof.setFile(new File(""), false);


   int _i0 = 0;
   //[BEGIN_LOOP_MODULE_PREPROCESSING, 0, service(SAMPLE_IN)]

   //[END_LOOP_MODULE_PREPROCESSING, 0, service(SAMPLE_IN)]
   for(com.patch.dto.SAMPLE_BODY_DTO var0 : myDof.getForwardList()) {
            doCall1 = var0;
   //[BEGIN_NODE_BLOCK, 2, service(SAMPLE_IN)]
   {
   //버추얼 모듈
    //[BEGIN_VIRTUAL_CODE_BLOCK, 2, service(SAMPLE_IN)]
    {
        // Loop Logic 구현
    }
    //[END_VIRTUAL_CODE_BLOCK, 2, service(SAMPLE_IN)]

   }
   //[END_NODE_BLOCK, 2, service(SAMPLE_IN)]
   }

   //[BEGIN_AFTER_CODE, 0, service(SAMPLE_IN)]

   //[END_AFTER_CODE, 0, service(SAMPLE_IN)]
   }
   //[END_NODE_BLOCK, 0, service(SAMPLE_IN)]
}
```

EMB Designer - File DOF Source - 루프 모드

다음은 File DOF 변수 속성과 Properties에 대한 설명이다.

- **DOF 변수 속성**

  DOF 변수 모듈을 클릭하면 아래와 같이 아이콘이 보이며 각각의 아이콘은 기능을 갖고 있다.




EMB Designer - File DOF 변수 - 속성

  - (삭제)

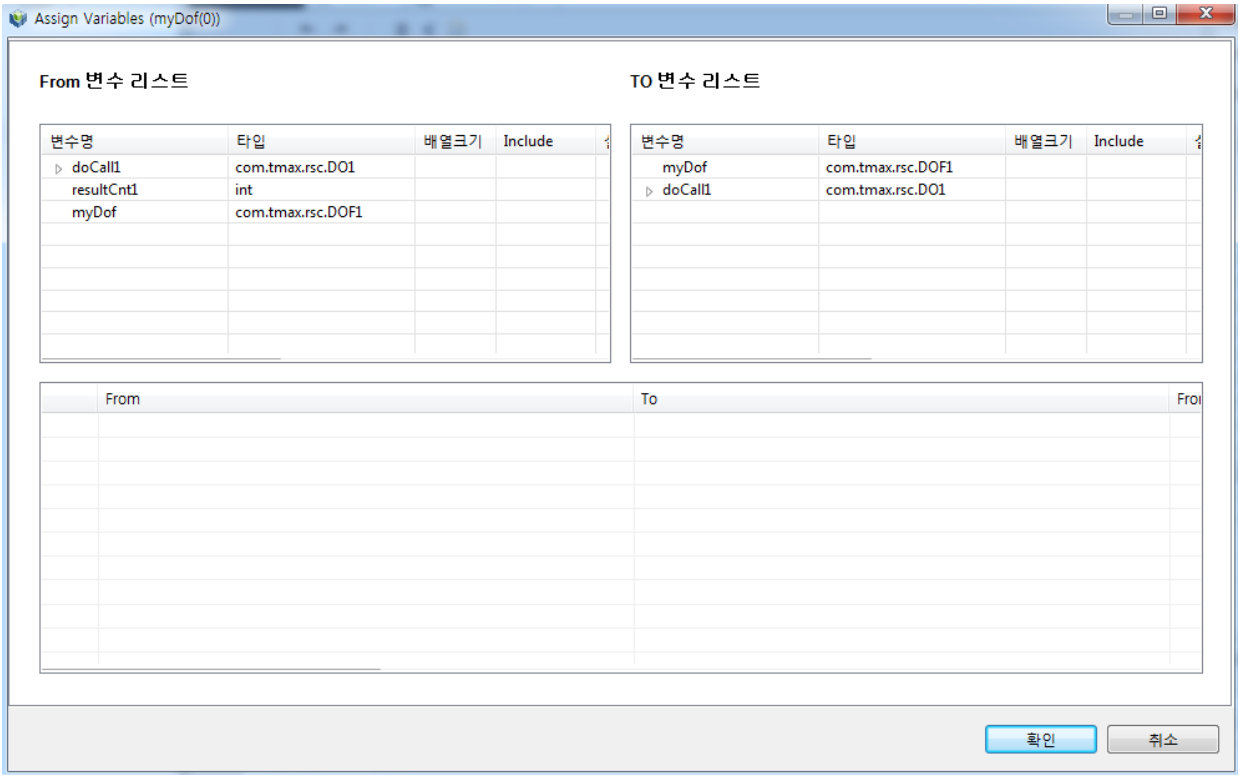
DOF 변수 모듈을 삭제한다.

- (주석 처리)

  DOF 변수 모듈을 주석 처리한다.

- (조회 파라미터 셋팅)

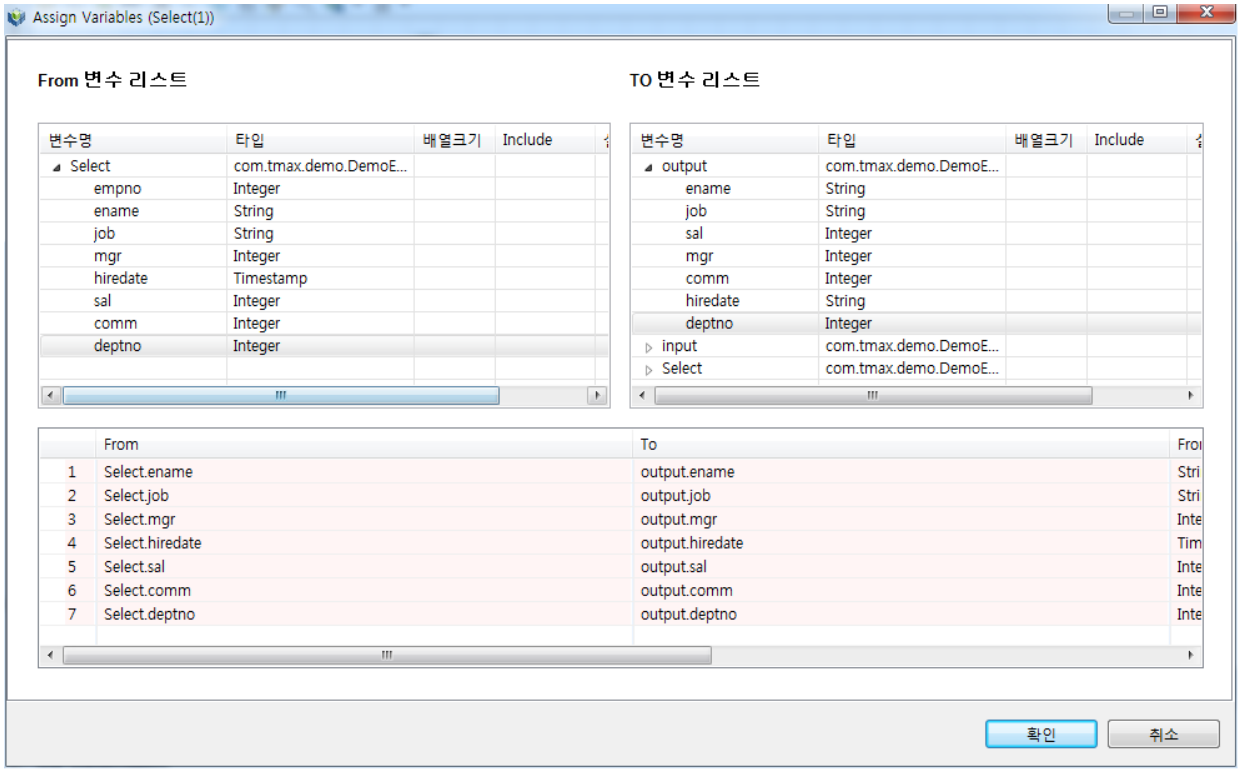
  DOF 조회 파라미터에 값을 설정한다. Assign 모듈과 기능이 유사하다.




EMB Designer - File DOF 변수 - 조회 파라미터 설정 화면

- (결과값 셋팅)

  DOF를 통해 가져온 결과 값을 설정한다. Assign 모듈과 기능이 유사하다.




EMB Designer - File DOF 변수 - 결과값 설정 화면

-  (호출 전 코드)

  DOF가 호출되기 전 수행해야 하는 코드를 입력한다.

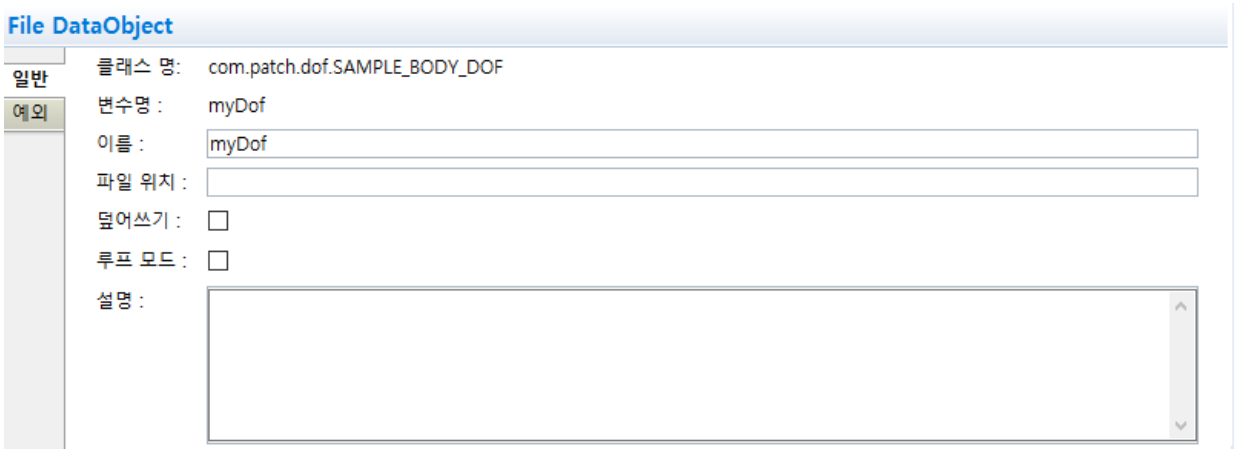
-  (호출 후 코드)

  DOF가 호출된 후 수행해야 하는 코드를 입력한다.

- **[Properties]** 탭

  DOF 변수 모듈을 선택한 후 나타나는 화면 하단의 **[Properties]** 탭에서 다음과 같은 특성을 설정할 수 있다.

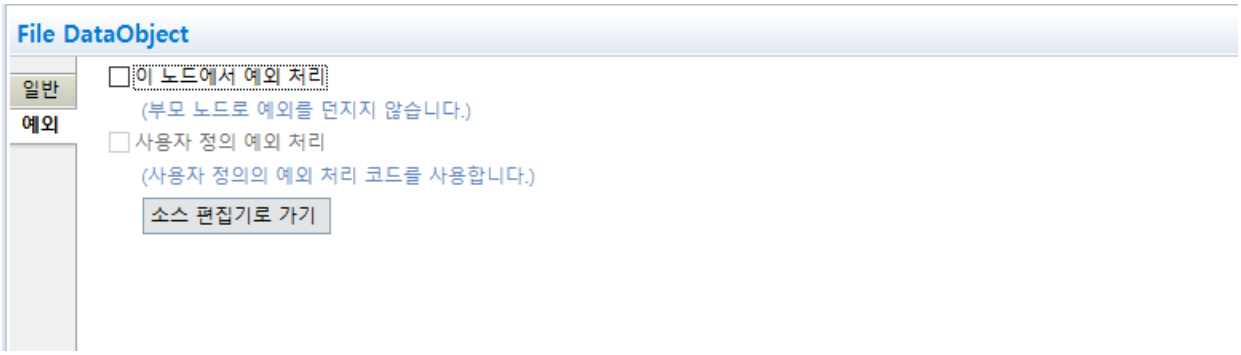
  - **[일반]** 탭




EMB Designer - File DOF 변수 - [Properties] - [일반] 탭

| 항목 | 설명 |
|---|---|
| 클래스명 | 호출된 DOF의 실제 리소스를 확인할 수 있다. |
| 변수명 | DOF 타입을 선언할 때 정의한 변수명을 보여준다. |
| 이름 | DOF Class명이다. |
| 파일위치 | File 위치를 설정한다. |
| 덮어쓰기 | File을 Write할 때 이전 데이터를 무시하고 Overwrate할지 여부를 선택한다. 체크박스를 체크하면 Overwrite한다. |
| 루프 모드 | Loop 처리를 할지 여부를 나타낸다. 체크박스를 체크하면 버추얼 모듈을 하위 노드로 추가할 수 있다. |
| 설명 | DOF 변수 모듈에 대한 내용을 입력한다. |

◦ **[예외]** 탭




EMB Designer - File DOF 변수 - [Properties] - [예외] 탭

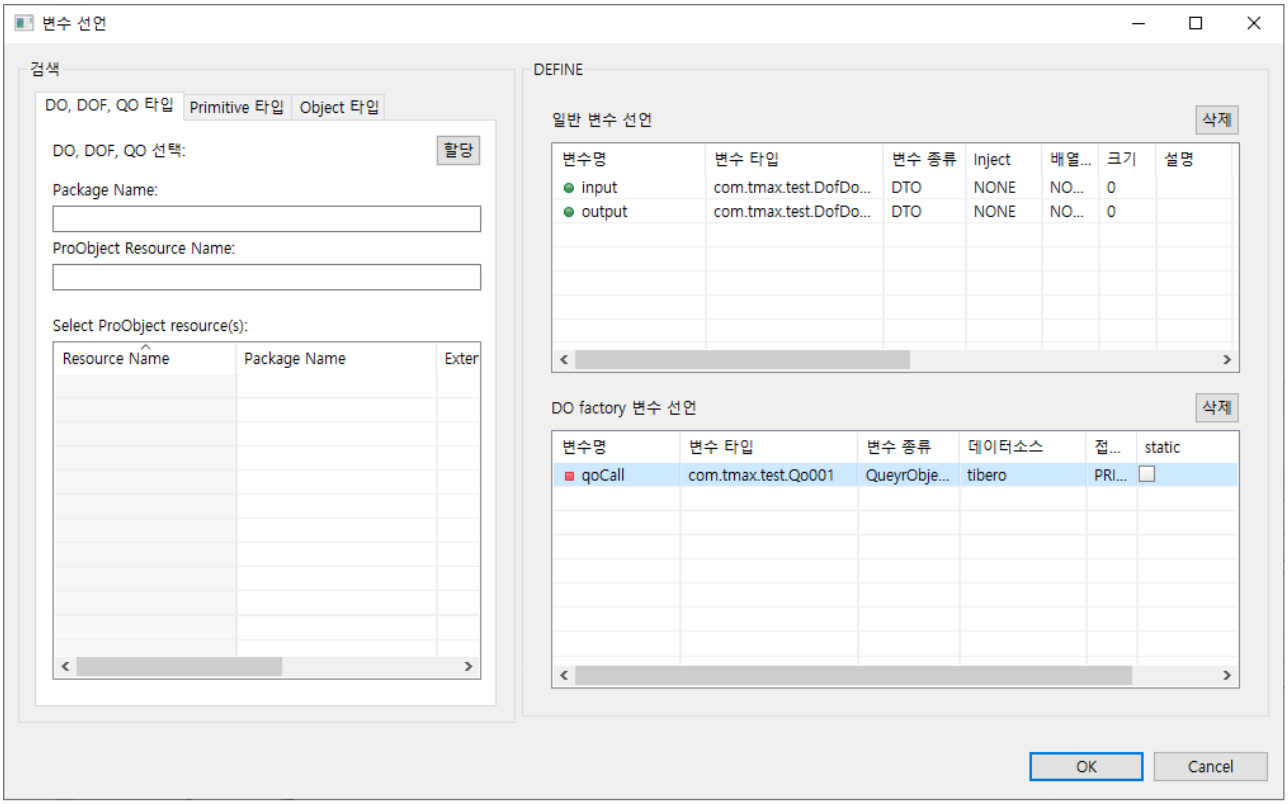
| 항목 | 설명 |
|---|---|
| 이 노드에서 예외 처리 | try-catch 문이 소스젠에 자동으로 생성되여 부모 노드로 예외를 발생하지 않는다. 해당 항목이 체크되어 있어야 **'사용자 정의 예외처리'** 항목이 활성화된다. |
| 사용자 정의 예외 처리 | 소스에서 개발자가 직접 catch 문을 추가할 수 있다. |
| **[소스 편집기로 가기]** | 모듈의 try-catch 해당 소스 라인으로 이동한다. |

### 4.3.4.12. Query Object 변수

EMB Designer에서 Query Object 사용을 위해서는 변수 선언이 선행 되어야 한다. **변수 정의 화면**에서 Query Object 변수를 설정한 후 EMB Designer에서 사용 할 수 있다.

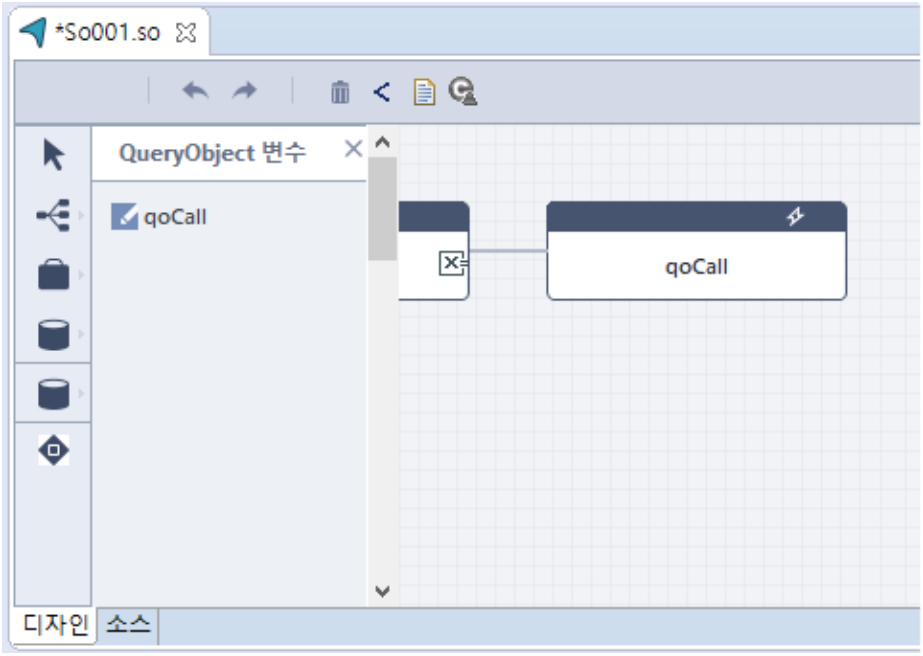
다음은 Query Object 변수를 추가하는 과정에 대한 설명이다.

1. + (**[멤버 변수 설정]**) 아이콘을 클릭하면 **변수 정의 화면**이 나타난다. 해당 화면에서 DO Type을 선택한 후 생성한 Query Object Type을 검색한다. 검색된 Query Object를 더블클릭하면 아래와 같이 변수가 생성된다 .추가된 변수에 변수명, 데이터소스 등을 설정한다.



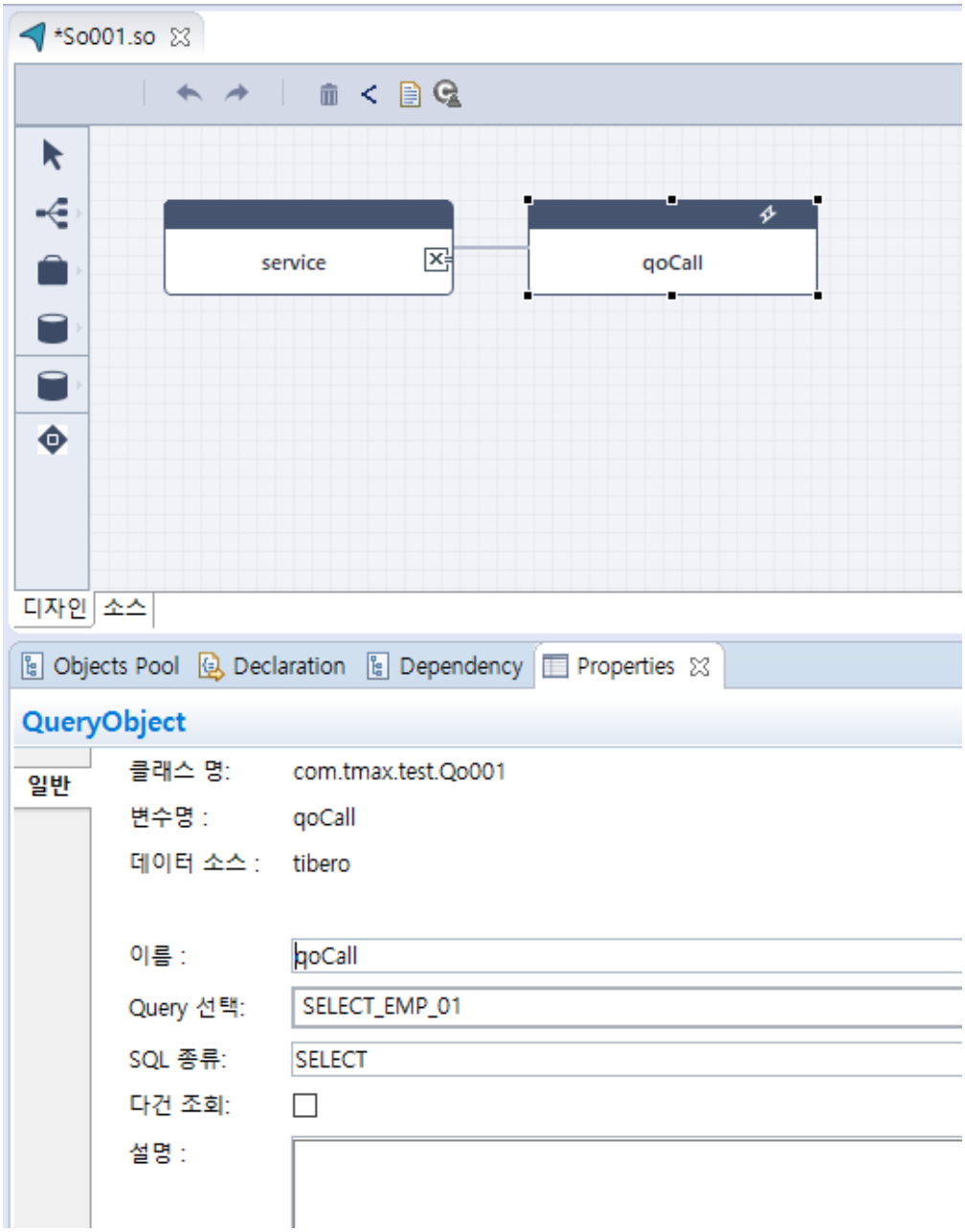

EMB Designer - Query Object 변수 - 변수 정의

2. **변수 정의 화면**을 저장한 후 Palette에서 Query Object 변수가 생성된 것을 확인할 수 있다.




EMB Designer - Query Object 변수 - Palette

3. **Palette**에서 ([**Query Object 변수**])를 클릭한 후 디자인 영역을 클릭한다. Query Object 변수 모듈을 클릭한 후 원하는 위치로 이동할 수 있다.



EMB Designer - Query Object 변수 - 디자인 영역

다음은 **[Properties]** 탭의 항목에 대한 설명이다.

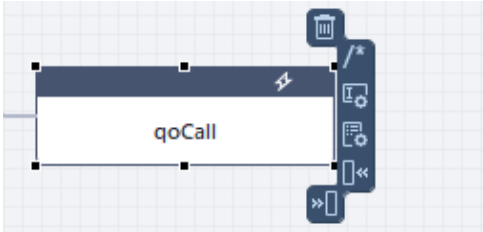
| 항목 | 설명 |
|---|---|
| 클래스명 | 호출된 Query Object의 실제 리소스를 확인할 수 있다. |
| 변수명 | Query Object 타입을 선언할 때 정의한 변수명을 보여준다. |
| 데이터 소스 | Query Object가 사용하는 데이터소스명이다. |
| 이름 | Query Object 변수의 논리명이다. |
| Query 선택 | Query Object가 실행할 쿼리를 선택한다. Query Object에 설정한 Alias 내역이 선택 대상이다. |
| SQL 종류 | Query Object가 실행할 쿼리의 타입을 보여준다. |
| 다건조회 | 조회된 결과가 다건일 경우 for 루프로 결과 값을 조회한다. **'SQL 종류'** 항목이 SELECT일 경우에만 활성화된다. |

| 항목 | 설명 |
| --- | --- |
| 설명 | Query Object 변수 모듈에 대한 내용을 입력한다. |

다음은 Query Object 변수 속성과 Properties에 대한 설명이다.

- **Query Object 변수 속성**

  Query Object 변수 모듈을 클릭하면 아래와 같이 아이콘이 보이며 각각의 아이콘은 기능을 갖고 있다.



EMB Designer - Query Object 변수 - 속성

  - (삭제)

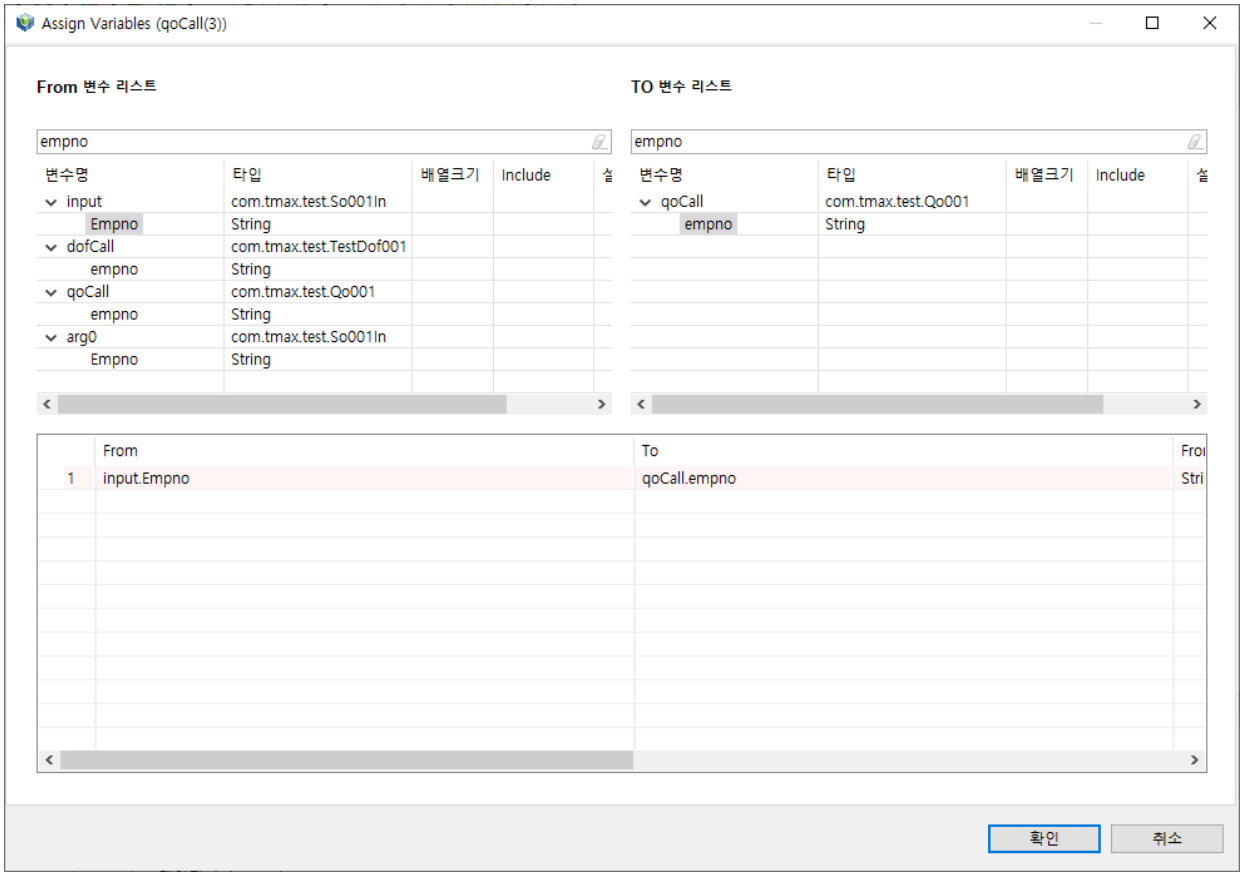
    Query Object 변수 모듈을 삭제한다.

  - (주석 처리)

    Query Object 변수 모듈을 주석 처리한다.

  - (조회 파라미터 셋팅)

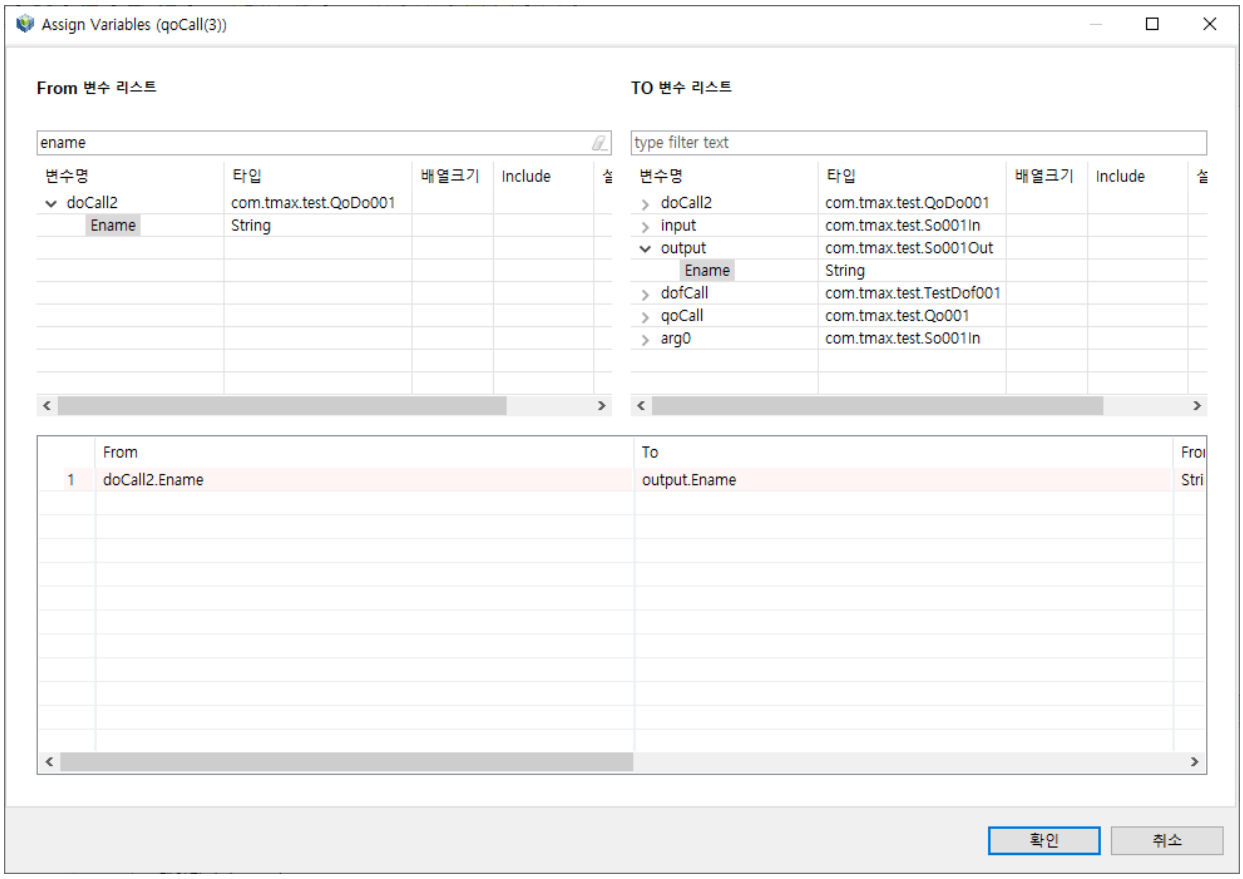
    Query Object 조회 파라미터에 값을 설정한다. Assign 모듈과 기능이 유사하다.




EMB Designer - Query Object 변수 - 조회 파라미터 설정 화면

- (결과값 셋팅)

Query Object를 통해 가져온 결과 값을 설정한다. Assign 모듈과 기능이 유사하다.




EMB Designer - Query Object 변수 - 결과값 설정 화면

- (호출 전 코드)

  Query Object가 호출되기 전 수행해야 하는 코드를 입력한다.

- (호출 후 코드)

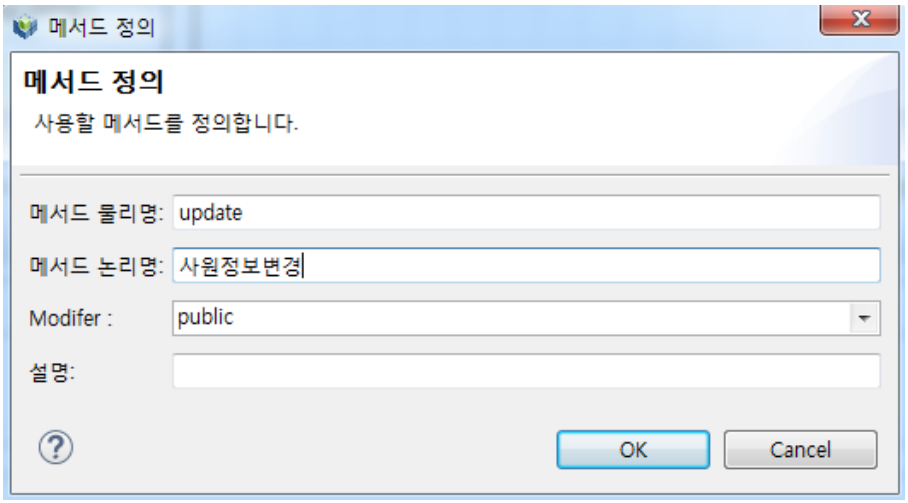
  Query Object가 호출된 후 수행해야 하는 코드를 입력한다.

#### 4.3.4.13. 로컬 메소드

메소드는 BO에 정의한 메소드 내역이 **Palette**의 로컬 메소드 영역에 나타난다. EMB Designer에 로컬 메소드를 추가하면 BO 내의 로컬 메소드를 호출하여 사용할 수 있으며 'this.method' 이름으로 소스가 생성된다.

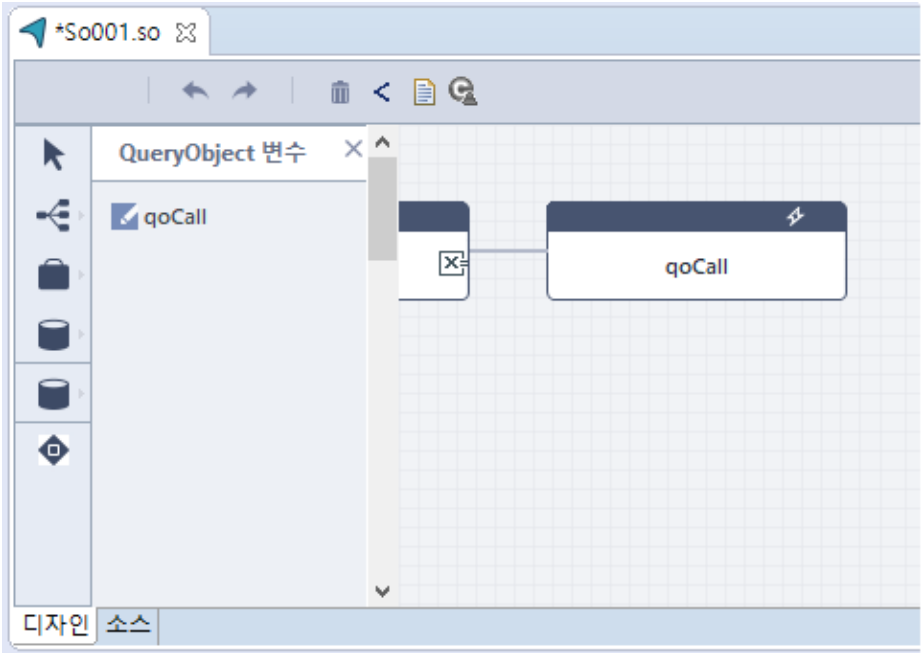
다음은 로컬 메소드를 추가하는 과정에 대한 설명이다.

1. **[메소드 정의]** 버튼을 클릭하면 나타나는 **메소드 정의 화면**에 기본 정보를 등록한다.



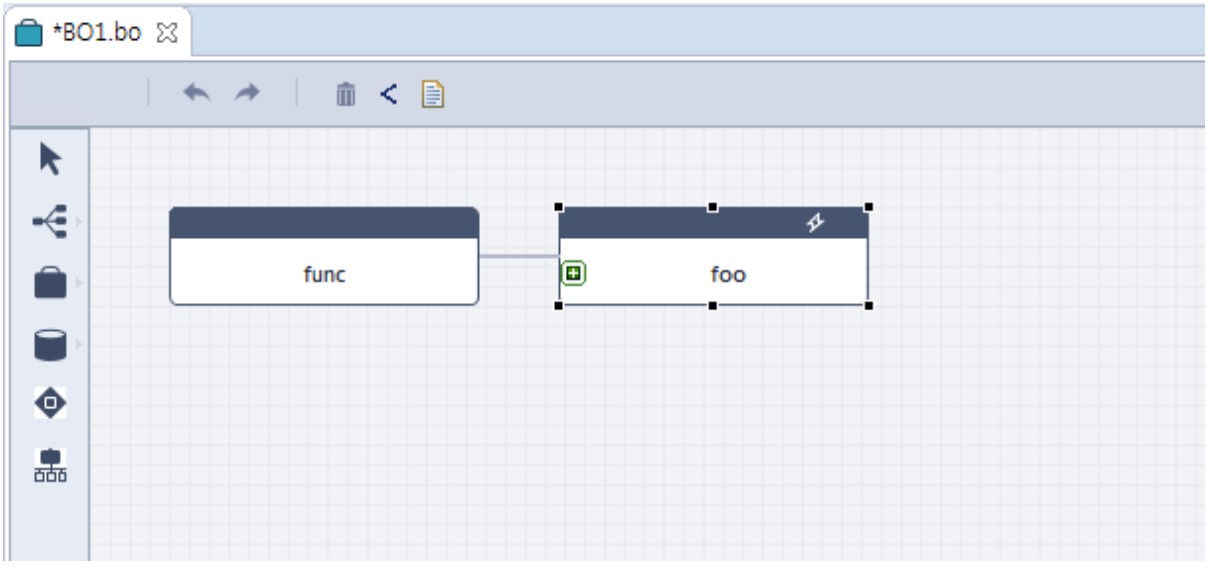

EMB Designer - 로컬 메소드 - 메소드 정의 화면

2. **메소드 정의 화면**을 저장한 후 Palette에서 로컬 메소드가 생성된 것을 확인할 수 있다.



EMB Designer - Query Object 변수 - Palette

3. **Palette**에서 **([로컬 메소드])**를 클릭한 후 디자인 영역을 클릭한다. 로컬 메소드를 디자인 화면에 삽입한다. 모듈을 클릭한 후 원하는 위치로 이동할 수 있다.



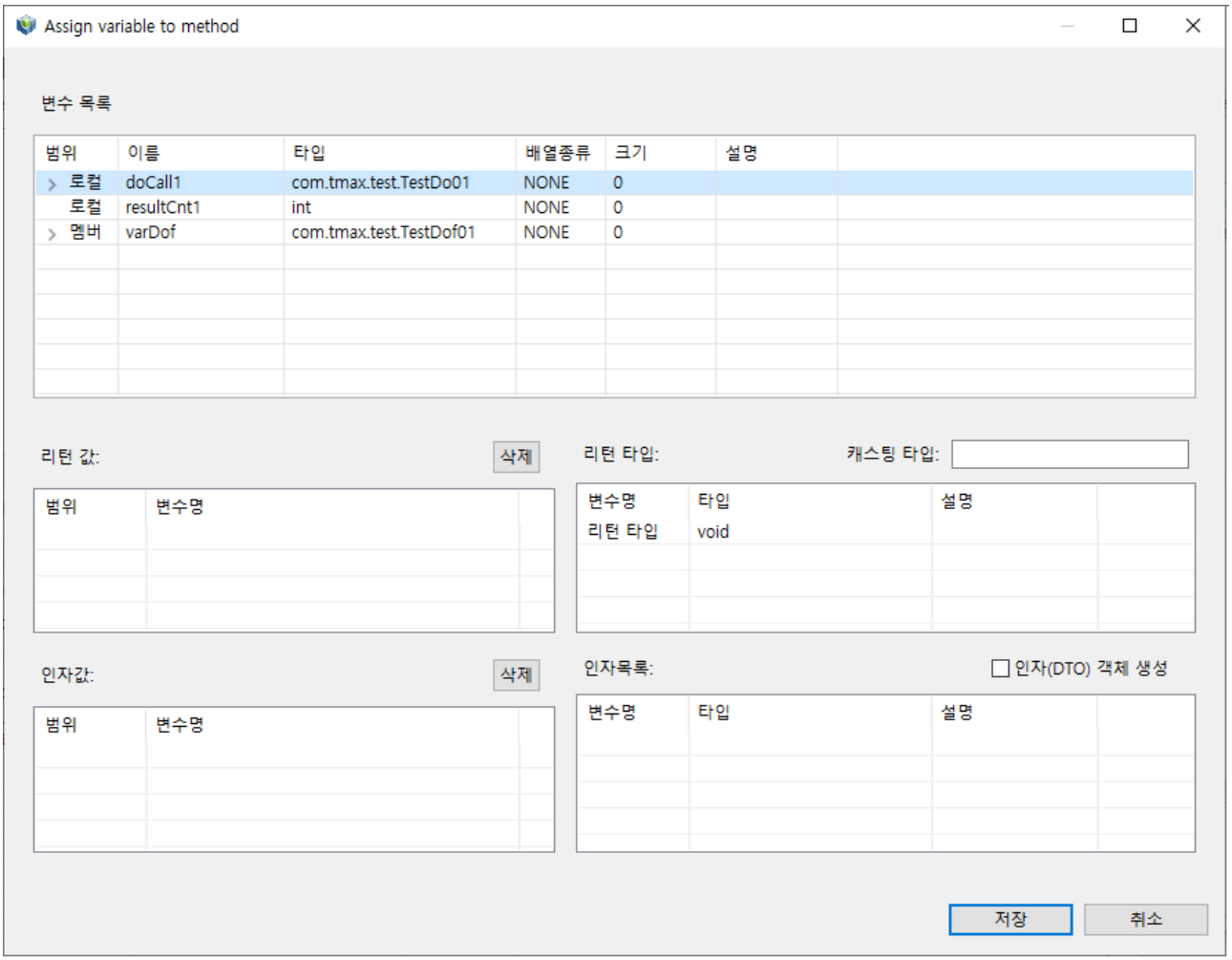
EMB Designer - 로컬 메소드 - 디자인 영역

로컬 메소드를 디자인 에디터에 추가하면 아래와 같이 자동 소스가 생성된다.

```
public void func() throws Throwable
{
//[BEGIN_NODE_BLOCK, , func()]
{

    //[BEGIN_NODE_BLOCK, 0, func()]
    {
        //[BEGIN_PRE_ASSIGN_BLOCK, 0, func()]

        //[END_PRE_ASSIGN_BLOCK, 0, func()]
        this.foo();
        //[BEGIN_POST_ASSIGN_BLOCK, 0, func()]

        //[END_POST_ASSIGN_BLOCK, 0, func()]
    }
    //[END_NODE_BLOCK, 0, func()]

}
//[END_NODE_BLOCK, , func()]
}
```

EMB Designer - 로컬 메소드 - 소스젠

4. 메소드 매핑이 필요한 경우 메소드 모듈의 **[+]**을 클릭하면 다음과 같은 매핑 화면이 나타난다.




EMB Designer - 로컬 메소드 - 매핑

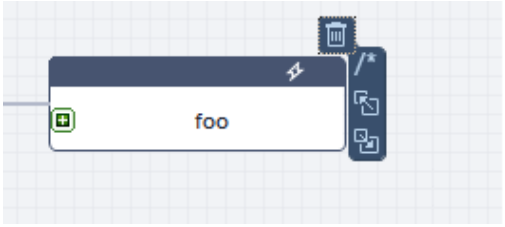
| 항목 | 설명 |
|---|---|
| 변수 목록 | 사용자가 BO에 선언한 멤버 변수 목록이 표시된다. |

| 항목 | 설명 |
| --- | --- |
| 리턴 값 | 해당 메소드의 리턴값을 설정할 수 있다. 변수 목록에서 리턴값으로 사용할 변수를 드래그 앤드 드롭하여 추가하거나, **[삭제]** 버튼을 클릭해서 리턴값에서 삭제할 수 있다. |
| 리턴타입 | 사용자가 설정한 리턴 타입을 표시한다. 해당 타입에 해당하는 변수만 리턴 값으로 추가할 수 있다. |
| 인자값 | 변수 목록에서 인자로 사용할 변수를 드래그 앤드 드롭하여 추가하거나, **[삭제]** 버튼을 클릭해서 인자값에서 삭제할 수 있다. |
| 인자목록 | 사용자가 설정한 인자 타입을 표시한다. 해당 타입에 해당하는 변수만 인자 값으로 추가할 수 있다. |
| 캐스팅 타입 | 리턴타입의 변수를 입력된 타입으로 캐스팅할 수 있다. |

다음은 로컬 메소드의 노드 속성과 Properties에 대한 설명이다.

- **로컬 메소드 노드 속성**

  로컬 메소드 모듈을 클릭하면 아래와 같이 아이콘이 보이며 각각의 아이콘은 기능을 갖고 있다.




EMB Designer - 로컬 메소드 - 노드 속성

  - (삭제)

    로컬 메소드를 삭제한다.

  - (주석 처리)

    로컬 메소드 호출부를 주석 처리한다.

  - (선할당 처리)

    로컬 메소드를 호출하기 전 set할 값을 설정한다. Assign 모듈과 기능이 유사하다.

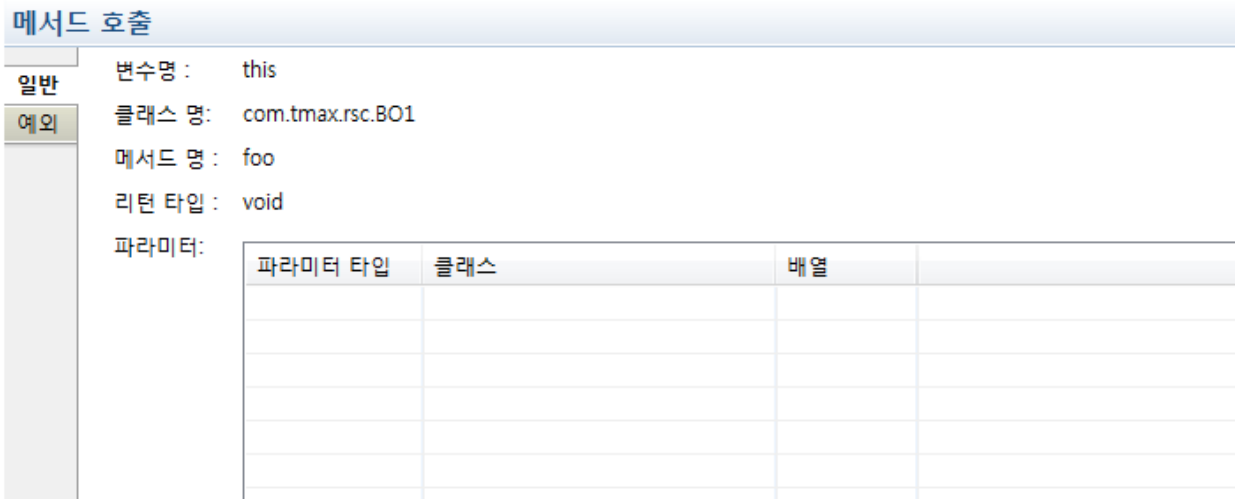
  - (후할당 처리)

    로컬 메소드를 호출한 후 get할 값을 설정한다. Assign 모듈과 기능이 유사하다.

- **[Properties]** 탭

  로컬 메소드을 선택하면 나타나는 화면 하단의 **[Properties]** 탭에서 다음과 같은 특성을 설정할 수 있다.

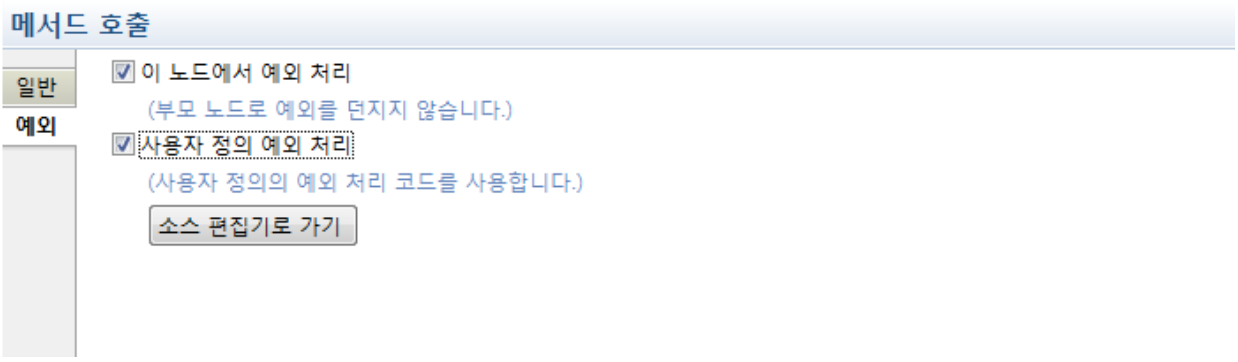
  - **[일반]** 탭




EMB Designer - 로컬 메소드 - [Properties] - [일반] 탭

| 항목 | 설명 |
|---|---|
| 변수명 | 로컬 메소드는 this 변수로 호출되므로 변수명은 'this'로 표기된다. |
| 클래스명 | 로컬 메소드가 생성된 클래스명을 나타낸다. |
| 메소드명 | 호출된 메소드명을 나타낸다. |
| 리턴 타입 | 메소드의 리턴 타입을 나타내며 리턴 타입 변경이 없을 경우 기본적으로 'void'를 사용한다. |
| 파라미터 | 메소드 생성할 때 설정된 agrs 값을 보여준다. |

◦ **[예외]** 탭




EMB Designer - 로컬 메소드 - [Properties] - [예외] 탭

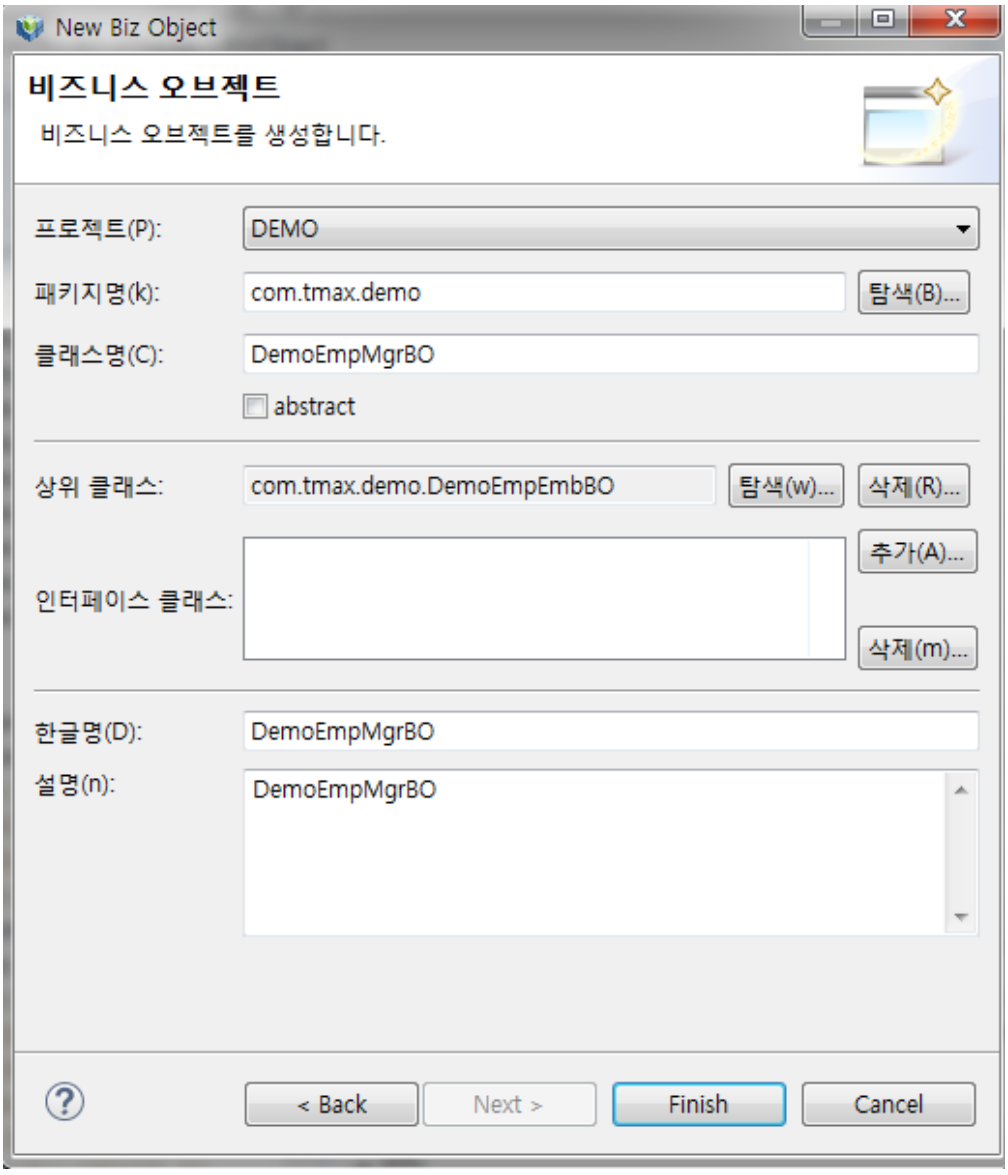
| 항목 | 설명 |
|---|---|
| 이 노드에서 예외 처리 | try-catch 문이 소스젠에 자동으로 생성되여 부모 노드로 예외를 던지지 않는다. 해당 항목이 체크되어 있어야 **'사용자 정의 예외처리'** 항목이 활성화된다. |
| 사용자 정의 예외 처리 | 소스에서 개발자가 직접 catch 문을 추가할 수 있다. |
| **[소스 편집기로 가기]** | 모듈의 try-catch 해당 소스 라인으로 이동한다. |

### 4.3.4.14. 상위 메소드

상위 메소드는 BO를 생성할 때에 지정한 상위 클래스의 메소드 내역이 **Palette**의 **상위 메소드**에 나타난다. EMB Designer에 상위 메소드를 추가하면 상위 클래스의 메소드를 호출하여 사용할 수 있으며 'super.method'로 소스가 생성된다.

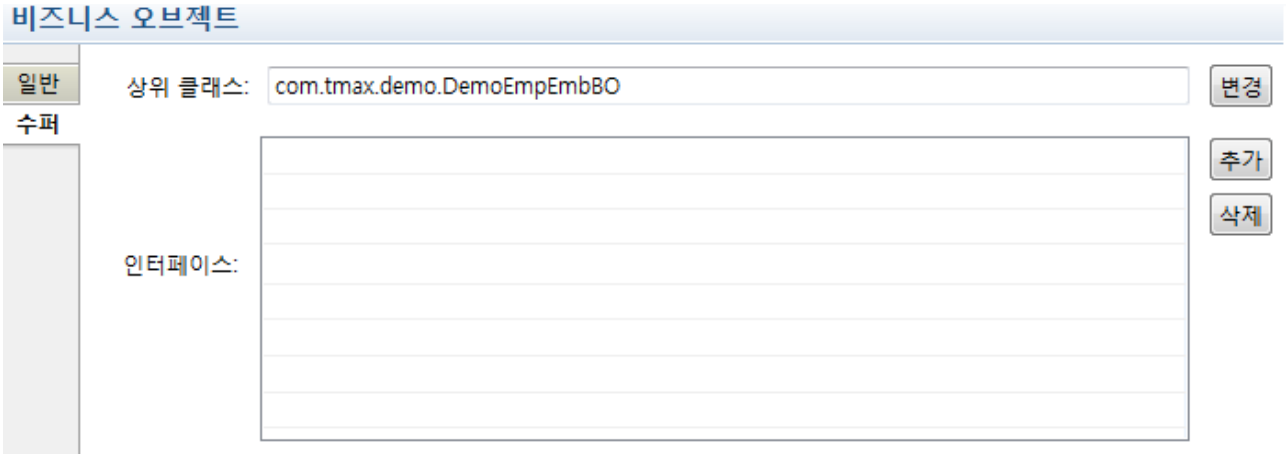
다음의 방법으로 상위 메소드를 등록할 수 있다.

1. BO를 생성할 때 상위 클래스를 등록한다.



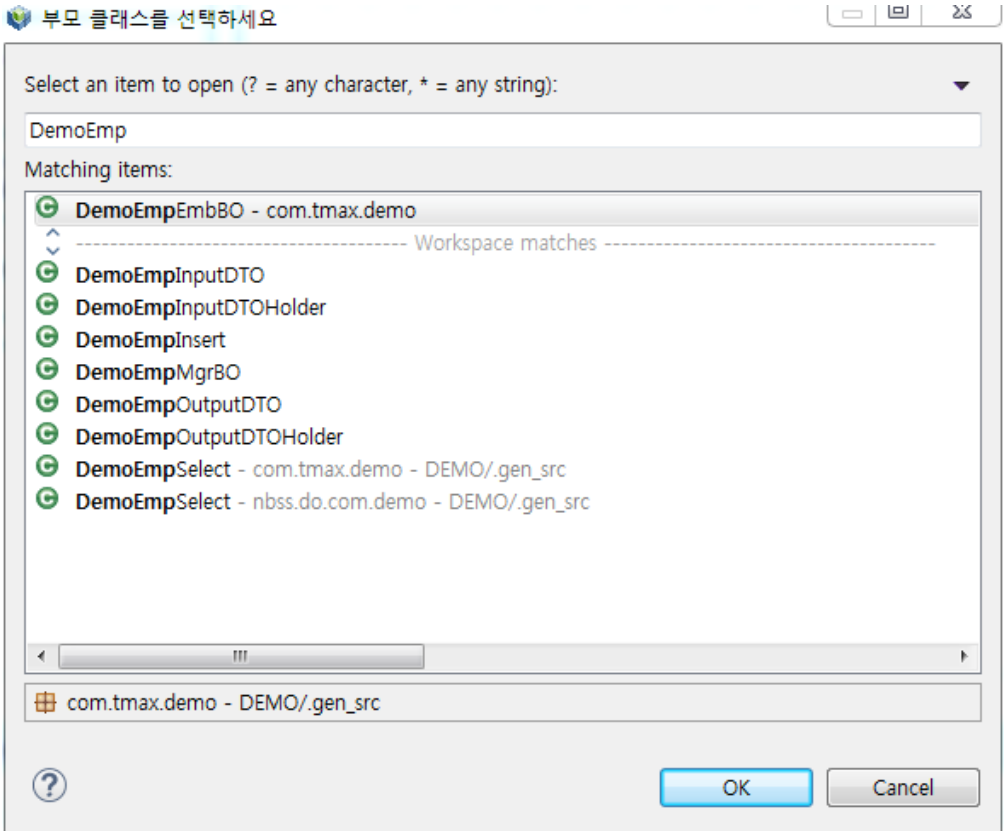
EMB Designer - 상위 메소드 - BO 생성 상위 클래스 등록

BO property의 **[수퍼]** 탭에서 **'상위 클래스'** 항목에 **[변경]** 버튼을 클릭한다.




EMB Designer - 상위 메소드 - 상위 클래스 등록

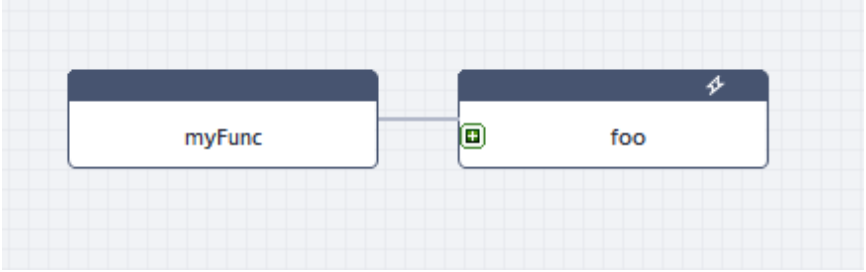
**부모 클래스 선택 화면**에서 상위 클래스를 등록한다.




EMB Designer - 상위 메소드 - 부모 클래스 선택 화면

2. **Palette**의  (**[상위 메소드]**)에 상위 클래스의 메소드가 추가된 것을 확인할 수 있다.

   Palette의 해당 로컬 메소드를 클릭한 후 디자인 영역을 클릭해 삽입한다. 상위 메소드 모듈을 클릭한 후 원하는 위치로 이동할 수 있다.



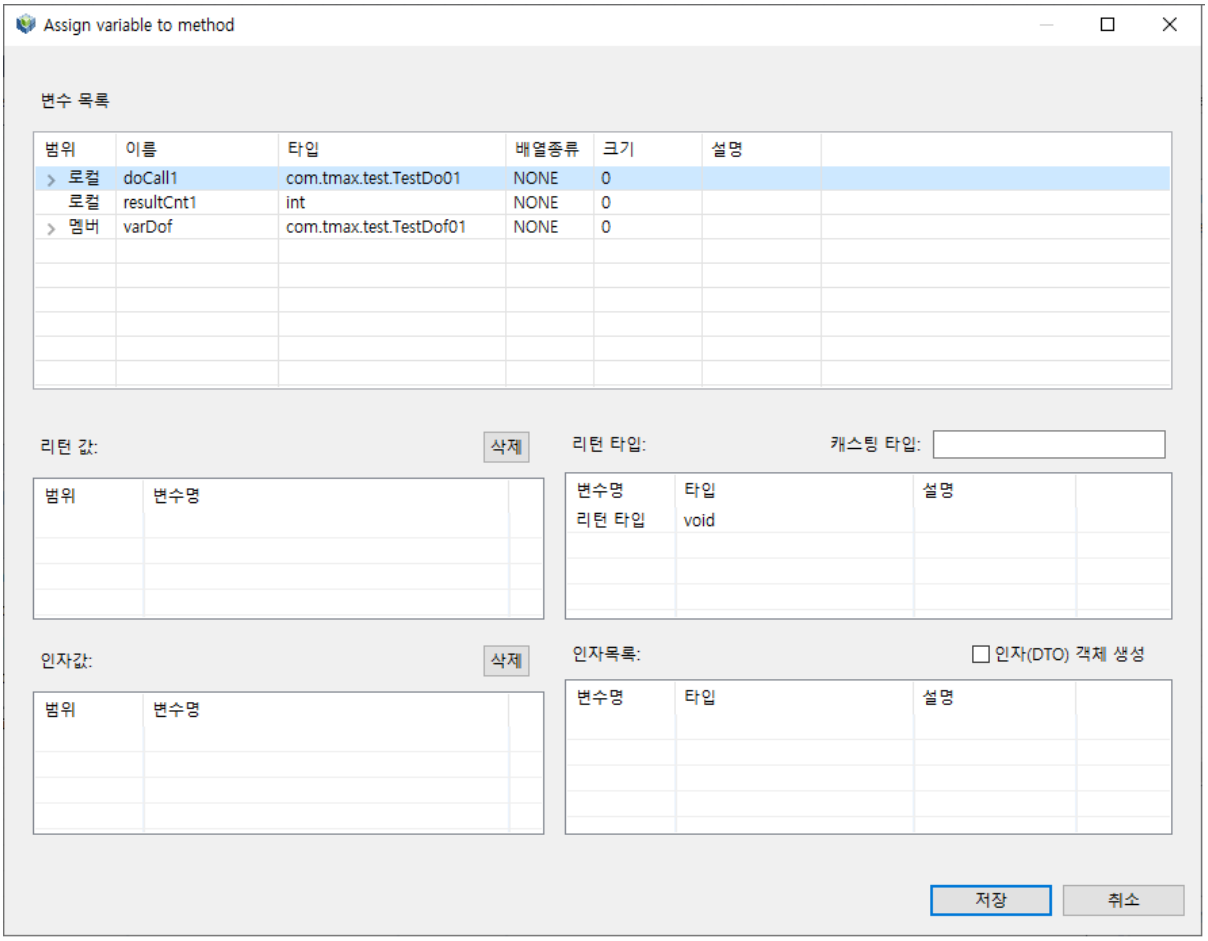

EMB Designer - 상위 메소드 - 디자인 영역

상위 메소드를 디자인 에디터에 생성하면 다음과 같이 자동 소스가 생성된다.

```
public void myFunc() throws Throwable
{
//[BEGIN_NODE_BLOCK, , myFunc()]
{

    //[BEGIN_NODE_BLOCK, 0, myFunc()]
    {
        //[BEGIN_PRE_ASSIGN_BLOCK, 0, myFunc()]

        //[END_PRE_ASSIGN_BLOCK, 0, myFunc()]
        super.foo();
        //[BEGIN_POST_ASSIGN_BLOCK, 0, myFunc()]

        //[END_POST_ASSIGN_BLOCK, 0, myFunc()]
    }
    //[END_NODE_BLOCK, 0, myFunc()]

}
//[END_NODE_BLOCK, , myFunc()]
}
```

EMB Designer - 상위 메소드 - 소스젠

3. 메소드 매핑이 필요한 경우 메소드 모듈의 **[+]**을 클릭하면 다음과 같은 매핑 화면이 나타난다.




EMB Designer - 상위 메소드 - 매핑

다음은 상위 메소드의 노드 속성과 Properties에 대한 설명이다.

- **상위 메소드 노드 속성**

  로컬 메소드 모듈을 클릭하면 아래와 같이 아이콘이 보이며 각각의 아이콘은 기능을 갖고 있다.


EMB Designer - 상위 메소드 - 노드 속성

  - (삭제)

    상위 메소드를 삭제한다.

  - (주석 처리)

    상위 메소드 호출부를 주석 처리한다.

  - (선할당 처리)

    상위 메소드를 호출하기 전 set할 값을 설정한다. Assign 모듈과 기능이 유사하다. 상위 메소드의 set 메소드에 전달될 변수 값을 assign한다. 선할당 처리 방법은 로컬 메소드의 선할당 처리동작과 동일한

방식으로 진행된다.

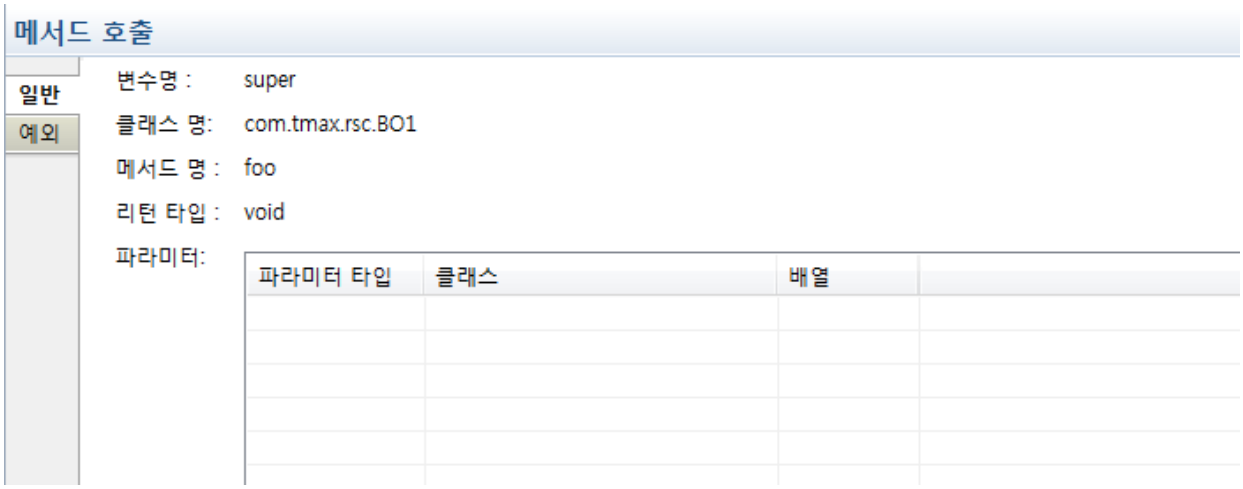
-  (후할당 처리)

  상위 메소드의 get 메소드에 전달될 변수 값을 assign한다. 로컬 메소드의 후할당 처리동작과 동일한 방식으로 진행된다.

- **[Properties]** 탭

  상위 메소드을 선택한 후 화면 하단의 **[Properties]** 탭에서 다음과 같은 특성을 설정할 수 있다.

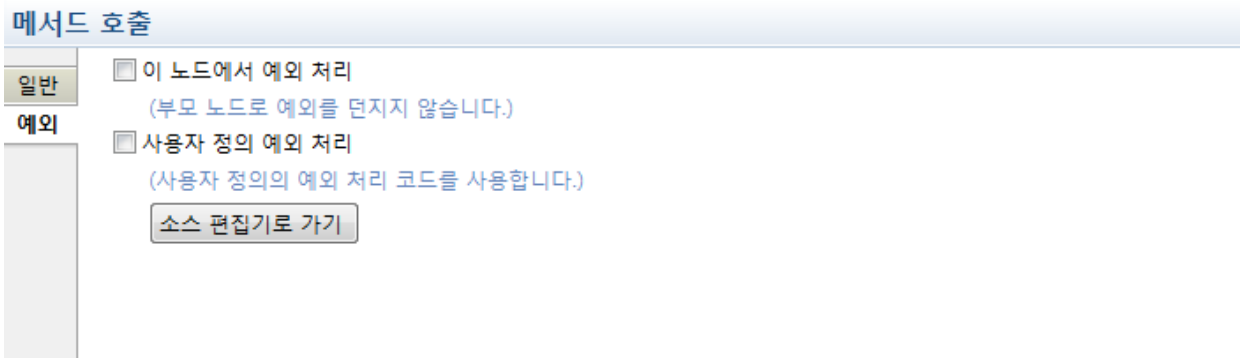
  - **[일반]** 탭




EMB Designer - 상위 메소드 - [Properties] - [일반] 탭

| 항목 | 설명 |
|---|---|
| 변수명 | 상위 메소드는 super 변수로 호출되므로 변수명은 'super'로 표기된다. |
| 클래스명 | 상위 메소드가 생성된 클래스명을 나타낸다. |
| 메소드명 | 호출된 메소드명을 나타낸다. |
| 리턴 타입 | 메소드의 리턴 타입을 나타내며 리턴 타입 변경이 없을 경우 기본적으로 'void"를 사용한다. |
| 파라미터 | 메소드 생성할 때 설정된 agrs 값을 보여준다. |

  - **[예외]** 탭




EMB Designer - 상위 메소드 - [Properties] - [예외] 탭

| 항목 | 설명 |
|---|---|
| 이 노드에서 예외 처리 | try-catch 문이 소스젠에 자동으로 생성되여 부모 노드로 예외를 던지지 않는다. 해당 항목이 체크되어 있어야 **'사용자 정의 예외처리'** 항목이 활성화된다. |
| 사용자 정의 예외 처리 | 소스에서 개발자가 직접 catch 문을 추가할 수 있다. |
| **[소스 편집기로 가기]** | 상위 메소드 모듈의 try-catch 해당 소스 라인으로 이동한다. |

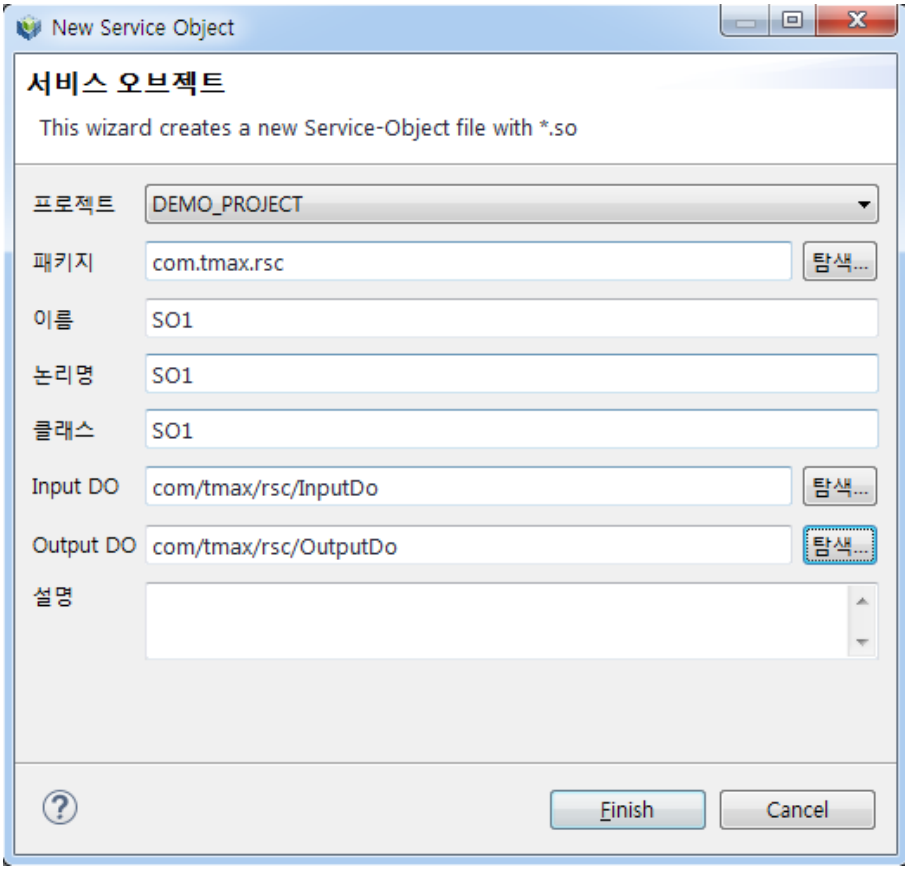
# 4.4. Service Object (SO)

Service Object(SO)는 서비스를 수행 및 관리하는 오브젝트로 비지니스 오브젝트를 Orchestration하는 플로우 중심 애플리케이션이다.

## 4.4.1. SO 생성

다음은 SO를 생성하는 과정에 대한 설명이다.

1. **Package Explorer**의 컨텍스트 메뉴에서 **[New] > [Service Object]**를 선택한 후 **New Service Object 화면**에서 SO를 생성하기 위한 모듈의 기본 정보를 등록한다.



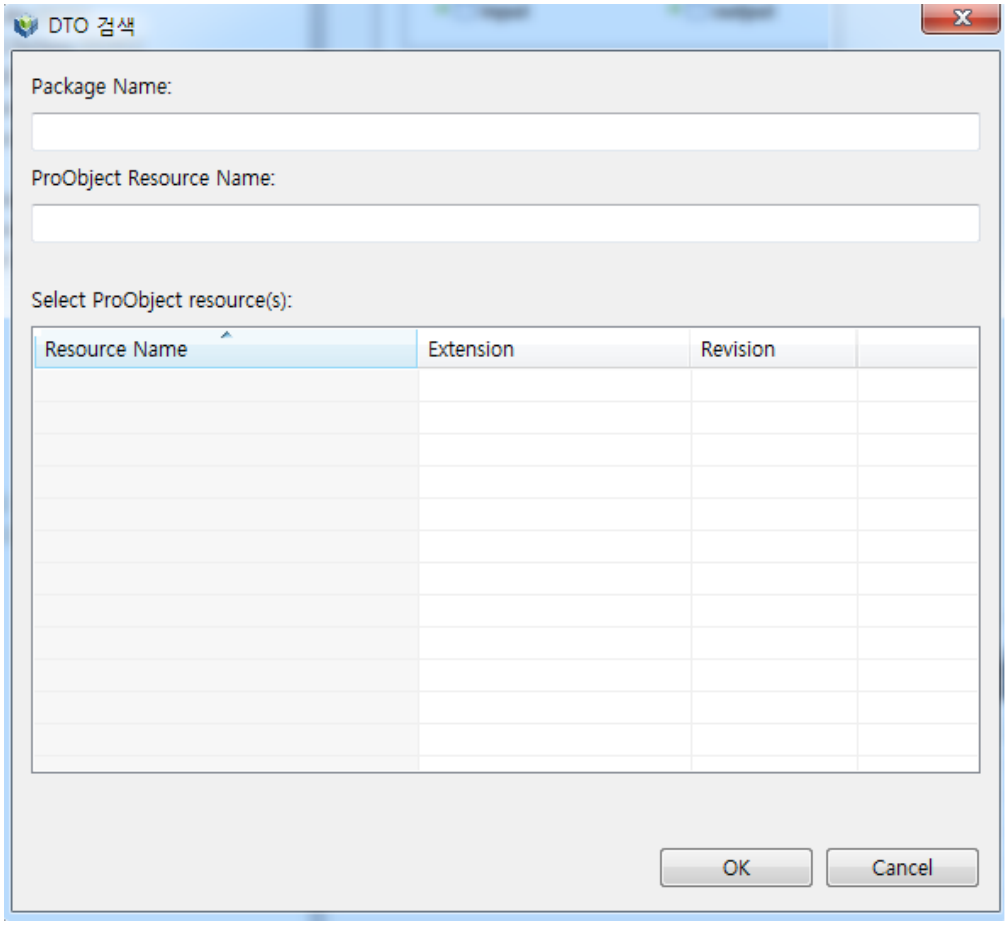

New Service Object 화면 - SO 생성

| 항목 | 설명 |
|---|---|
| 프로젝트 | 생성할 SO가 위치할 실제 프로젝트명을 입력한다. |
| 패키지 | 실제 사용하는 패키지 네이밍 규칙에 따라 패키지명을 정의한다. |

| 항목 | 설명 |
| --- | --- |
| 이름 | 생성할 SO의 물리명을 입력한다. |
| 논리명 | 생성할 SO의 논리명을 입력한다. |
| 클래스 | 생성할 SO의 클래스명을 입력한다. |
| Input DO | 생성할 SO의 입력 DO를 지정한다.<br><br>외부로 호출 받아 처리되는 부분이므로 입력 전문에 대한 반드시 정의해야 한다. |
| Output DO | 생성할 SO의 출력 DO를 지정한다.<br><br>외부로 호출 받아 처리되는 부분이므로 출력 전문에 대한 반드시 정의해야 한다. |
| 설명 | 생성할 SO에 대한 부가 설명을 입력한다. |

다음은 Input DTO를 지정하는 과정에 대한 설명이다.

a. **New Service Object 화면**에서 **'Input DO'** 항목의 **[탐색...]** 버튼을 클릭하면 **Search DO 화면**이 나타난다.



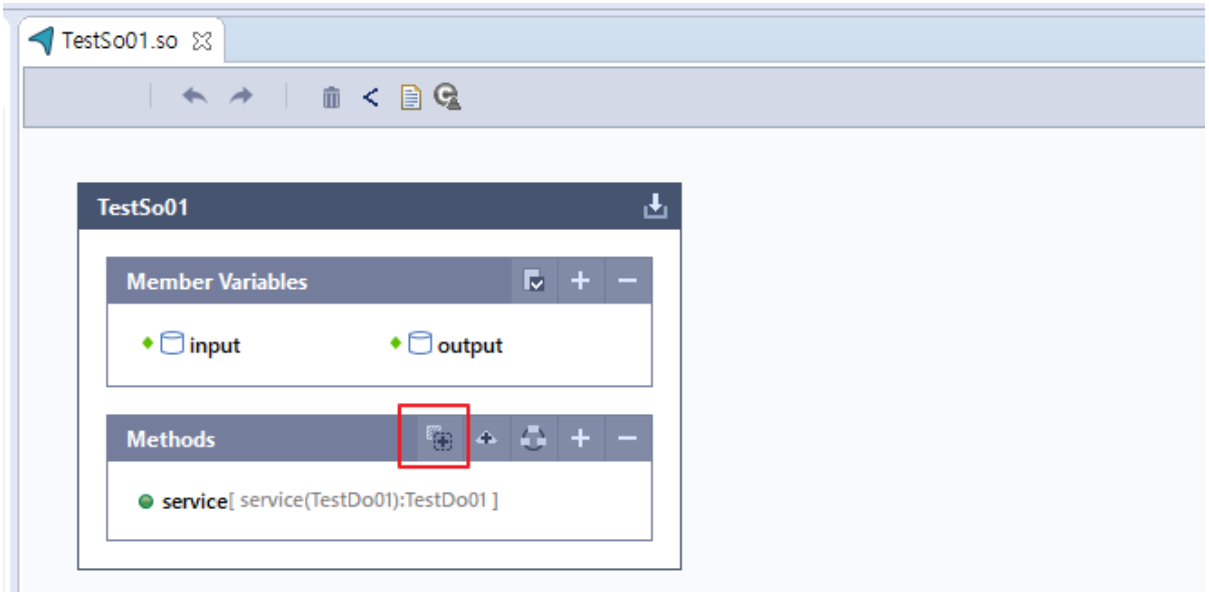

New Service Object - Search DO 화면

b. **'리소스명'** 항목에 내용을 입력하면 자동으로 리소스가 검색되어 **리소스 선택 화면**에 리소스 목록이 나열된다.

c. 리소스를 선택한 후 **[OK]** 버튼을 클릭하면 **New Service Object 화면**의 **'Input DO'**에 리소스 내역이 지정된 것을 확인할 수 있다. Input DTO와 동일한 방법으로 Output DO를 지정한다.

2. **New Service Object 화면**에서 모듈의 기본 정보를 모두 설정한 후 **[Finish]** 버튼을 클릭하면 **Service Flow Editor 화면**으로 이동한다.

   **[비구현 메서드 추가]** 아이콘을 선택하면 기본 SO 메소드 이름은 'service', 리턴 타입은 'Output DO'로 기본 설정된다.



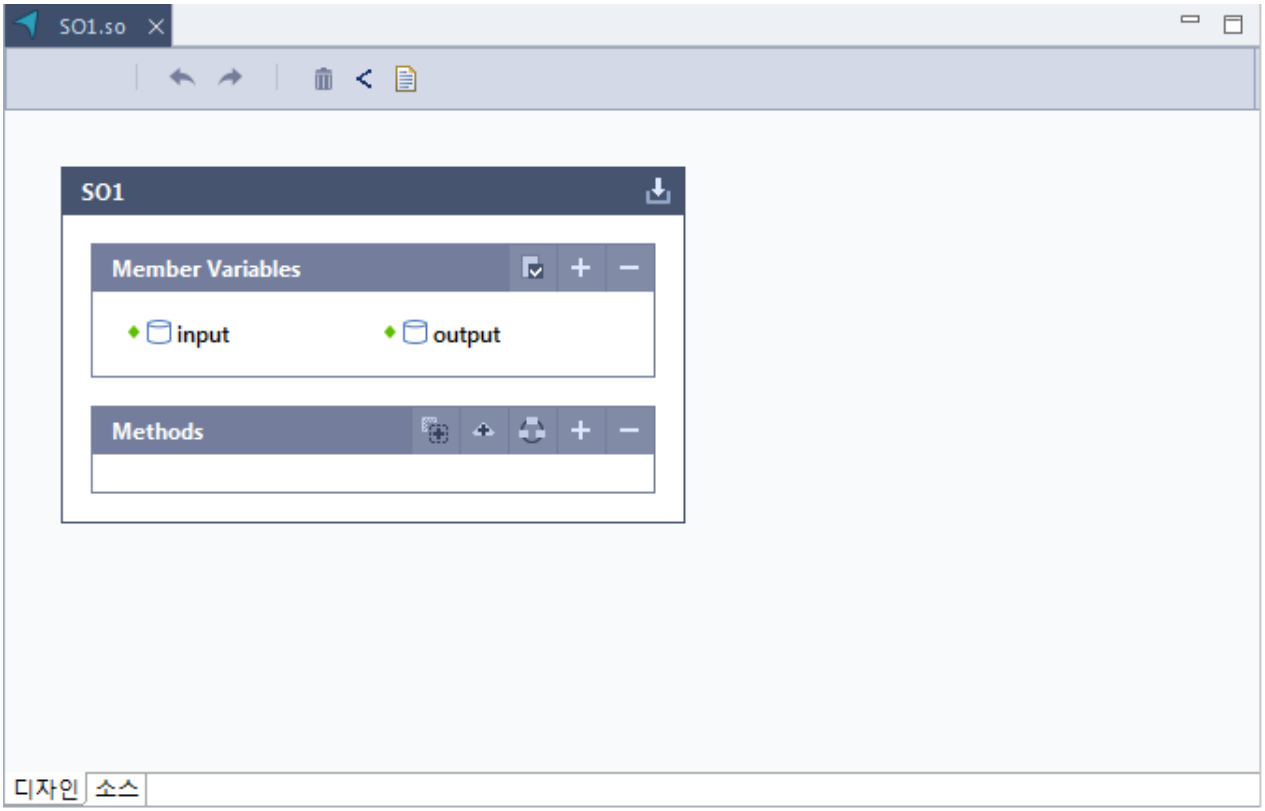

Service Object - 비구현 메소드 추가

## 4.4.2. Design Editor

다음은 기본 디자인 에디터이며 기본 정보를 등록한 후 첫 화면으로 보이는 디자인 에디터이다.

기본 정보를 등록할 때 입력했던 물리명으로 클래스를 생성하며, 논리명은 최상위 박스의 이름으로 보인다. SO를 생성하는 경우 지정한 Input DO 및 Output DO가 맴버변수로 추가된다.

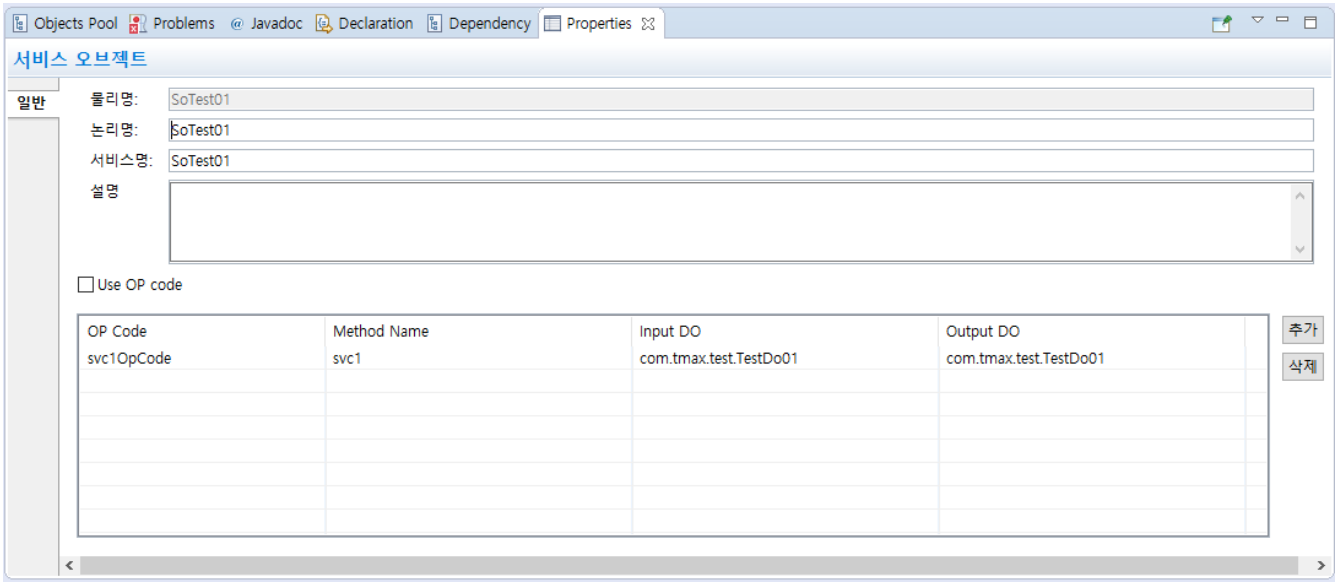
멤버 변수와 메소드 박스에서 **[+]** 버튼을 클릭하면 해당 SO의 멤버 변수와 메소드를 선언할 수 있다. 자세한 내용은 툴바, 멤버 변수, 메소드를 참고한다. SO의 Design Editor의 추가 기능은 내용은 하위 단락(서비스 오브젝트 일반 속성)에서 설명하도록 한다.




SO 디자인 에디터

디자인 에디터의 여백에 마우스 오른쪽 버튼을 클릭하면 아래와 같은 **서비스 오브젝트 화면**에 표시된다.

**[Properties] > [일반]** 탭에서 SO의 리소스 정보 및 Op Code Method를 관리한다. **[추가]** 버튼을 클릭하면 **OP code 목록**에 OP Code를 추가하고, **[삭제]** 버튼을 클릭하면 **OP code 목록**에서 선택된 OP Code를 삭제할 수 있다.




디자인 에디터 - 일반 속성

- **기본 정보**

| 항목 | 설명 |
|---|---|
| 물리명 | ProStudio에서 생성하는 리소스의 식별 기준이 되는 Physical Name이다. |

| 항목 | 설명 |
|---|---|
| 논리명 | 리소스의 가독성을 위한 물리명과 1:1로 매치되는 간략한 이름이다. |
| 서비스명 | 서비스의 위한 기준이 되는 이름이다. |
| 설명 | 서비스에의 상세 Spec 등을 기술한다. |
| Use OP code | Op Code Method 기능의 사용 여부이다.<br><br>체크박스를 체크하는 경우 OP code를 사용하며 Executor Source가 전문의 opCode 값에 의해 분기 가능하도록 변경된다. 자세한 설명은 "OP Code 사용"을 참고한다. |

- **OP code 목록**

| 항목 | 설명 |
|---|---|
| Op Code | OP Code명이다. |
| Method Name | OP Code Method Name이다. Method Name 기준으로 디자인 에디터의 메소드로 추가되어 관리된다. |
| Input DO | Input DO명을 입력한다. **[...]** 버튼을 클릭하여 대상 DO를 선택한다. |
| Output DO | Output DO명을 입력한다. **[...]** 버튼을 클릭하여 대상 DO를 선택한다. |

### OP Code 사용

OP Code는 하나의 서비스만 처리 가능한 구조를 확장하기 위해서 사용한다. 하나의 서비스에는 ServiceObject를 Overrice한 1개의 서비스를 만드는 것이 일반적이다. 여러 개의 서비스 메소드의 지원을 위한 기능이 OP Code이다.

디자인 에디터의 일반 속성 화면(디자인 에디터 - 일반 속성)의 **'Use OP code'** 항목을 체크하는 경우 아래 예시와 같이 Executor Source가 전문의 opCode 값에 의해 분기 가능하도록 변경된다. 아래 예시와 같이 입력된 Header에 "opCode" 노드에 값이 존재하는 경우 해당 매칭되는 OP Code Method를 처리하게 된다. Default(Use OP code가 체크 되지 않았을 때)는 ServiceObject를 Overrice한 메소드를 사용한다.

다음은 **'Use OP code'** 항목을 체크하는 경우 Executor 발췌한 내용이다.

```
package com.tmax.test;

import com.tmax.proobject.engine.service.executor.ServiceExecutor;
import com.tmax.proobject.engine.promapper.ProMapperUtil;
import com.tmax.proobject.engine.servicemanager.ServiceContext;
import com.tmax.proobject.engine.servicemanager.thread.ServiceWorkerThread;

public class TestSo01Executor extends ServiceExecutor
{
   public TestSo01Executor() {
     serviceObject = new TestSo01();
   }

   public Object execute(Object serviceInput, String serviceExecutionMethod) throws Throwable {
      ServiceWorkerThread worker = (ServiceWorkerThread)Thread.currentThread();
      ServiceContext serviceContext = (ServiceContext)worker.getExecutingServiceInfo();
      String opCode =
serviceContext.getServiceRequestContext().getRequest().getHeader().get("opCode");
```

```
      ClassLoader classLoader = serviceContext.getServiceRequestEvent().getServiceGroupClassLoader();

      Object unmarshaledInput = ProMapperUtil.unmarshal(serviceContext.getServiceRequestEvent(),
serviceInput,
         serviceContext.getServiceRequestEvent().getServiceMeta().getInputDto(), classLoader);

      Object output;
         switch( opCode ) {
          case "svc1OpCode" :
             output = ((TestSo01) serviceObject).svc1( (com.tmax.test.TestDo0001) unmarshaledInput);
             break;
          default :
            output = serviceObject.service(unmarshaledInput);
          }
     if(serviceContext.getDelayedServiceCall() != null) {
          return null;
       } else {
          Object marshaledOutput = ProMapperUtil.marshal(serviceContext.getServiceRequestEvent(),
output,
             serviceContext.getServiceRequestEvent().getServiceMeta().getOutputDto(), classLoader);
          return marshaledOutput;
       }
  }

   public String getRendezvousMethodName(String service){
      return null;
   }
}
```

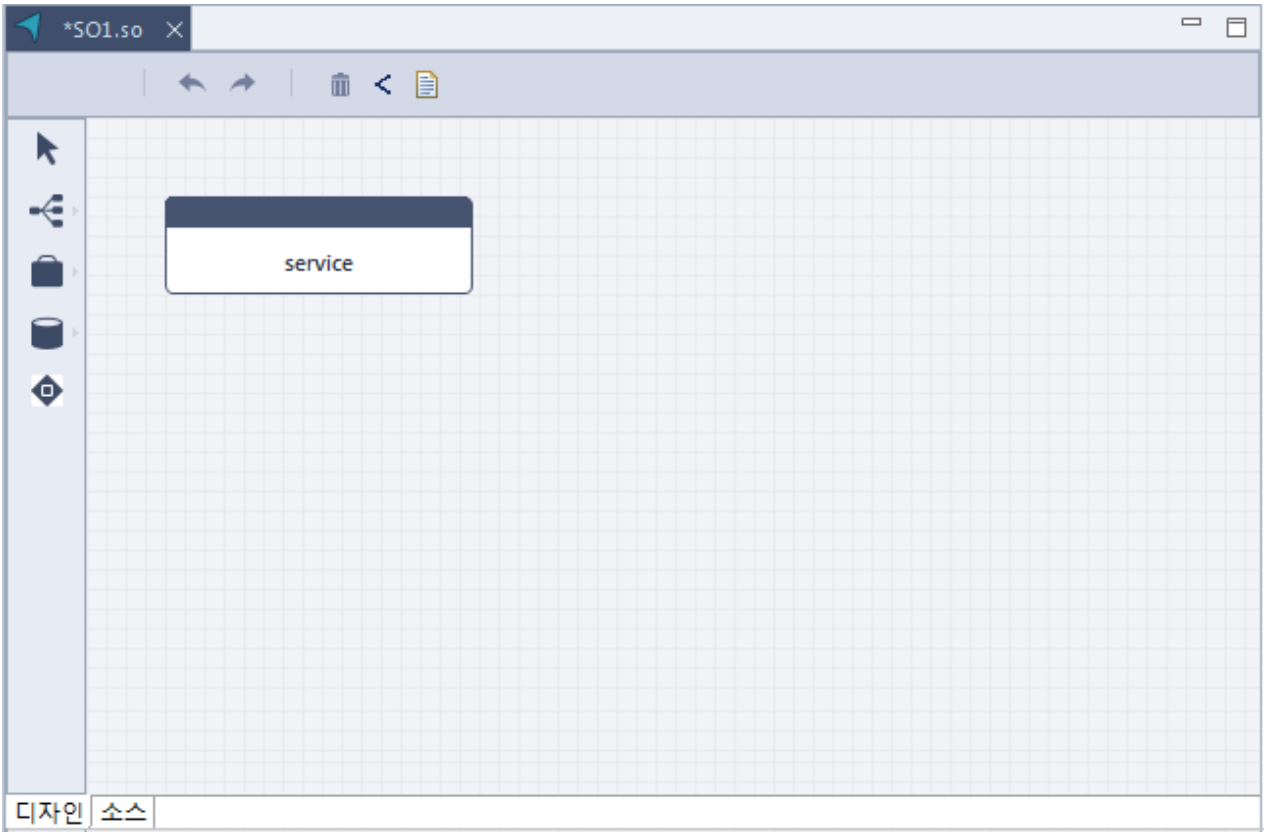
### 4.4.3. EMB Designer

**EMB Designer**는 메소드 내용을 플로우 설계, 플로우 편집, 소스 편집, 플로우 삭제 등 개발자가 실질적으로 개발할 수 있는 영역으로 EMB 모듈의 플로우를 가시적으로 나타낸다. 사용자는 Palette에서 다양한 모듈을 가져다 드래그 앤드 드롭으로 추가할 수 있다. 자세한 내용은 EMB Designer를 참고한다.

EMB Designer의 소스 에디터와 디자인 에디터를 이동하면서 편집하기에는 불편하므로 소스 에디터의 원하는 위치에서 이너 모듈, 버츄얼 모듈, 루프 모듈을 추가할 수 있다.

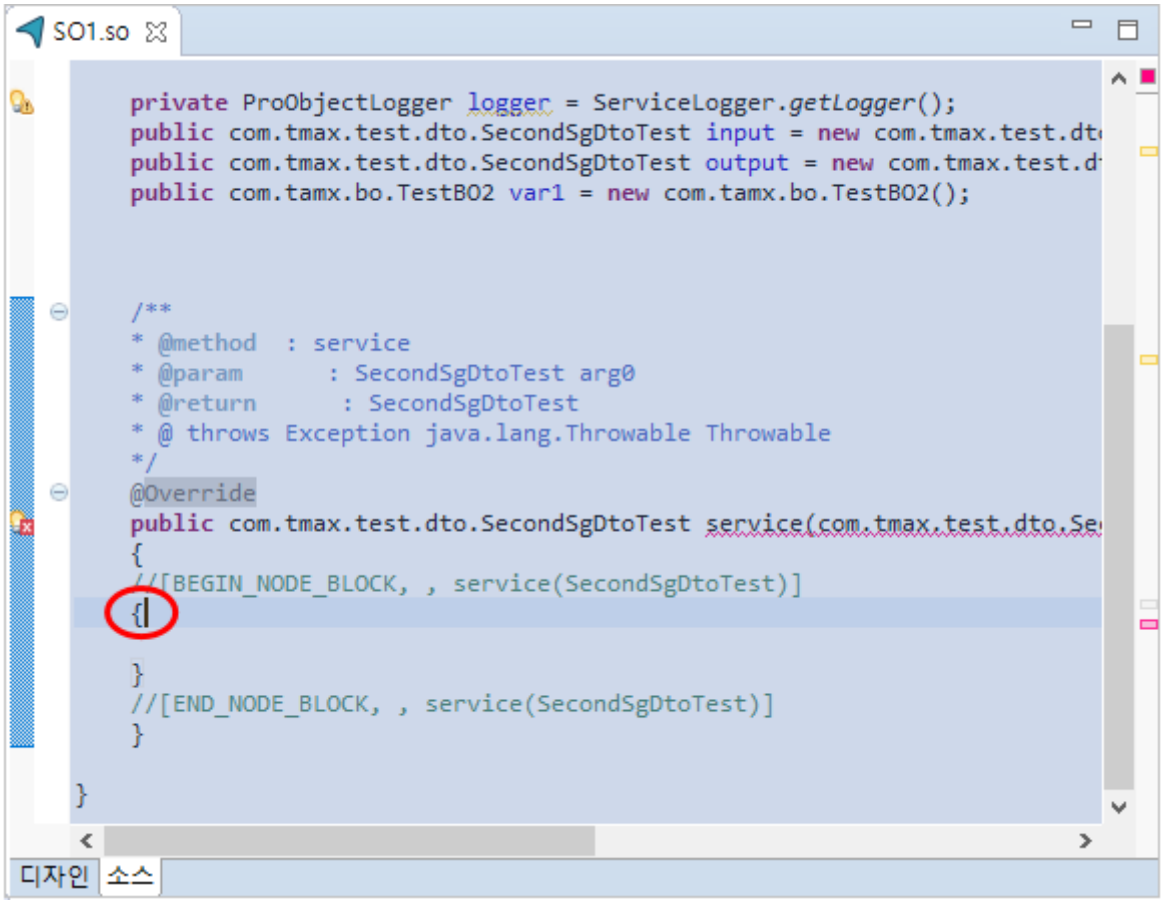
다음은 소스 에디터에서 모듈을 추가하는 방법에 대한 설명이다.

1. EMB Designer에서 모듈을 추가할 SO를 생성한다(SO 생성 참고).



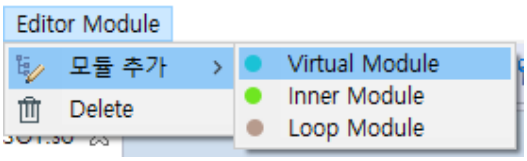
EMB Designer

2. EMB Designer의 **[소스]** 탭을 선택한 후 모듈을 추가할 위치에 마우스를 이동한다.



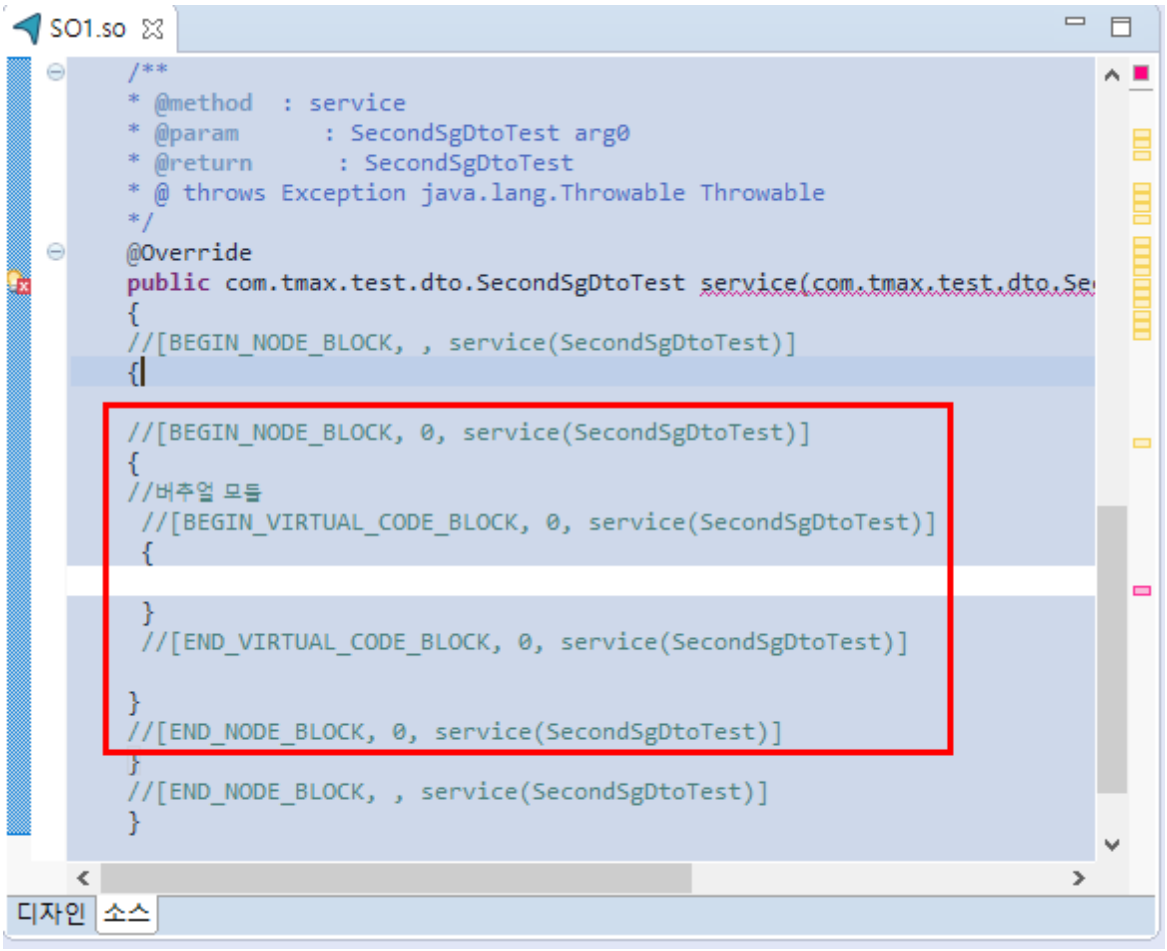
EMB Designer - [소스] 탭

3. **[에디터 모듈 생성]** 버튼을 클릭한 후 **[모듈 추가] > [Vertual Modeule]**을 선택한다. **[에디터 모듈 생성]** 버튼은 **[소스]** 탭에서만 활성화되고 모듈을 생성하거나 삭제할 수 있다.

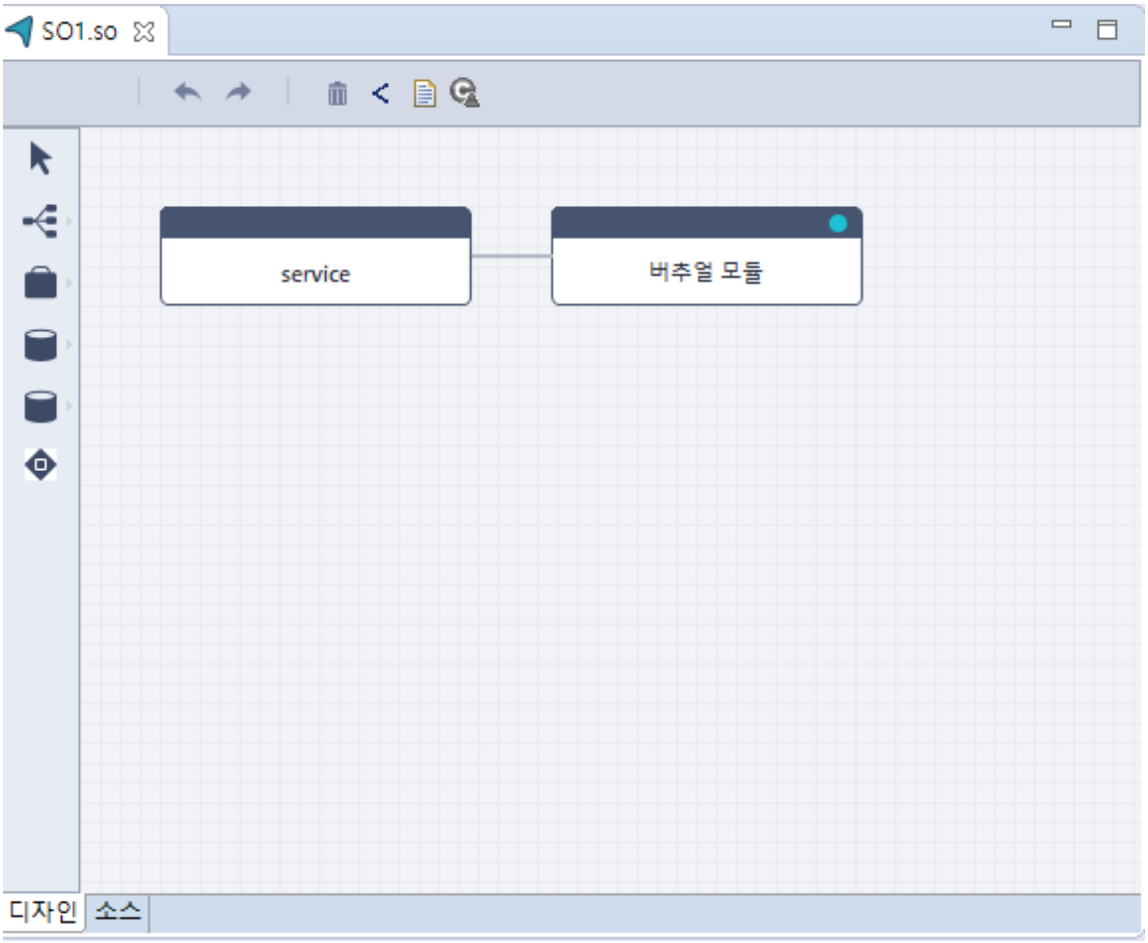



EMB Designer - [소스] 탭 - 에디터 모듈 생성 버튼

4. 선택한 모듈별 소스 코드가 소스 에디터에 추가된다. 다음은 소스 에디터에 버츄얼 모듈(소스)이 추가된 내용이다.




EMB Designer - [소스] 탭 - 버츄얼 모듈 추가 (소스)



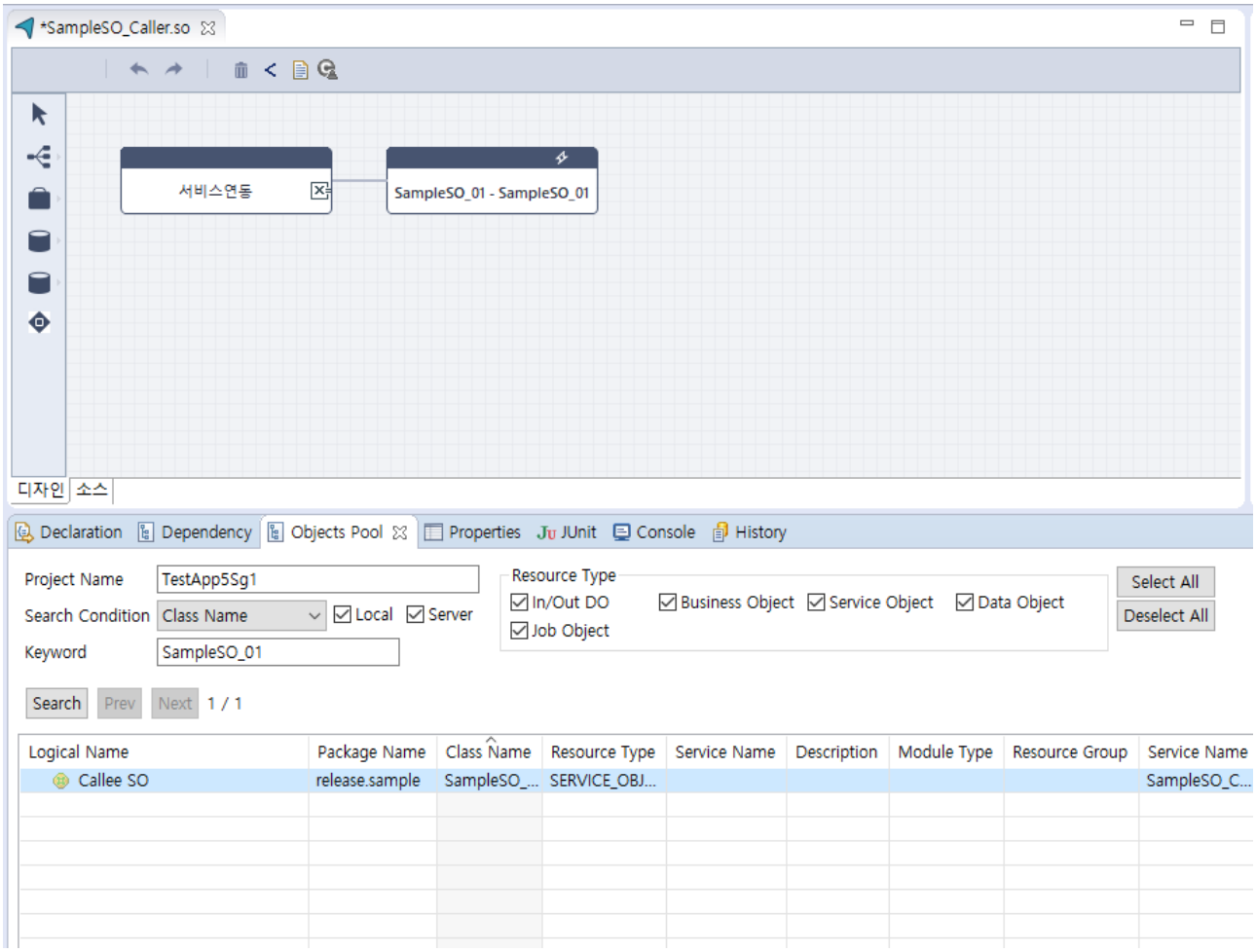
EMB Designer - [소스] 탭 - 버츄얼 모듈 추가 (디자인)

## 4.4.4. Service Object(SO) 연동

이미 개발된 SO를 연동하여 동기(Synchronous), 비동기(Asynchronous) Call 등을 수행하여 다양한 유형의 업무 유형을 처리할 수 있다.

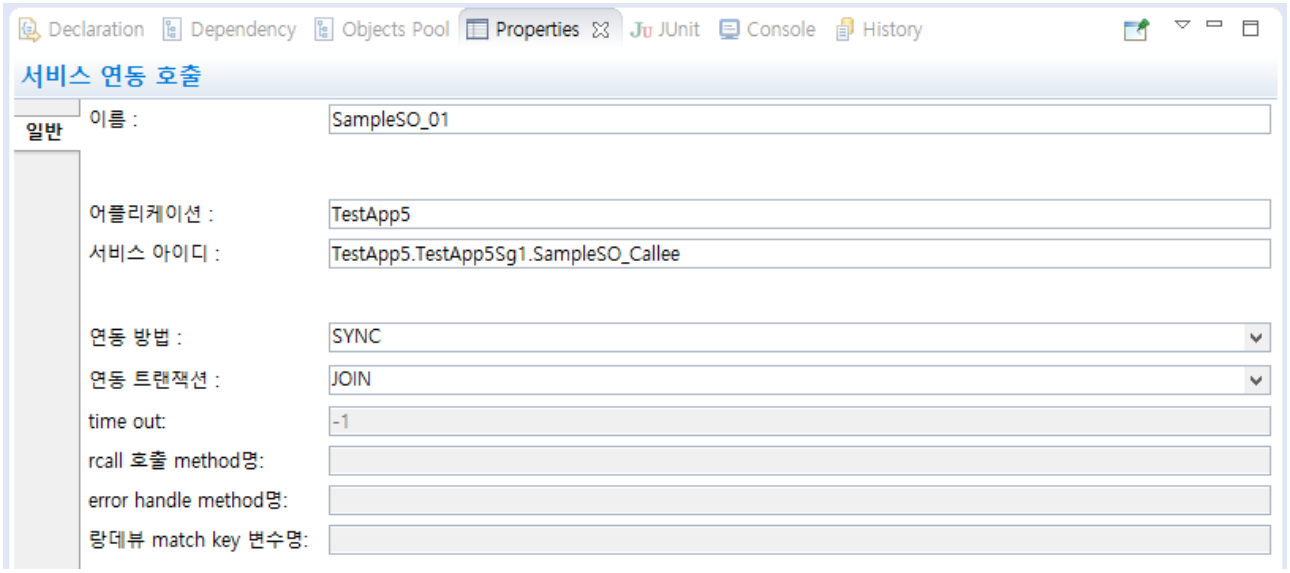
다음은 EMB의 노드에서 연동 모듈을 추가하는 과정에 대한 설명이다.

1. View 영역의 **[Object Pool]** 탭에서 연동할 SO를 조회한 후 연동할 SO를 드래그 앤드 드롭으로 EMB Flow Editer에 추가한다.




SO연동 - SO 연동 노드 추가

2. 추가된 연동 Node를 업무 목적에 맞게 설정한다. 자세한 개발 방법은 "ProObject 런타임 엔진 개발자 안내서"를 참고한다.




SO연동 - SO 연동 노드 설정

| 항목 | 설명 |
|---|---|
| 이름 | EMB Flow에 표시 및 소스의 주석에 사용되는 이름이다. |
| 어플리케이션 | Callee의 애플리케이션명이다. |

| 항목 | 설명 |
| --- | --- |
| 서비스 아이디 | 실제로 Call되는 SO이며, Package+Class명(SO명)이다. |
| 연동방법 | SO의 연동 방법은 다음과 같다.<br>• SYNC : 동기(Synchronous)을 의미한다. 런타임 API기준 ServiceManager.call이다.<br>• ASYNCREPLY : 비동기(Asynchronous) Call을 의미하며, 결과 값을 받는다. 런타임 API는 ServiceManager.acall이다. 결과값을 받기 위해서는 연동 호출 이후 ServiceManager.getReply를 해야 한다.<br>• ASYNCNOREPLY : 비동기(Asynchronous) Call을 의미하며, 결과 값을 받지 않는다. 런타임 API는 ServiceManager.acall_noreply이다.<br>• RETURNCALL : call 방식과 유사하며, 서비스를 동기적(Synchronous)으로 요청하는 경우에 사용한다. 연동할 때도 단일 Thread로 처리 가능하다. 런타임 API는 ServiceManager.rcall이다.<br>• FORWARD : 전문 및 Response 권한을 Callee에게 넘기긴다. 런타임 API는 ServiceManager.forward이다. |
| 연동 트랜잭션 | Caller와 Callee의 Transaction을 하나로 할 것인지를 나타 낸다.<br>• NONE : 해당 사항이 없음(acall, acall_noreply에서 지원하지 않는다.)<br>• JOIN : Caller와 Callee의 트랜잭션을 하나로 처리한다. 런타임의 Type은 TransactionType.JOIN이다.<br>• SPLIT : Caller와 Callee의 트랜잭션을 각각 처리한다. 런타임의 Type은 TransactionType.SPLIT이다. |
| time out | Time out 시간을 설정한다. |
| rcall 호출 methode명 | RCall한 메소드명을 설정한다. |
| error handle method명 | RCall할 때 사용할 Error Handler를 설정한다. |
| 랑데뷰 match key 변수명 | 랑데뷰 Match Key를 설정한다. |

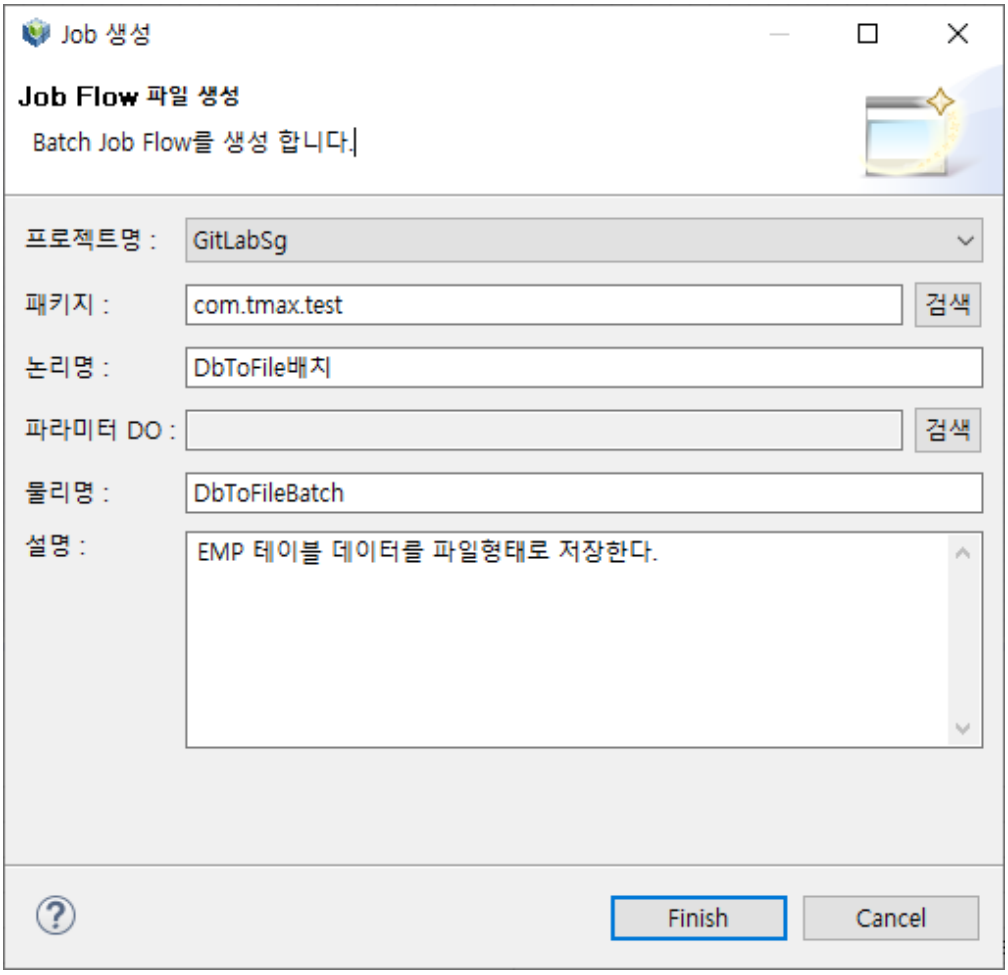
# 4.5. Job Object(JO)

배치 JOB 개발을 위한 표준 프레임 및 대용량 배치 처리를 위한 아키텍처 제공하며 배치 개발 및 운영 편의성을 위한 스케줄러를 제공한다. Job Object는 배치를 수행할 때 태스크를 정의하고 데이터 가공할 BO의 수행 순서 및 런타임에 Thread 및 DB 세션관리를 제어할 수 있다.

## 4.5.1. JO 생성

다음은 JO를 생성하는 과정에 대한 설명이다.

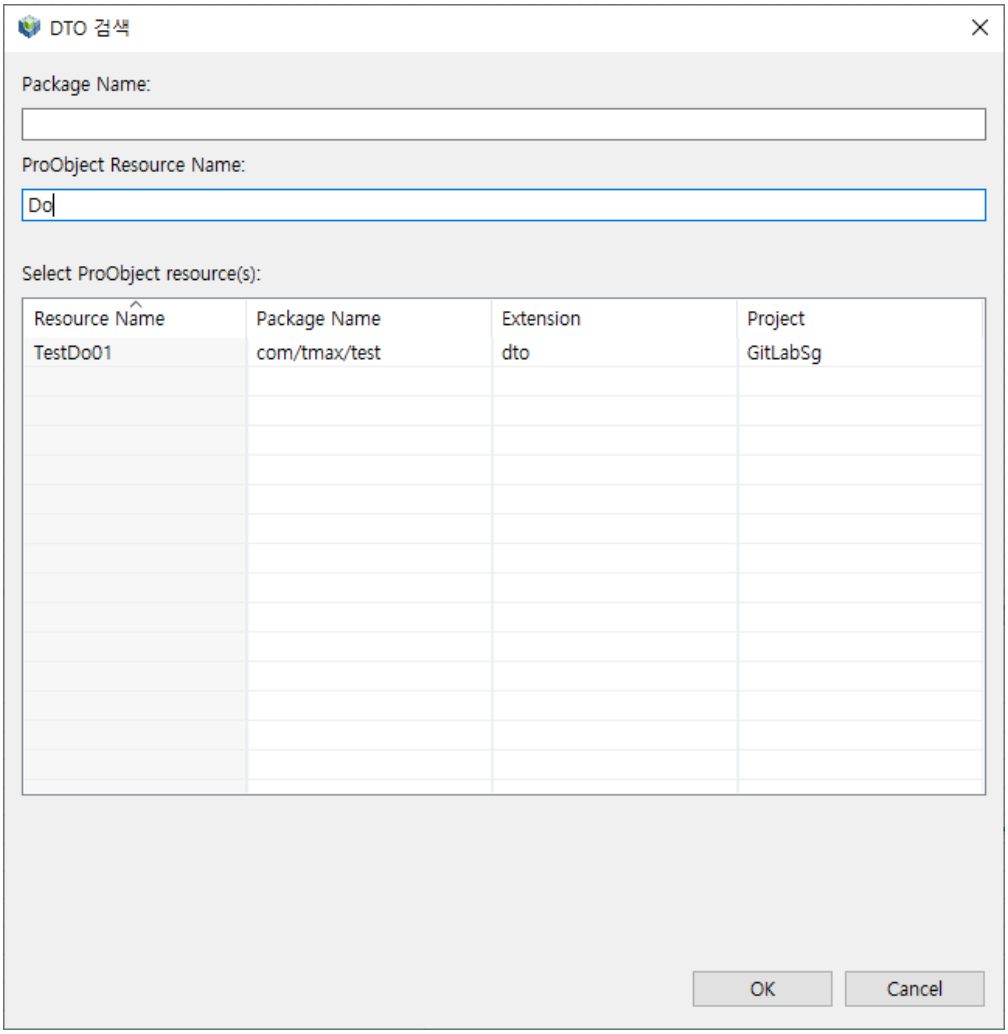
1. **Package Explorer**의 컨텍스트 메뉴에서 **[New] > [JOB Object]**를 선택한 후 **JOB 생성 화면**에서 Batch Job 플로우를 생성하기 위한 모듈의 기본 정보를 등록한다.



JOB 생성 화면

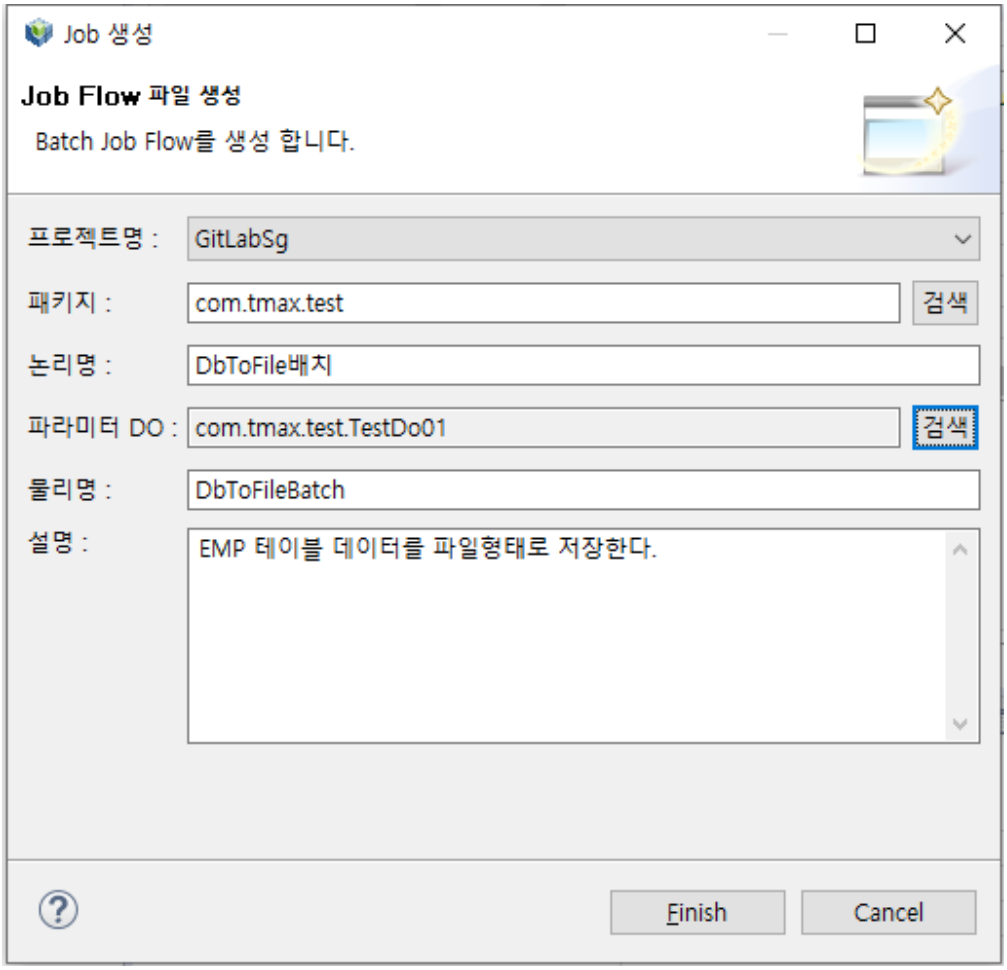
| 항목 | 설명 |
|---|---|
| 프로젝트명 | 생성할 JO가 위치할 실제 프로젝트명을 입력한다. |
| 패키지 | 실제 사용하는 패키지 네이밍 규칙에 따라 패키지명을 정의한다. |
| 논리명 | 생성할 Job 플로우의 논리명을 입력한다. |
| 파라미터 DO | JO에서 사용될 배치 입력 파라미터 값을 설정한다. 설정할 파라미터는 **[검색]** 버튼을 클릭하면 **Search DTO 화면**(JOB 생성 화면 - Search DTO화면)에서 선택한다. |
| 물리명 | 생성할 Job 플로우의 논리명을 입력한다. |
| 설명 | 생성할 Job 플로우에 대한 부가 설명을 입력한다. |

**'파라미터 DO'**항목에서 **[검색]** 버튼을 클릭하면 **Search DTO 화면**이 나타난다. 목록에서 등록할 항목을 선택한 후 **[OK]** 버튼을 클릭한다.



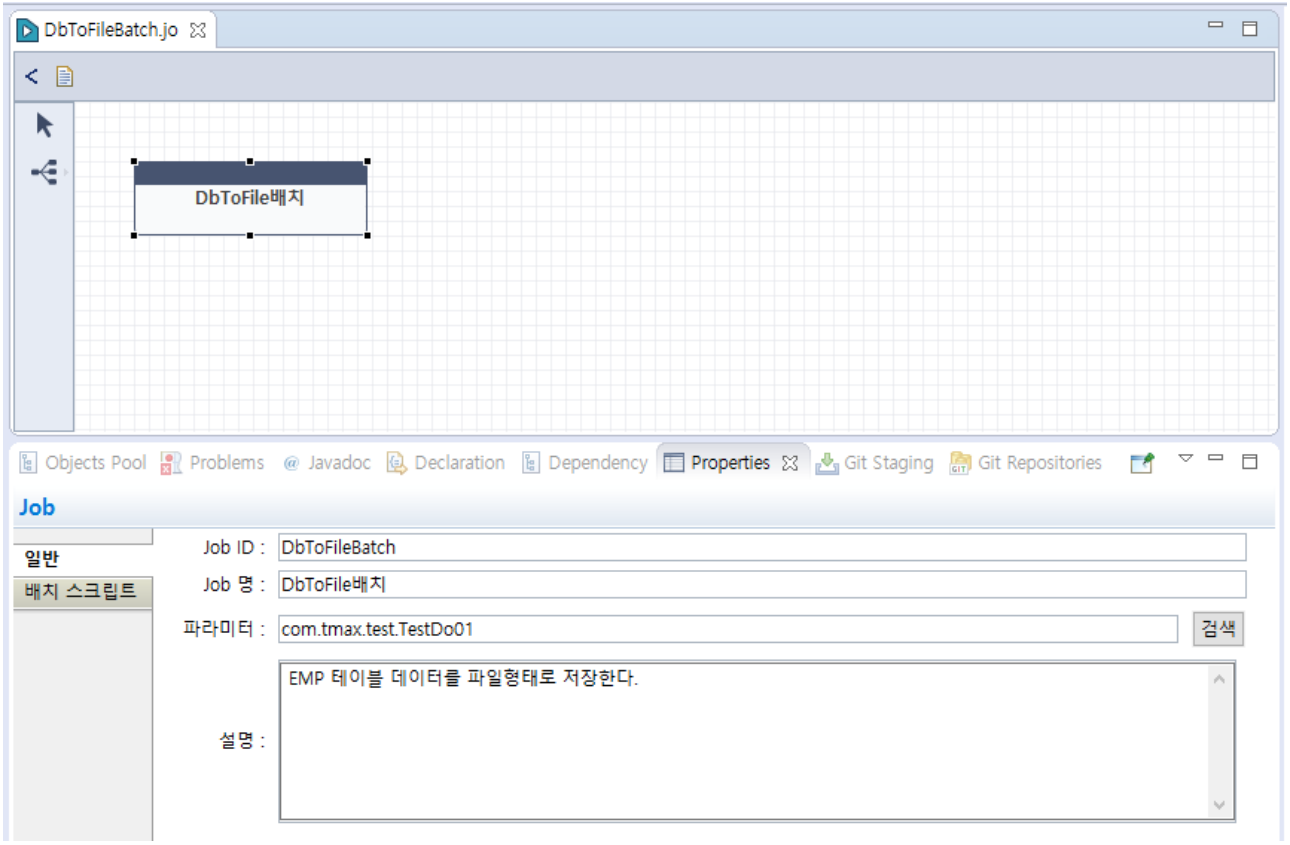

JOB 생성 화면 - Search DTO화면

**JOB 생성 화면**에서 파라미터 DO가 등록된 것을 확인할 수 있다.



JOB 생성 화면 - 파라미터 설정

2. **JOB 생성 화면**에서 모듈의 기본 정보를 모두 설정한 후 **[Finish]** 버튼을 클릭하면 Job Object Task를 편집할 수 있는 에디터로 이동한다.




Job 일반 화면

3. 생성한 JO의 Job 일반 화면에서 **[배치 스크립트]** 탭을 클릭한 후 **[Shell 배포]** 버튼을 클릭하면 배치 Shell을 배포할 수 있다.


Job 배포

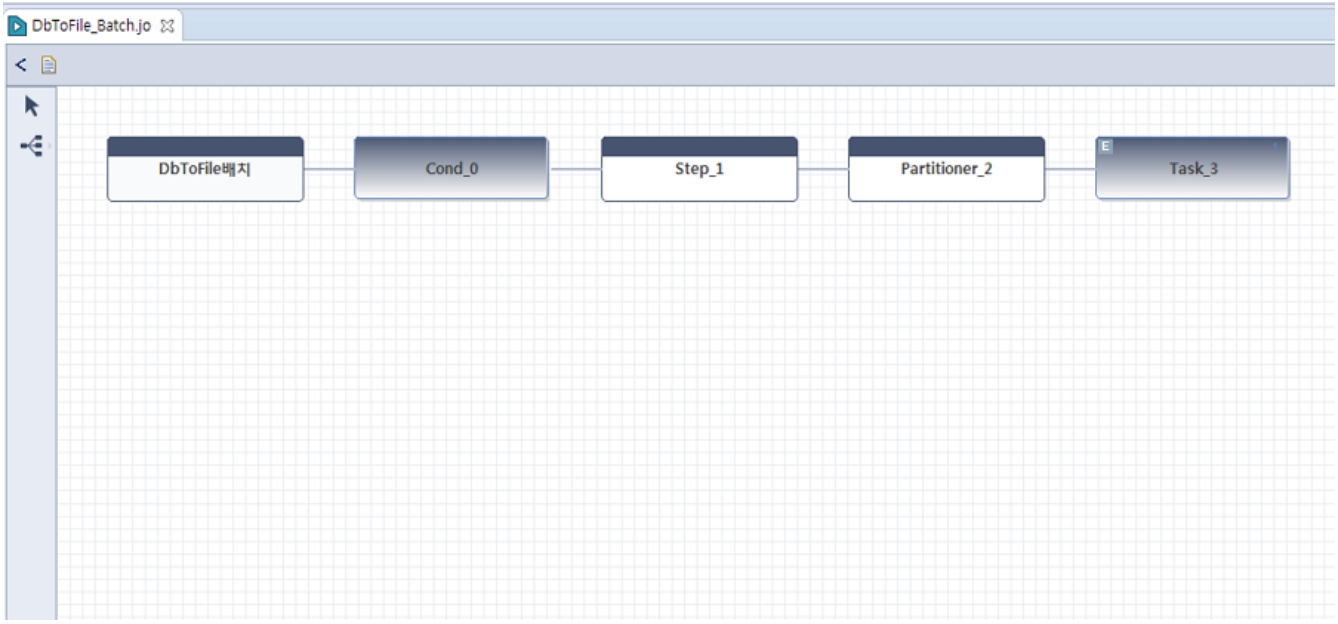
배치 Shell 배포를 위해서는 아래와 같이 PoDevSvr.xml에 배포할 Path Config인 GENERATE_JOB_SCRIPT_LOCATION가 설정되어 있어야 한다.

```
<?xml version="1.0" encoding="EUC-KR" standalone="yes"?>
<serverConfig xmlns="http://www.tmax.co.kr/proobject/serverConfig">
    :
  중략
    :
  <configField id="GENERATE_JOB_SCRIPT_LOCATION"
value="/home/newdev/proobject/resources/jobShell" type="String" xmlns=""/>
</serverConfig>
```

## 4.5.2. Design Editor

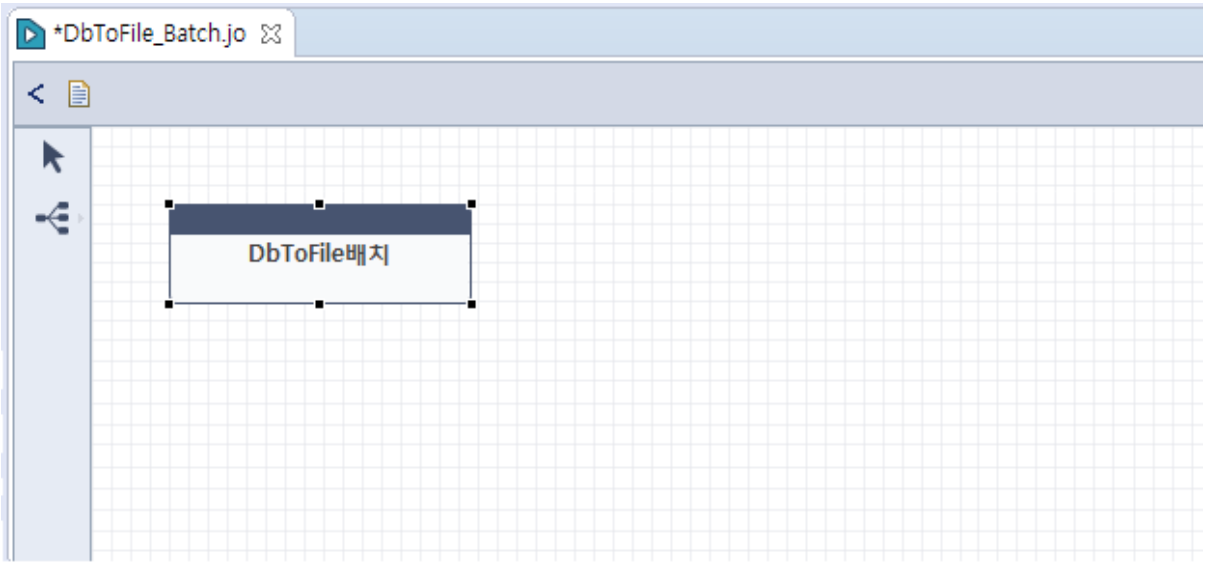
**디자인 에디터**는 JO 생성하는 경우 조회되는 초기 화면으로 Job Task를 등록, 구성하는 화면이다.

JO를 디자인할 수 있는 에디터로 Task를 조립, 연결할 수 있으며 Properties를 이용하여 사용할 오브젝트를 선택할 수 있다.




JO 디자인 에디터 예

JO 최상위 노드로 JOB명, 파라미터 등록 등 기본 정보를 설정한다. JO의 시작점으로 다음과 같이 JO 디자인 노드의 시작이다.




JO 디자인 에디터

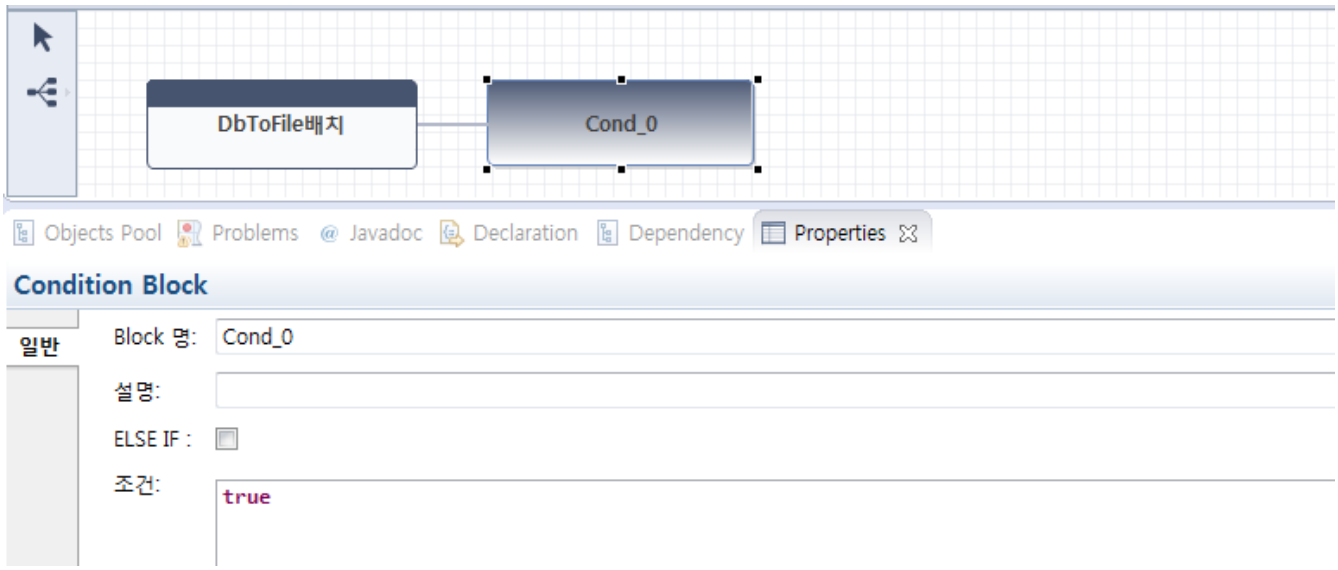
JO 디자인 영역에 추가할 수 있는 모듈은 Condition Block, Step Block, Partitioner Block, ETL Task, Online Task, Normal Task 6가지로 구분된다.

| 메뉴 | 설명 |
| --- | --- |
| (Nodes) | ◦ **[Condition Block]** : 조건에 따른 분기를 지원한다. 자세한 내용은 Condition Block을 참고한다.<br>◦ **[Step Block]** : 자세한 내용은 Step Block을 참고한다.<br>◦ **[Partitioner Block]** : 자세한 내용은 Partitioner Block을 참고한다.<br>◦ **[ETL Task]** : 사용자의 설정에 따라 수집/가공/적재 순의 데이터 처리를 병렬로 처리한다. 자세한 내용은 ETL Task를 참고한다.<br>◦ **[Online Task]**: 단일 건수 처리 온라인 서비스를 병렬로 일괄 기동하여, 비즈니스 로직의 재사용성과 처리 성능을 극대화한다. 자세한 내용은 Online Task를 참고한다.<br>◦ **[Normal Task]**: ETL Task와 같이 수집/가공/적재로 데이터를 처리하는 경우 데이터를 가공할 때 별도의 BO 등록 및 개발해야 하는 불편을 해소하고 기존 프레임워크에 익숙한 개발자에게 편의성을 제공한다. 자세한 내용은 Normal Task를 참고한다. |

#### 4.5.2.1. Condition Block

Condition Block을 통해 조건에 따른 분기를 지원한다.

**Palette**의 **[Condition Block]**을 선택한 후 디자인 화면에 드래그 앤드 드롭하면 모듈이 생성된다. Condition Block을 클릭하면 **[Properties]** 탭에서 내용을 확인할 수 있고, Condition Block의 이름 변경도 가능하다.

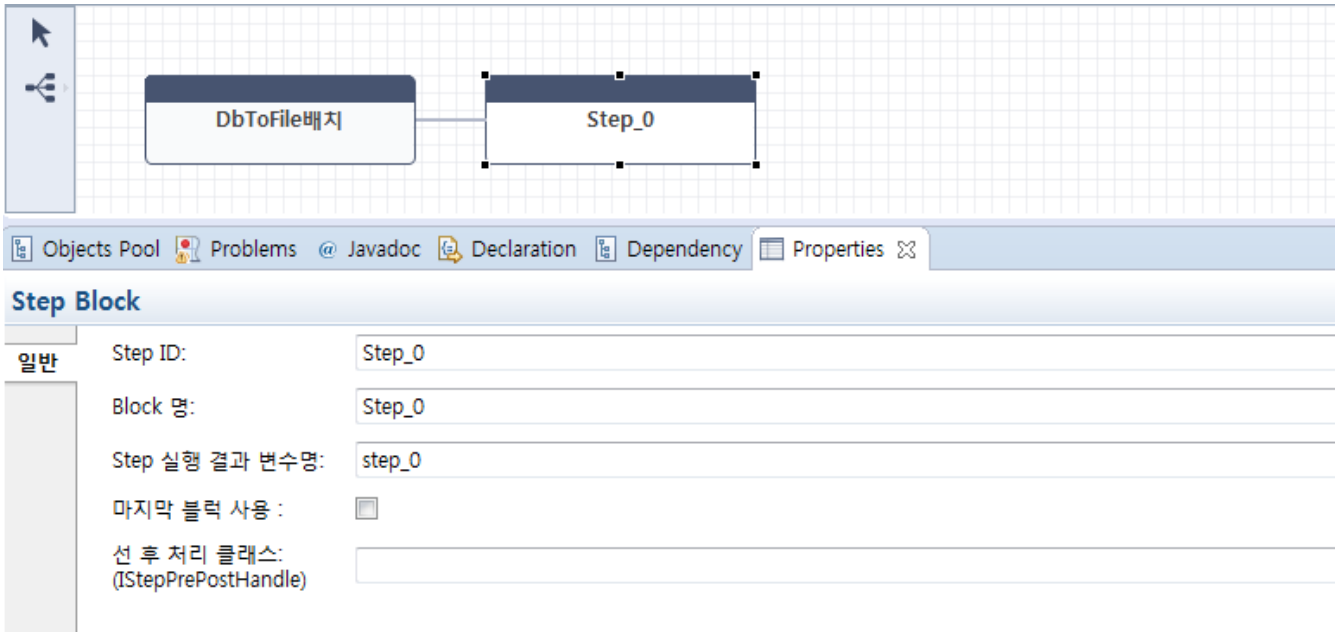


JO 디자인 에디터 - Condition Block

| 항목 | 설명 |
| --- | --- |
| Block 명 | Condition Block의 이름을 정의한다. |
| 설명 | Condition Block의 설명을 넣는다. |
| ELSE IF | **'ELSE IF'** 체크박스는 XOR 처리로 Condition Block이 3개 이상 묶인 경우 마지막 블록에 ELSE 조건이 아닌 ELSE IF 조건을 설정하는 경우 선택한다. |
| 조건 | 해당 Condition Block이 수행 되기 위한 조건을 작성하는 영역이다. |

#### 4.5.2.2. Step Block

**Palette**의 **[Step Block]**을 선택한 후 디자인 화면에 드래그 앤드 드롭하면 모듈이 생성된다. Step Block을 클릭하면 **[Properties]** 탭에서 내용을 확인할 수 있고, Step Block의 이름 변경도 가능하다.

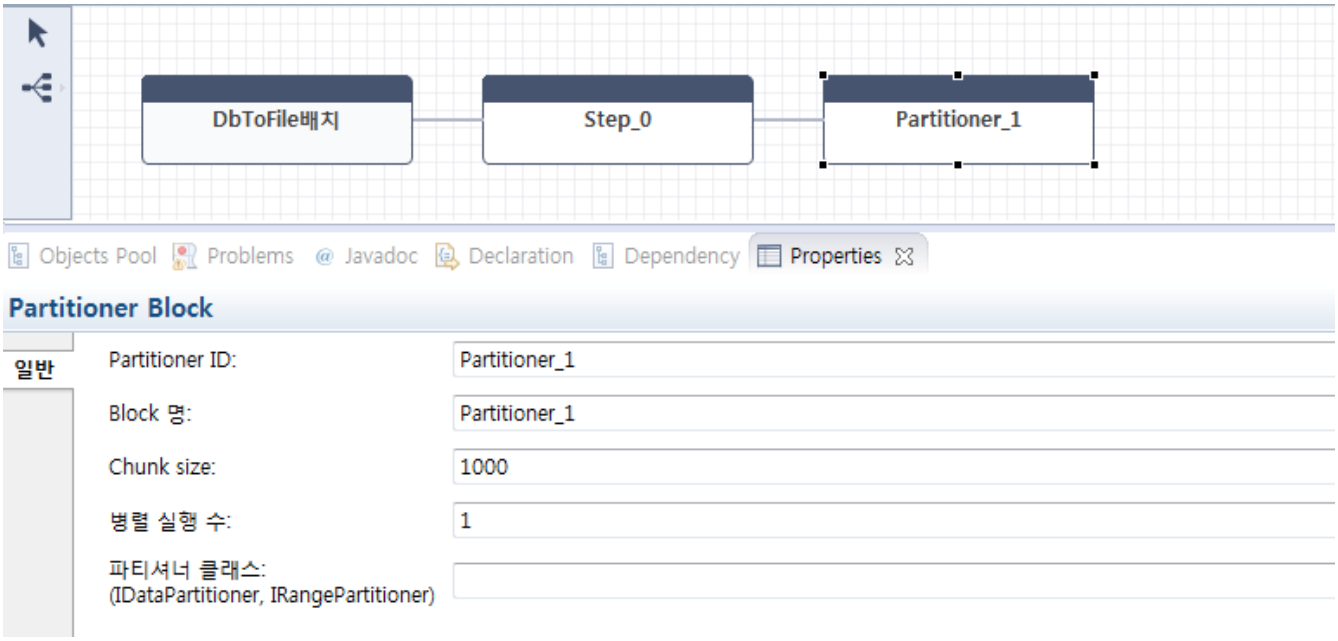



JO 디자인 에디터 - Step Block

| 항목 | 설명 |
|---|---|
| Step ID | Step Block의 식별자이다. |
| Block 명 | 개발자가 인식하기 쉬운 논리적인 이름이다. |
| Step 실행 결과 변수명 | Step Block 수행 결과를 받을 변수명을 정의한다. |
| 마지막 블록 사용 | 마지막 블록의 사용 여부를 설정한다. |
| 선 후 처리 클래스 | 해당 Step Block의 선 후 처리를 할 수 있는 클래스를 지정한다.<br><br>IStepPrePostHandle 인터페이스를 implements받은 클래스여야 한다. |

#### 4.5.2.3. Partitioner Block

**Palette**의 **[Partitioner Block]**을 선택한 후 디자인 화면에 드래그 앤드 드롭하면 모듈이 생성된다. Partitioner Block을 클릭하면 **[Properties]** 탭에서 내용을 확인할 수 있고, Partitioner Block의 이름 변경도 가능하다.



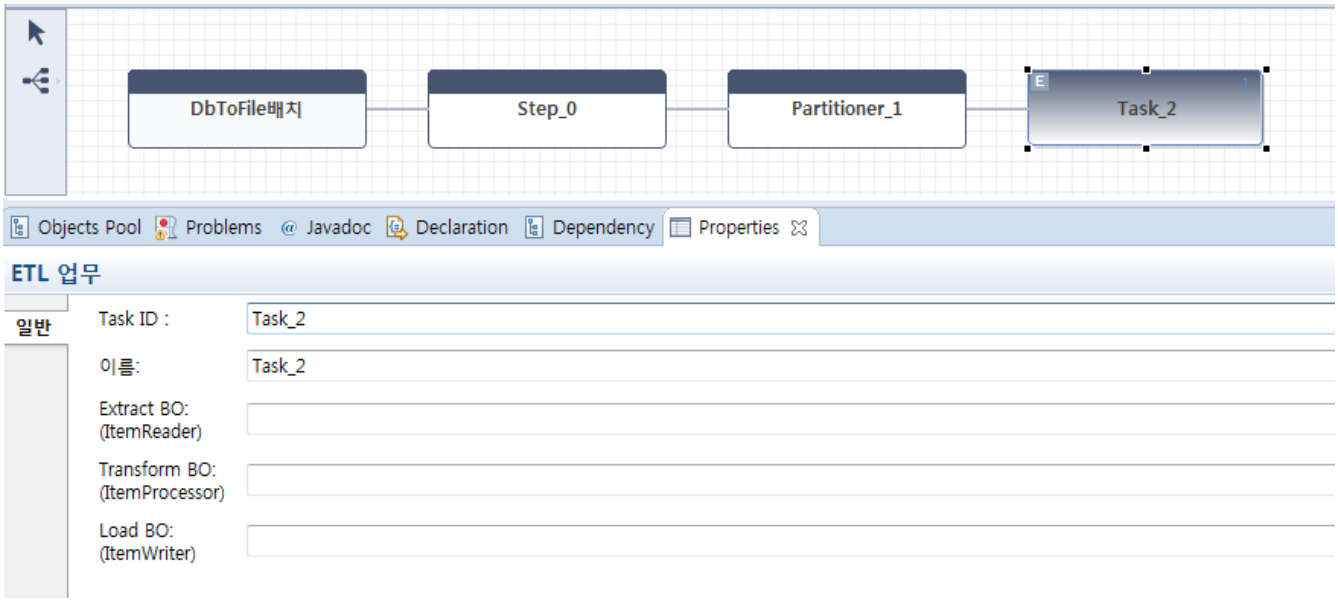

JO 디자인 에디터 - Partitioner Block

| 항목 | 설명 |
|---|---|
| Partitioner ID | Partitioner Block의 식별자이다. |
| Block 명 | 개발자가 인식하기 쉬운 논리적인 이름이다. |
| Chunk Size | 한개의 처리(Call)시에 순차적으로 처리 되는 처리 개수이다. 예를 들어 200개의 처리 대상 중 Chunk Size가 10이면 20개의 Block이 수행된다. |
| 병렬 실행 수 | 동시에 최대 수행될 수 있는 병렬 처리 수행 횟수를 정의한다. |
| 파티셔너 클래스 | IDataPartitioner 혹은 IRangePartitioner 인터페이스를 implements 받은 클래스여야 한다. |

#### 4.5.2.4. ETL Task

ETL Task는 사용자의 설정에 따라 수집/가공/적재 순의 데이터 처리를 병렬로 처리하는 Task이다. 수집, 가공, 적재는 사용에 따라 구성, 조립하여 사용할 수 있다.

**Palette**의 **[ETL Task]**을 선택한 후 디자인 화면에 드래그 앤드 드롭하면 수집/가공/적재 순으로 BO를 등록할 수 있다. ETL Task를 클릭하면 **[Properties]** 탭에서 내용을 확인할 수 있다.



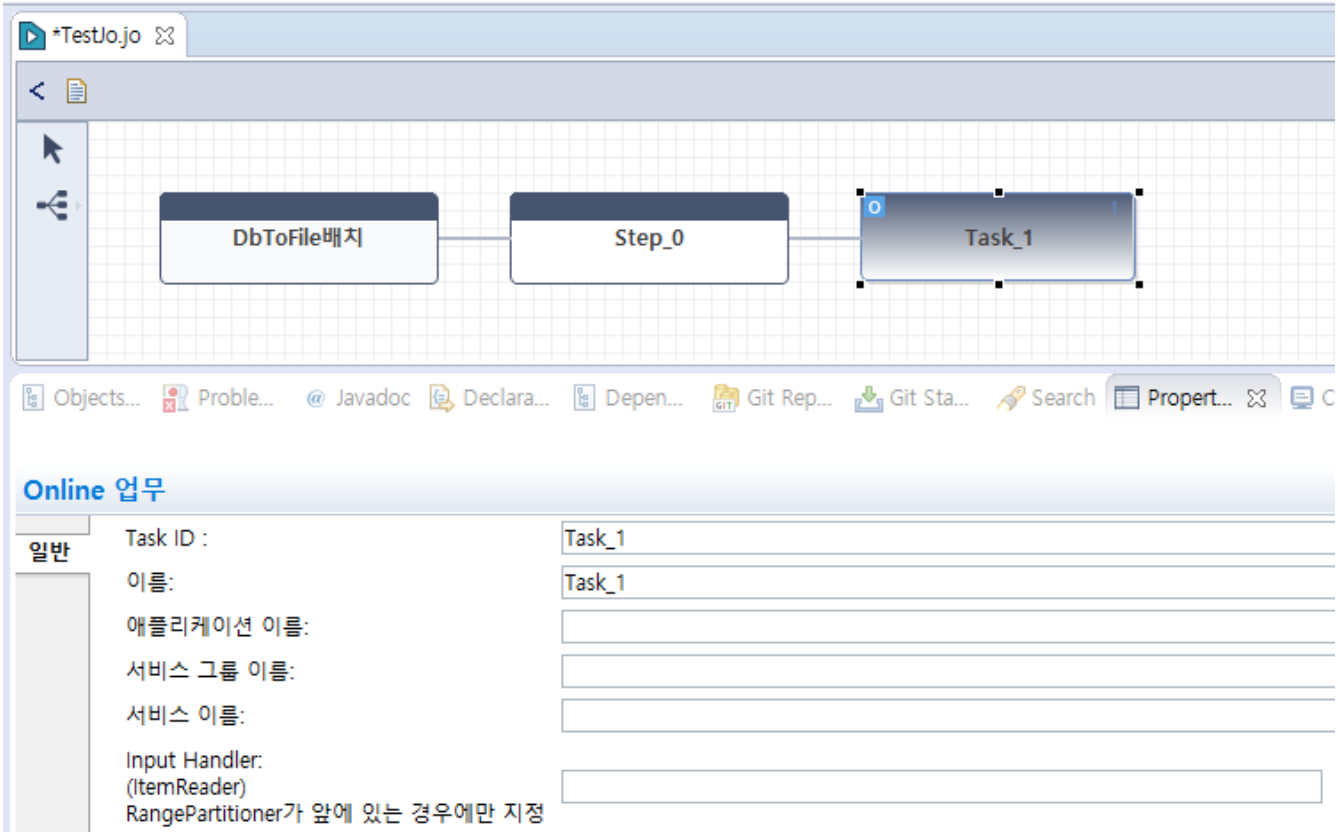

JO 디자인 에디터 - ETL Task

| 항목 | 설명 |
|---|---|
| Task ID | ETL Task의 식별자이다. |
| 이름 | 개발자가 인식하기 쉬운 논리적인 이름이다. |
| Extract BO | 처리할 대상의 내역을 추출하는 BO를 설정한다. |
| Transform BO | Extract에서 추출된 내역을 처리하는 BO를 설정한다. |
| Load BO | Transform에서 처리된 결과를 Load 처리하는 를 설정한다. |

#### 4.5.2.5. Online Task

Online Task는 단일 건수 처리 온라인 서비스를 병렬로 일괄 기동하여, 비즈니스 로직의 재사용성과 처리 성능을 극대화하는 Task 배치이다.

**Palette**의 **[Online Task]**을 선택한 후 디자인 화면에 드래그 앤드 드롭하면 모듈을 등록할 수 있다. Online Task를 클릭하면 **[Properties]** 탭에서 내용을 확인할 수 있다.



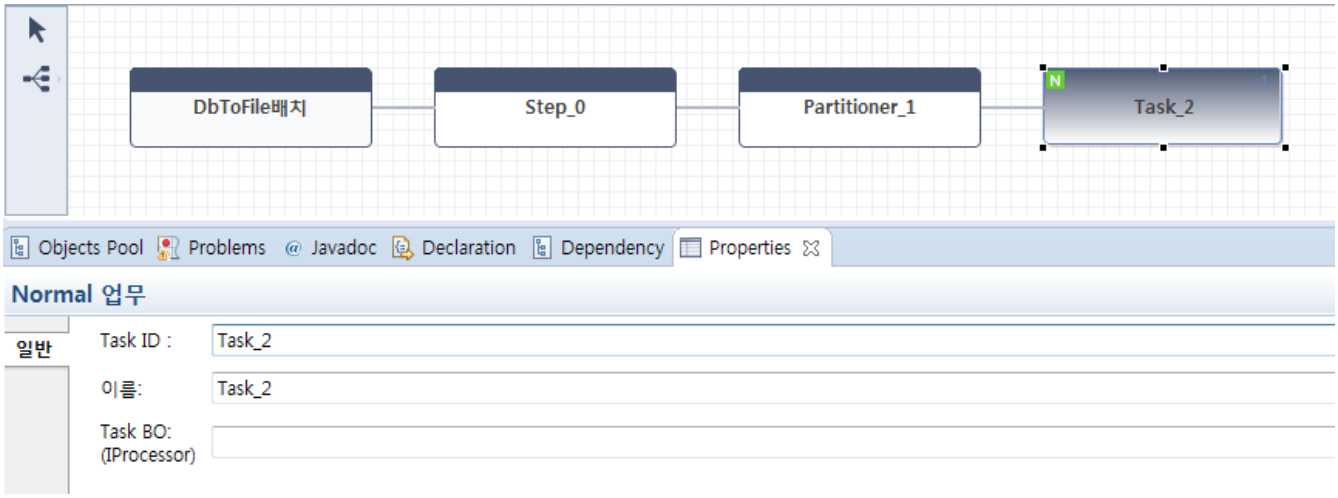
JO 디자인 에디터 - Online Task

| 항목 | 설명 |
|---|---|
| Task ID | Online Task의 식별자이다. |
| 이름 | 개발자가 인식하기 쉬운 논리적인 이름이다. |
| 애플리케이션 이름 | 서비스 이름에 소속된 애플리케이션을 설정한다. |
| 서비스 그룹 이름 | 서비스 이름에 소속된 서비스 그룹 을 설정한다. |
| 서비스 이름 | Online Task에서 수행할 서비스 이름을 설정한다. |
| Input Handler | RangePartitioner가 앞에 있는 경우만 지정하며, 서비스의 입력값 처리를 위한 Handler이다. |

#### 4.5.2.6. Normal Task

Normal Task는 ETL Task와 같이 수집/가공/적재로 데이터를 처리하는 경우 데이터를 가공할 때 별도의 BO 등록 및 개발해야 하는 불편을 해소하고 기존 프레임워크에 익숙한 개발자에게 편의성을 제공한다.

**Palette**의 **[Normal Task]**을 선택한 후 디자인 화면에 드래그 앤드 드롭하면 Task BO를 등록할 수 있다. Normal Task를 클릭하면 **[Properties]** 탭에서 내용을 확인할 수 있다.




JO 디자인 에디터 - Normal Task

| 항목 | 설명 |
| --- | --- |
| Task ID | Normal Task의 식별자이다. |
| 이름 | 개발자가 인식하기 쉬운 논리적인 이름이다. |
| Task BO | Normal Task에서 처리할 BO를 설정한다. |

# 5. 지원 기능

본 장에서는 주요 기능 외 지원 기능에 대해 설명한다.

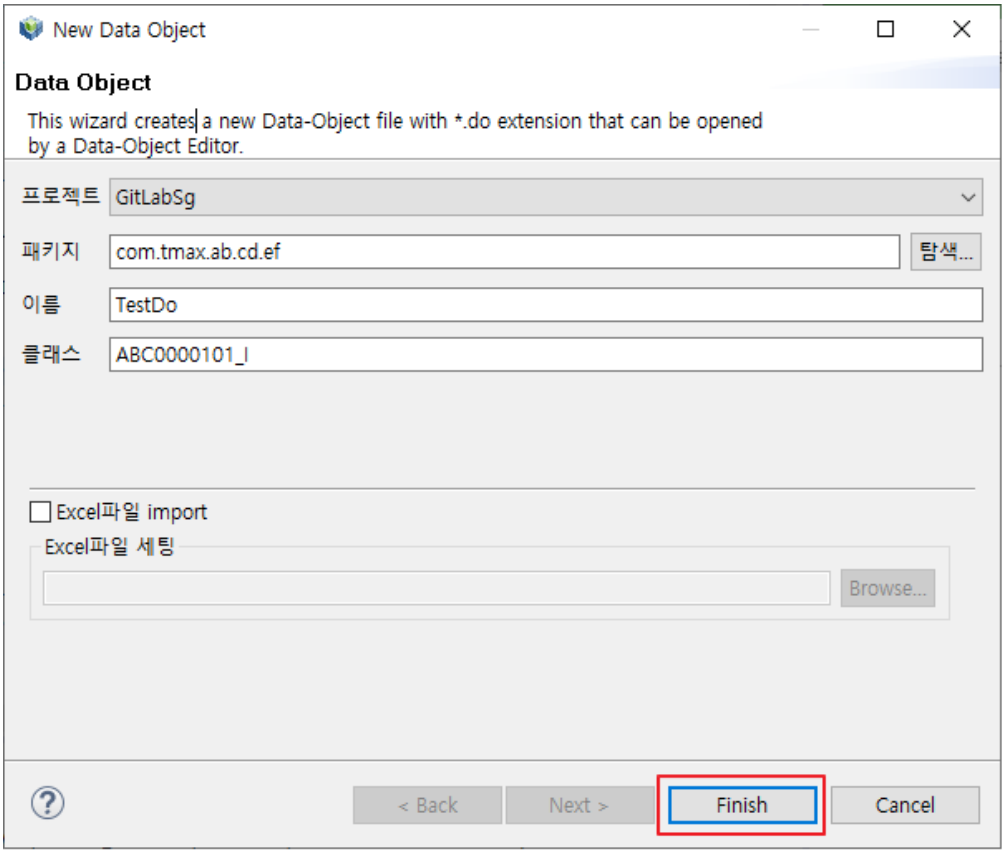
## 5.1. 리소스 생성

본 절에서는 리소스 생성 및 개발 편의 기능에 대해서 설명한다.

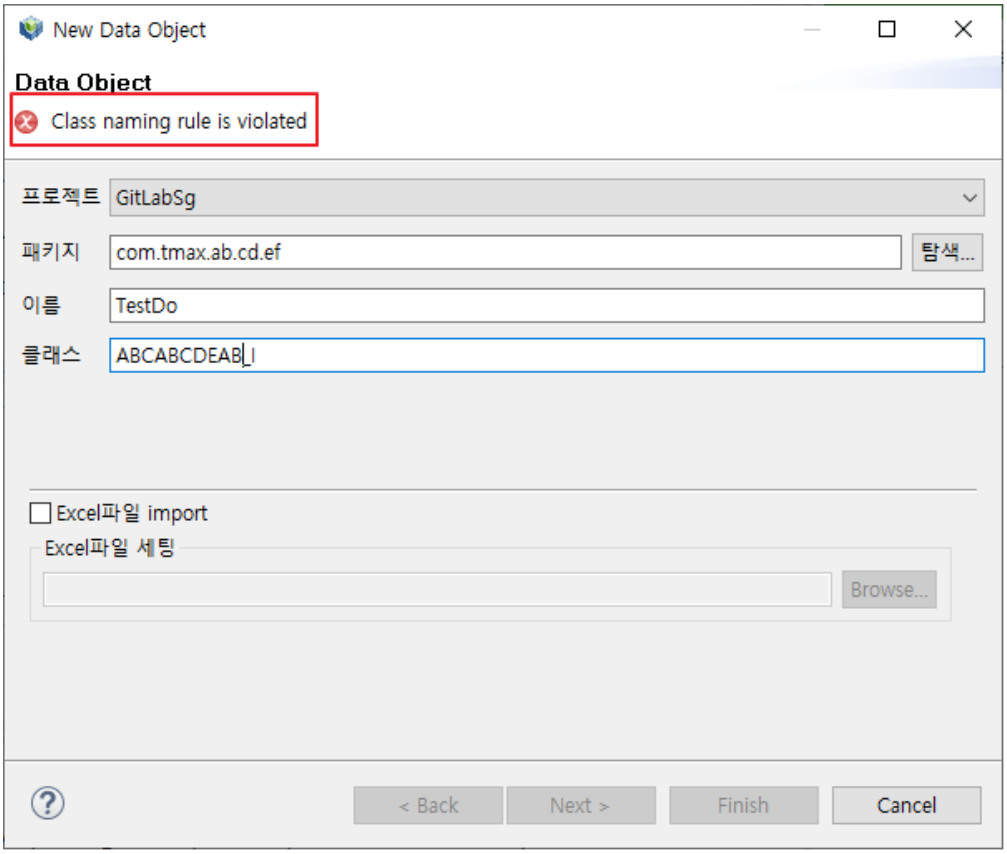
### 5.1.1. Class Naming Rule Check

ProObject 리소스를 생성하는 경우 클래스명을 정의된 Rule에 맞게 강제할 수 있다.

Naming Rule Check가 활성화되어 있으면, Naming Rule에 맞지 않는 규칙에 대한 리소스 생성을 막을 수 있다.

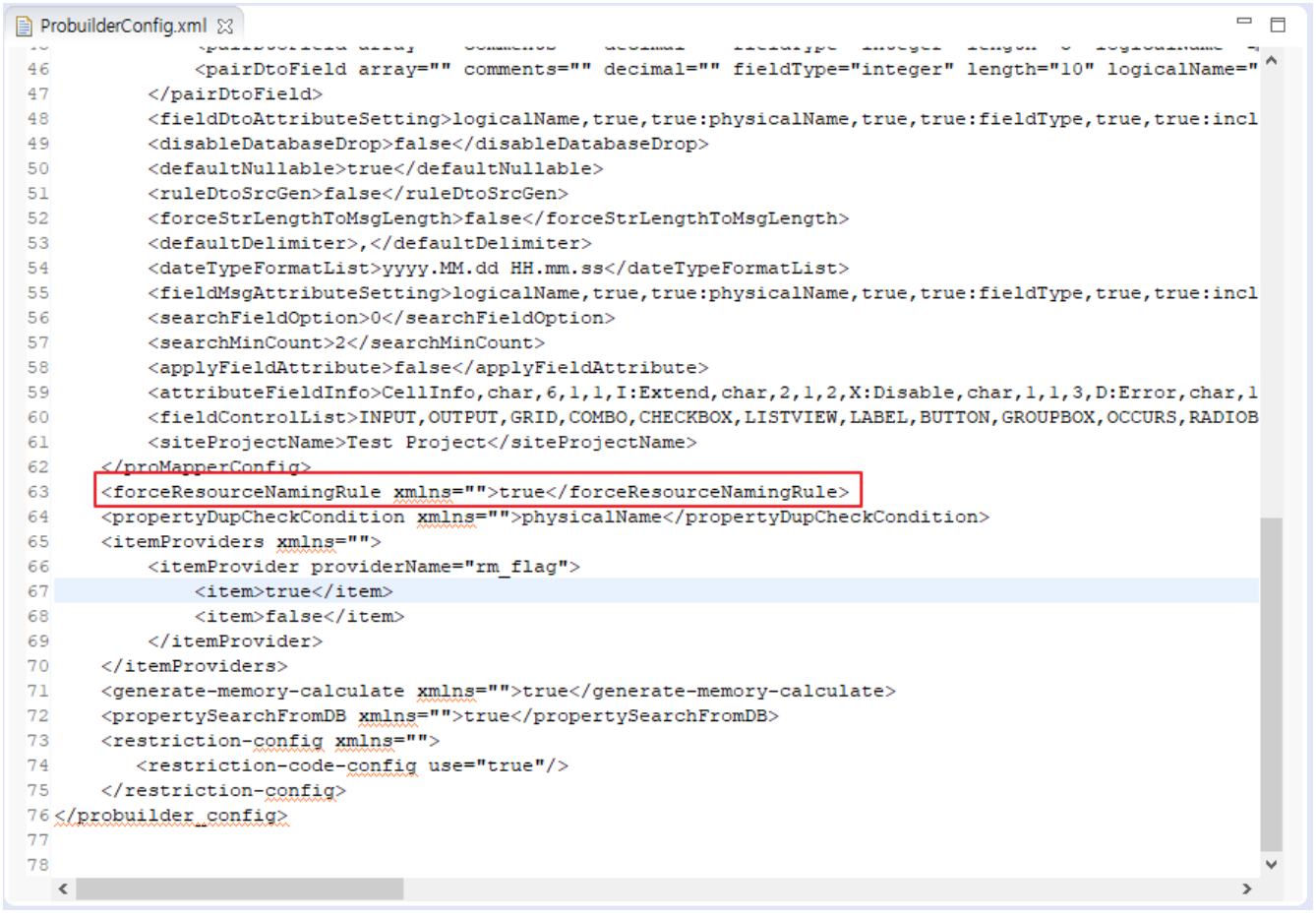



Naming Rule Check - Rule이 맞을 때




Naming Rule Check - Rule이 맞지 않을 때

Naming Rule Check의 활성화를 위해서는 ProbuilderConfig.xml 파일의 **<forceResourceNamingRule>** Element가 true로 설정되어야 한다.




EMB Designer - 이너 노드 속성

다음은 각 리소스별 Naming Rule에 대한 설명이다.

- DO

```
L2(2) + L3(1) + NUM_CODE(5) + FUNC_CODE(2) + _I/_O
L2(2) + L3(1) + NUM_CODE(5) + FUNC_CODE(2) + DOF_D
L2(2) + L3(1) + NUM_CODE(5) + _I/_O/_S
```

- DOF

```
L2(2) + L3(1) + NUM_CODE(5) + FUNC_CODE(2) + DOF
```

- BO

```
L2(2) + L3(1) + NUM_CODE(5) + FUNC_CODE(2)
L2(2) + L3(1) + J + NUM_CODE(4) + FUNC_CODE(2)
```

- SO

```
L2(2) + L3(1) + NUM_CODE(5)
```

- JO

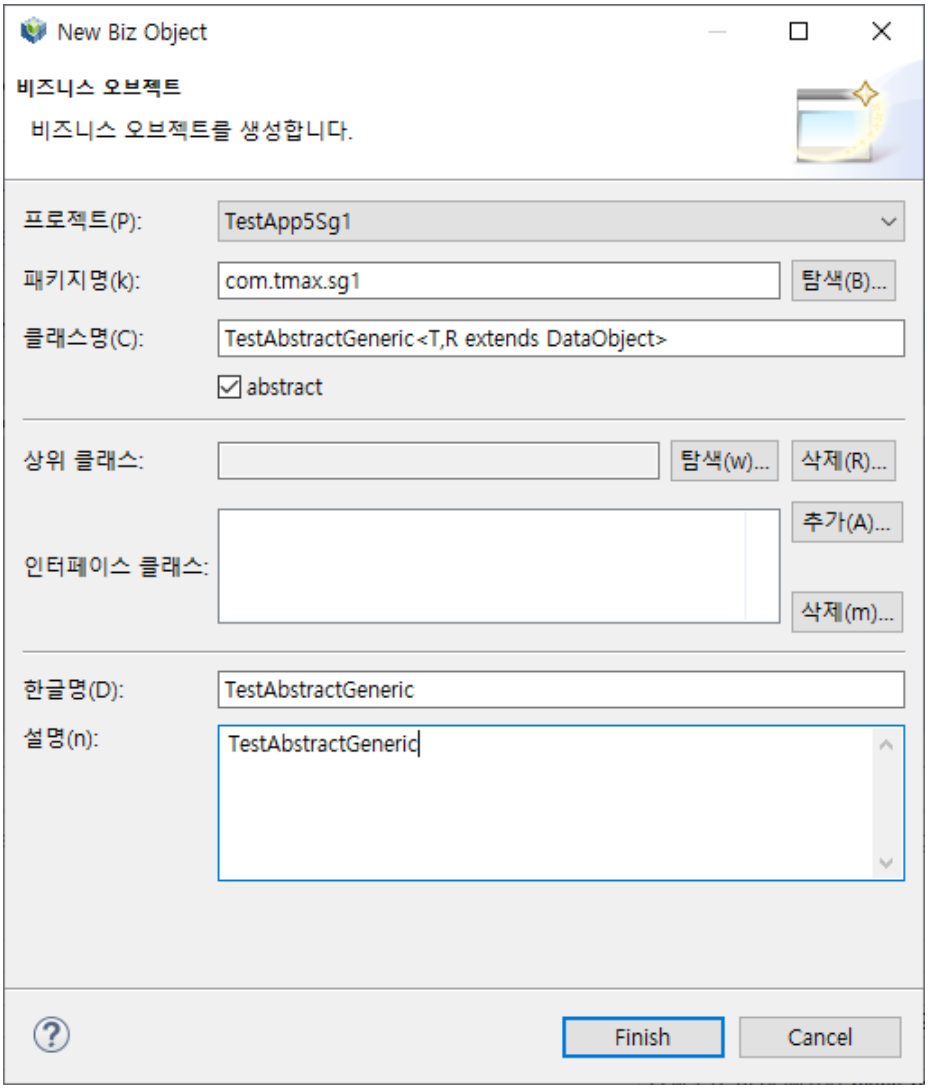
```
L2(2) + L3(1) + J + NUM_CODE(4)
```

| 구분 | Rule |
|---|---|
| () 안의 숫자 | 길이 |
| L2 | 대문자 알파벳 |
| L3 | 대문자 알파벳 (하나) |
| NUM_CODE | 0부터 9까지 숫자 |
| FUN_CODE | 0부터 9까지 숫자 |
| _I/_O/_S | _I or _O or _S |

## 5.1.2. Generic Type 리소스 생성

다음은 Generic Type을 가지는 BO 생성하는 과정에 대한 설명이다.

1. 트리 메뉴에서 **[New] > [Biz Object (Designer)]**를 클릭한다. **New Biz Object 화면**에서 각 항목을 입력한 후 **[Finish]** 버튼을 클릭한다.



Generic Type BO 생성(1)

클래스명을 입력하고, 클래스는 **'abstract'**로 생성해야 한다. 클래스명은 Generic Type 정의를 함께 넣는다.

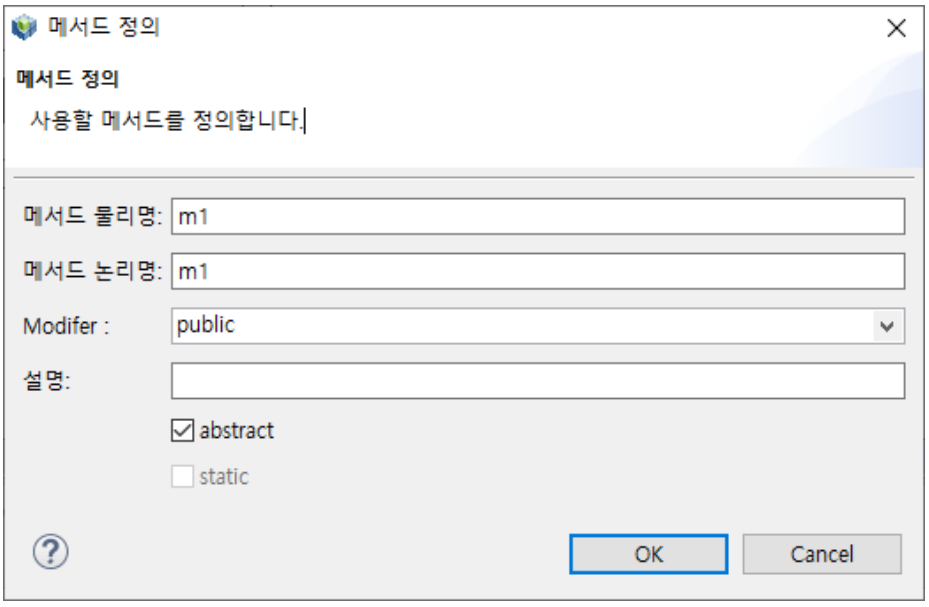
다음은 클래스명에 설정할 Generic Type이다.

| Generic Type | 가능 여부 |
|---|---|
| "<", "<>", "<T,>" | 불가 - 규칙에 안맞음 |
| "<T>", "<T, R>", <T,R extends DataObject> | 가능 |

2. 다음은 Generic Type BO가 생성된 소스의 예이다.

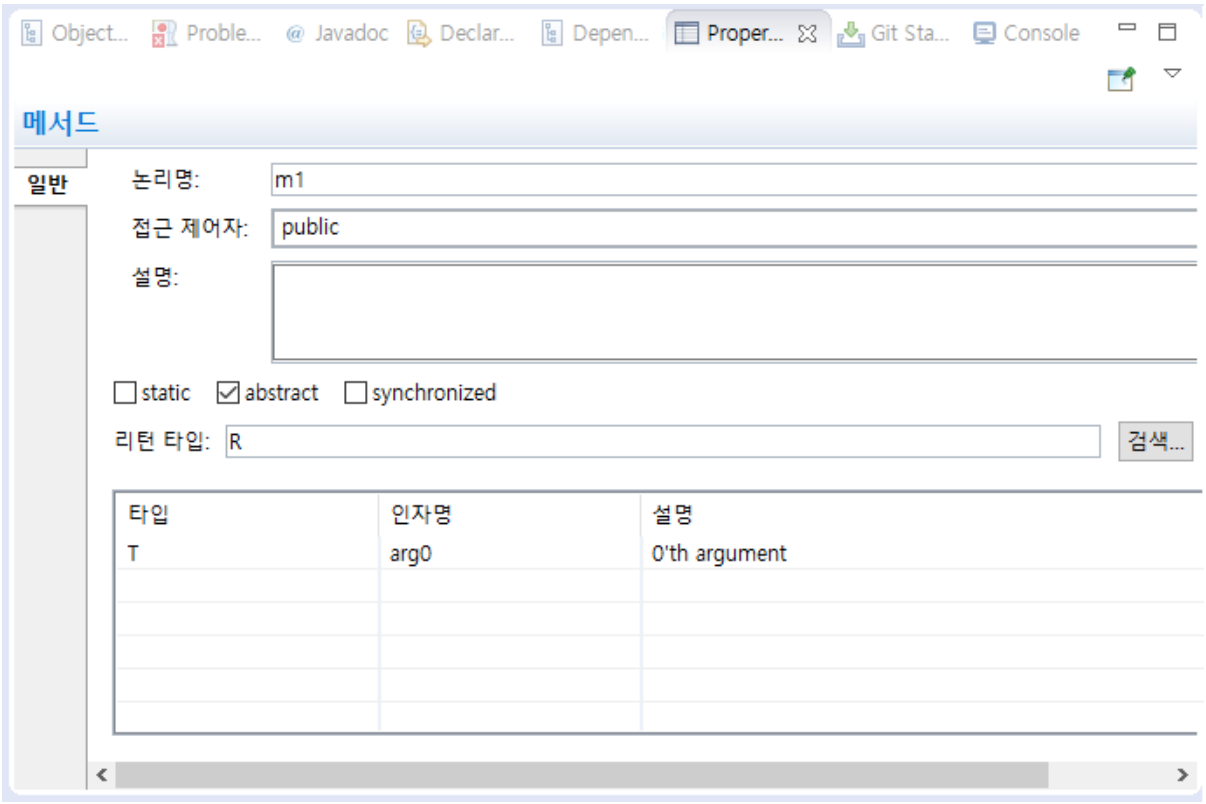

Generic Type BO 생성(3)

3. 메소드를 Generic Type으로 생성한다. **디자인 에디터**의 **메소드 영역**에서 **[+]** 버튼을 클릭해서 메소드를 정의한다. **'abstract'**로 생성해야 한다.



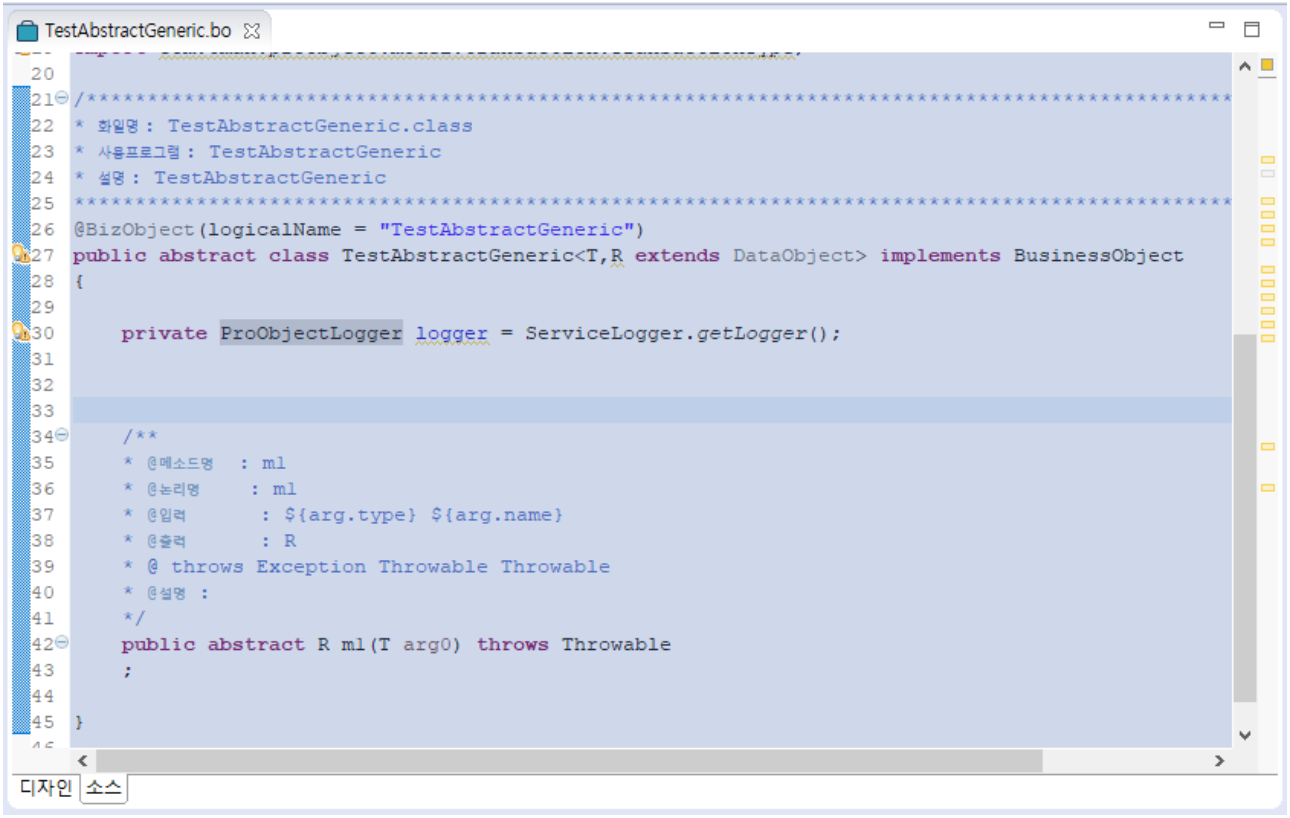

Generic Type BO 생성(4)

4. **[Properties]** 탭을 선택한 후 리턴 타입을 입력한다.




Generic Type BO 생성(5)

5. 생성된 Generic Type의 생성된 Java Source는 다음과 같다.



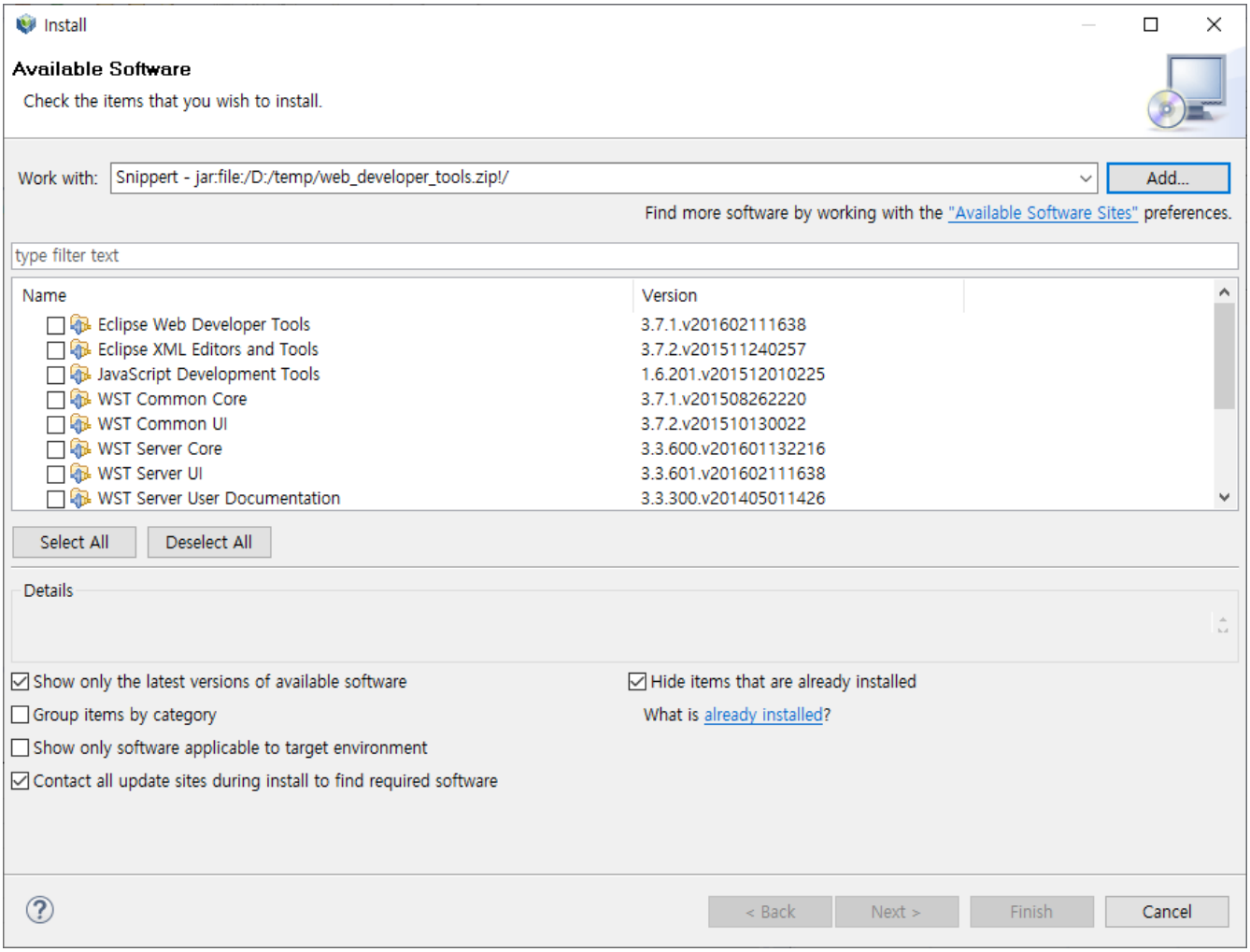

Generic Type BO 생성(6)

## 5.1.3. Code Snippet 기능

다음은 Snippet 기능 설치, Code Templet 정의, 정의된 템플릿의 BO/SO 영역에 적용하는 방법에 대한 설명이다.

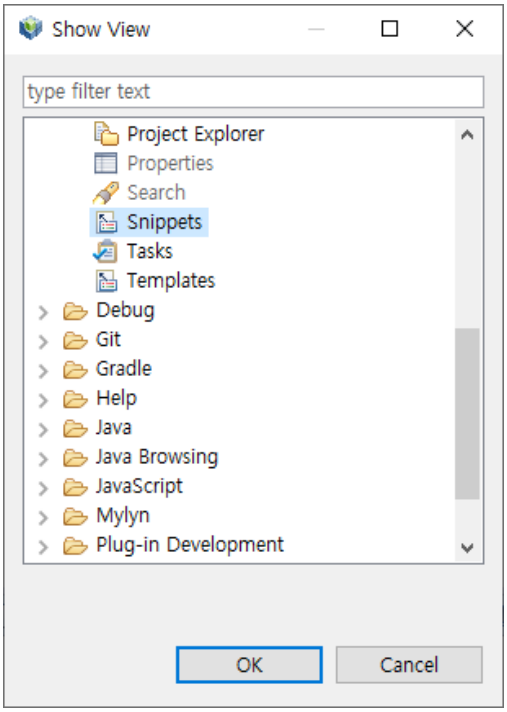
### 5.1.3.1. 설치

**[Help] > [install new software] > [add ] > [archive]** 메뉴를 선택한 후 "web_developer_tools.zip"을 선택하여 설치 준비를 한다. **'Group items by category'**를 체크해야 하며, 설치할 때 모든 패키지를 체크해야 한다. 설치가 완료되면 ProStudio를 재시작한다.




Code Snippet 설치

**[Window] > [show view] > [other..] > [General] > [Snippets ]**을 선택하여 Snippets View를 Open한다.

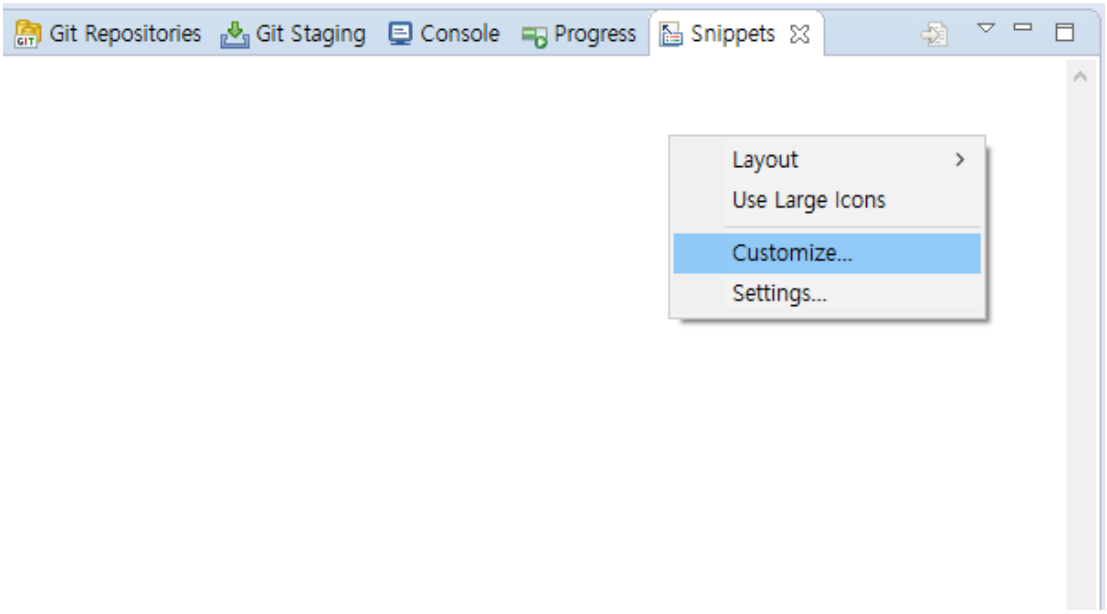



Code Snippet 기능 - View Open

### 5.1.3.2. 템플릿 적용

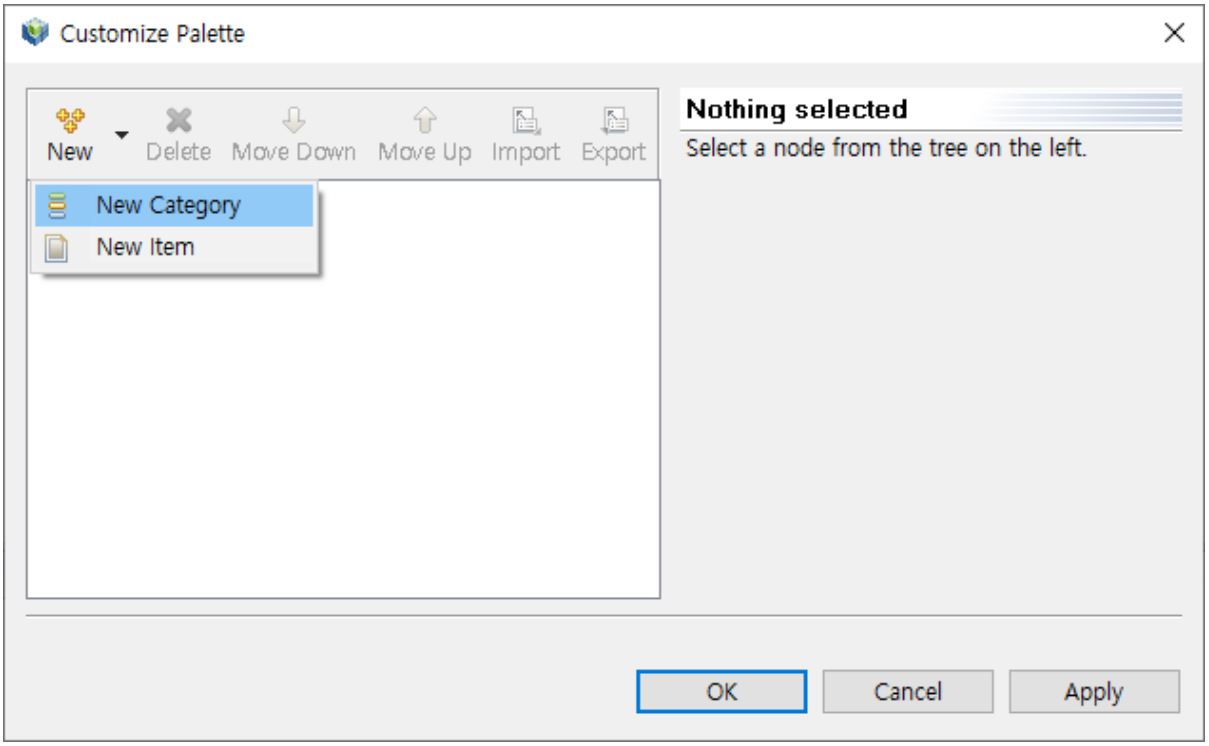
다음은 템플릿을 적용하는 과정에 대한 설명이다.

1. **[Customize...]** 메뉴를 선택하여 Snippet View를 선택하여 템플릿을 "Category" 및 "Item"을 생성한다.



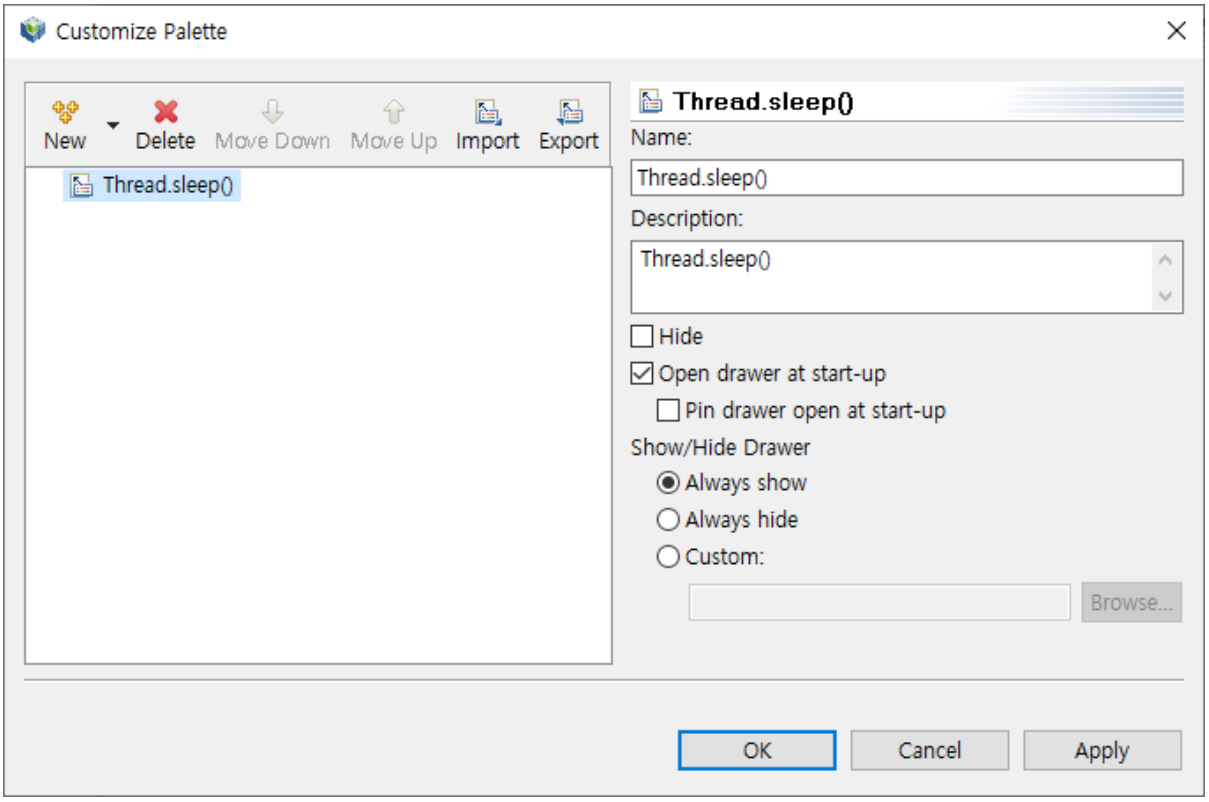

Code Snippet 기능 - 템플릿 적용 (1)

2. **[New Category]** 버튼을 클릭해서 그룹핑하는 이름을 정의한다. Category는 템플릿 내역을 그룹핑하는 단위이고, Item은 템플릿의 단위이다.



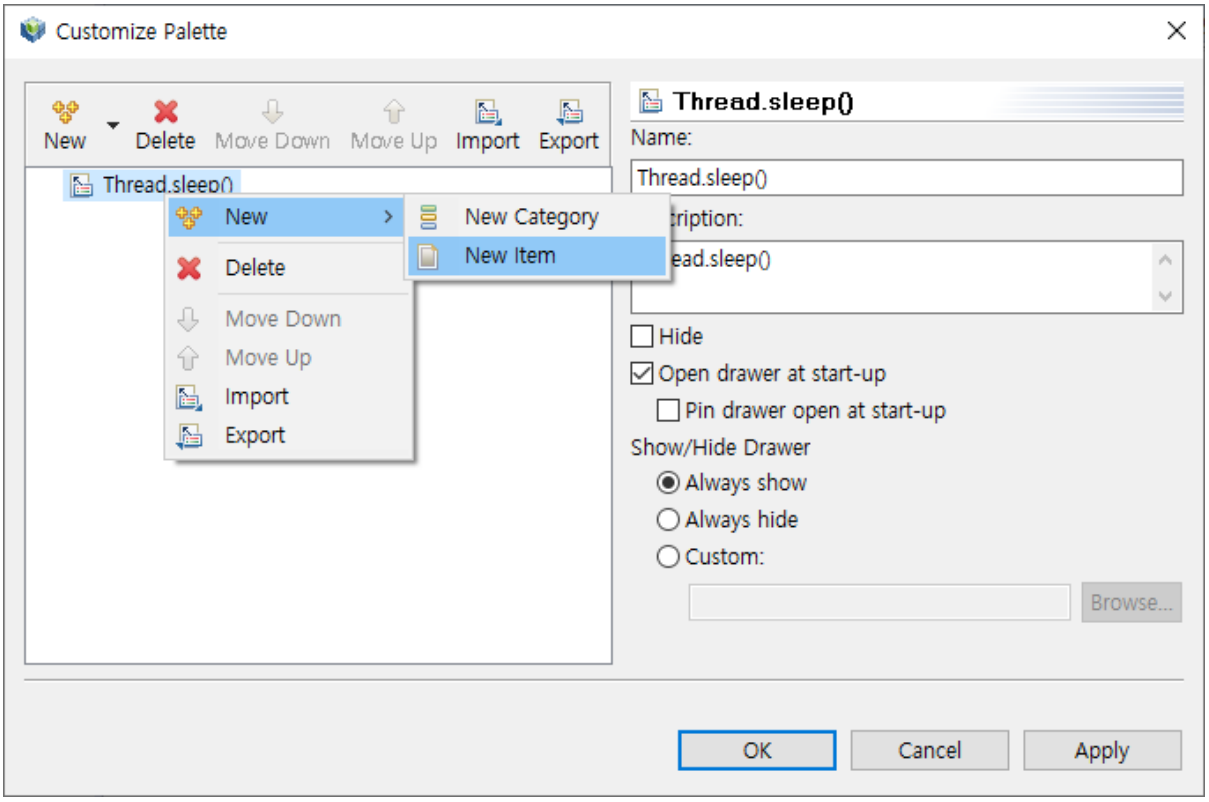

Code Snippet 기능 - 템플릿 적용 (2)

**'Name'**과 **'Description'** 항목을 입력한 후 **[OK]** 버튼을 클릭한다.



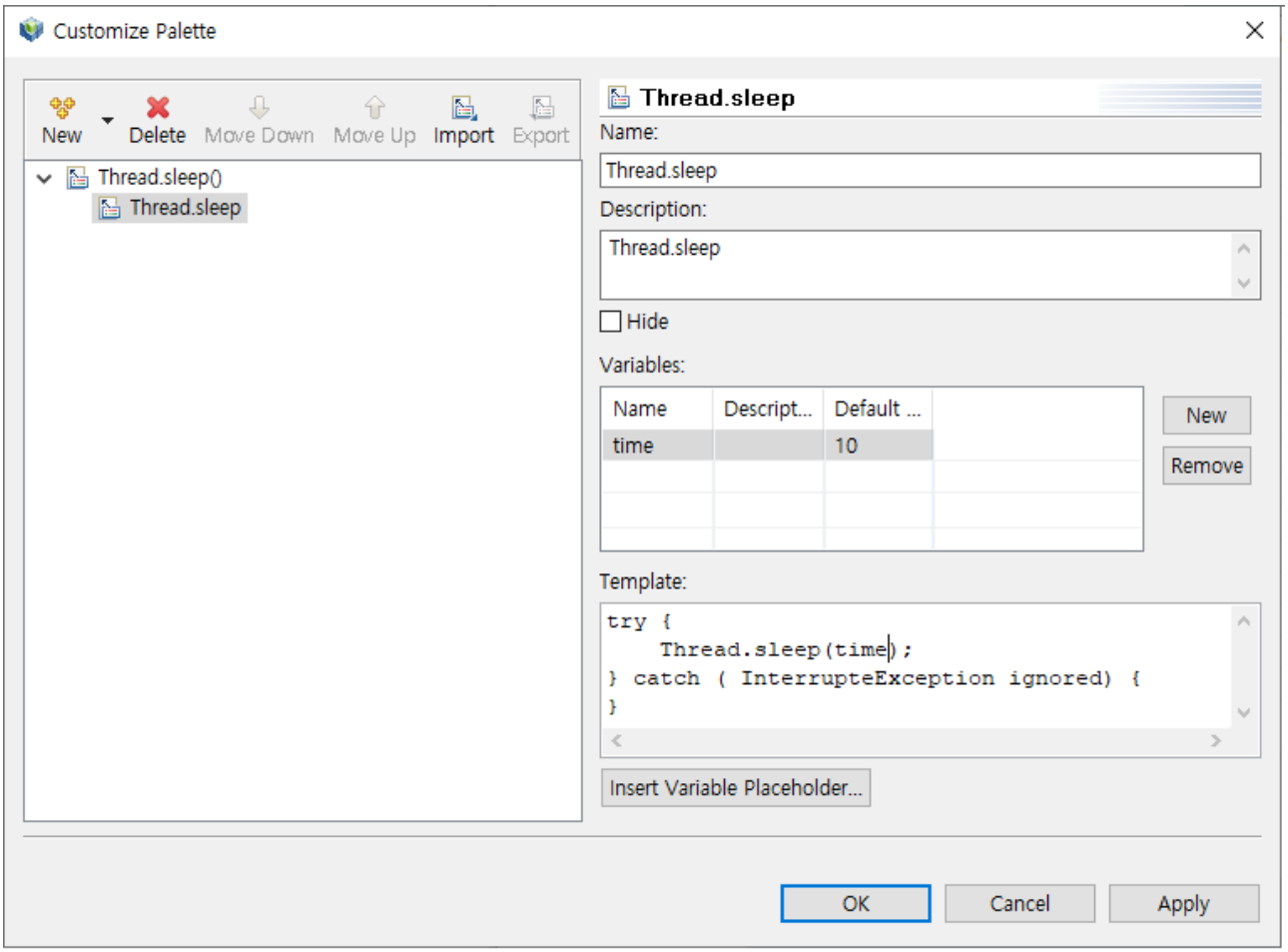

Code Snippet 기능 - 템플릿 적용 (3)

3. **[New Item]** 버튼을 클릭해서 Item을 추가한다.



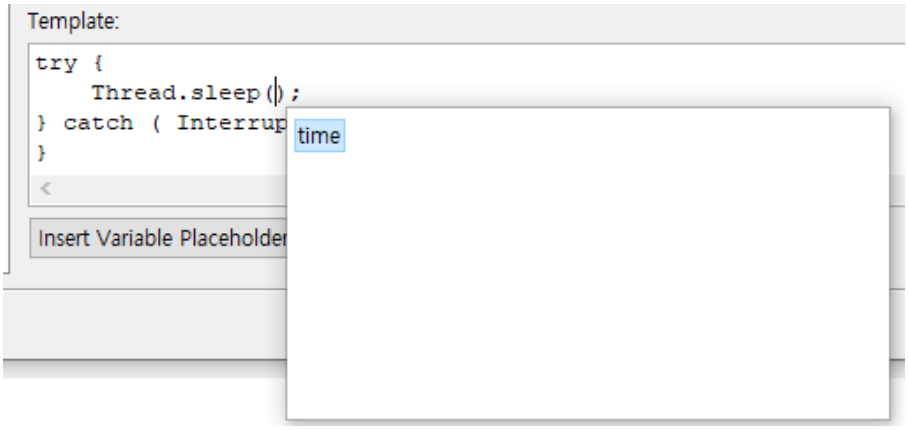

Code Snippet 기능 - 템플릿 적용 (4)

**'Name'**, **'Template'**, **'Variables'**를 추가한다.



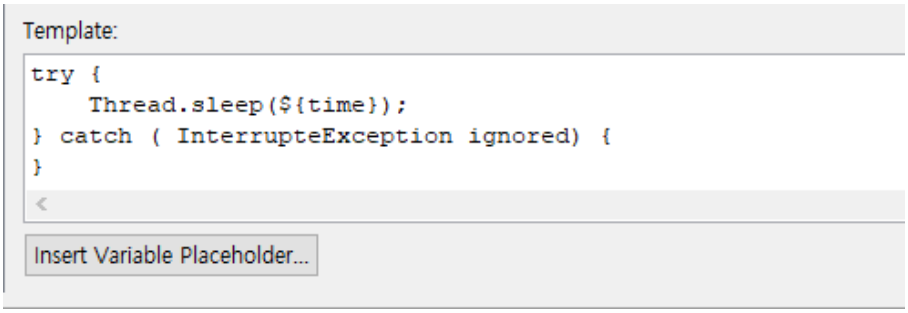

Code Snippet 기능 - 템플릿 적용 (5)

4. **'Template'**의 변수에 Focus를 두고 **[Insert Variable Placeholder..]** 버튼을 클릭하면 변수 내역이 조회된다. 변수 목록에서 설정할 변수를 클릭한다.



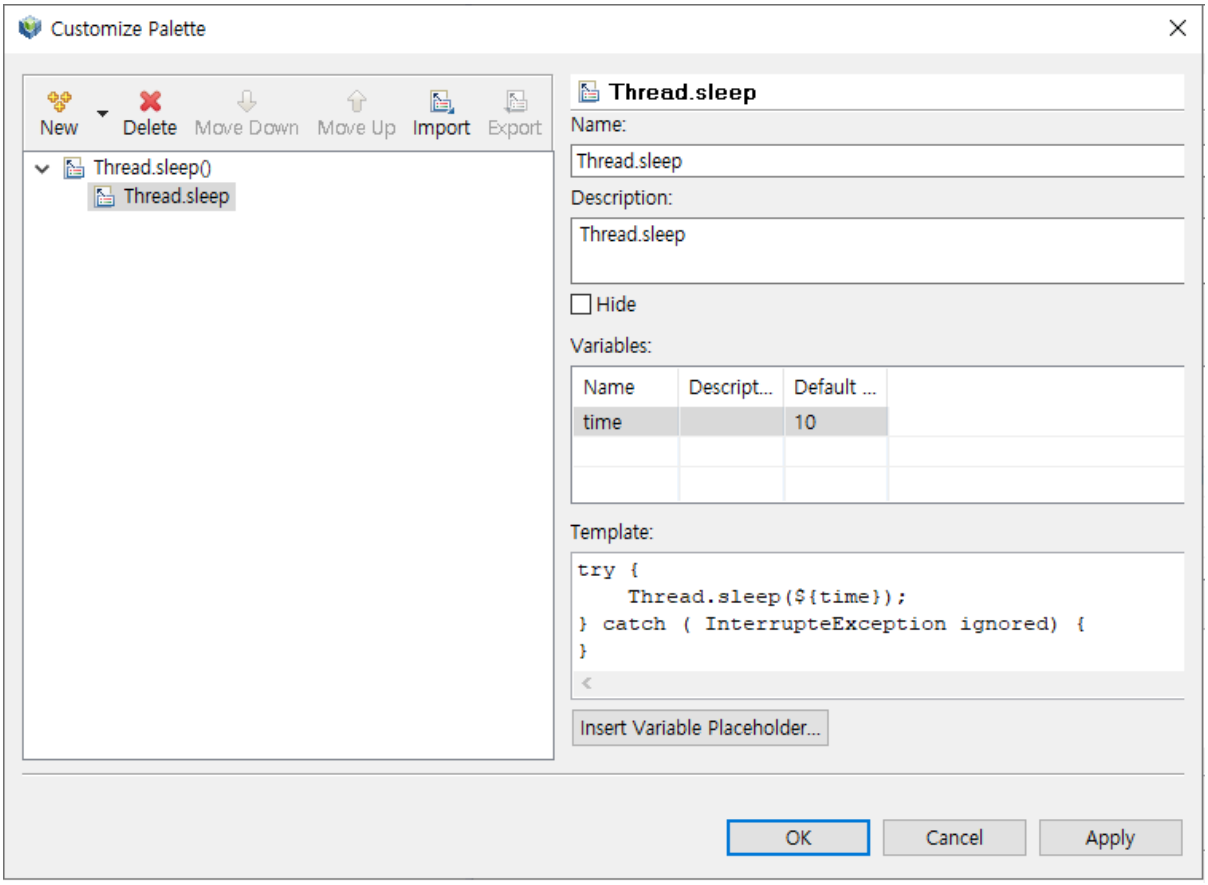

Code Snippet 기능 - 템플릿 적용 (6)

5. 다음과 같이 해당 변수는 '${변수}'의 형태로 추가된다.




Code Snippet 기능 - 템플릿 적용 (7)

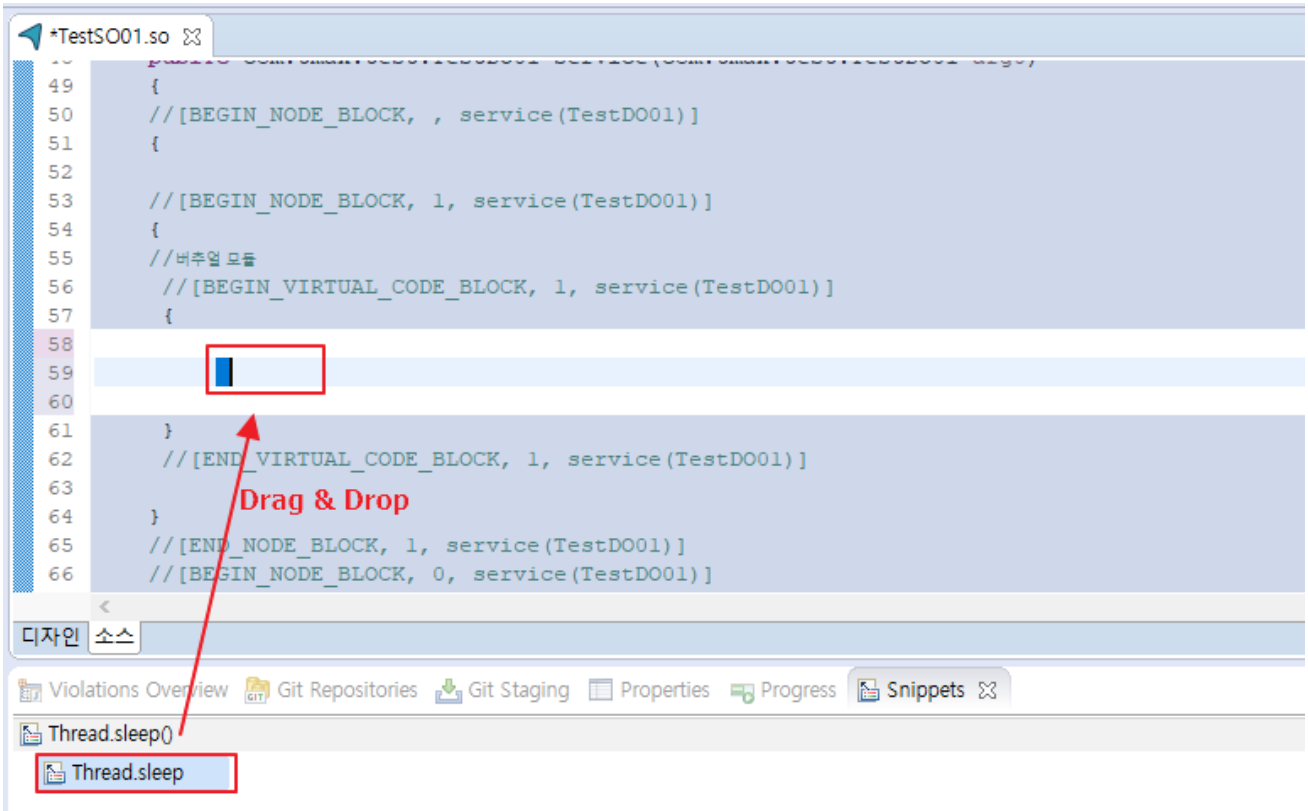
6. 완성된 템플릿이며, **[OK]** 버튼을 클릭하면 템플릿이 생성된다.




Code Snippet 기능 - 템플릿 적용 (8)

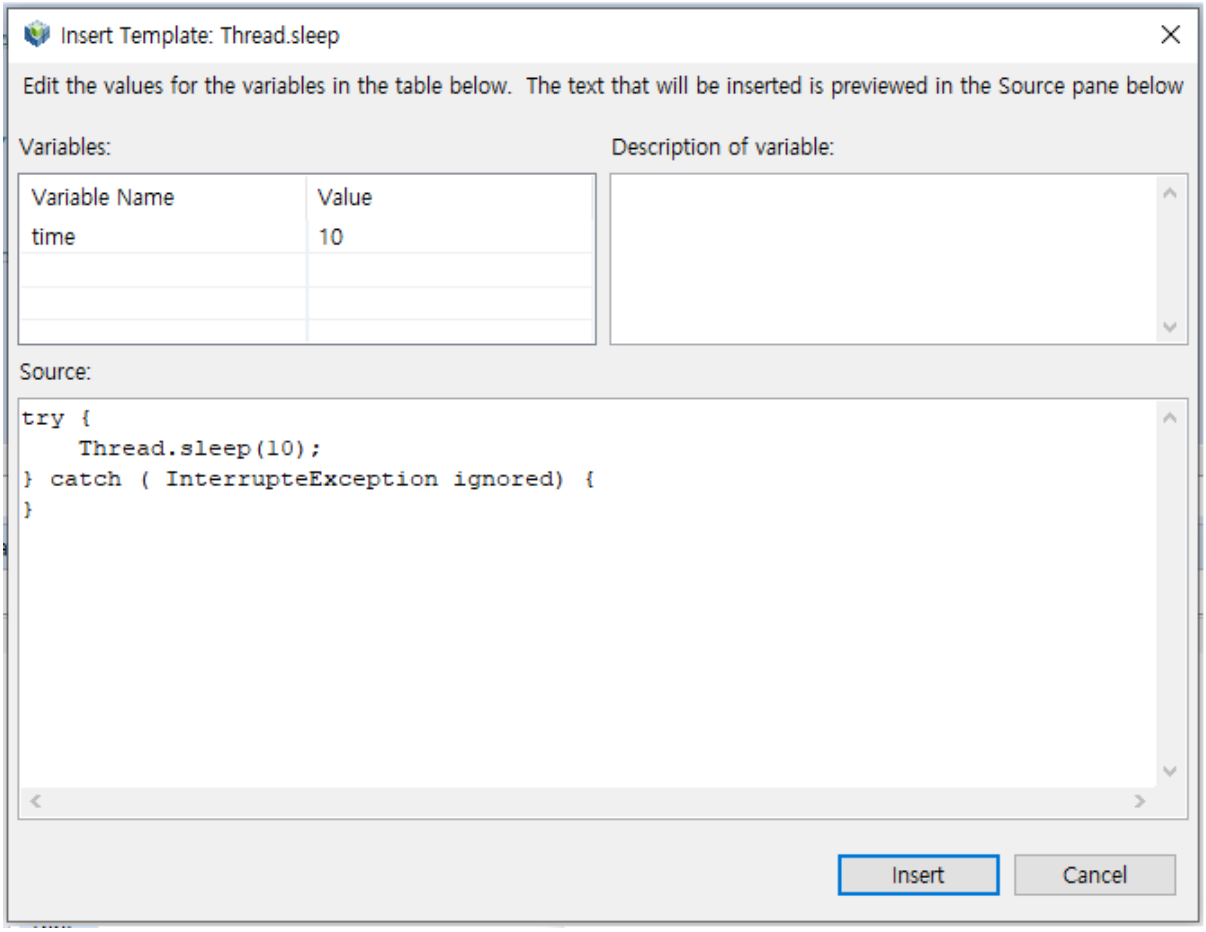
### 5.1.3.3. BO/SO 적용

BO/SO의 프로그램 가능 영역에 적용할 Item을 드래그 앤드 드롭하면 변수를 입력할 수 있는 화면이 조회된다.




Code Snippet 기능 - 프로그램 적용 (1)

입력할 변수를 선택한 후 **[Insert]** 버튼을 클릭한다.




Code Snippet 기능 - 프로그램 적용 (2)

다음은 Snippets에 의해 템플릿이 적용된 소스의 예이다.

Code Snippet 기능 - 프로그램 적용 (3)

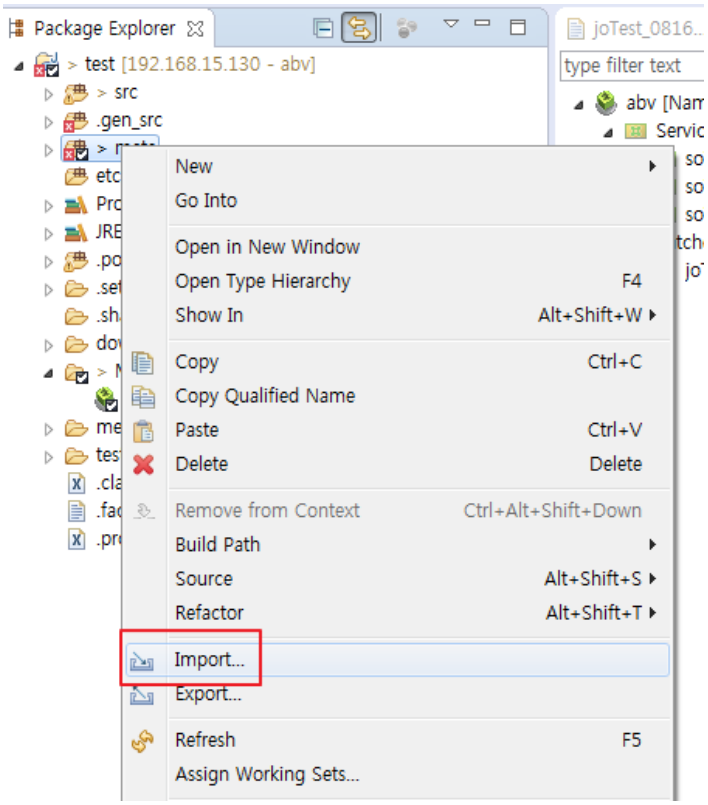
# 5.2. 리소스 관리

본 절에서는 리소스의 주요 관리 기능에 대해서 설명한다.

## 5.2.1. Resource Import

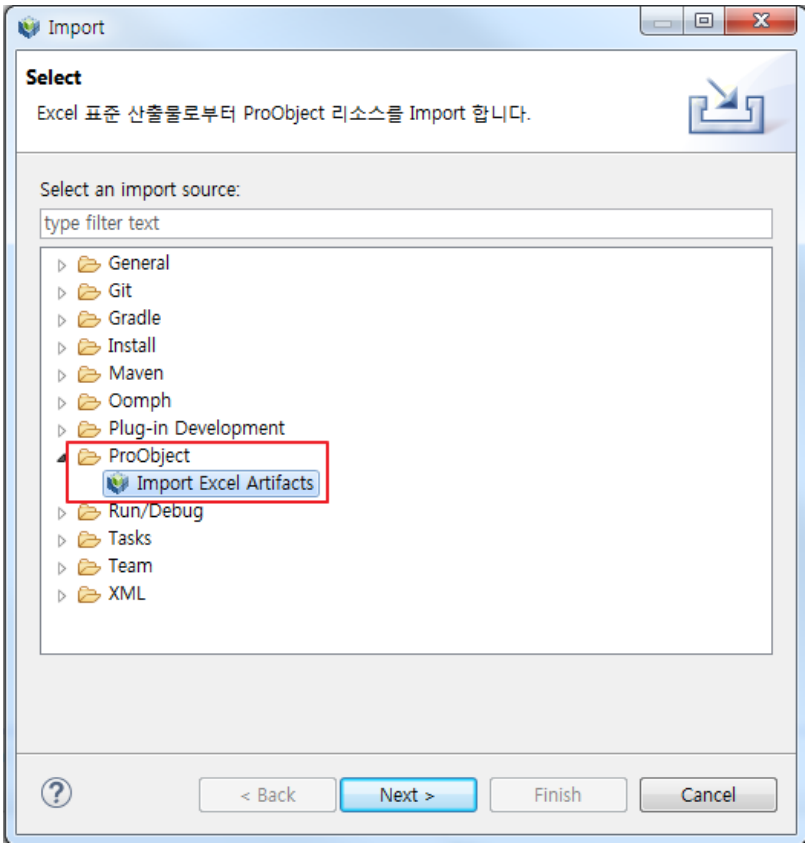
Excel로 Import할 파일을 생성한 후 다음의 과정으로 파일을 ProObject 리소스에 Import한다. 생성할 Excel 파일 포맷에 대한 자세한 설명은 "리소스 엑셀 파일 포맷"을 참고한다.

1. **Package Explorer**의 컨텍스트 메뉴에서 **[Import]**를 선택한다.



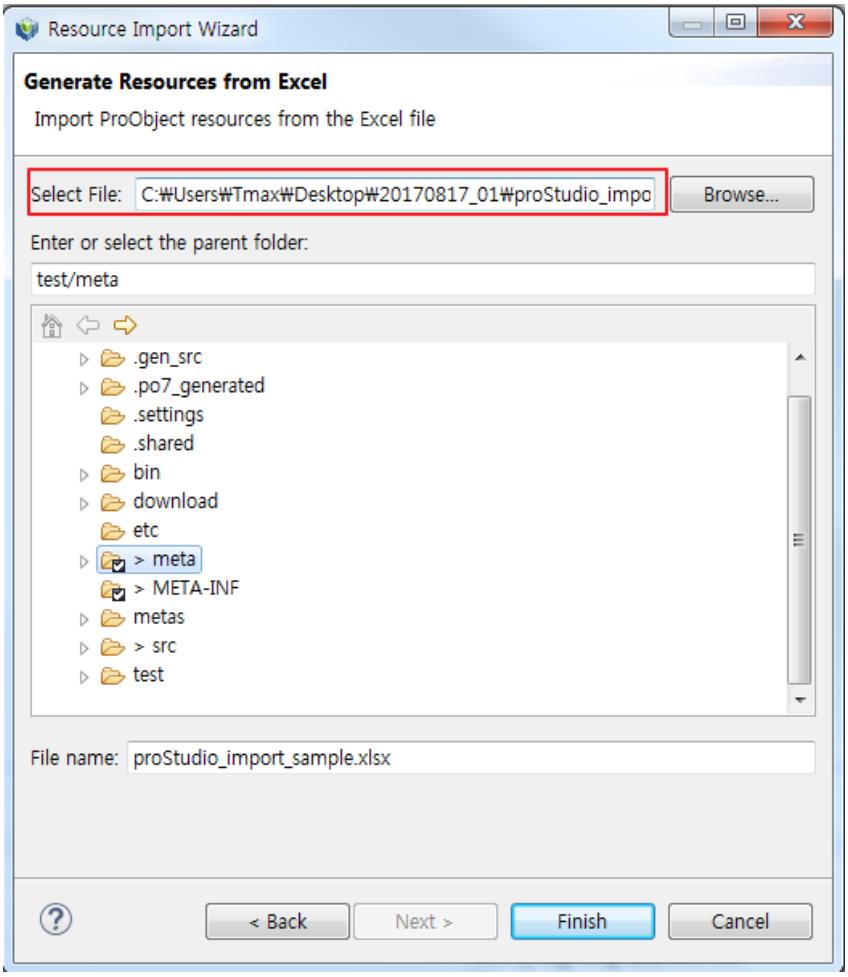

Resource Import - 메뉴 선택

2. **Import 화면**에서 **[Proobject] > [Import Excel Artifacts]**를 선택한 후 **[Next]** 버튼을 클릭한다.




Resource Import - Import 화면

3. **Resource Import 화면**에서 **[Browse..]** 버튼을 클릭해서 Import할 Excel 파일을 선택한다. 타겟 프로젝트를 선택한 후 **[Finish]** 버튼을 클릭한다.




Resource Import - Select Excel File

### 리소스 엑셀 파일 포맷

다음의 포맷으로 Excel 파일을 작성하면 해당 파일을 Import하여 ProObject 리소스를 생성할 수 있다.

| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 업무영역 | | | | | 프로그램 정보 | | | Biz Logic | | | | |
| 2 | L1 | L2 | L3 | L4 | 그룹명 | 프로그램ID class Name | 프로그램명 (한글) logicalName | | 유형 | Seq | 처리명 (logical) displayName | 처리모듈ID (physical) methodName | 처리로직 |
| 3 | com | tmax | obj | artifact | | ArtifactBO01 | ArtifactBO01 | | BO | 0 | 테스트메소드11 | testMethod11 | comment_테스트메소드11_1 |
| 4 | | | | | | | | | IM | 1 | 이너01 | Inner1 | comment_이너01_1.1 |
| 5 | | | | | | | | | VM | 2 | 버추얼01 | Virtual1 | comment_버추얼01_1.2 |
| 6 | com | tmax | obj | artifact | | ArtifactBO02 | | | BO | 3 | 알리 | m1_phy | comm_m1 |
| 7 | | | | | | | | | LM | 1.2 | 루프02 | Loop2 | comment_루프01_1.1.4 |
| 8 | | | | | | | | | IM | 1.1.1 | 이너02 | Inner2 | comment_루11프01_1.1.2 |
| 9 | | | | | | | | | LM | 1.1 | 루프01 | Loop1 | comment_루프01_1.1.2 |
| 10 | | | | | | | | | AM | 1.3 | 어사인01 | Assign1 | cd |
| 11 | | | | | | | | | GRM | 1.4 | 겟리01 | GetReply1 | cdfdfe |
| 12 | | | | | | | | | BO | 0 | 테스트메소드21 | testMethod21 | comment_테스트메소드11_1 |
| 13 | | | | | | | | | IM | 1 | 이너01 | Inner1 | comment_이너01_1.1 |
| 14 | | | | | | | | | VM | 2 | 버추얼01 | Virtual1 | comment_버추얼01_1.2 |

Import용 Excel 형식

| 컬럼 | 항목명 | 설명 |
|---|---|---|
| A | L1 | 업무영역에 대한 정보를 입력한다. (예: com) |
| B | L2 | 업무영역에 대한 정보를 입력한다. (예: tmax) |
| C | L3 | 업무영역에 대한 정보를 입력한다. (예: obj) |
| D | L4 | 업무영역에 대한 정보를 입력한다. (예: artifact) |

| 컬럼 | 항목명 | 설명 |
|---|---|---|
| E | 그룹명 | 프로그램 그룹 정보를 입력한다. |
| F | 프로그램ID | 프로그램 ID를 입력한다. 리소스 Import에 사용되는 컬럼이다.<br><br>(예: ArtifactBO01) |
| G | 프로그램명 (한글) | 프로그램 이름을 한글로 입력한다. 리소스 Import에 사용되는 컬럼이다. (예: Artifact테스트BO01) |
| H | blank | 공란 |
| I | 유형 | Biz Logic 정보를 입력한다. 입력된 값을 기준으로 ProObject 리소스를 생성한다.<br>◦ AM : Assign Module<br>◦ BO<br>◦ COND : Condition<br>◦ ET : ETL Task<br>◦ GRM : Get Reply Module<br>◦ IM : Inner Module<br>◦ JO<br>◦ LM : Loop Module<br>◦ NT : Normal Task<br>◦ PART : Partitioner<br>◦ SO<br>◦ STEP<br>◦ VM : Virtual Module |
| J | Seq | Biz Logic 정보를 입력한다. (예: 0) |
| K | 처리명 (logical) | Biz Logic에서 처리되는 논리적 이름을 입력한다. 리소스 Import에 사용되는 컬럼이다. (예: 테스트메소드11 ) |
| L | 처리모듈ID (physical) | Biz Logic에서 처리되는 물리적 이름을 입력한다. 리소스 Import에 사용되는 컬럼이다. (예: testMethod11) |
| M | 처리로직 | 처리로직 정보를 입력한다. (예: comment_테스트메소드11_1) |

작성된 Excel 파일의 컬럼으로 BO, SO, JO 리소스를 생성하고 메소드를 선언한다.

- **BO/SO**

| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 업무영역 | | | | 프로그램 정보 | | | | Biz Logic | | | | |
| 2 | L1 | L2 | L3 | L4 | 그룹명 | 프로그램ID class Name | 프로그램명 (한글) logicalName | | 유형 | Seq | 처리명 (logical) displayName | 처리모듈ID (physical) methodName | 처리로직 |
| 3 | com | tmax | obj | artifact | | ArtifactBO01 | ArtifactBO01테스트 | | BO | 0 | 테스트메소드11 | testMethod11 | comment_테스트메소드11_1 |
| 4 | | | | | | | | | IM | 1 | 이너01 | Inner1 | comment_이너01_1.1 |
| 5 | | | | | | | | | VM | 2 | 버추얼01 | Virtual1 | comment_버추얼01_1.2 |
| 6 | com | tmax | obj | artifact | | ArtifactBO02 | | | BO | 3 | 알리 | m1_phy | comm_m1 |
| 7 | | | | | | | | | LM | 1.2 | 루프02 | Loop2 | comment_루프01_1.1.4 |
| 8 | | | | | | | | | IM | 1.1.1 | 이너02 | Inner2 | comment_루11프01_1.1.2 |
| 9 | | | | | | | | | LM | 1.1 | 루프01 | Loop1 | comment_루프01_1.1.2 |
| 10 | | | | | | | | | AM | 1.3 | 어사인01 | Assign1 | cd |
| 11 | | | | | | | | | GRM | 1.4 | 겟리01 | GetReply1 | cdfdfe |
| 12 | | | | | | | | | BO | 0 | 테스트메소드21 | testMethod21 | comment_테스트메소드11_1 |
| 13 | | | | | | | | | IM | 1 | 이너01 | Inner1 | comment_이너01_1.1 |
| 14 | | | | | | | | | VM | 2 | 버추얼01 | Virtual1 | comment_버추얼01_1.2 |
| 15 | | | | | | | | | LM | 1.2 | 루프02 | Loop2 | comment_루프01_1.1.4 |
| 16 | | | | | | | | | LM | 1.1 | 루프01 | Loop1 | comment_루프01_1.1.2 |
| 17 | | | | | | | | | AM | 1.3 | 어사인01 | Assign1 | cd |

BO Import 예제

| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 업무영역 | | | | 프로그램 정보 | | | | Biz Logic | | | | |
| 2 | L1 | L2 | L3 | L4 | 그룹명 | 프로그램ID class Name | 프로그램명 (한글) logicalName | | 유형 | Seq | 처리명 (logical) displayName | 처리모듈ID (physical) methodName | 처리로직 |
| 21 | com | tmax | obj | artifact | | ArtifactSO01 | ArtifactSO01테스트 | | SO | 0 | 테스트메소드11 | testMethod11 | comment_테스트메소드11_1 |
| 22 | | | | | | | | | IM | 1 | 이너01 | Inner1 | comment_이너01_1.1 |
| 23 | | | | | | | | | VM | 2 | 버추얼01 | Virtual1 | comment_버추얼01_1.2 |
| 24 | com | tmax | obj | artifact | | ArtifactBO02 | | | BO | 3 | | m1_phy | comm_m1 |
| 25 | | | | | | | | | LM | 1.2 | 루프02 | Loop2 | comment_루프01_1.1.4 |
| 26 | | | | | | | | | LM | 1.1 | 루프01 | Loop1 | comment_루프01_1.1.2 |
| 27 | | | | | | | | | AM | 1.3 | 어사인01 | Assign1 | cd |
| 28 | | | | | | | | | GRM | 1.4 | 겟리01 | GetReply1 | cdfdfe |
| 29 | | | | | | | | | SO | 0 | 테스트메소드21 | testMethod21 | comment_테스트메소드11_1 |
| 30 | | | | | | | | | IM | 1 | 이너01 | Inner1 | comment_이너01_1.1 |
| 31 | | | | | | | | | VM | 2 | 버추얼01 | Virtual1 | comment_버추얼01_1.2 |
| 32 | | | | | | | | | LM | 1.2 | 루프02 | Loop2 | comment_루프01_1.1.4 |
| 33 | | | | | | | | | LM | 1.1 | 루프01 | Loop1 | comment_루프01_1.1.2 |
| 34 | | | | | | | | | AM | 1.3 | 어사인01 | Assign1 | cd |
| 35 | | | | | | | | | GRM | 1.4 | 겟리01 | GetReply1 | cdfdfe |

SO Import 예제

| 작업 | 설명 |
|---|---|
| 리소스 생성 | 다음의 컬럼을 Import해서 리소스를 생성한다.<br>• 프로그램ID(F컬럼)<br>• 프로그램명(G컬럼)<br>• 처리명(K컬럼)<br>• 처리모듈ID(L컬럼) |
| 리소스 내부의 Method 선언 | 다음의 컬럼을 Import해서 리소스 내부의 Method를 선언한다.<br>• 처리명(K컬럼)<br>• 처리모듈ID(L컬럼) |
| Method 호출 | 다음의 컬럼을 Import해서 Method를 호출한다.<br>• 프로그램ID(F컬럼)<br>• 처리명(K컬럼)<br>• 처리모듈ID(L컬럼) |

- **JO**

| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 업무영역 | | | | 프로그램 정보 | | | | Biz Logic | | | | |
| 2 | L1 | L2 | L3 | L4 | 그룹명 | 프로그램ID<br>class Name | 프로그램명 (한글)<br>logicalName | | 유형 | Seq | 처리명 (logical)<br>displayName | 처리모듈ID (physical)<br>methodName | 처리로직 |
| 36 | com | tmax | obj | artifact | | ArtifactBO02 | | | BO | 3 | | m1_phy | comm_m1 |
| 37 | com | tmax | obj | artifact | | ArtifactBO03 | | | BO | 4 | | m2_phy | comm_m2 |
| 38 | com | tmax | obj | artifactjo | | ArtifactJO01 | Artifact테스트JO01 | | JO | 0 | | | comment_1 |
| 39 | | | | | | | | | COND | 1 | 블록1 | Cond_1 | comment_블록1_1.1 |
| 40 | | | | | | | | | STEP | 2 | 스텝2 | Step_2 | comment_ETLTask1_1.1.1 |
| 41 | | | | | | | | | STEP | 1.1 | 스텝1 | Step_1 | comment_ETLTask1_1.1.1 |
| 42 | | | | | | | | | PART | 1.1.1 | 파티셔너1 | Partitioner_1 | comment_온라인Task1_1.1.2 |
| 43 | | | | | | | | | ET | 1.1.1.1 | ETLTask2 | ETL_1 | comment_ETLTask2_1.2.1 |
| 44 | | | | | | | | | PART | 2.1 | 파티셔너2 | Partitioner_2 | comment_온라인Task1_1.1.2 |
| 45 | | | | | | | | | NT | 2.1.1 | 노멀1 | NO_1 | comment_ETLTask2_1.2.1 |

JO Import 예제

| 작업 | 설명 |
|---|---|
| 리소스 생성 | 다음의 컬럼을 Import해서 리소스를 생성한다.<br>• 프로그램ID(F컬럼)<br>• 프로그램명(G컬럼) |

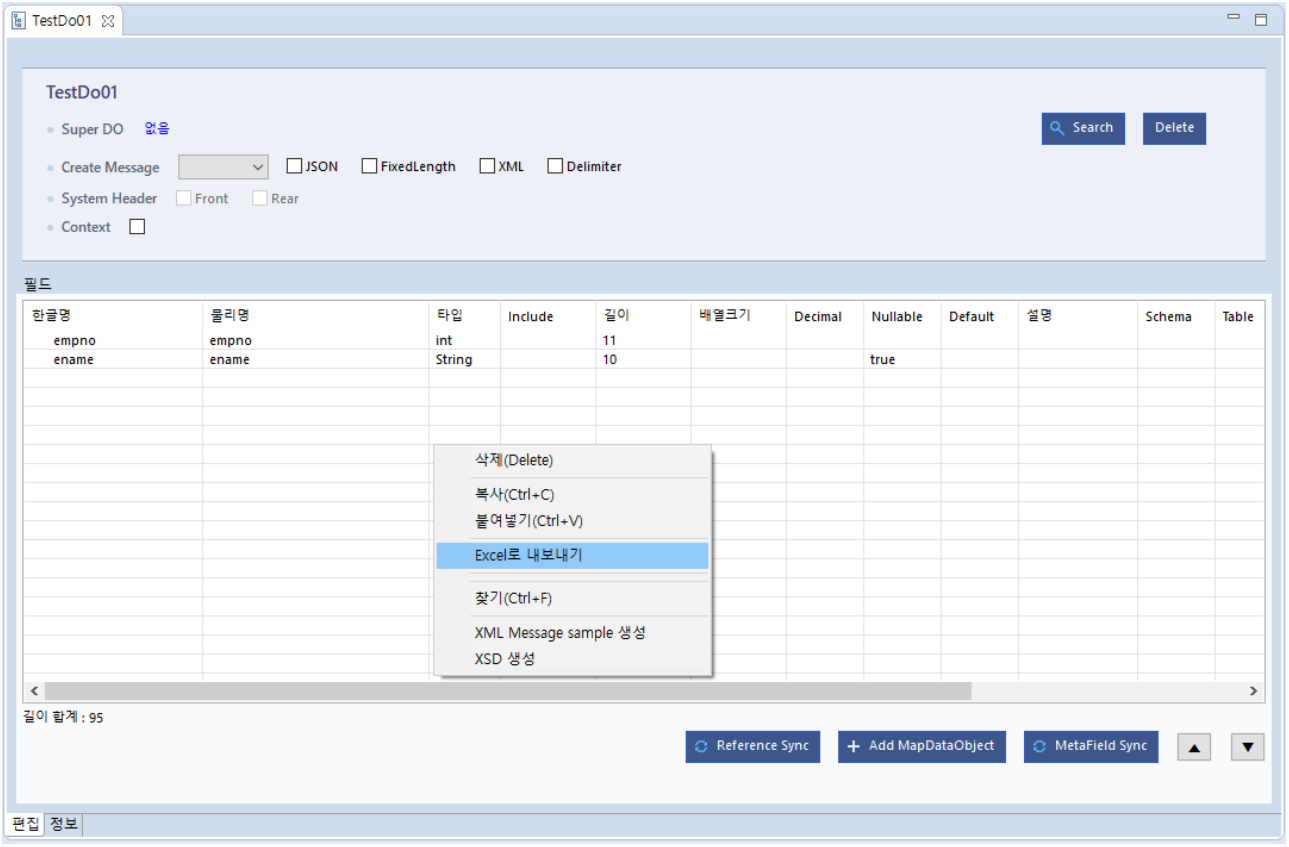
## 5.2.2. Excel Import/Export

DO를 Excel로 Import/Export할 수 있다. DO에 컬럼이 많은 사이트 환경의 경우 해당 기능은 반복되는 작업을 최소화하여 생상성을 높여줄 수 있다.

### 5.2.2.1. Export

다음의 과정으로 Excel 파일을 Export한다.

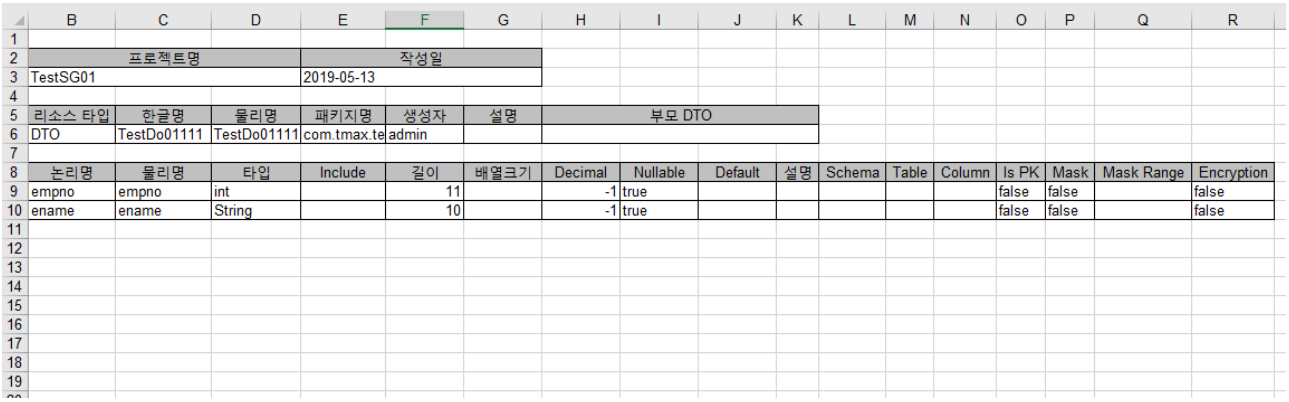
1. Export할 DO를 조회한 후 Grid의 컨텍스트 메뉴에서 **[Excel로 내보내기]**를 클릭한다.




DO Excel Export (1)

2. 윈도우 탐색기에서 저장할 위치를 선택한 후 **[저장]** 버튼을 클릭한다.

   다음은 DO 정보를 Excel로 Export한 파일의 예이다.



| | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | | | | |
| 2 | 프로젝트명 | | | 작성일 | | | | | | | | | | | | | |
| 3 | TestSG01 | | | 2019-05-13 | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | | | |
| 5 | 리소스 타입 | 한글명 | 물리명 | 패키지명 | 생성자 | 설명 | 부모 DTO | | | | | | | | | | |
| 6 | DTO | TestDo01111 | TestDo01111 | com.tmax.te | admin | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | | | |
| 8 | 논리명 | 물리명 | 타입 | Include | 길이 | 배열크기 | Decimal | Nullable | Default | 설명 | Schema | Table | Column | Is PK | Mask | Mask Range | Encryption |
| 9 | empno | empno | int | | 11 | | -1 | true | | | | | | false | false | | false |
| 10 | ename | ename | String | | 10 | | -1 | true | | | | | | false | false | | false |
| 11 | | | | | | | | | | | | | | | | | |
| 12 | | | | | | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | | | | | | |
| 14 | | | | | | | | | | | | | | | | | |
| 15 | | | | | | | | | | | | | | | | | |
| 16 | | | | | | | | | | | | | | | | | |
| 17 | | | | | | | | | | | | | | | | | |
| 18 | | | | | | | | | | | | | | | | | |
| 19 | | | | | | | | | | | | | | | | | |

DO Excel Export (2)

### 5.2.2.2. Import

다음은 Export된 Excel 파일을 편집한 후 Import하는 과정에 대한 설명디다. Export한 DO TestDo01를 TestDo01New로 Import한다.

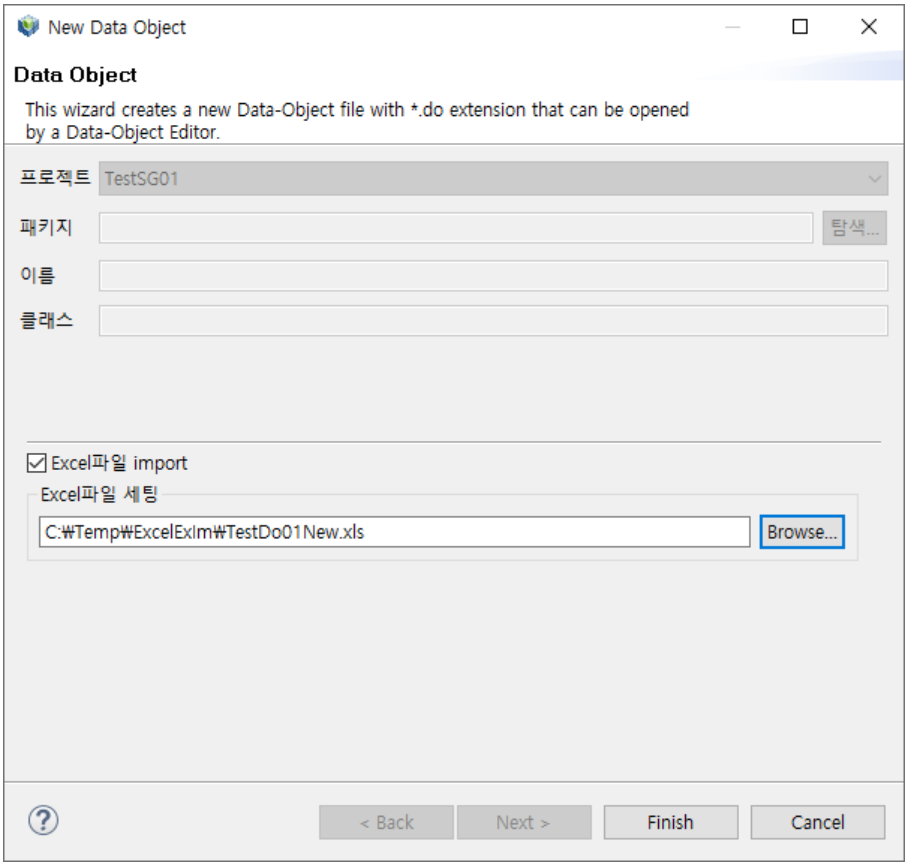
1. Export한 파일의 내용을 수정한다.

| | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | | | | |
| 2 | 프로젝트명 | | | 작성일 | | | | | | | | | | | | | |
| 3 | TestSG01 | | | 2019-05-13 | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | | | |
| 5 | 리소스 타입 | 한글명 | 물리명 | 패키지명 | 생성자 | 설명 | 부모 DTO | | | | | | | | | | |
| 6 | DTO | TestDo01111 | TestDo01111 | com.tmax.te | admin | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | | | |
| 8 | 논리명 | 물리명 | 타입 | Include | 길이 | 배열크기 | Decimal | Nullable | Default | 설명 | Schema | Table | Column | Is PK | Mask | Mask Range | Encryption |
| 9 | empno | empno | int | | 11 | | -1 | true | | | | | | false | false | | false |
| 10 | ename | ename | String | | 10 | | -1 | true | | | | | | false | false | | false |
| 11 | | | | | | | | | | | | | | | | | |
| 12 | | | | | | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | | | | | | |
| 14 | | | | | | | | | | | | | | | | | |
| 15 | | | | | | | | | | | | | | | | | |
| 16 | | | | | | | | | | | | | | | | | |
| 17 | | | | | | | | | | | | | | | | | |
| 18 | | | | | | | | | | | | | | | | | |
| 19 | | | | | | | | | | | | | | | | | |

DO Excel Import(1)

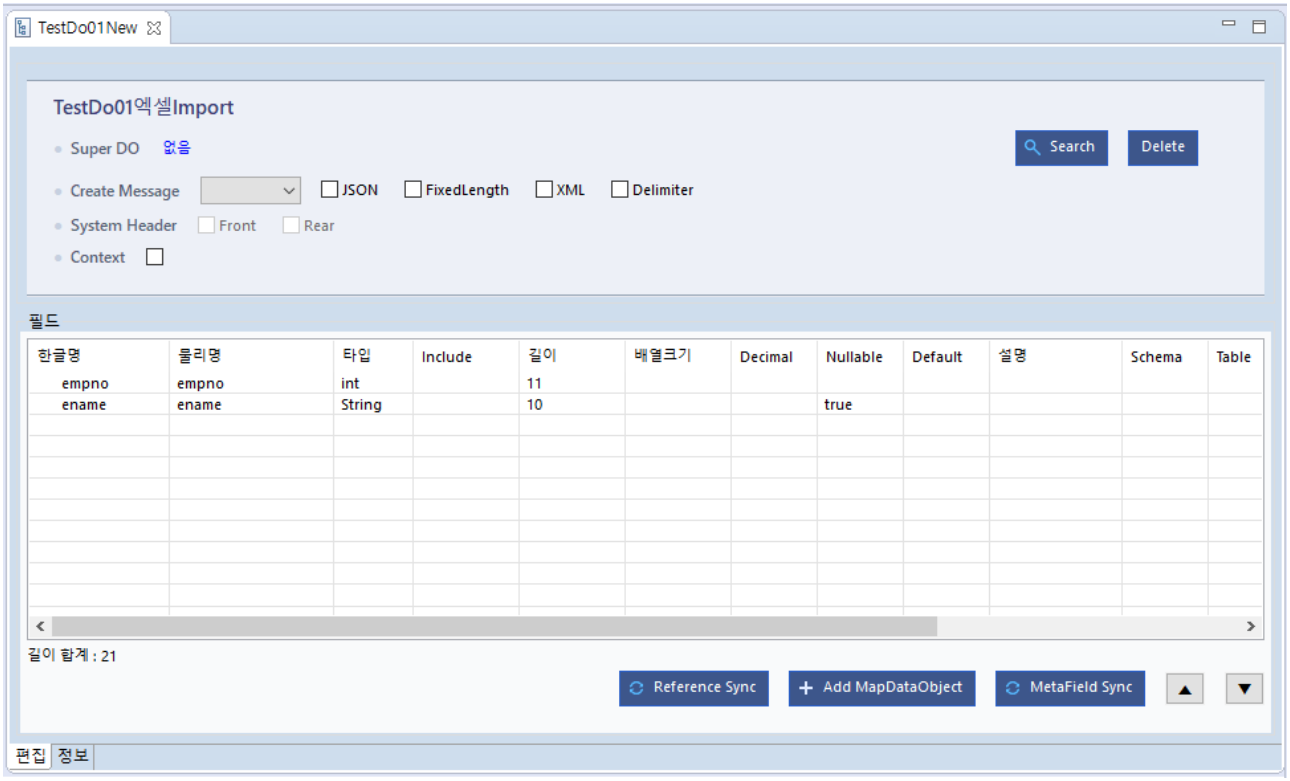
2. **[Proobject] > [New] > [Data Object]**를 클릭하면 **New Data Object 화면**이 열린다.

   **'Excel 파일 Import'** 항목을 체크한 후 **[Browse..]** 버튼을 클릭해서 Import할 Excel 파일을 선택한 후 **[Finish]** 버튼을 클릭한다. Import할 파일을 Import 포맷에 맞아야 한다.




DO Excel Import(2)

3. Import가 완료되면 **DO 화면**에서 정보를 확인할 수 있다.



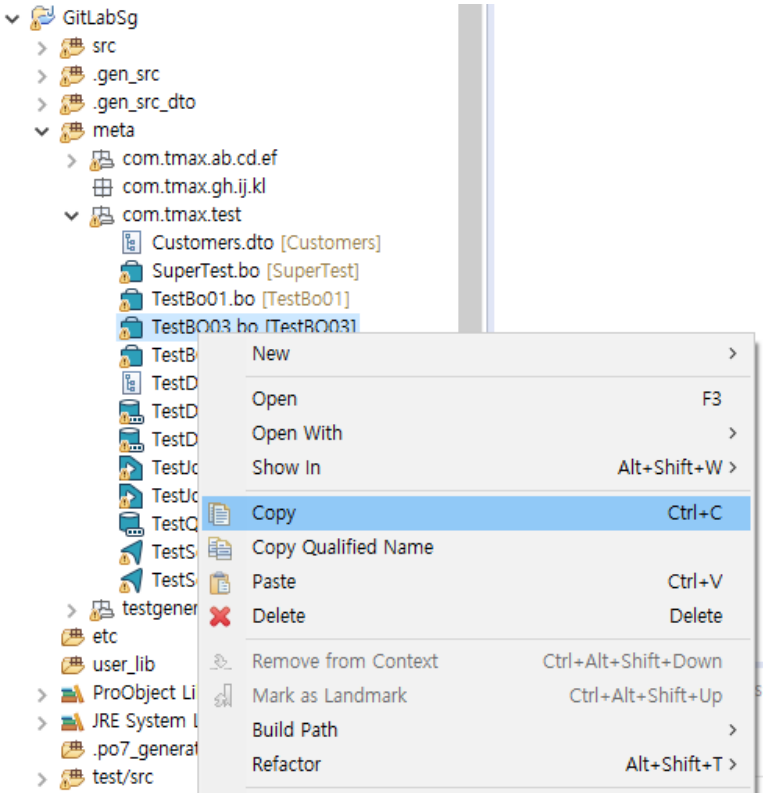
DO Excel Import(3)

## 5.2.3. Resource 복사/붙여넣기

리소스의 메타 내부에 들어있는 path 정보의 노드 값이 변경되어야 하기 때문에 패키지 간의 Resource 복사/붙여넣기 기능을 제공한다.

- **Resource 복사**

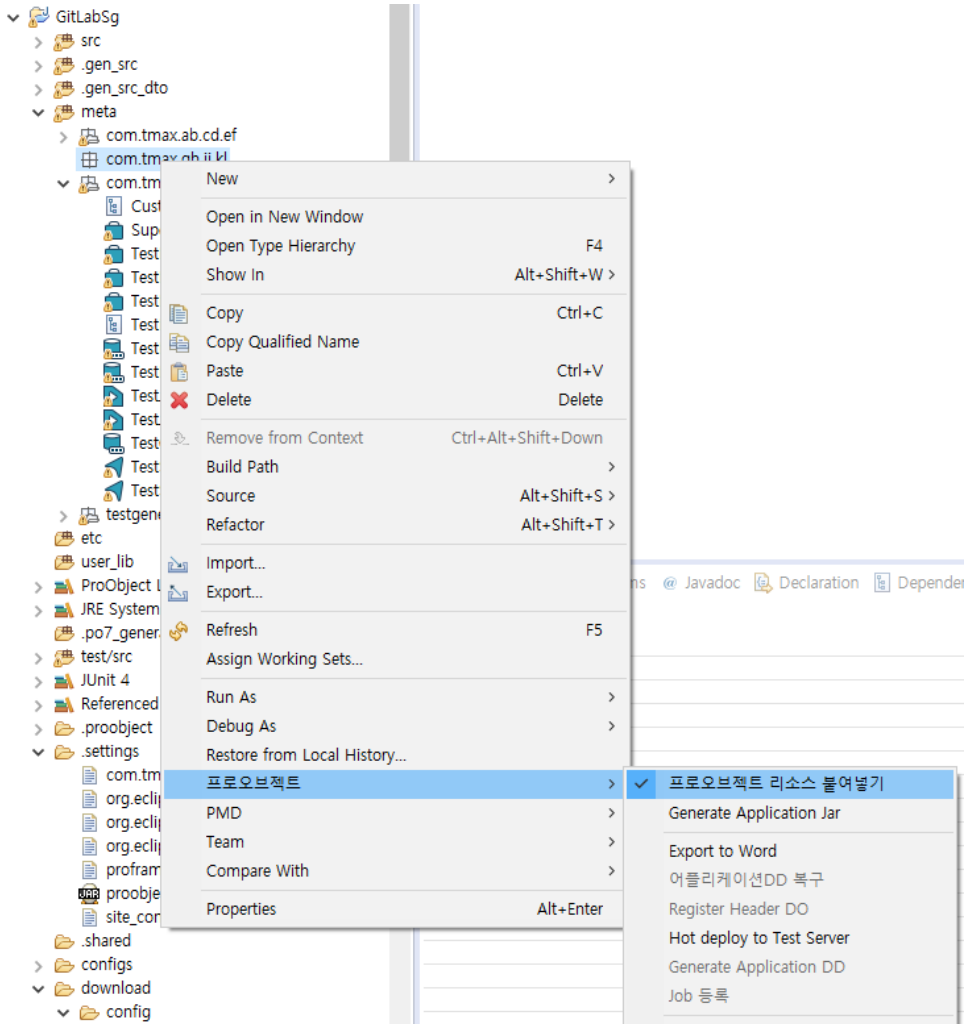
  **Package Explorer**의 복사할 리소스를 선택한 후 컨텍스트 메뉴에서 **[Copy]**를 선택한다. 해당 리소스를 선택한 후 <Ctrl> + C를 눌러도 동일하게 동작한다.



Resource 복사하기

- **Resource 붙여넣기**

  **Package Explorer**에서 붙여넣기할 패키지를 선택한 후 컨텍스트 메뉴에서 **[프로오브젝트] > [프로오브젝트 리소스 붙여넣기]**를 선택한다.



Resource 붙여넣기

**Package Explorer**에서 컨텍스트 메뉴에 **[paste]**는 리소스의 메타를 그대로 복사하기 때문에 정상적인 복사가 되지 않기 때문에 리소스를 붙여넣는 경우 위의 **[프로오브젝트] > [프로오브젝트 리소스 붙여넣기]** 메뉴를 사용해야 한다.

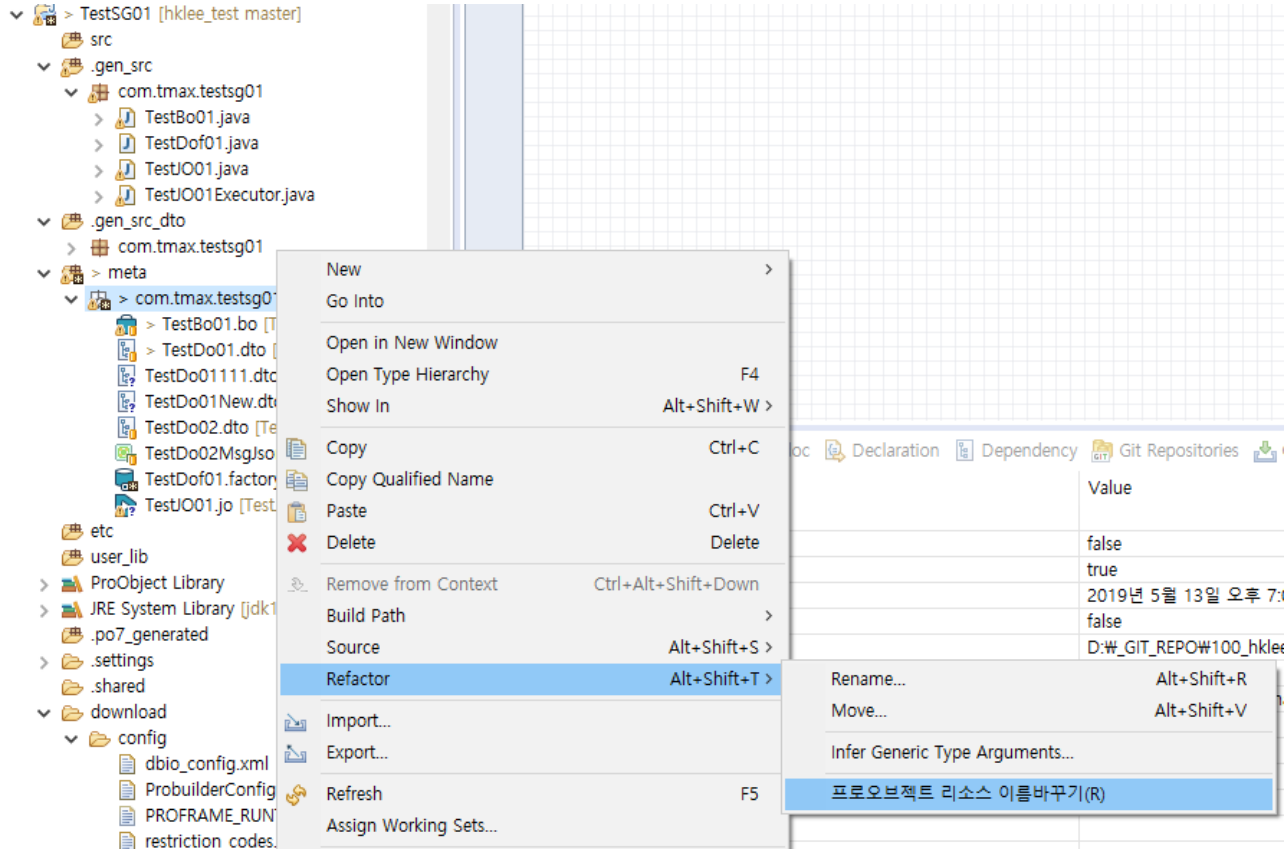
## 5.2.4. Resource Refactoring

ProObject 리소스의 이름을 변경하거나 패키지의 이름을 변경하는 방법에 대해서 설명한다.

### 5.2.4.1. 패키지/리소스명 변경

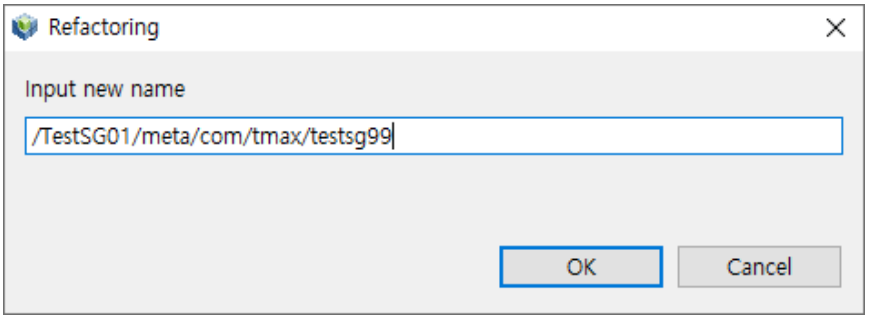
다음은 서비스 그룹이나 애플리케이션의 패키지/리소스 이름을 변경하는 방법에 대한 설명이다.

1. **Package Explorer**의 변경할 패키지를 선택한 후 컨텍스트 메뉴에서 **[Refactor] > [프로오브젝트 리소스 이름바꾸기]** 메뉴를 선택한다.



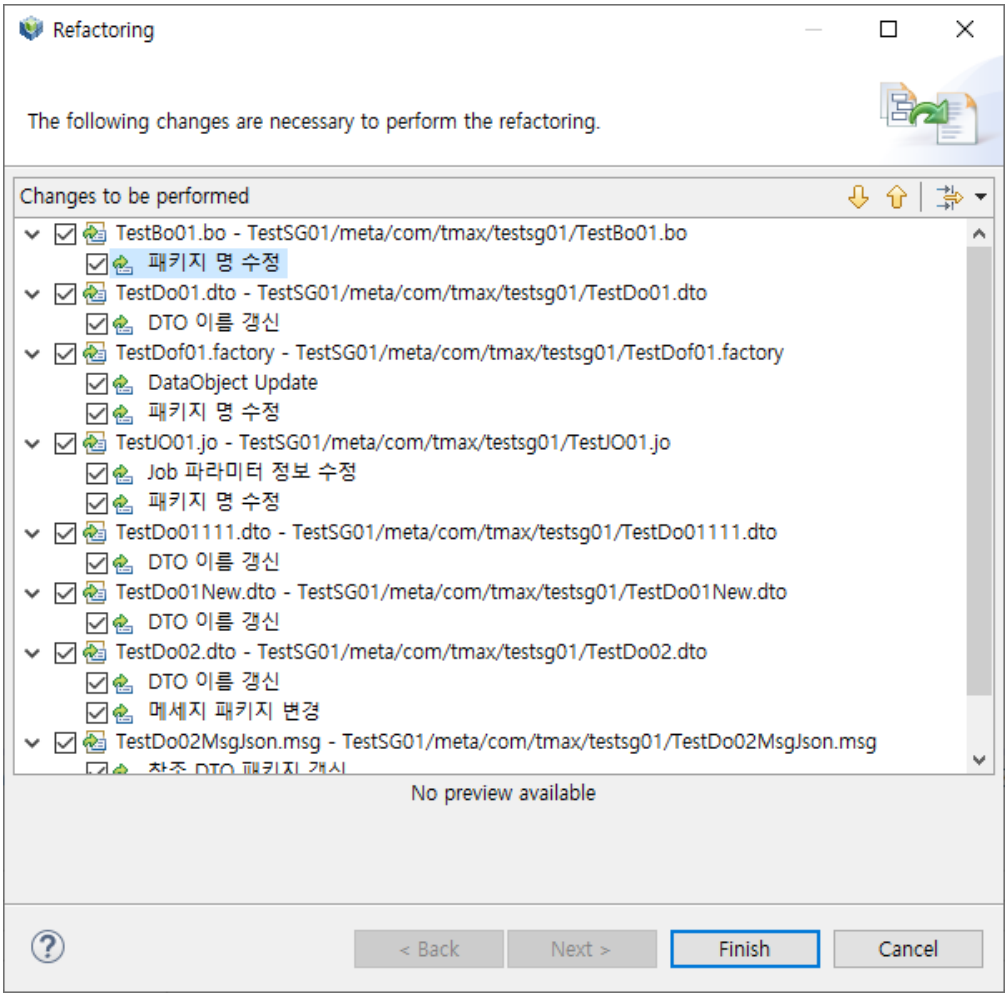

Package명 변경(1)

2. **Refactorying 화면**에서 변경할 패키지 이름을 입력한 후 **[OK]** 버튼을 클릭한다.




Package명 변경(2)

3. 다음과 같이 변경할 리소스별 상세 변경 정보가 조회된다. 목록에서 변경할 리소스를 선택한 후 **[Finish]** 버튼을 클릭한다.



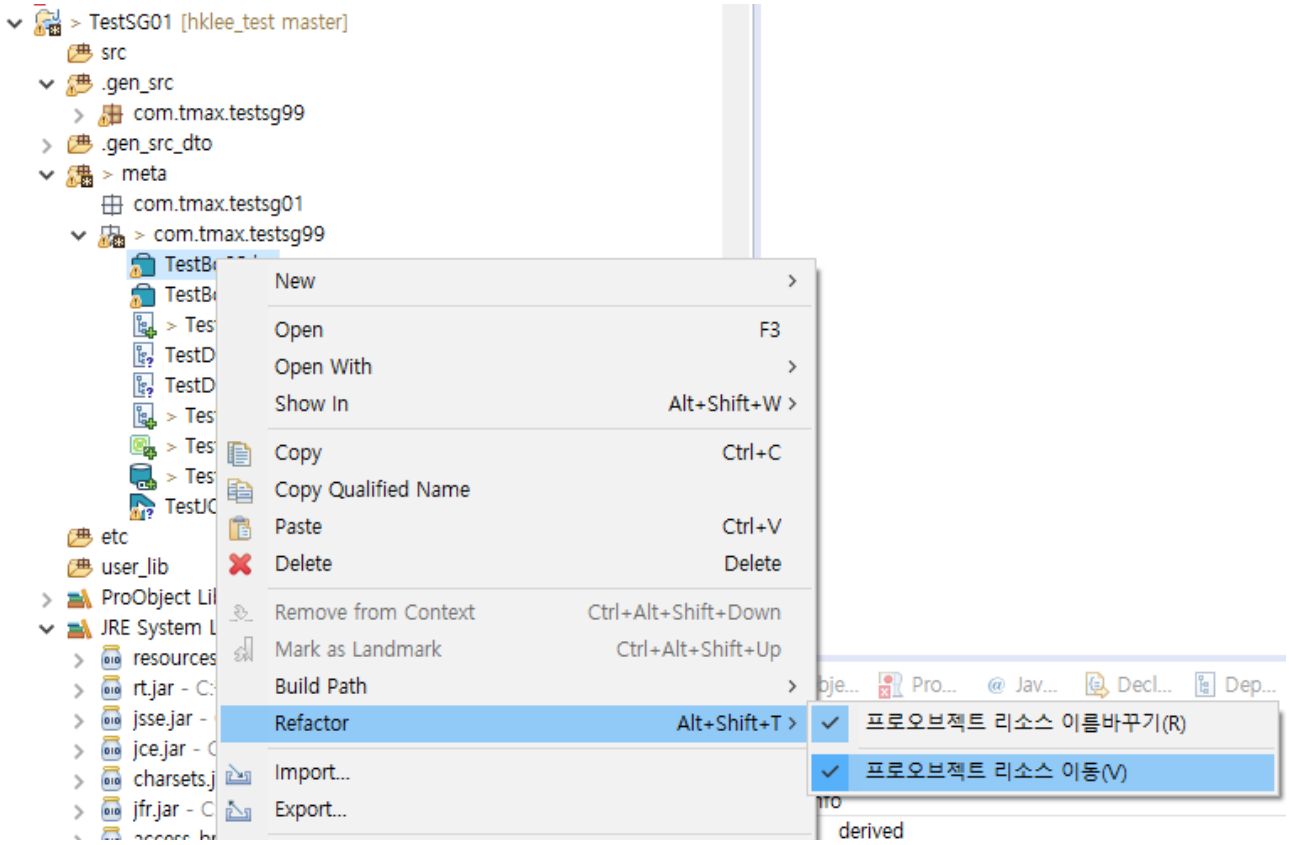
Package명 변경(3)

4. Refactorying이 완료되면 선택한 패키지의 모든 리소스가 수정한 패키지로 된다.

### 5.2.4.2. 리소스 패키지 이동

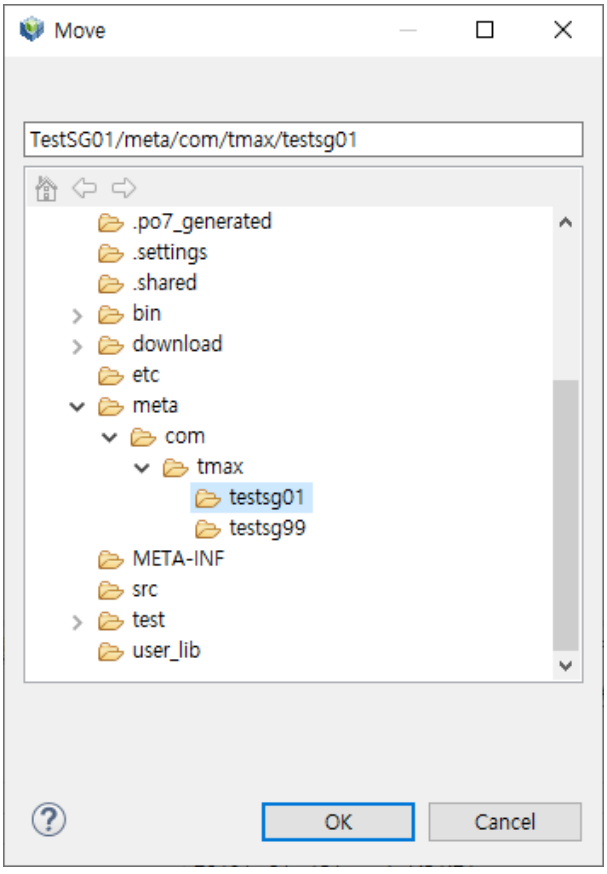
다음은 리소스 패키지를 이동하는 과정에 대한 설명이다.

1. **Package Explorer**의 변경할 패키지를 선택한 후 컨텍스트 메뉴에서 **[Refactor] > [프로오브젝트 리소스 이동]**을 선택한다.



리소스 패키지 이동(1)

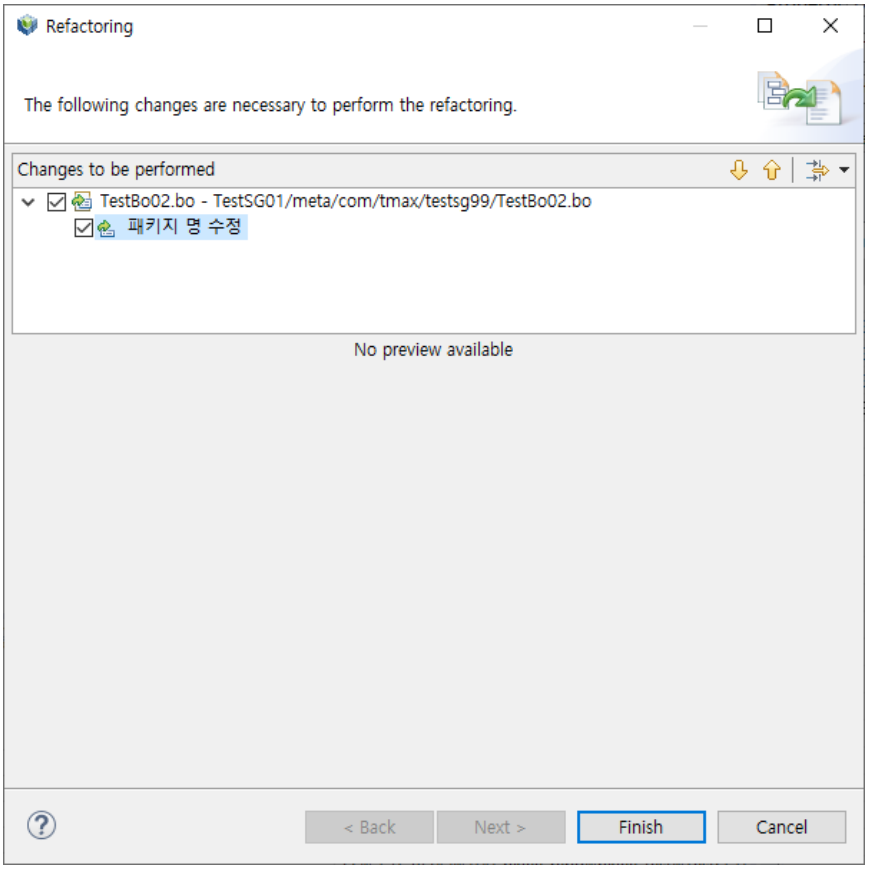
2. **Refactorying 화면**에서 변경할 패키지를 선택한 후 **[OK]** 버튼을 클릭한다.



리소스 패키지 이동 (2)

3. 다음과 같이 변경할 리소스별 상세 변경 정보가 조회된다. 목록에서 변경할 리소스를 선택한 후 **[Finish]** 버튼을

클릭한다.




리소스 패키지 이동 (3)

## 5.2.5. Commit/Push 금지 기능

본 절에서는 Commit/Push할 때 XML이나 SQL에 금지 함수를 사용한 경우 작업을 중단하는 기능에 대해서 설명한다. 금지함수 사용을 위한 설정이 선행되어야 하지만, 일반적으로 개발환경 서버의 접근 권한이 있는 담당자가 공통 설정을 한 후 함수를 추출한다.

### 5.2.5.1. 금지 함수 검출

XML에서 금지 함수를 사용한 경우 Commit/Push되지 않도록 하려면 환경설정이 필요하다. 개발자가 해당 금지 함수 목록에 있는 금지 함수를 사용한 경우 Push가 중단된다. 금지 함수는 SO 또는 BO, serviceobject interface 또는 buisinessobject interface의 구현체 java 파일에서 추출한다.

다음은 금지 함수 검출하는 과정에 대한 설명이다.

1. 금지 함수를 사용하기 위해서는 $PROOBJECT_HOME/system/config/ProbuilderConfig.xml에 사용 여부가 체크되어 있어야 한다.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<probuilder_config xmlns= .. 중략 ..
    <restriction-config xmlns="">
       <restriction-code-config use="true"/>
    </restriction-config>
```

```
</probuilder_config>
```

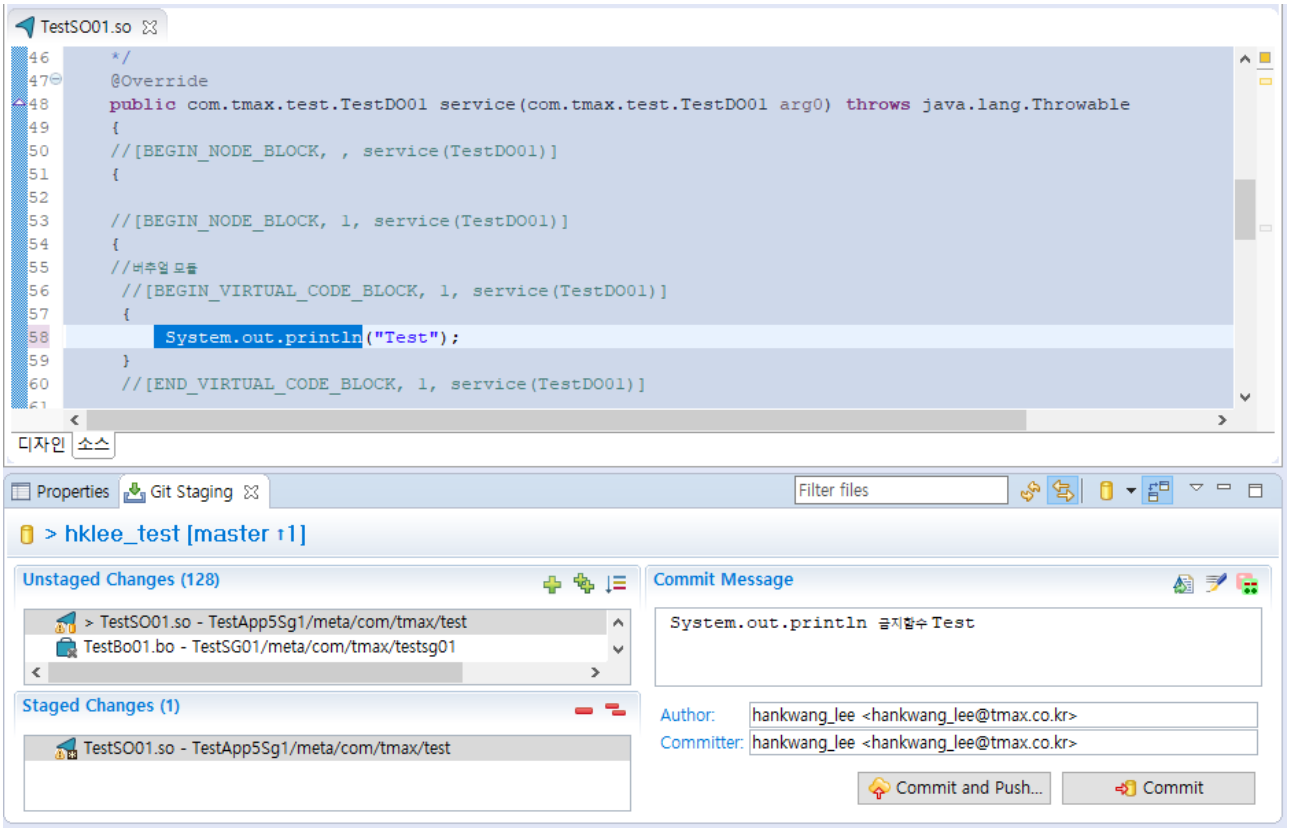
2. 다음은 시스템을 종료시키는 exit 및 logger를 사용하지 않고 println을 사용하도록 상세한 금지함수 내역을 ${PROOBJECT_HOME}/system/config/restriction_codes.xml에 설정한다.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
     :
   중략
     :
    <ns22:restriction-code code="java.lang.System.exit" action="restrict"/>
    <ns22:restriction-code code="java.io.PrintStream.println" alternative="logger.info"
action="restrict"/>
</ns22:restriction-codes>
```

다음은 금지 함수 설정에 사용되는 태그에 대한 설명이다.

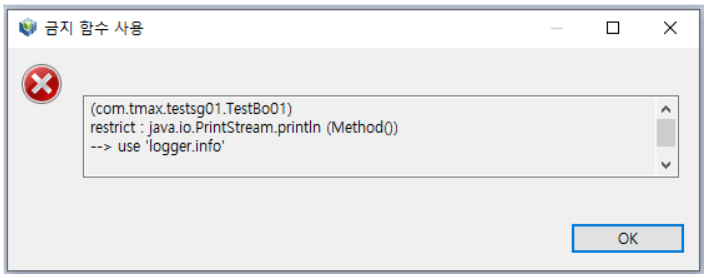
| 항목 | 설명 |
|---|---|
| code | 금지 함수는 패키지명부터 클래스명 메소드명까지 java naming 규칙에 따라 작성한다. (argument는 필요없음) |
| action | ◦ restrict : 커밋 금지<br>◦ warn : 커밋 시 경고 메시지 출력<br>◦ skip : 제한하지 않음 |
| alternative | 대체할 함수가 있을 경우에 추가하면 되며, action 시 해당 정보를 표시해준다. |

3. 금지 함수의 검출은 Push 단계에 확인할 수 있다. 2번 과정의 Bold된 부분과 같이 설정하고 다음과 같이 'System.out.println' API를 입력한 후 **[Commit and Push...]** 버튼을 클릭하면 금지 함수사용으로 인해 Push가 되지 않는다.




금지함수 검출 - action 설정

금지함수를 사용하고 Push를 요청한 경우 다음의 경고 화면이 나타난다.




금지함수 검출 - 경고

### 5.2.5.2. SQL 금지 함수 검출

개발자가 작성한 Query 및 Hint절에 SQL에 금지 문자가 있을 때 Push를 막을 수 있다.

다음은 SQL 금지 함수 검출하는 과정에 대한 설명이다.

1. $PROOBJECT_HOME/system/config/dbio_config.xml에 Query의 및 Hint절에 금지 함수를 설정한다.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dbio-config xmlns="http://www.tmax.co.kr/proobject/dbio-config">
      :
    중략
     :
   <tool-config>
       :
     중략
      :
```

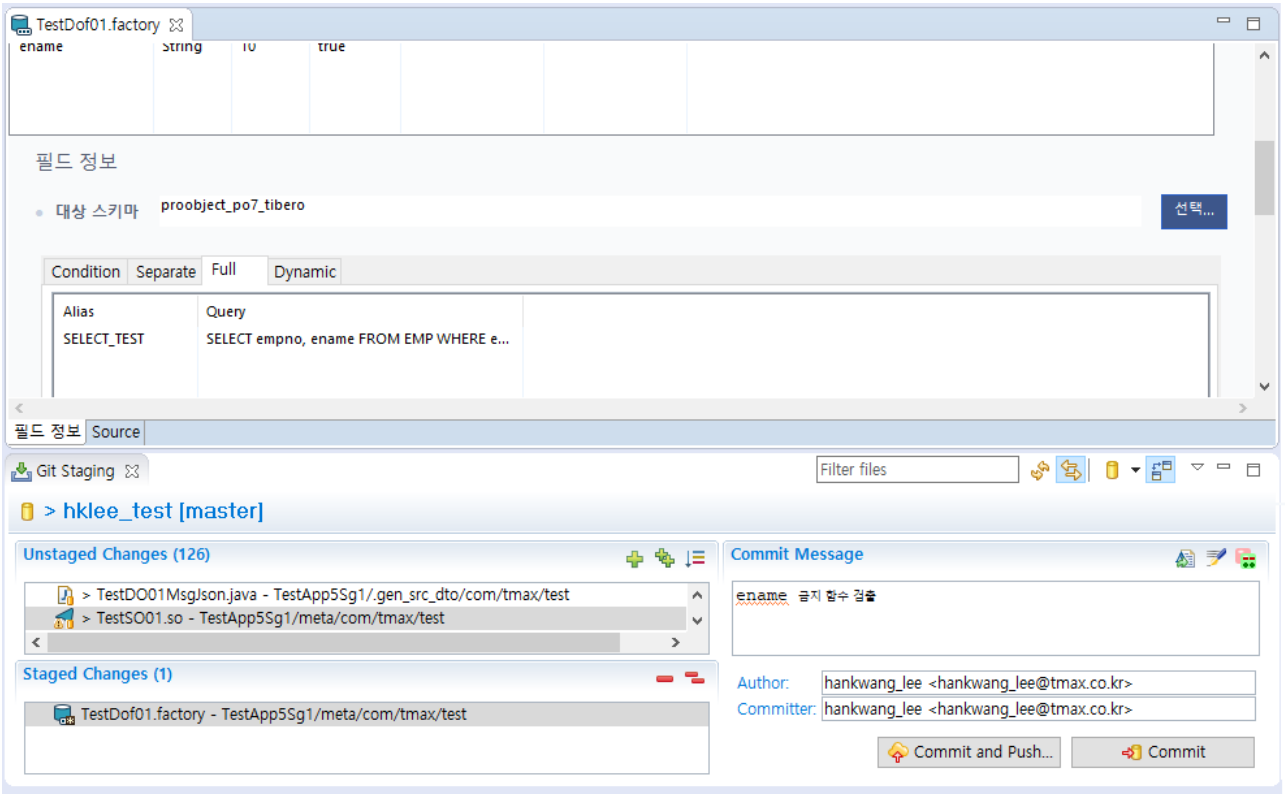
```
        <sqlRestrictedKeyword>ename</sqlRestrictedKeyword>
        <sqlRestrictedHint>FULL</sqlRestrictedHint>
    </tool-config>
</dbio-config>
```

다음은 금지 함수 설정에 사용되는 태그에 대한 설명이다.

| 항목 | 설명 |
|---|---|
| <sqlRestrictedKeyword> | Hint절 외의 변수에 대해 금지 물자를 설정한다. |
| <sqlRestrictedHint> | Query의 Hint절(/*+ Hint */)내에 금지 문자를 설정한다. |

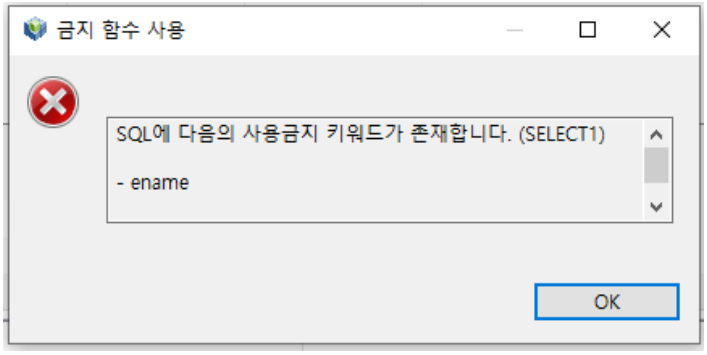
2. 금지 함수의 검출은 Push 단계에 확인할 수 있다.

   2번 과정의 Bold된 부분과 같이 설정하고 다음과 같이 Query 및 Hint 절에 설정한 금지 문자를 입력한 후 **[Commit and Push...]** 버튼을 클릭하면 금지 함수사용으로 인해 Push가 되지 않는다.



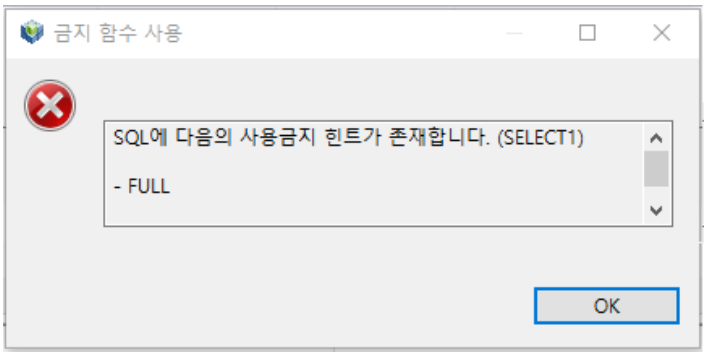

금지 SQL 검출 - 변

SQL에 금지함수를 사용하고 Push를 요청한 경우 다음의 경고 화면이 나타난다.



금지 SQL 검출 - 변수

Hint 절에 금지함수를 사용하고 Push를 요청한 경우 다음의 경고 화면이 나타난다.



금지 SQL 검출 - Hint절

## 5.2.6. PMD

PMD는 작성된 소스 코드를 정해진 규칙에 따라 자동으로 분석해 잘못된 코드의 문제점을 유형별로 진단하고 간략한 설명을 제공해줌으로써 코드의 품질을 향상시키는 데 도움을 주는 툴로 개발자가 프로그래밍을 하는데 가이드 역할을 한다.

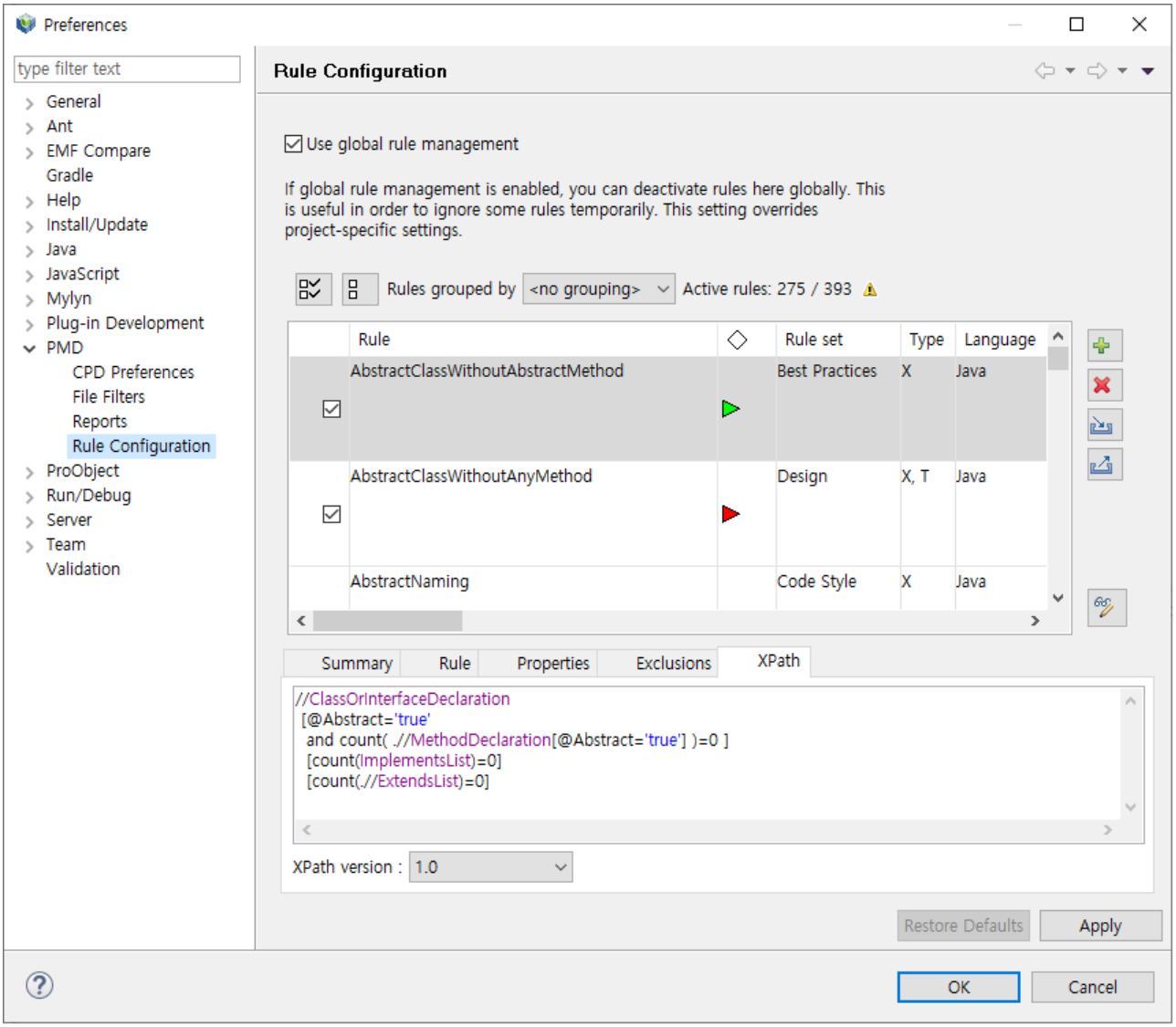
PMD를 사용하기 위해서 PMD를 생성하고 PMD Rule 적용한 후 Rule에 의한 Check Code를 수행한다. Custom Rule 생성 및 Rule 배포는 OwnerShip을 가지는 담당자가 하며, 공통 Rule 파일(ruleset.xml) 적용은 개발자가 각각 진행한다.

### 5.2.6.1. Custom Rule 생성

PMD는 기본 Rule을 활성화할 수 있지만, Customizing 기능도 함께 제공한다.

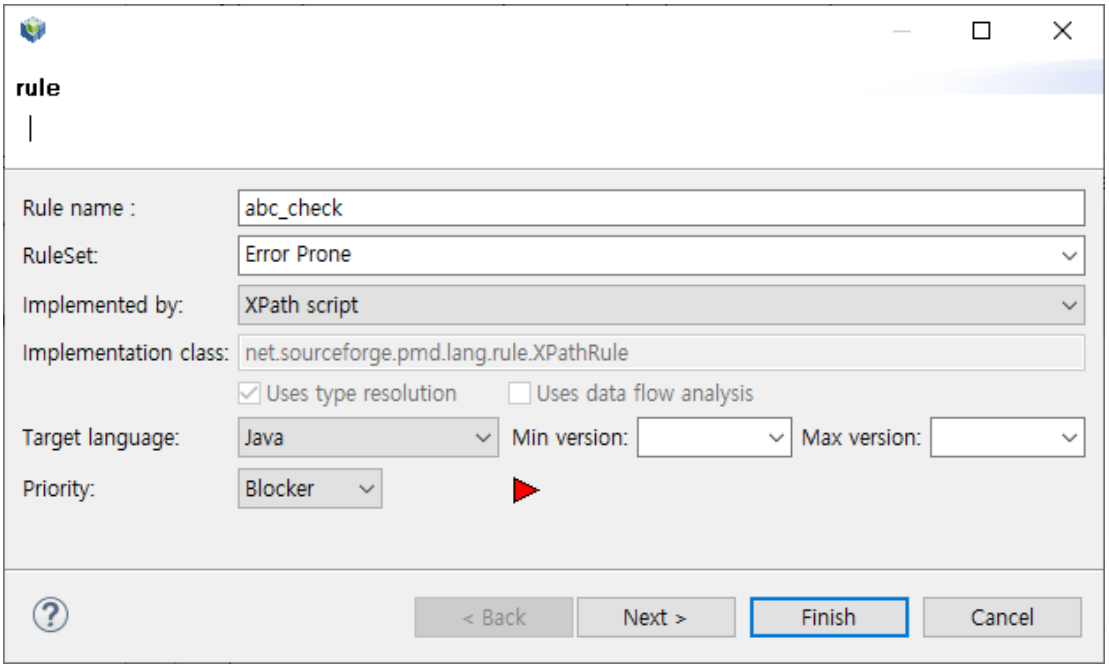
다음은 Custom Rule을 생성하는 과정에 대한 설명이다.

1. **[Window] > [Preferences] > [ProObject] > [ProMapper]** 메뉴를 선택하면 현재 정의된 Rule 내역 및 편집 가능한 화면이 나타난다. **Rule Configuration 영역**에서 **[+]** 버튼을 클릭한다.




Custom Rule 생성(1)

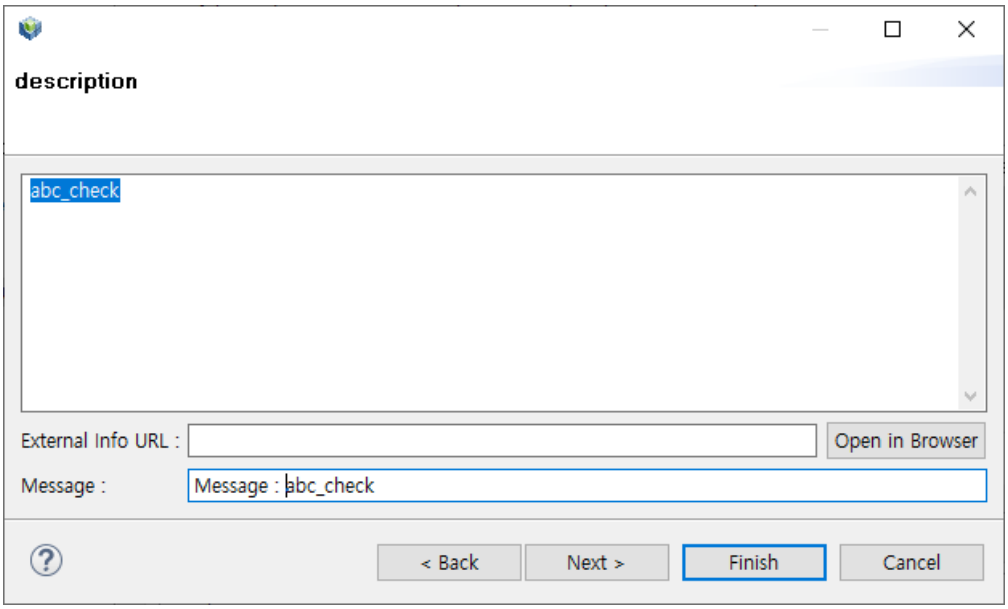
2. Rule의 기본 정보를 설정한다. 각 항목을 입력한 후 **[Finish]** 버튼을 클릭한다.




Custom Rule 생성(2)

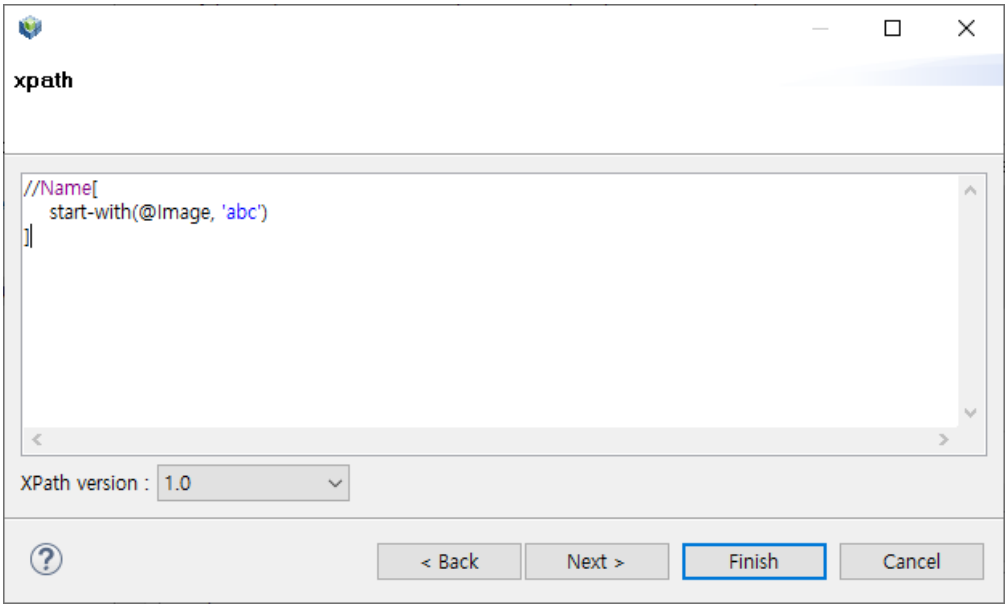
| 항목 | 설명 |
| --- | --- |
| Rule name | 정의할 Rule 이름을 입력한다. |
| RuleSet | Rule을 그룹핑하는 단위이며, 생성할 유형과 맞는 RuleSet을 선택한다. |
| Implemented by | 스크립트 종류를 선택하며, 현재 장표에서는 XPath script의 기준으로 설정한다. |
| Implementation class | Implemented by의 Class 구현체이다. |
| Priorty | 규칙에 매치되는 결과의 중요도를 선택한다.<br>◦ Blocker(High): 심각한 버그가 발생할 수 있는 코드이기 때문에 반드시 Rule을 준수해야 한다. (빨강색)<br>◦ Critical(Medium) : 심각하지는 않지만 버그가 발생할 수 있는 코드이기 때문에 Rule을 준수해야 한다. (하늘색)<br>◦ Urgent(Medium) : 복잡한 코딩, Best Practice 및 보안, 성능 등에 관련 된 내용으로 준수 할 것을 권장한다. (초록색)<br>◦ Important(Medium Low) : 버그가 아니며 표준, 코딩 스타일, 불필요한 코드 및 미사용 코드에 관련된 내용이다. (분홍색)<br>◦ Warning(Low) : 패키지, 클래스, 필드 등 Naming에 관련된 내용이다. (파랑색) |

3. Rule의 설명과 메시지를 입력한 후 **[Next]** 버튼을 클릭한다. 만약 빈 화면이 나오면 **[Next]** 버튼을 한번 더 클릭한다.



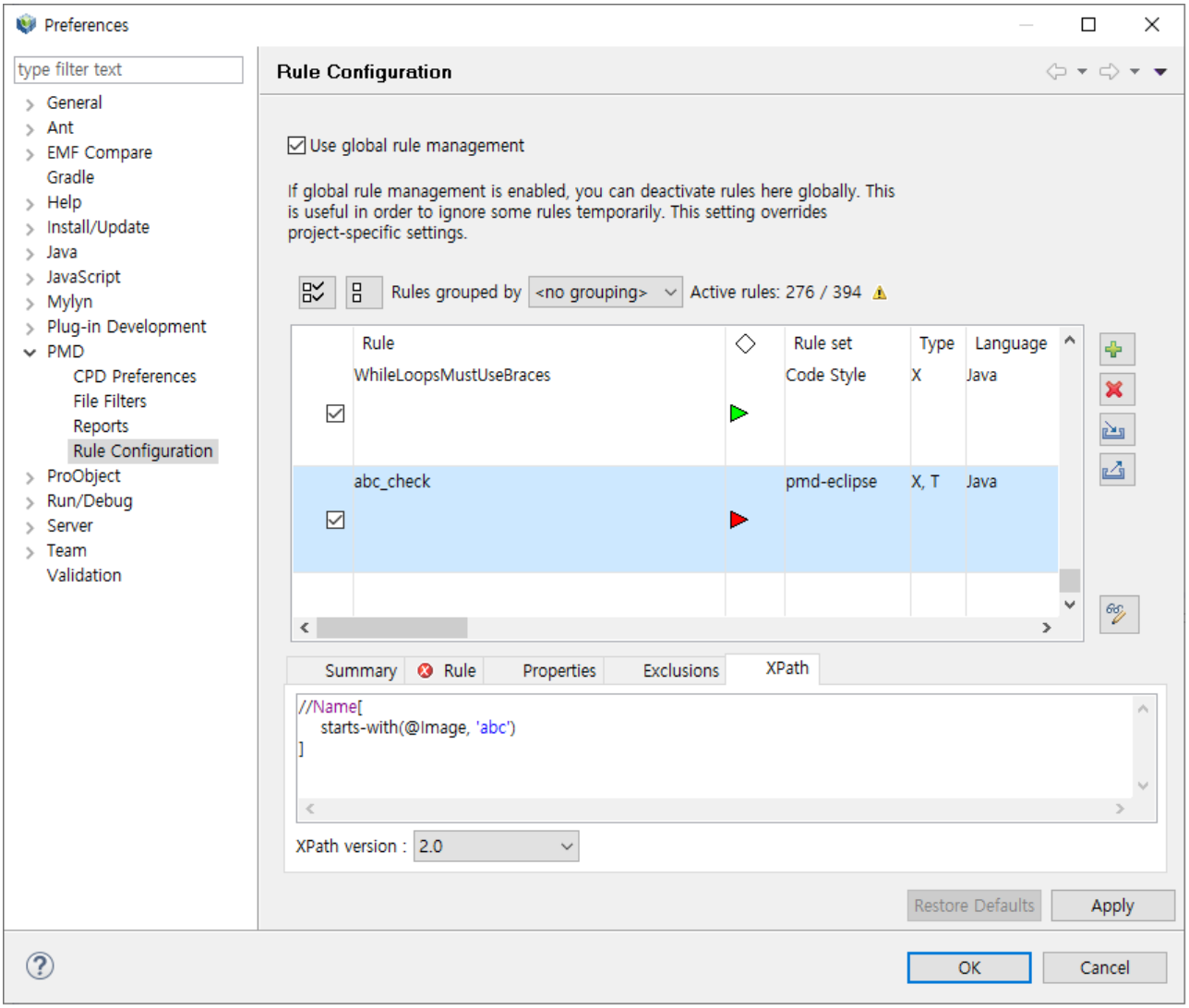

Custom Rule 생성(3)

4. 다음은 실제로 체크되는 Rule을 적용하는 화면으로 시작문자가 'abc'로 시작했을 때 Rule에 match를 체크한 예이다. Xpath Script 문법을 사용하여 스크립트를 작성한 후 **[Finish]** 버튼을 클릭한다.




Custom Rule 생성(4)

5. 다음은 **'abc_check Rule'**이 생성된 화면이다.

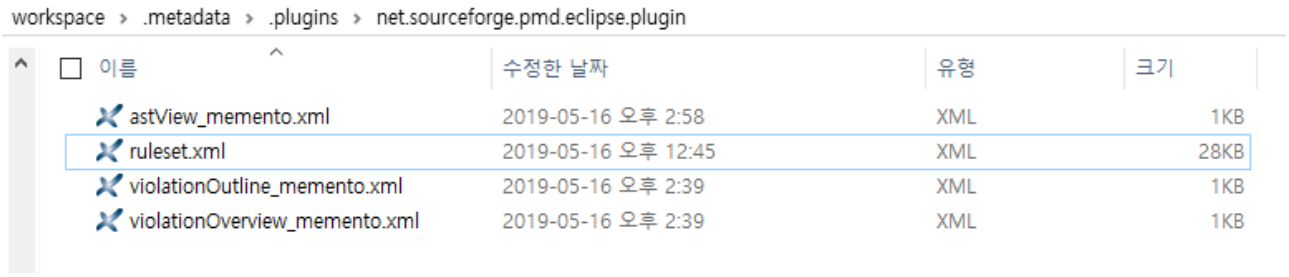



Custom Rule 생성(5)

### 5.2.6.2. Rule 배포

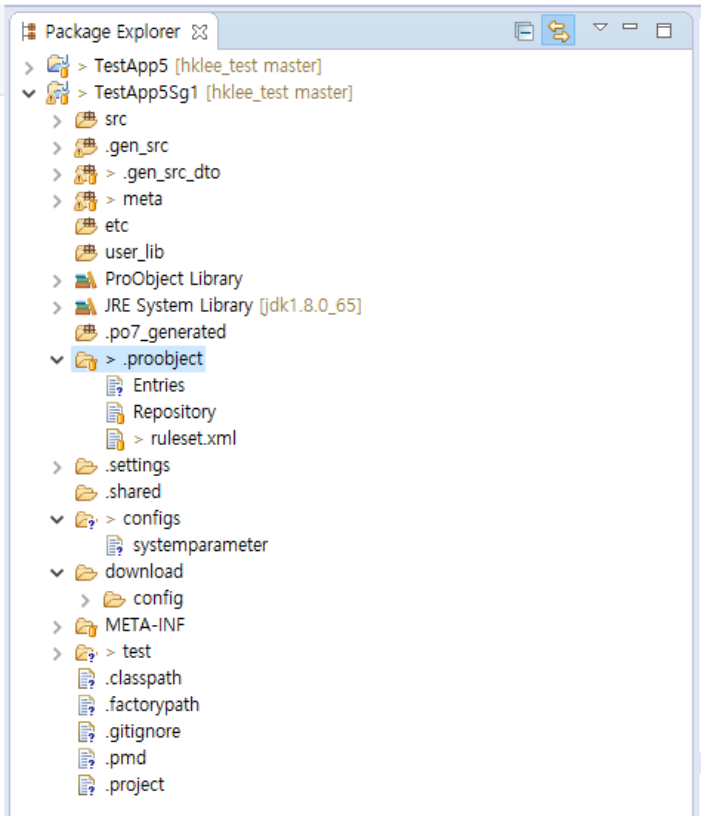
다음은 생성한 PMD Rule을 배호하는 과정에 대한 설명이다. PMD Rule 생성, 배포는 OwnerShip을 가지는 담당자가 대표로 진행한다.

1. PMD Rule File은 ProStudio workspace 하위에 생성된다.




Rule 배포 (1)

2. **Package Explorer**에서 프로젝트를 선택한 후 컨텍스트 메뉴에서 **[프로오브젝트] > [PMD Rule Download]**를 선택하면 workspace의 ruleset.xml 파일이 ".project" 하위 폴더에 생성된다. 해당 파일은 Git Grpository에 Push하면, 모든 개발자와 해당 파일을 공유할 수 있다.



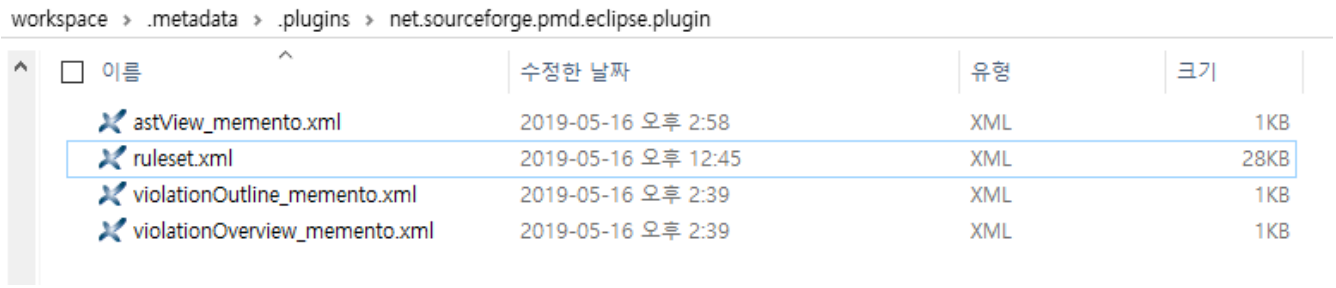

Rule 배포 (2)

### 5.2.6.3. 공통 Rule 파일 적용

Git Repository에 Push된 ruleset.xml 파일은 Pull할 때 개발자의 ProStudio에 생성(변경)된다.

Local에 생성(변경)된 파일의 적용은 각각의 개발자의 추가 작업이 필요한다. Check Rule을 위한 공통 Rule 파일(ruleset.xml) 적용은 각각의 개발자가 수행하도록 한다.

**Package Explorer**에서 프로젝트를 선택한 후 컨텍스트 메뉴에서 **[프로오브젝트] > [PMD Rule Upload]**를 선택하면 workspace의 ruleset.xml 파일이 적용된다.




공통 Rule파일(ruleset.xml) 적용

Rule을 적용하기 위해서는 ProStudio를 재부팅해야 한다.

## 5.2.7. 산출물 내보내기

Project를 진행하는 경우 단계별로 리소스에 대한 산출물을 생성이 필요하다. 이에 대한 지원을 ProStudio에서는 산출물 내보내기 기능으로 제공한다. 산출물 내보내기 가능한 리소스는 SO, BO, JO, DO, QO, JO이며 Default 또는 설정에 따라 리소스 단일 또는 일괄 산출물 생성이 가능하다.

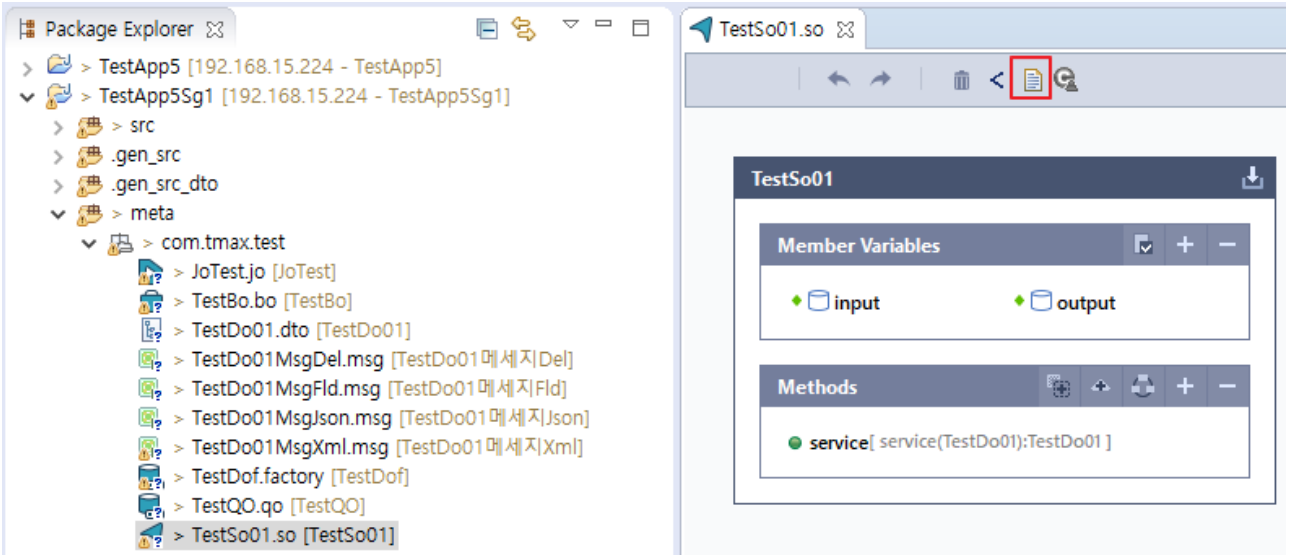
산출물을 생성하는 경우 Project별로 템플릿의 수정할 필요가 있을 경우에는 확정점 플러그인 설치 및 구현을 참고한다.

### 5.2.7.1. 단일 산출물 내보내기

산출물 내보내기는 각각을 리소스를 Open할 수 수행한다. Open된 리소스에서는 각 리소스별로 해당되는 **[내보내기]** 버튼을 클릭해서 산출물을 내보낼 수 있다.

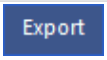
다음은 ServiceObject(SO)를 산출물로 내보내는 과정에 대한 설명이다(Export 방식 및 생성 위치는 패키지별로 동일하다).

1. 산출물 내보내기할 리소스를 더블클릭하여 Open 및  (**[산출물 내보내기]** 버튼)을 클릭한다.



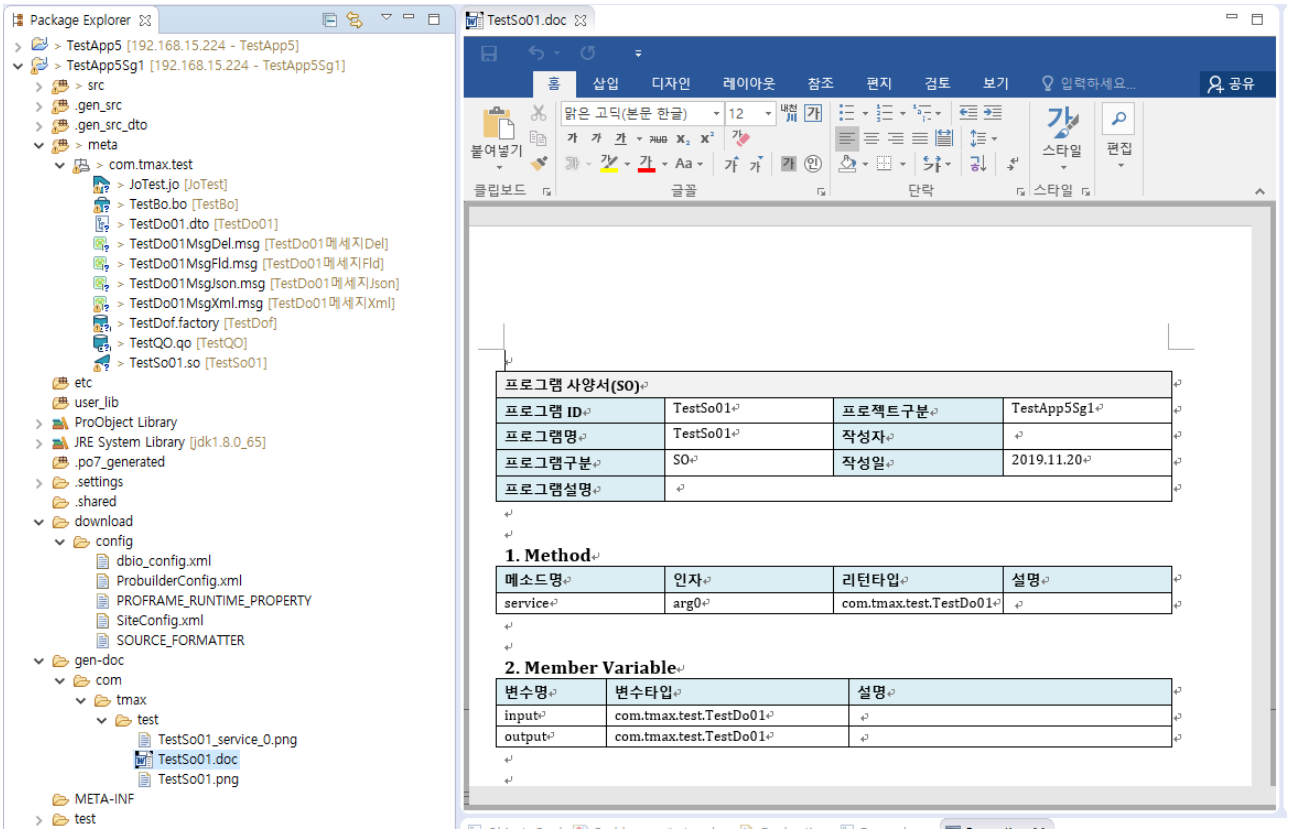
Export Document

다음은 각 리소스별 내보내기한 기본 산출물의 구성에 대한 설명이다.

| 리소스 유형별 버튼 | 설명 |
|---|---|
|  | ◦ Service Object : 기본정보, Method, Member Variable, Input Output Info, Input Data, Output Data, Flow, 처리 로직<br>◦ Business Object : 기본정보, Method, Member Variable<br>◦ Job Object : 기본정보, Flow, Job Parameter Info, Job Parameter Field, Condition Info |
|  | ◦ Data Object : 기본정보 (Default 환경에서는 해당 버튼은 보이지 않으며, 버튼 활성화를 위해서는 관련 설정을 참고한다.)<br>◦ Data Object Factory : 기본정보, Data Object Info, Data Object Field, Full Statement<br>◦ Query Object : 기본정보, Full Statement |

2. 생성된 산출물은 **Project Exploer**의 gen-doc 아래에 내보내기한 SO의 Package에 .doc으로 생성된다.

   다음은 SO(Service Object)의 예제이며, 각각의 리소스 유형별 산출물 Layout은 다르다.

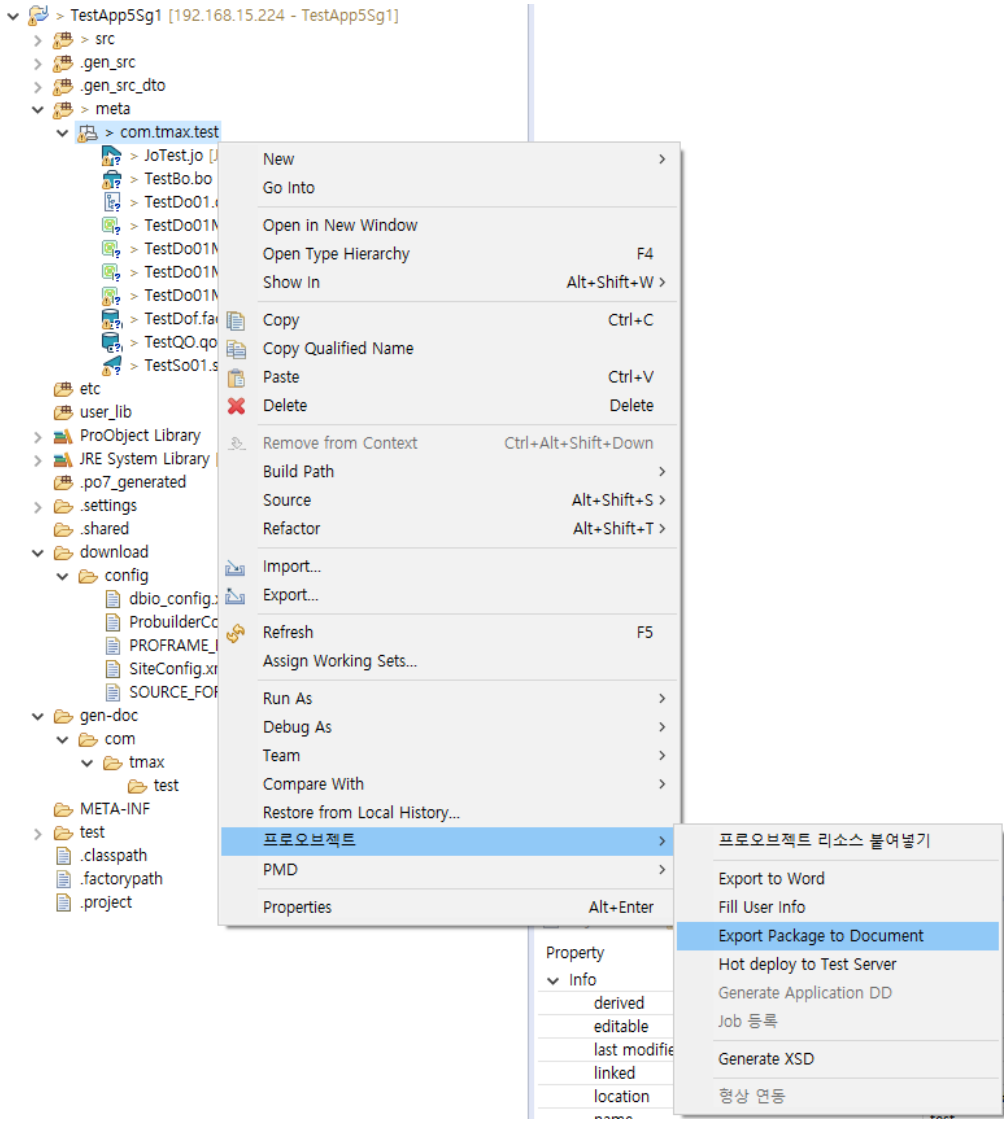



Export Document - 산출물

### 5.2.7.2. 일괄 산출물 내보내기

다음의 과정으로 패키지별로 산출물을 일괄로 내보낼 수 있다.

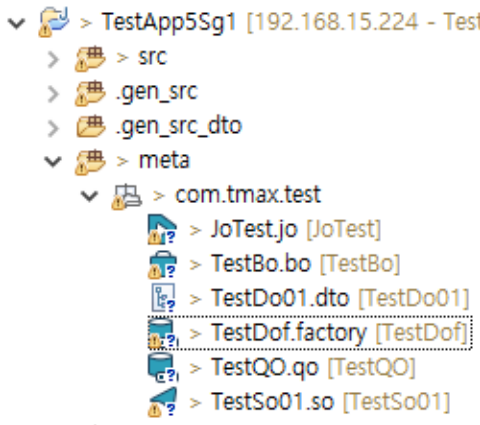
1. **Project Exploer**의 PO 리소스 메타의 컨텍스트 메뉴에서 **[프로오브젝트] > [Export Package to Document]**를 선택한다.




Export Document (1)

2. 생성된 산출물은 **Project Exploer**의 gen-doc 아래에 내보내기한 프로젝트 아래에 .doc으로 일괄 생성된다.

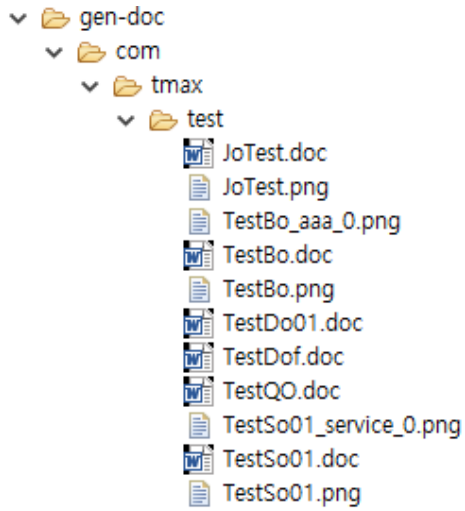
   산출물 내보내기할 대상은 아래와 같이 6개의 리소스이다.




Export Document (2)

SO, BO, JO, DO, QO, JO 유형에 대해서 산출물 내보내기 가능하다.

3. 최종적으로 생성된 산출물 내역은 아래와 같이 각 리소스별로 1개씩 총 6개가 생성된다. 문서에 추가하기 위해 생성한 중간 파일인 ".png"은 필요한 경우 다른 용도로 사용할 수 있다.




Export Document - 산출물 내역

#### 5.2.7.3. 관련 설정

DO에 산출물 버튼 활성화 및 메타(XML Meta)에 생성/변경 정보를 추가하기 위해서 다음과 같은 산출물 관련 설정을 해야 한다.

### DO 산출물 버튼 활성화

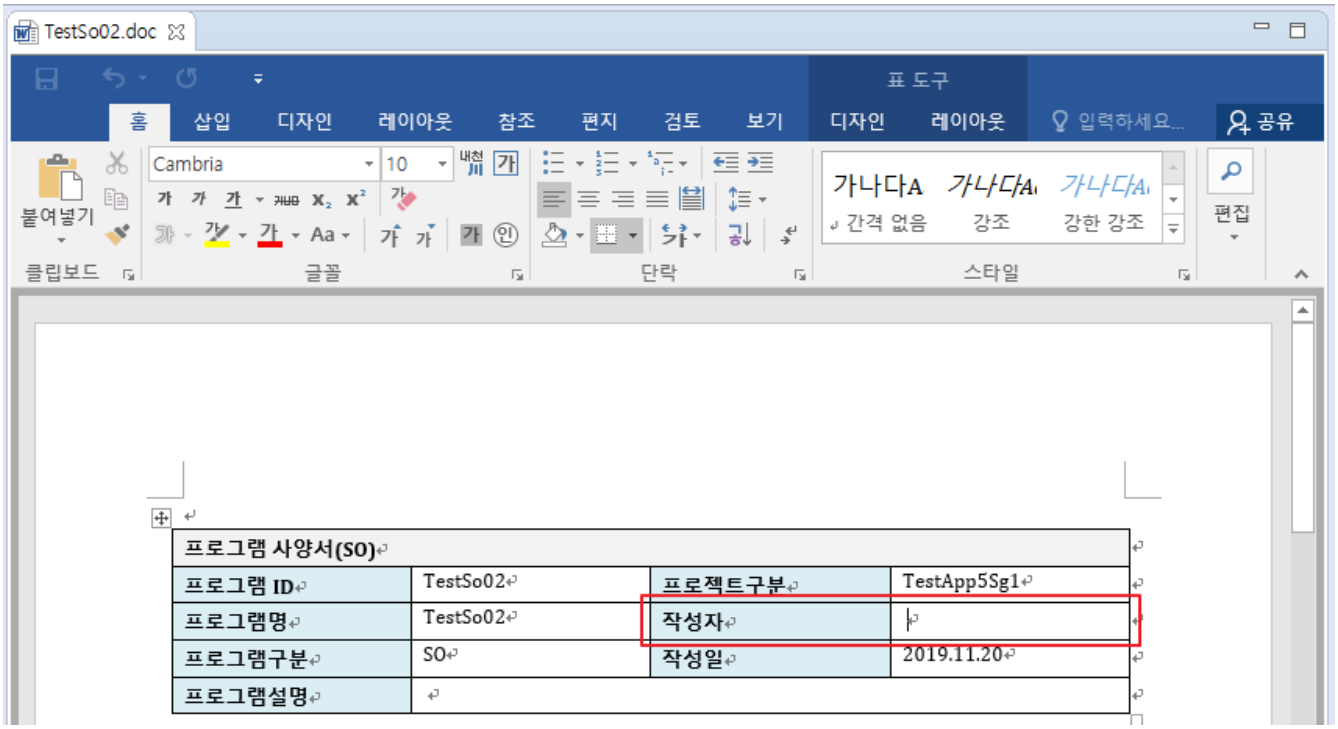
DO 작성 화면에서 **[산출물 내보내기]** 버튼 활성화를 위해서는 아래와 같이 SiteConfig.xml의 USE_STURUCTURE_DOC_EXPORT 설정이 true가 되어야 한다.

```
<?xml version="1.0" encoding="EUC-KR" standalone="yes"?>
<siteConfig xmlns="http://www.tmax.co.kr/proobject/siteConfig">
   :
  중략
   :
<siteElement id="USE_STURUCTURE_DOC_EXPORT" value="true" type="boolean" xmlns=""/>
</siteConfig>
```

### 리소스 메타(XML Node 정보)에 권한정보 추가

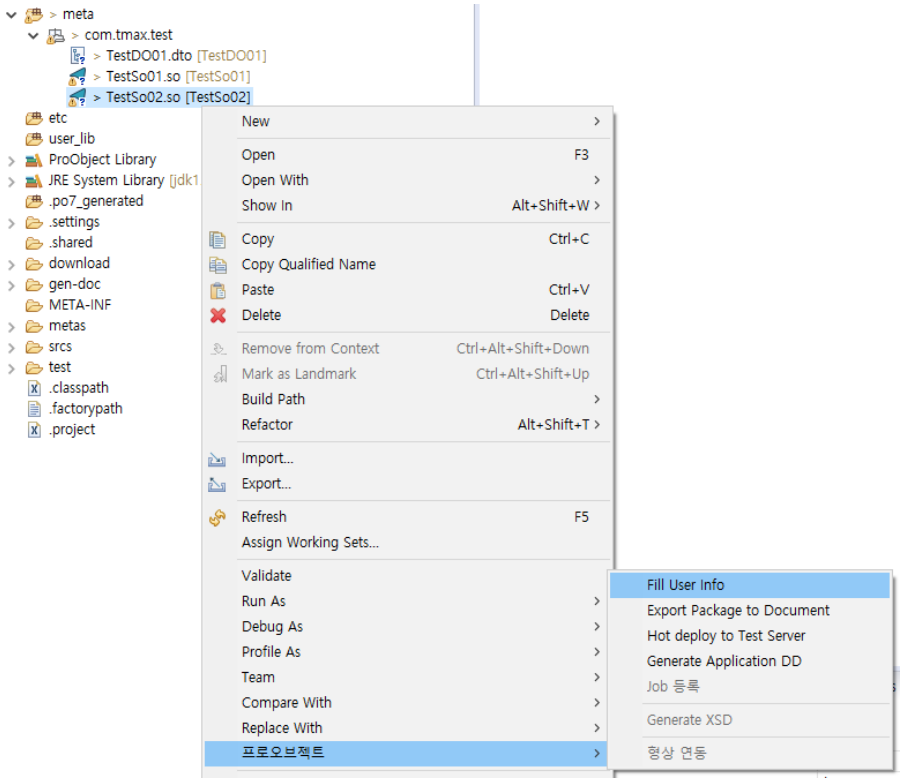
기본 메타에는 생성자(creator), 최종변경자(modifier), 권한사용자(owner) 정보가 없다. 해당 정보가 없을 때에는 산출물에도 표시 되지 않아, 아래 그림과 같이 공란으로 표시된다.

이미 만들어진 리소스에 대해 권한정보를 추가 하면 해당 정보가 산출물에 표시할 수 이다.



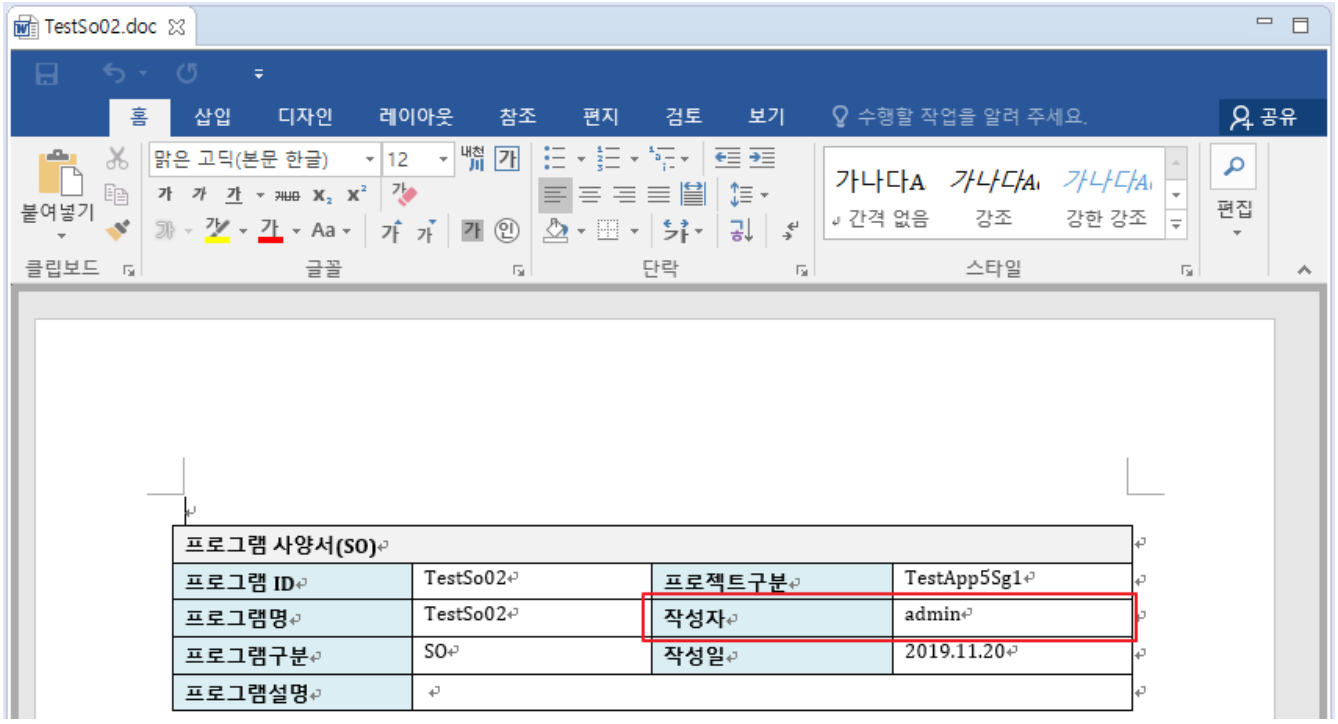

Export Document - 작성자 공란

다음과 같이 PO 리소스를 선택한 후 컨텍스트 메뉴에서 **[프로오브젝트] > [Fill User Info]**를 선택한다.




Export Document - 권한정보 추가

리소스 메타(XML Node정보)에 권한정보가 추가 되면, 해당 리소스의 산출문에도 아래와 같이 개발자 정보가 표시된다.



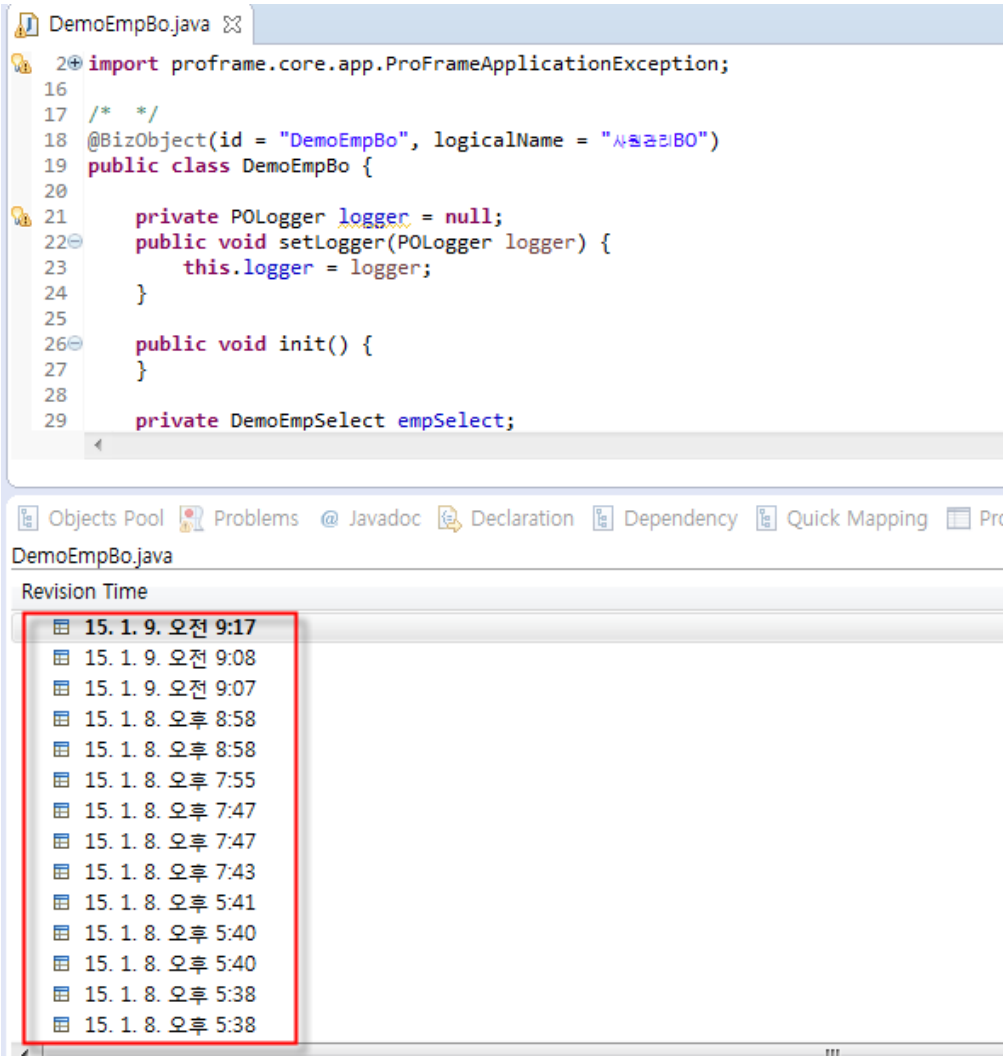

Export Document - 작성자 정상 표시

신규 리소스를 생성하는 경우에도 메타(XML Node정보)에 권한정보가 추가 하기 위해서는 $PROOBJECT_HOME/system/config/ProbuilderConfig.xml에 saveResourceUserInfo가 true로 되어 있어야 한다.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<probuilder_config xmlns="http://www.tmax.co.kr/proobject/probuilder_config">
    <default-dto-names dtoPrefix="Pre" dtoSuffix="Dto" inDtoPrefix="InPre" inDtoSuffix="ListDto"
xmlns=""/>
    <real-type xmlns="">true</real-type>
    <all-property-from-server xmlns="">false</all-property-from-server>
    <proMapperConfig xmlns="">
        :
      중략
        :
<saveResourceUserInfo>true</saveResourceUserInfo>
</probuilder_config>
```

## 5.2.8. Local History

날짜별로 ProObject 리소스의 변경된 사항을 확인할 수 있다. 리소스를 선택한 후 **Package Explorer**의 컨텍스트 메뉴에서 **[Team] > [Show Local History]**를 선택한다.



Team action - Local History

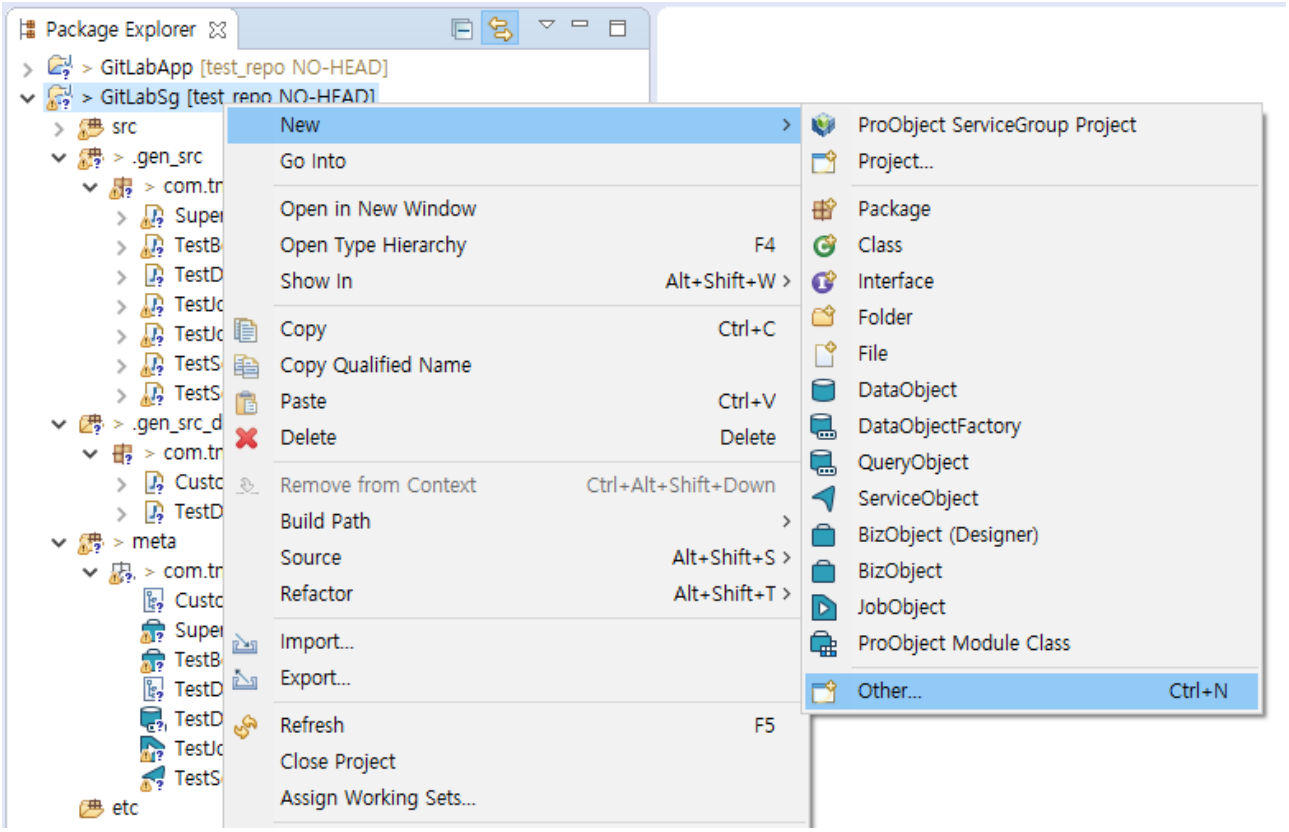
# 5.3. Test Case

본 절에서는 테스트 케이스를 생성하고 테스트를 수행하는 기능에 대해서 설명한다.

## 5.3.1. BO Unit Test

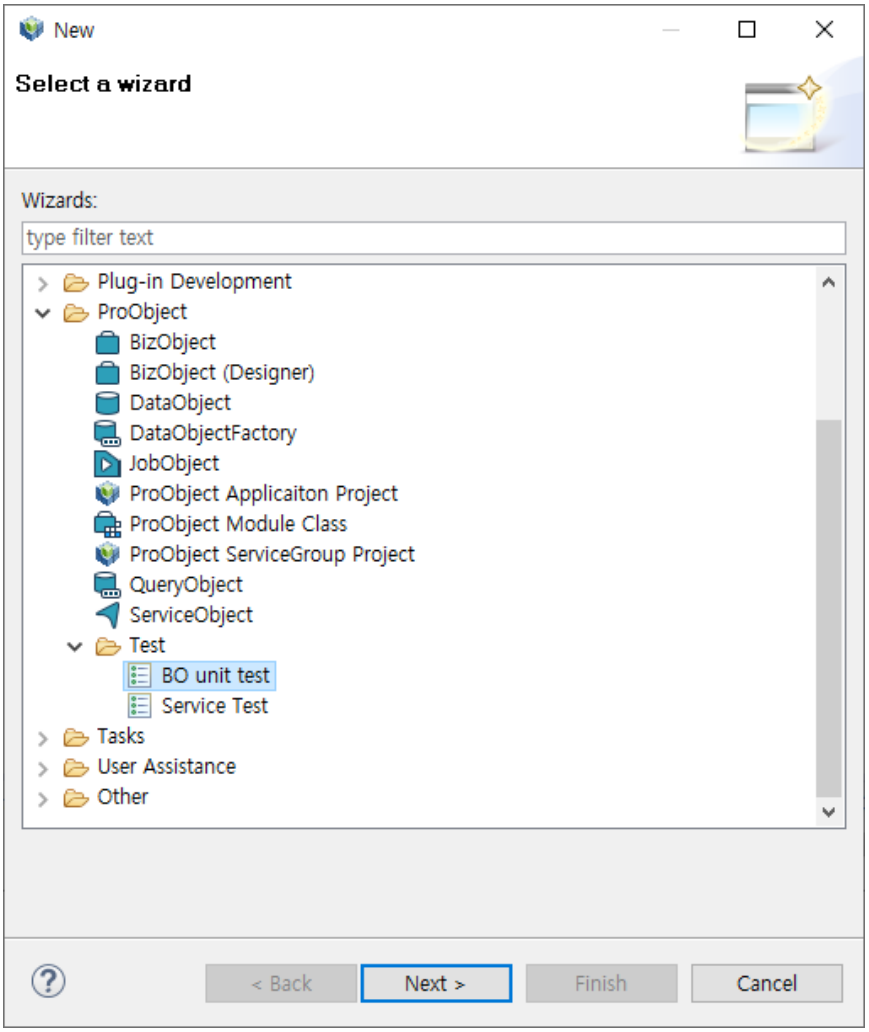
완료된 Biz Object는 단위 테스트를 수행할 수 있다.

1. **Package Explorer**의 컨텍스트 메뉴에서 **[New] > [Other...]**를 선택한다.



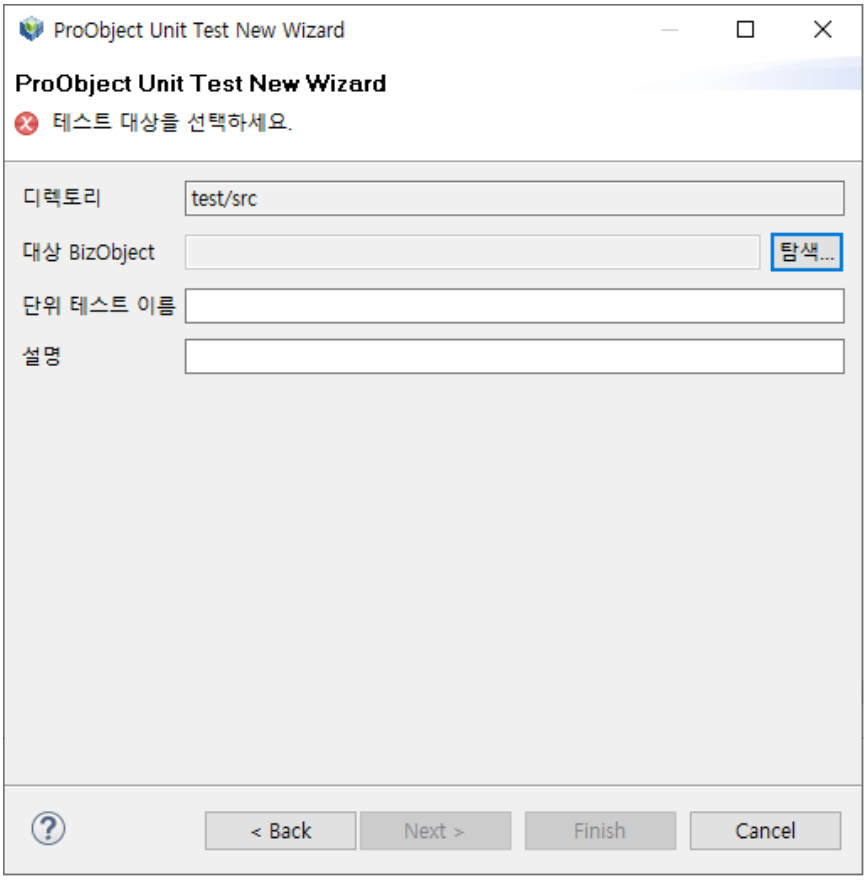
BO Unit Test 선택(1)

2. 트리 메뉴에서 **[Proobject] > [Test] > [BO Unit test]**를 선택한 후 **[Next]** 버튼을 클릭한다.



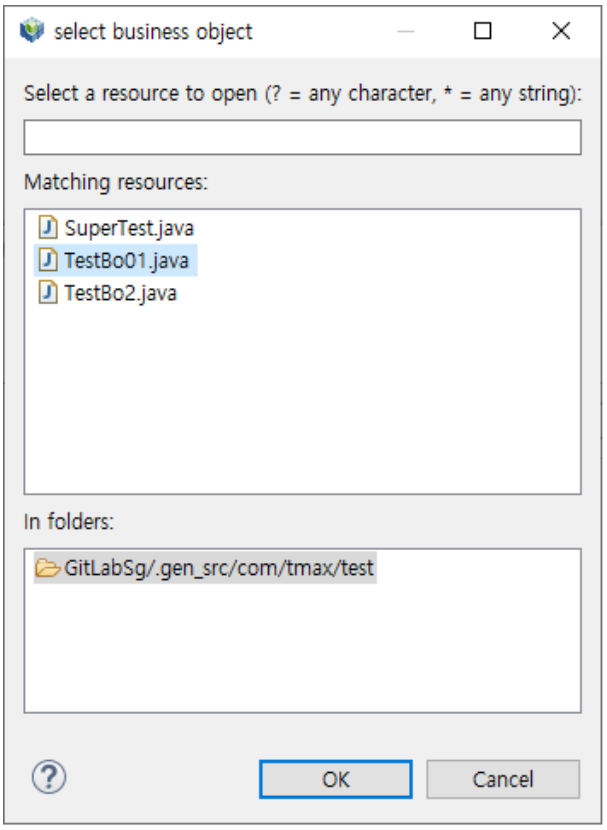
BO Unit test 선택(2)

3. 테스트 대상이 되는 BO를 설정할 수 있는 화면이 열리면 **[탐색]** 버튼을 클릭한다.



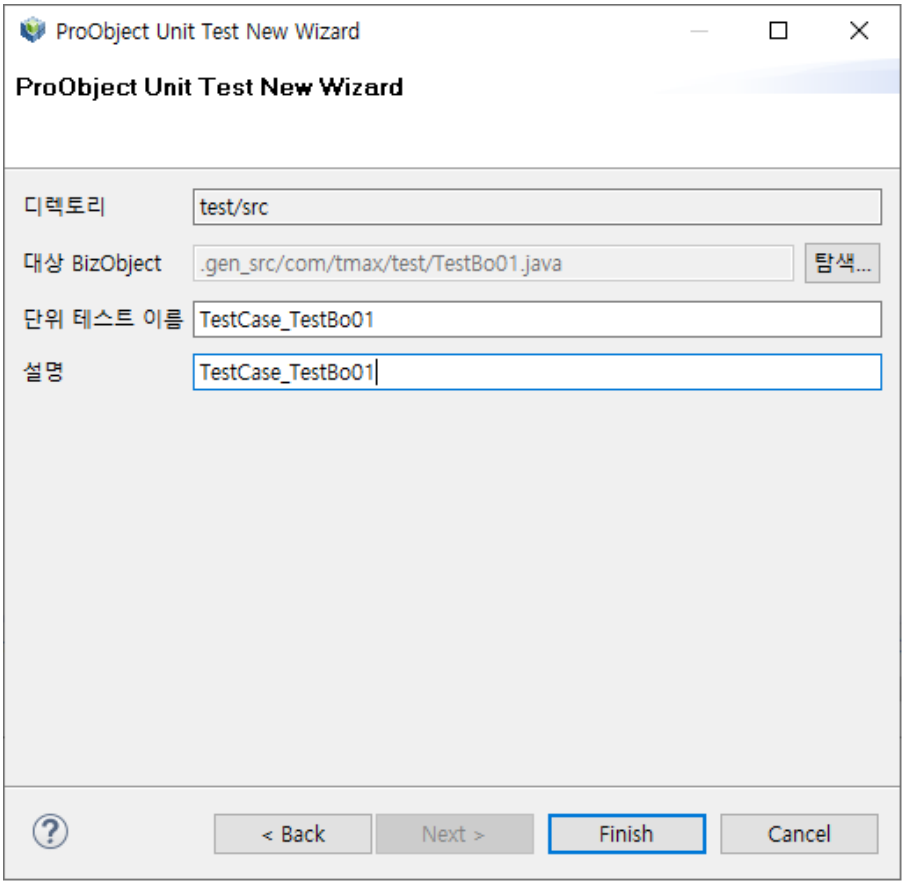

BO Unit Test 설정 화면

4. 조회된 BO 내역 중 Unit Test할 BO(java)를 선택한 후 **[OK]** 버튼을 클릭한다.



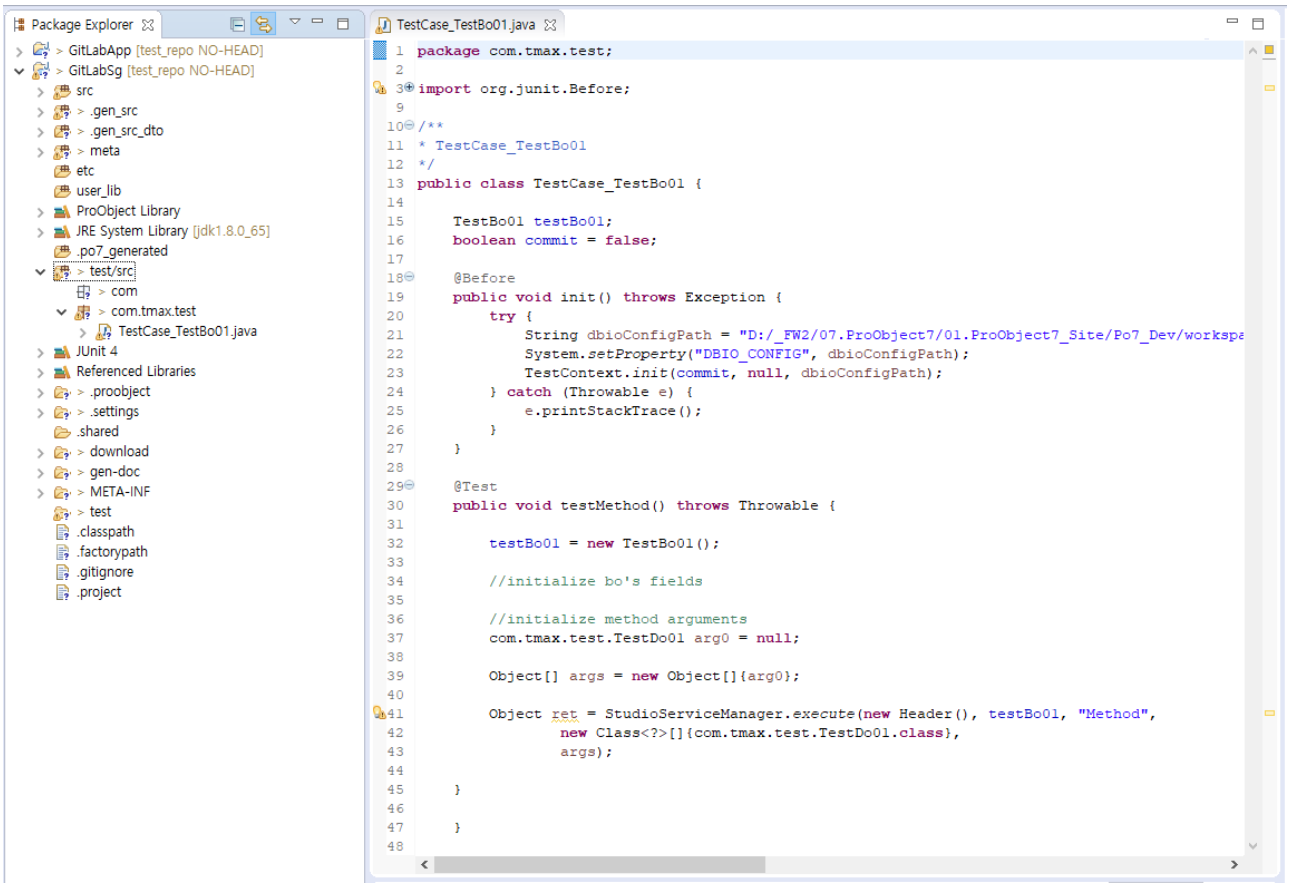

Test 대상 Bo 선택

5. **'단위 테스트 이름'**, **'설명'**을 입력한 후 **[Finish]** 버튼을 클릭한다.



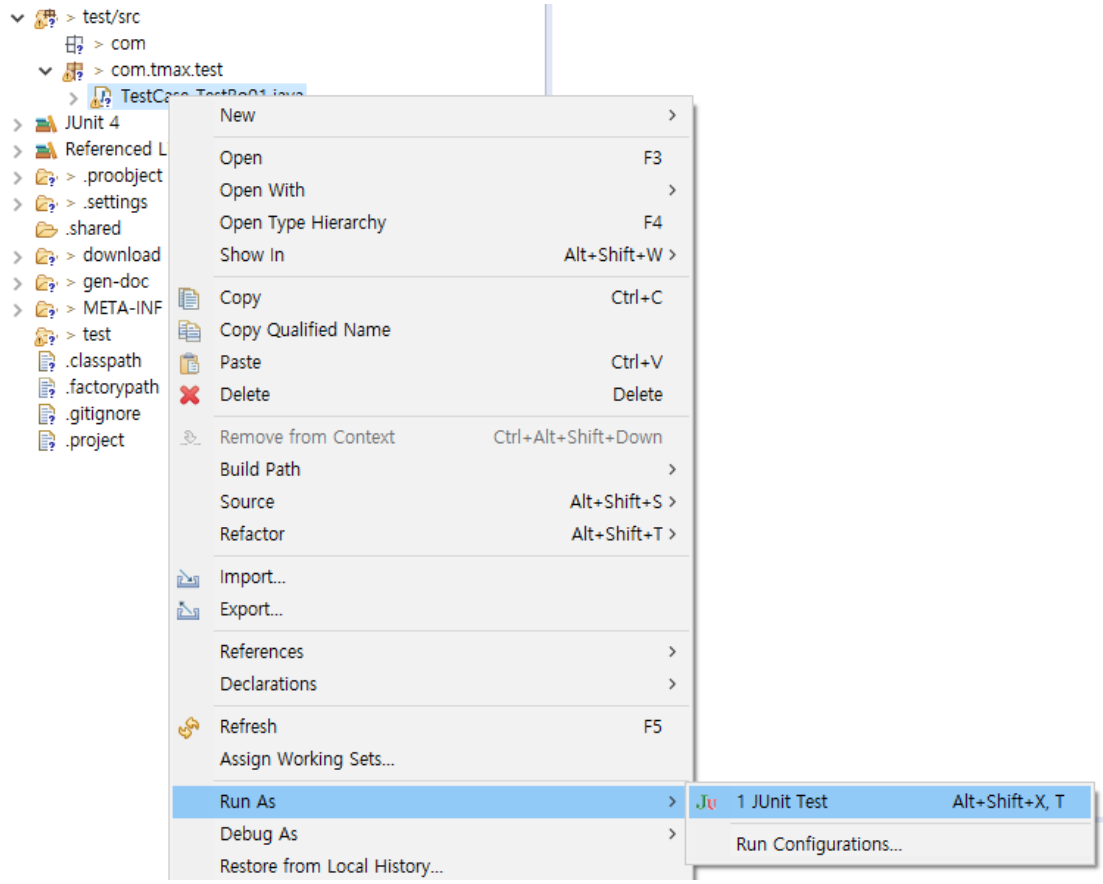

BO Unit Test Case 생성

6. 아래 그림과 같이 Unit Test Case Source가 생성된다. 해당 테스트 케이스는 [test/src]/[class path] 이하에 생성된다.



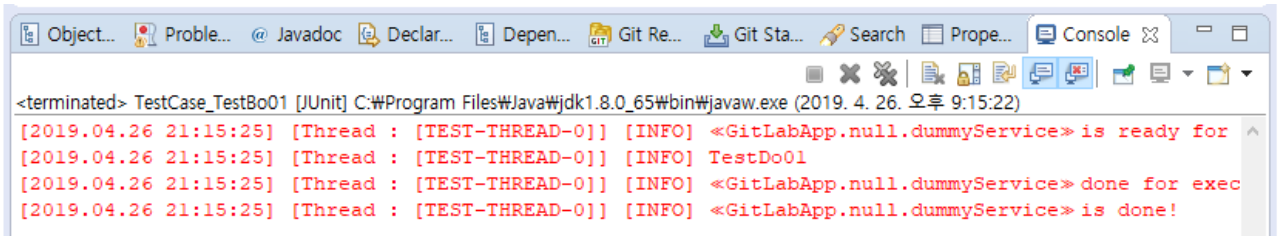

BO Unit Test Case Source

7. 생성된 Unit Test Case File을 선택한 후 컨텍스트 메뉴에서 **[Run As] > [JUnit Test]**를 선택한다.

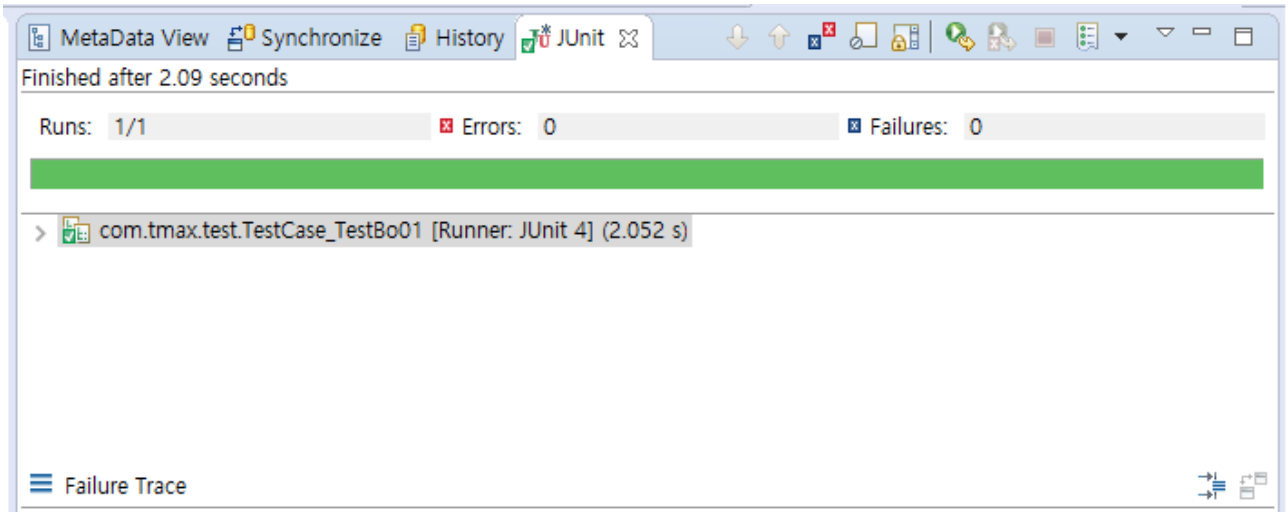



BO Unit Test 실행

8. BO Unit Test 결과 및 집계를 확인할 수 있다.

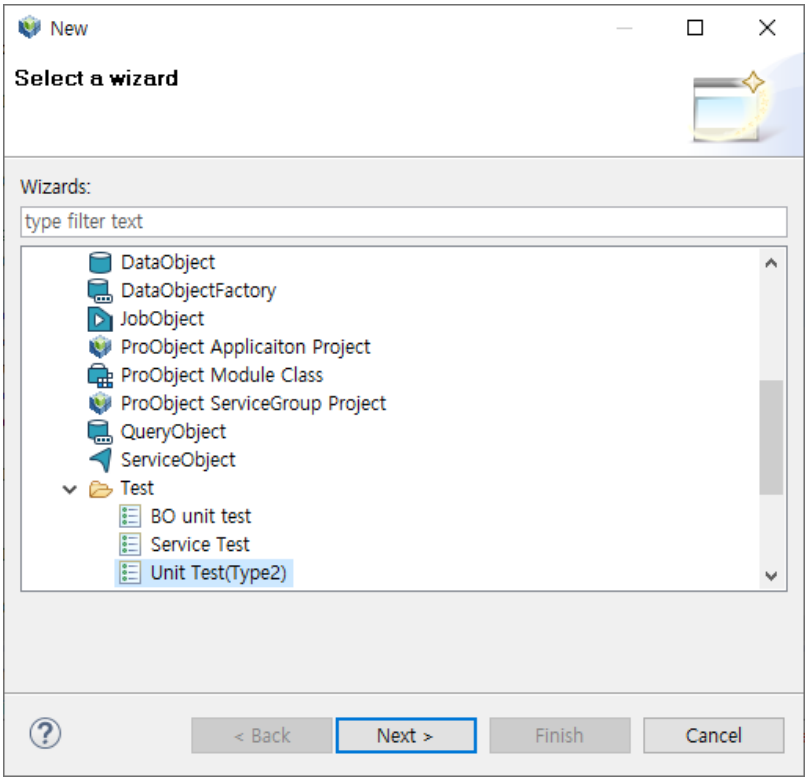



BO Unit Test 실행 결과 (1)



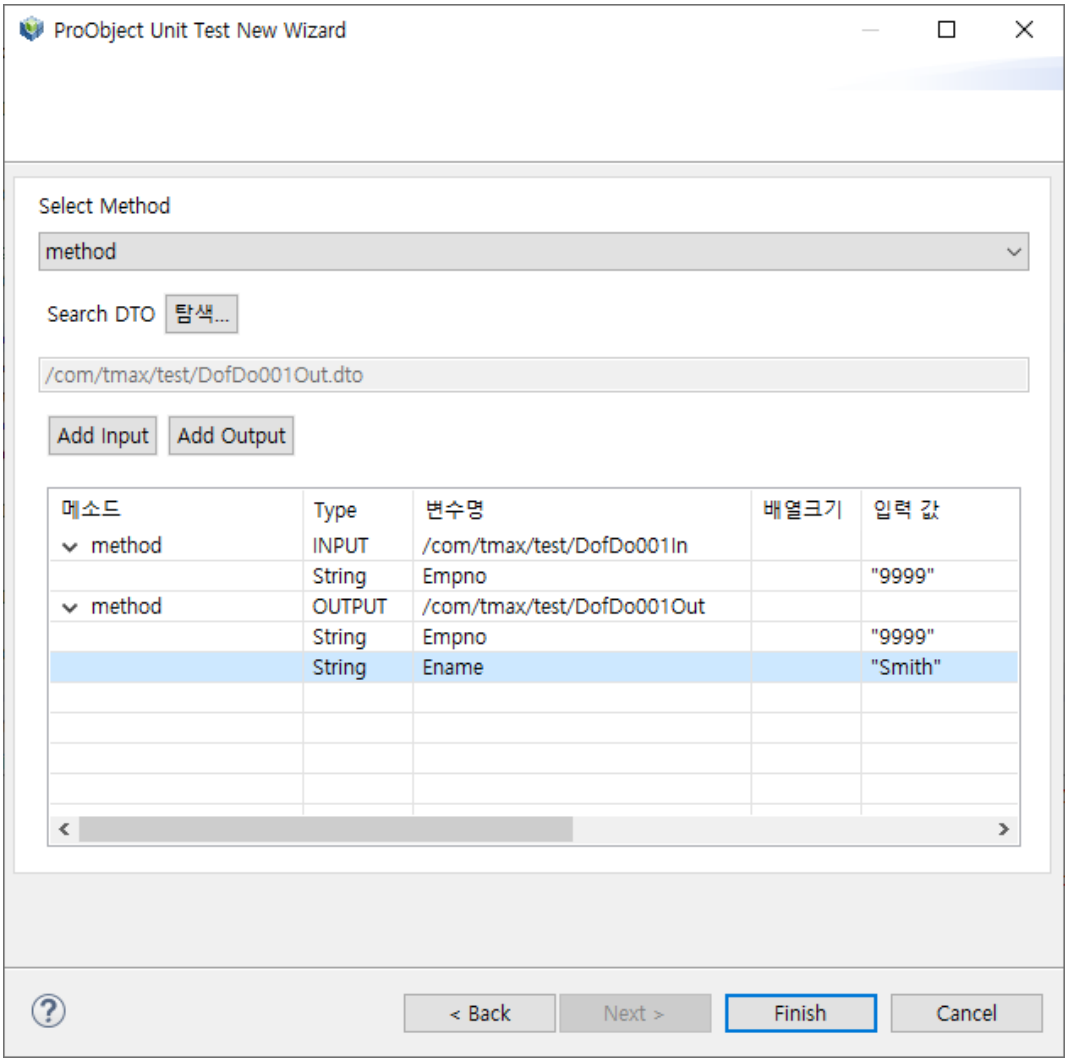

BO Unit Test 실행 집계 (2)

9. Unit Test 편의기능은 기존 기능과 동일 하지만 In/Out 출력 값을 UI로 설정한다.




Unit Test(Type2)

입력 출력 구조체 설정 및 Input/Output 값으로 설정한다.




Input/Output값 세팅

아래와 같이 세팅한 입력값이 포함된 Unit Test Source가 생성된다.

```
@Test
public void testMethod() throws Throwable {

    testBo01 = new TestBo01();

    //initialize bo's fields

    //initialize method arguments


        com.tmax.test.DofDo001In arg0 = new DofDo001In();

        arg0.setEmpno("9999");



        Object[] args = new Object[]{arg0};



       Object ret = StudioServiceManager.execute(new Header(), testBo01, "method",

        new Class<?>[]{com.tmax.test.DofDo001In.class}, args);


        com.tmax.test.DofDo001Out retObj = (DofDo001Out)ret;

        retObj.setEmpno("9999");

        retObj.setEname("Smith");
}
```

Input/Output값 세팅

## 5.3.2. Service Test 설정

본 절에서는 Local Service Test할 때에도 공통 헤더 및 시스템 컨텍스트를 사용하기 위한 환경설정을 설명한다.

### 5.3.2.1. 공통 헤더 설정

Service Test하는 경우 공통 헤더를 입력하려면 공통 헤더 배포 및 PoDevSvr.xml가 설정되어 있어야 한다.

다음은 공통 헤더 생성 및 PoDevSvr.xml을 설정하는 과정에 대한 설명이다.

1. com.tmax.proobject.network.protocol.proobject.request 패키지를 만들고 Header.dto를 복사한다.
2. Header.dto 상속한 Custom DO를 만든다.
3. Header.dto를 상속한 $PROOBJECT_HOME/system/config/PoDevSvr.xml에 아래 내용을 넣는다.

   이때 custom header 정보를 "Package + Class명"으로 입력한다.

```
<?xml version="1.0" encoding="EUC-KR" standalone="yes"?>
<serverConfig xmlns="http://www.tmax.co.kr/proobject/serverConfig">
     :
    중략
```

```
     :
    <configField id="custom-header" value="com.tmax.test.CustDto" type="String" xmlns=""/>
</serverConfig>
```

#### 5.3.2.2. 시스템 컨텍스트 정보 사용

Service Test 시스템 컨텍스트 사용을 위해서는ProStudio 의 application.xml에 <system-context> 태그에 정보가 설정되어 있어야 한다.

다음은 Service Test를 위한 <system-context>를 설정하는 과정에 대한 설명이다. ProStudio 애플리케이션 프로젝트에 application.xml 설정에 <system-context>를 추가한다.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    :
   중략
    :
<ns16:system-context>
    <ns16:name>LAB_CONTEXT</ns16:name>
    <ns16:context-table>
        <ns16:schema>tibero</ns16:schema>
        <ns16:table>LAB_CONTEXT</ns16:table>
    </ns16:context-table>
</ns16:system-context>
</ns16:application>
```

예시의 DB Context 정보는 다음과 같다.

| 항목 | 설명 |
|---|---|
| Schema명 | DB의 스키마 명이다. (예: : tibero) |
| Table명 | Key, Value를 가지는 컨텍스트 정보를 가지는 DB Table명이다. (예: LAB_CONTEXT) |

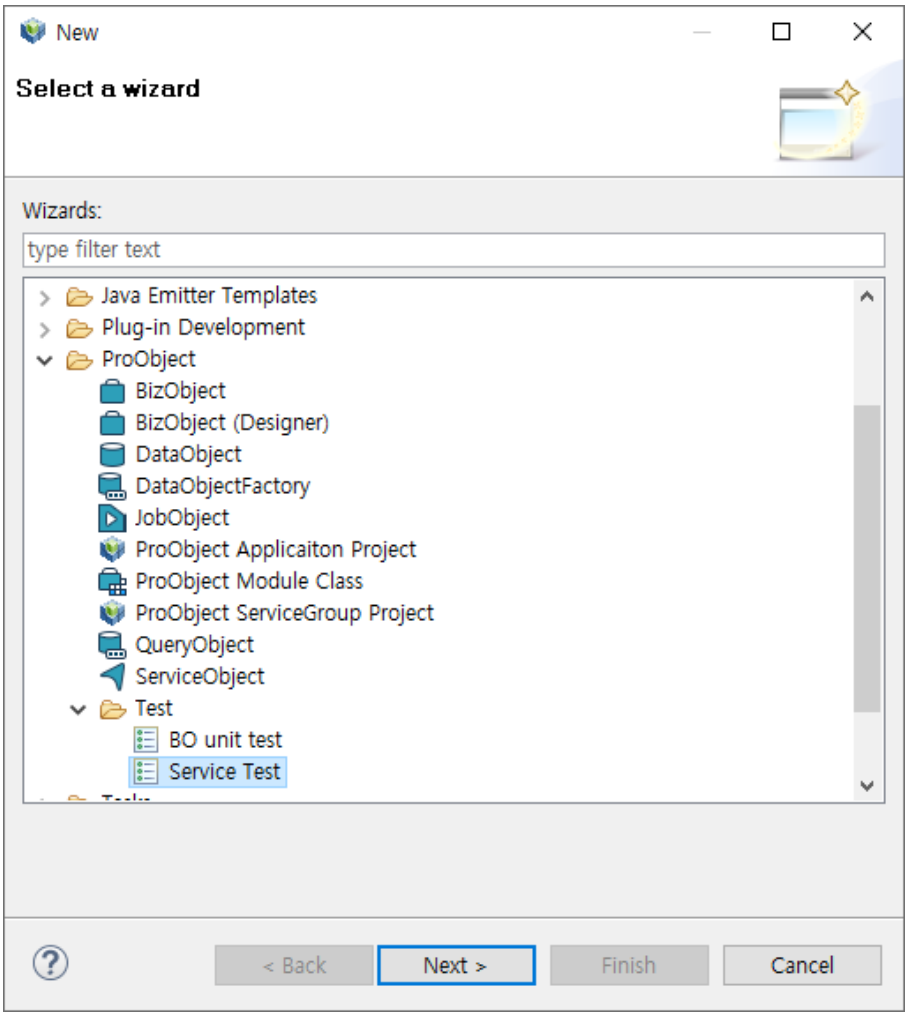
다음은 SO, BO에서 LAB_CONTEXT인 <system-context>에서 Key가 EMS_MSG_LEN인 Value를 가져오는 예제이다.

```
com.tmax.proobject.model.network.RequestContext reqContext =
ServiceManager.getCurrentRequestContext();
com.tmax.proobject.model.context.SystemContext systemContext =  reqContext.getSystemContext();
com.tmax.proobject.dataobject.context.ContextDataObject context =
(com.tmax.proobject.dataobject.context.ContextDataObject)systemContext.getContext("LAB_CONTEXT");

if(context != null){
   logger.info("context EMS_IP : " +  context.get("EMS_MSG_LEN"));
}else {
   logger.info("context is null!");
}
```

## 5.3.3. Service Test

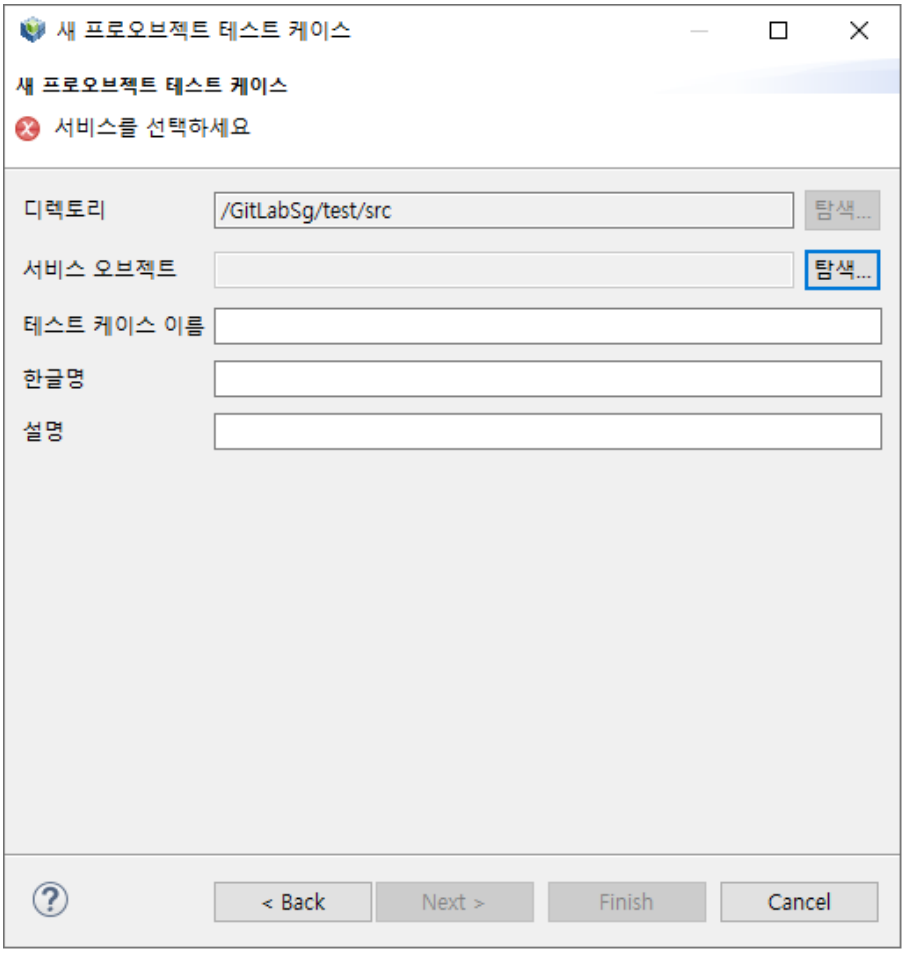
Local에서 서비스 단위 테스트를 수행할 수 있다.

1. **Package Explorer**의 컨텍스트 메뉴에서 **[New] > [Other...]**를 선택한다(BO Unit Test 선택(1) 참고).
2. 트리 메뉴에서 **[Proobject] > [Test] > [Service unit test]**를 선택한 후 **[Next]** 버튼을 클릭한다.



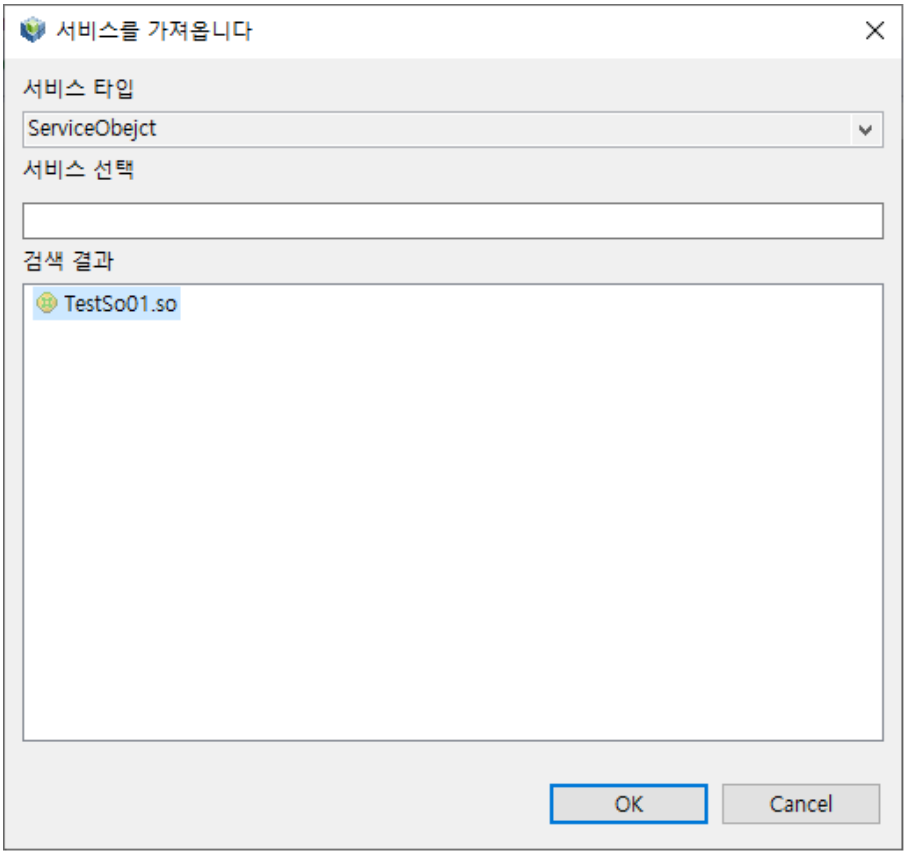

Service test 선택

3. 테스트 대상이 되는 SO를 설정할 수 있는 화면이 열리면 **[탐색]** 버튼을 클릭한다.



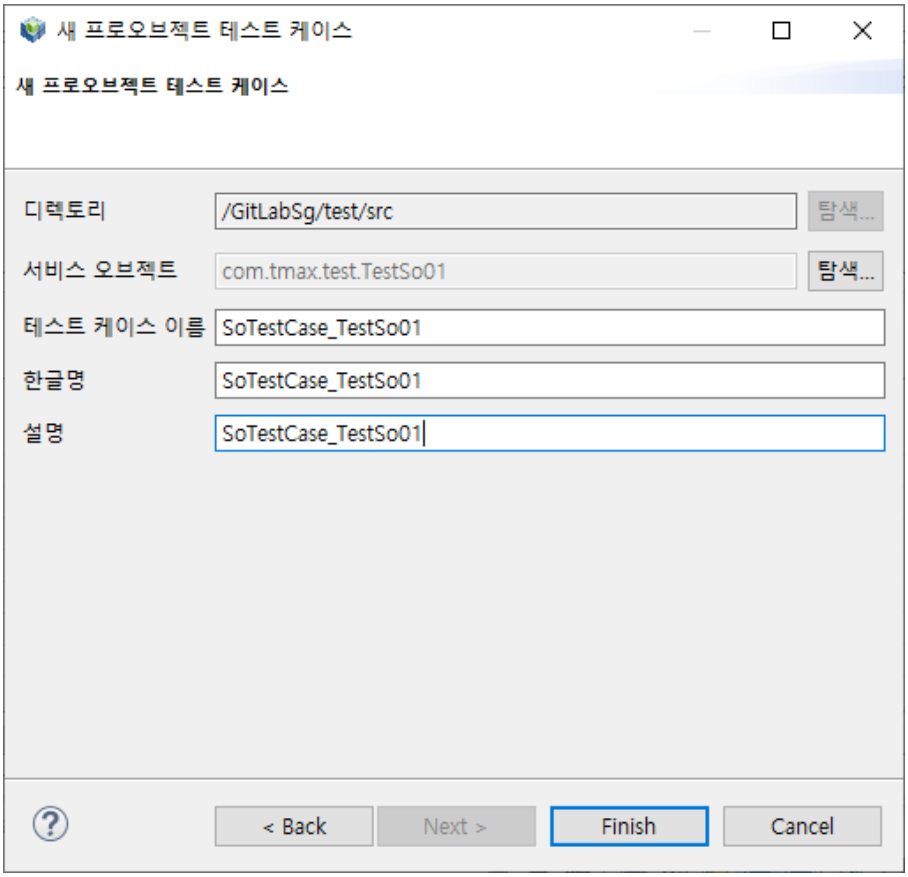

Service Test 설정 화면

4. 조회된 SO 내역 중 테스트할 SO를 선택한 후 **[OK]** 버튼을 클릭한다.



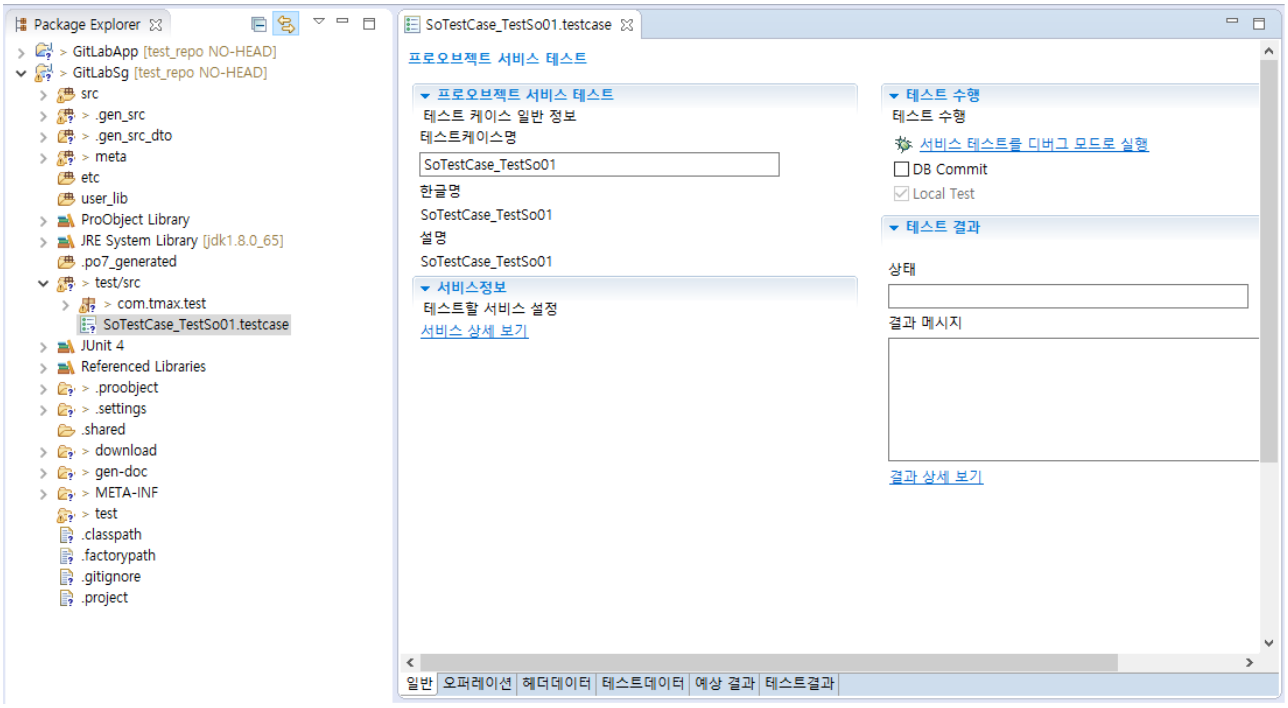

Test 대상 SO 선택

5. **'테스트 케이스 이름'**, **'한글명'**, **'설명'**을 입력하고 **[Finish]** 버튼을 클릭한다.



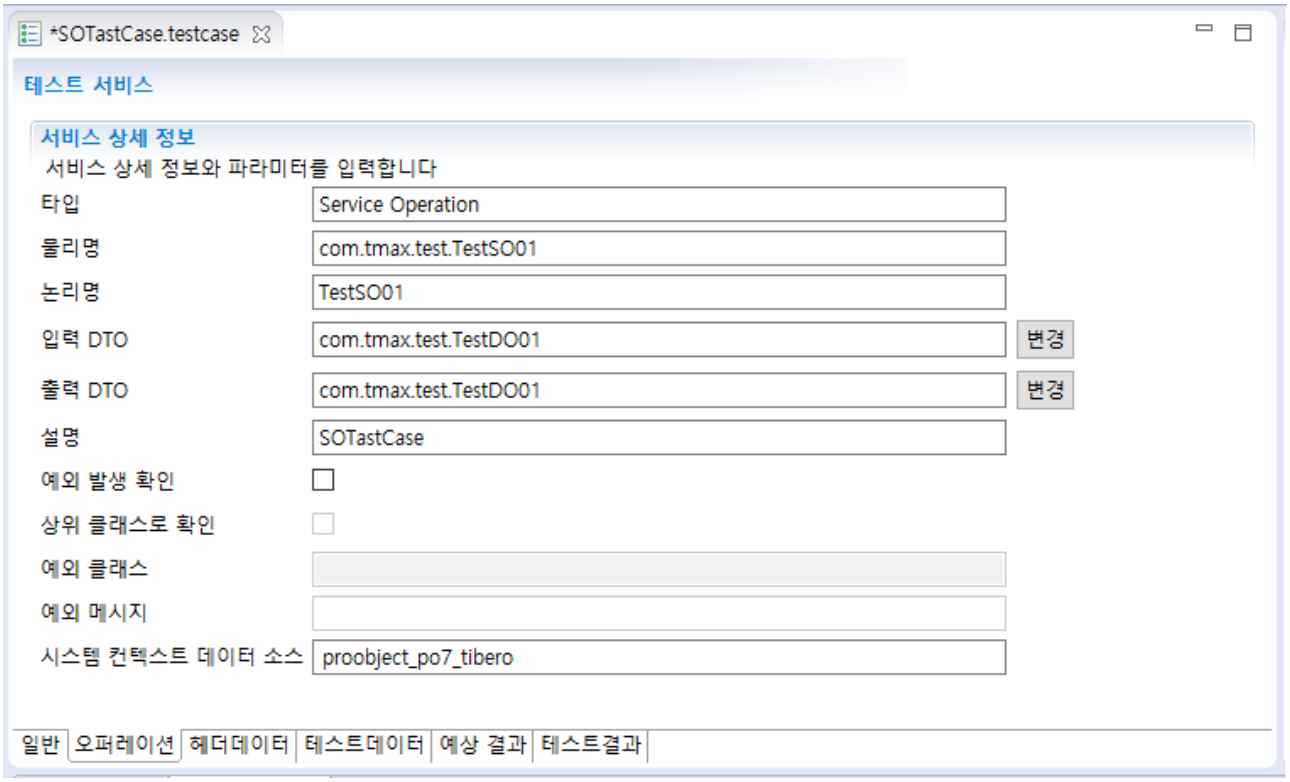

Service Test Case 생성

6. 아래 그림과 같이 Service Test Case가 생성되었다. 해당 테스트 케이스는 [test/src] 이하에 생성된다.




Service Test Case

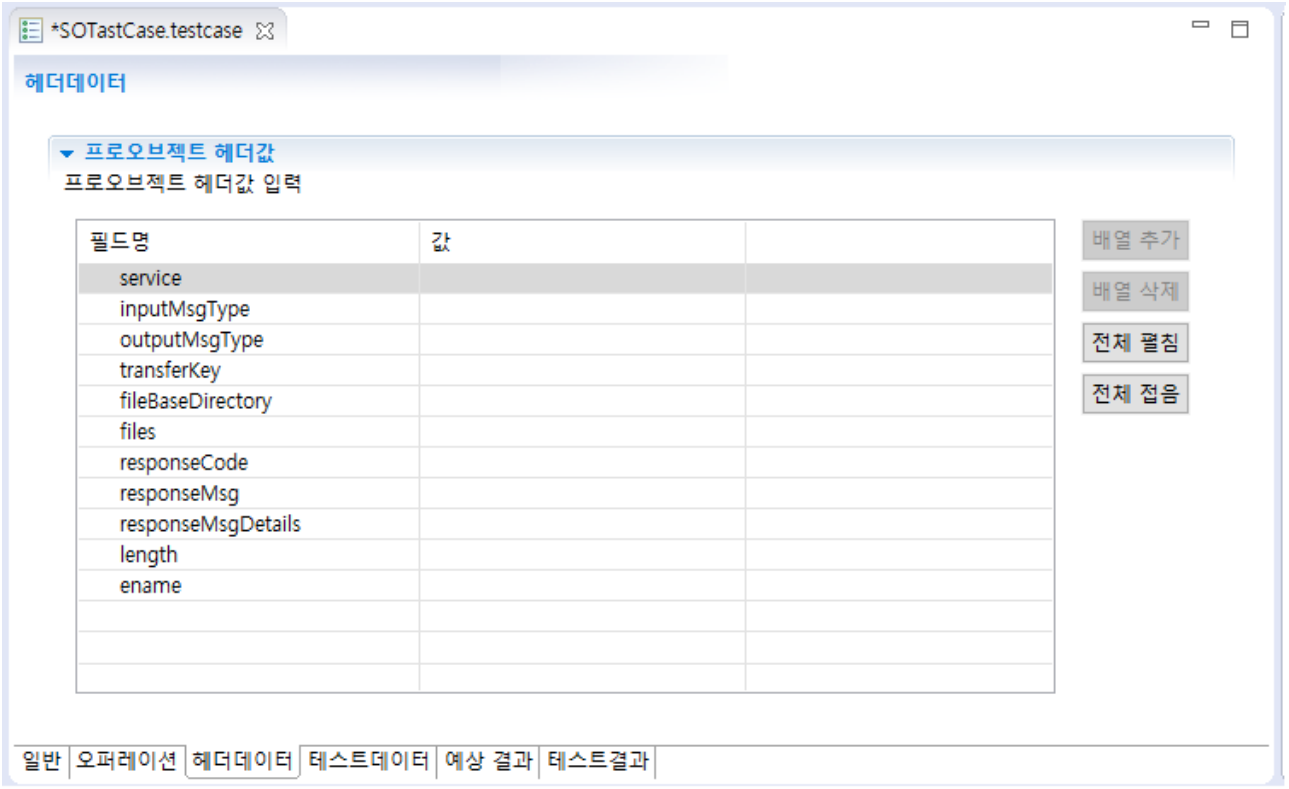
7. **[오퍼레이션]** 탭을 선택하면 서비스 테스트의 상세 정보를 확인할 수 있다.



Service Test Case 상세정보 - [오퍼레이션] 탭

| 항목 | 설명 |
|---|---|
| 타입 | Test Type이며, 서비스 Test이기 때문에 Service Operation이다. |
| 물리명 | 테스트할 SO의 물리명이다. |
| 논리명 | 테스트할 SO의 논리명이다. |
| 입력 DTO | 테스트할 SO의 입력 DTO이다. |
| 출력 DTO | 테스트할 SO의 출력 DTO이다. |
| 설명 | 테스트할 SO의 설명이다. |
| 예외 발생 확인 | 예외 발생 확인 여부를 체크한다. |
| 상위 클래스로 확인 | 예외 발생을 확인하는 경우 활성화되며, Exception을 상위 Class로 확인한다. |
| 예외 클래스 | 예외 클래스를 지정한다. |
| 예외 메시지 | 예외 메시지 내용을 입력한다. |
| 시스템 컨텍스트 데이터 소스 | 시스템 컨텍스트 데이터소스를 지정한다. dbio_config.xml 파일에 studio-jdb Node의 Key가 conn_name인 Value를 넣는다. |

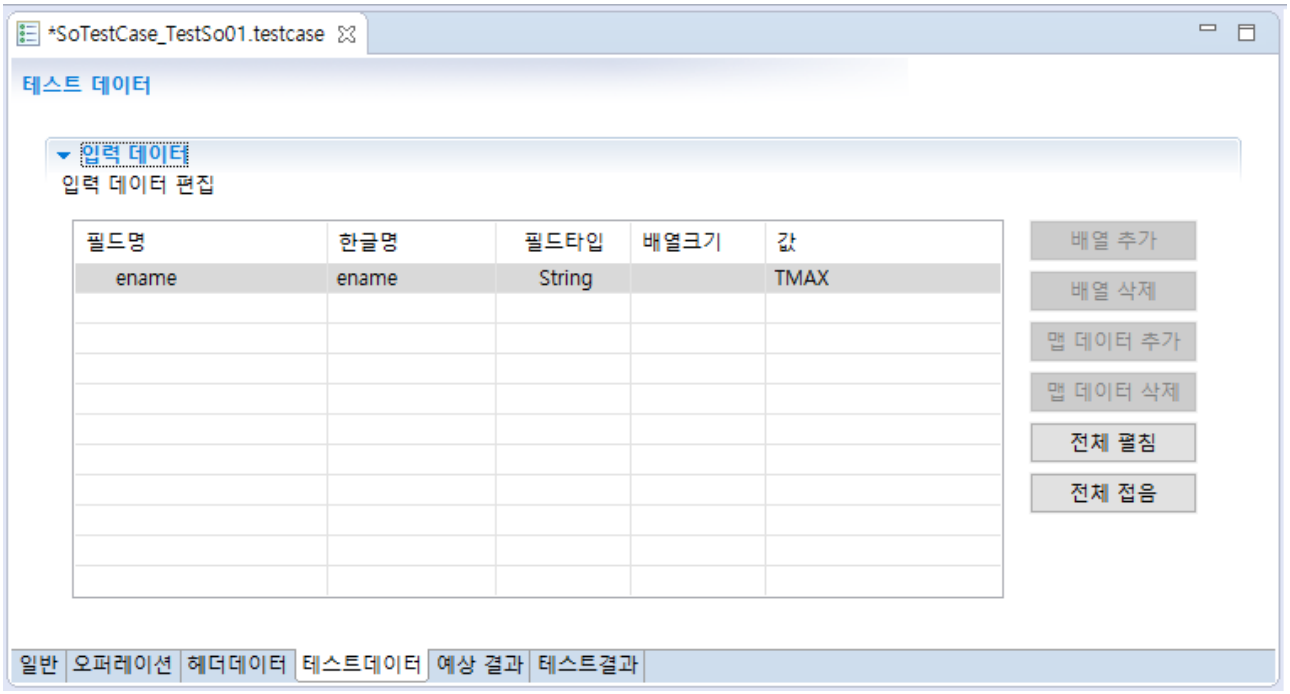
8. **[헤더데이터]** 탭을 선택하면 PO 공통 헤더 아래에 site 공통 헤더(예: ename) 내역을 확인할 수 있다. 해당 설정은 공통 헤더 설정을 참고한다.




Service Test Case 상세정보 - [헤더데이터] 탭

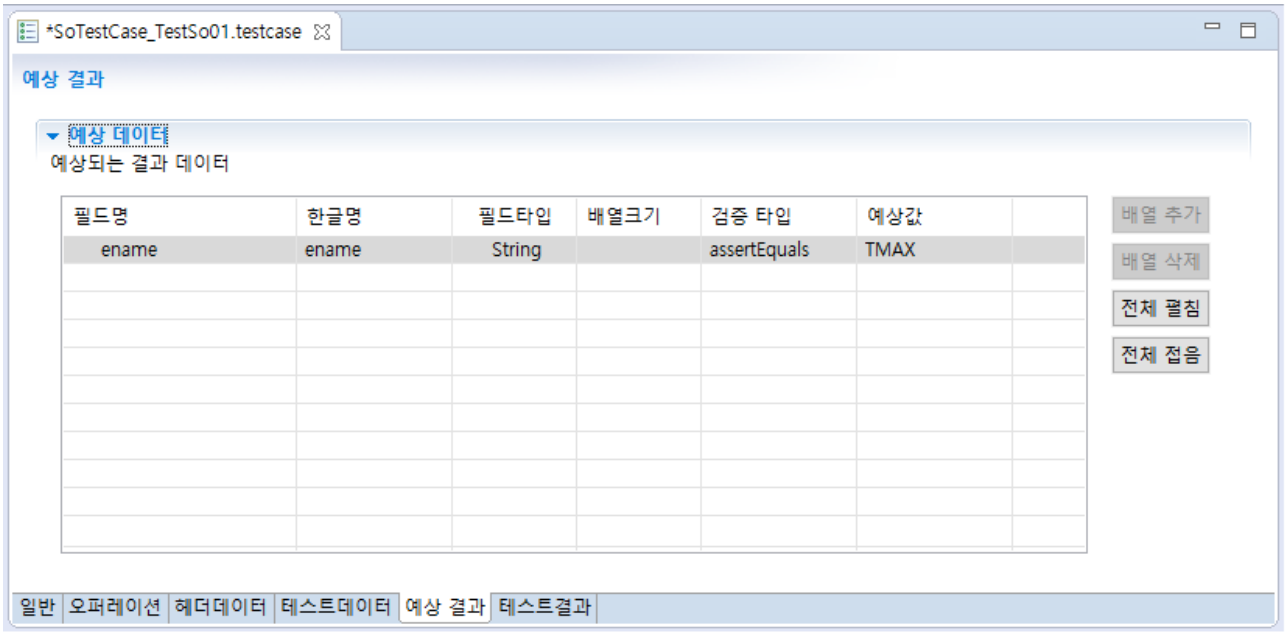
| 항목 | 설명 |
|---|---|
| 타입 | Test Type이며, 서비스 Test이기 때문에 Service Operation이다. |
| 물리명 | 테스트할 SO의 물리명이다. |
| 논리명 | 테스트할 SO의 논리명이다. |
| 입력 DTO | 테스트할 SO의 입력 DTO이다. |
| 출력 DTO | 테스트할 SO의 출력 DTO이다. |
| 설명 | 테스트할 SO의 설명이다. |

9. **[테스트데이터]** 탭을 선택한 후 **'값'** 컬럼에 서비스 입력값을 설정한다.




Service Test 입력값 세팅

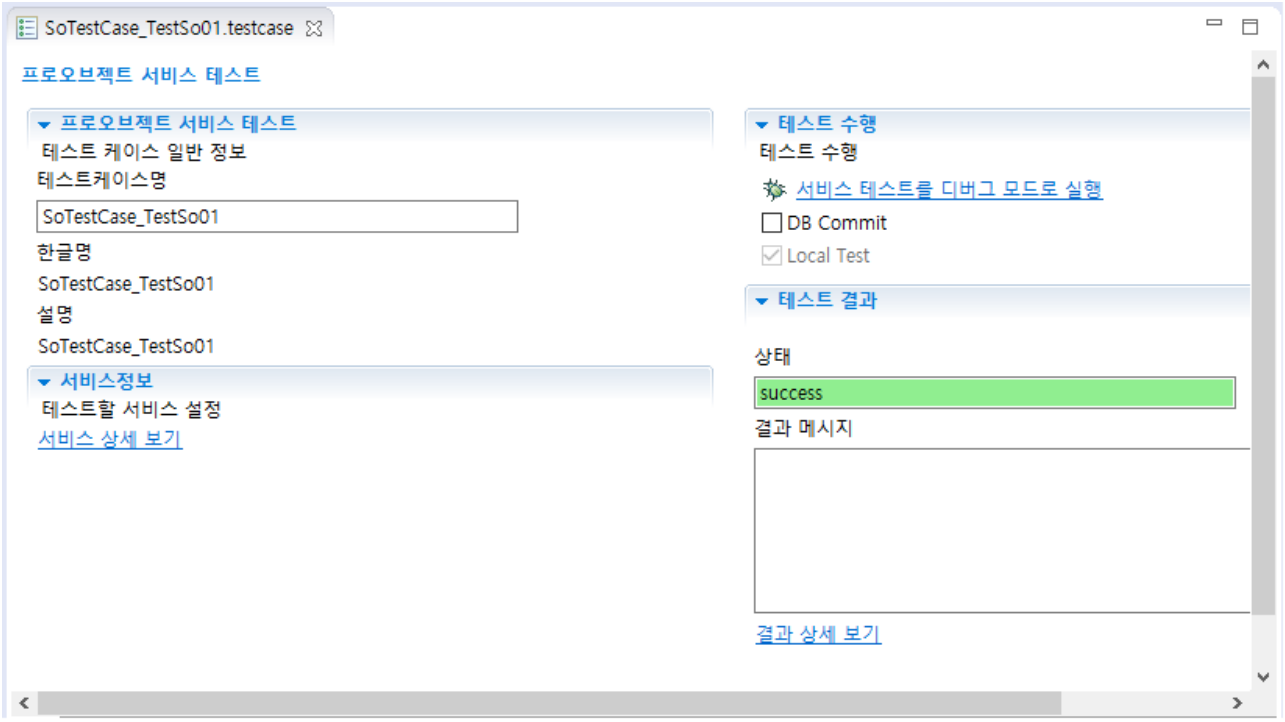
10. **[예상 결과]** 탭을 선택해서 Test 예상 결과 값을 설정한다. 예상값이 검증타입의 조건과 맞아야 해당 결과가 Success가 된다.




Test Success 기준값 세팅

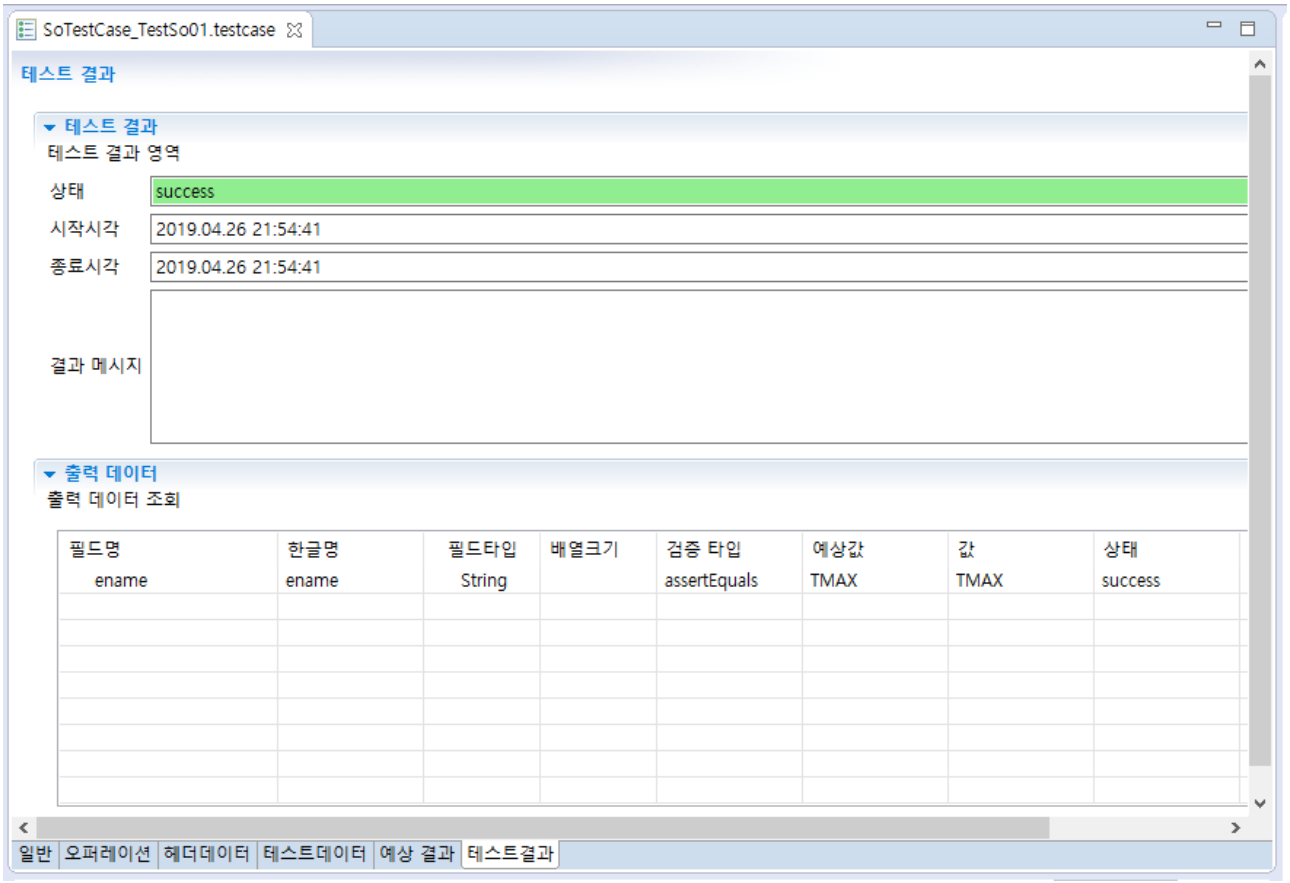
| 검증 타입 | 설명 |
|---|---|
| assertEquals | 예상값이 동일할 경우 Success이다. |
| assertNotNull | 결과 값이 NotNull일 경우 Success이다. |
| assertNull | 결과 값이 Null일 경우 Success이다. |

11. **'서비스 테스트를 디버그 모드로 실행'**을 클릭하여 SO 테스트를 수행한다.




Service Test 실행

12. **[테스트결과]** 탭에서 테스트 결과를 확인할 수 있다.



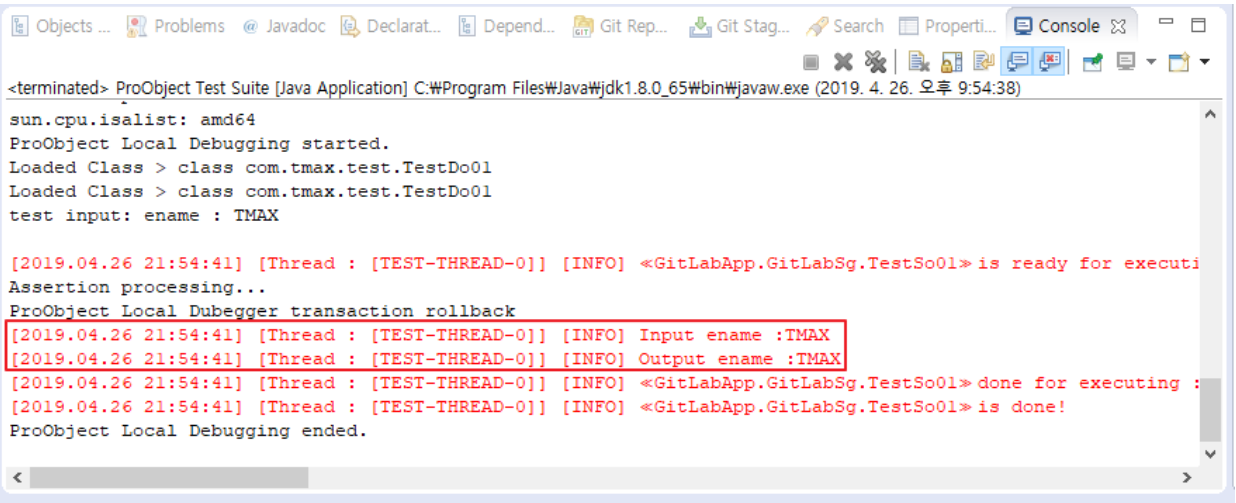

Service Test 실행 결과

- **테스트 결과**

| 항목 | 설명 |
| --- | --- |
| 상태 | 테스트 성공(success), 실패(failure) 결과이다. |
| 시작시각 | 테스트 시작시각 이다. |
| 종료시각 | 테스트 종료시각 이다. |
| 결과 메시지 | 전체 테스트 결과 메시지 내용이다. |

- **출력데이터**

| 항목 | 설명 |
| --- | --- |
| 필드명 | 결과 값의 필드명(물리명)이다. |
| 한글명 | 결과 값의 한글명(논리명)이다. |
| 필드타입 | 필드명의 유형이다. |
| 배열크기 | 필드명의 배열 크기이다. |
| 검증타입 | 필드명의 결과를 판정한 기준이다.<br>◦ Equals<br>◦ NotNull<br>◦ Null |
| 예상값 | **[예상결과]** 탭에서 입력한 입력값이다. |
| 값 | 예상값과 비교되는 테스트의 결과 값이다. |
| 상태 | 해당 필드에 해당하는 테스트 결과(success, failure)이다. |
| 결과 메시지 | 필드에 해당하는 예상값, 결과값의 비교 결과 메시지이다. |




Service Test 로그

# 5.4. 개발 편의

본 절에서는 ProStudio에서 제공하는 개발 편의 기능에 대해서 설명한다.

## 5.4.1. Deploy Descriptor Editor

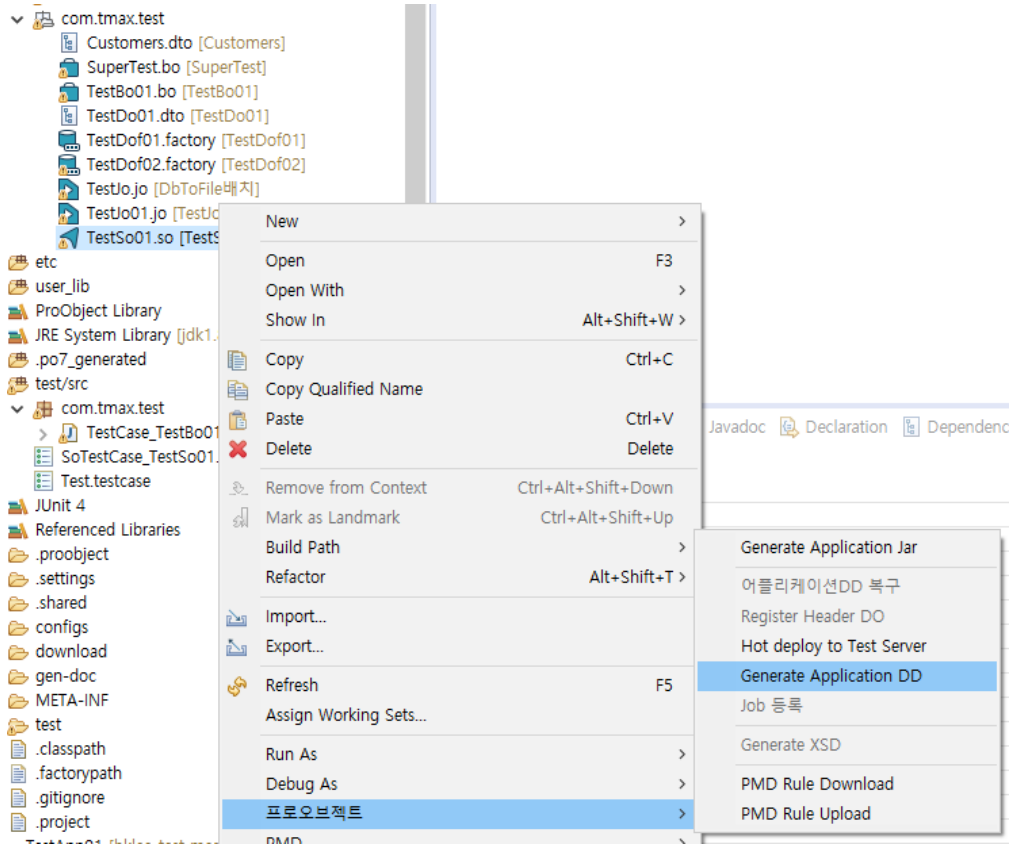
Deployment Descripter에 등록된 SO, JO를 등록/수정/삭제할 수 있는 기능을 제공한다.

### 5.4.1.1. SO/JO 추가

Deployment Descripter Editor를 사용해서 servicegroup.xml에 SO/JO의 추가할 수 있다.

- **SO 추가**

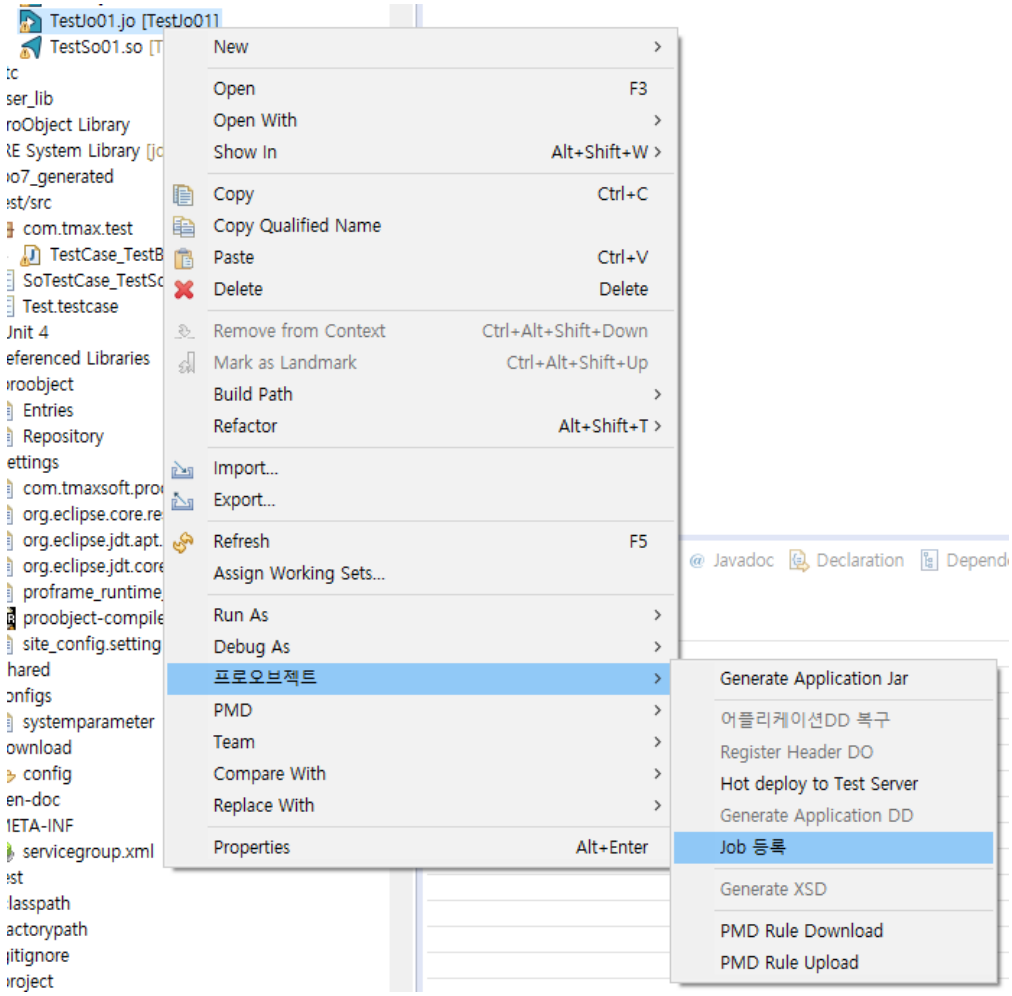
  **Package Explorer**에서 DD를 생성할 SO를 선택한 후 컨텍스트 메뉴에서 **[프로오브젝트] > [Generate Application DD]**를 선택하여 등록할 SO를 추가한다.



Deploy Descriptor Editor - SO 추가

- **JO 추가**

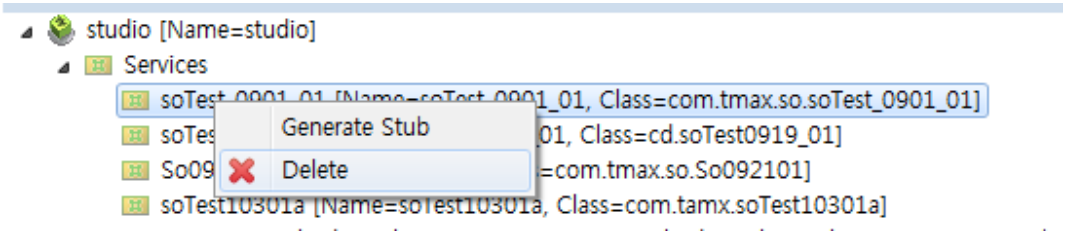
  **Package Explorer**에서 DD를 생성할 JO를 선택한 후 컨텍스트 메뉴에서 **[프로오브젝트] > [Job 등록]**을 선택하여 등록할 JO를 추가한다.



Deploy Descriptor Editor - JO 추가

### 5.4.1.2. SO 삭제

Deployment Descripter Editor를 사용해서 servicegroup.xml에 등록된 SO를 삭제할 수 있다. 삭제할 SO를 선택한 후 컨텍스트 메뉴에서 **[Delete]**를 선택한다.



Deploy Descriptor Editor - 등록된 SO 삭제

### 5.4.1.3. SO/JO 속성 수정

Deployment Descripter Editor를 사용해서 servicegroup.xml에 등록된 서비스 속성을 수정할 수 있다.

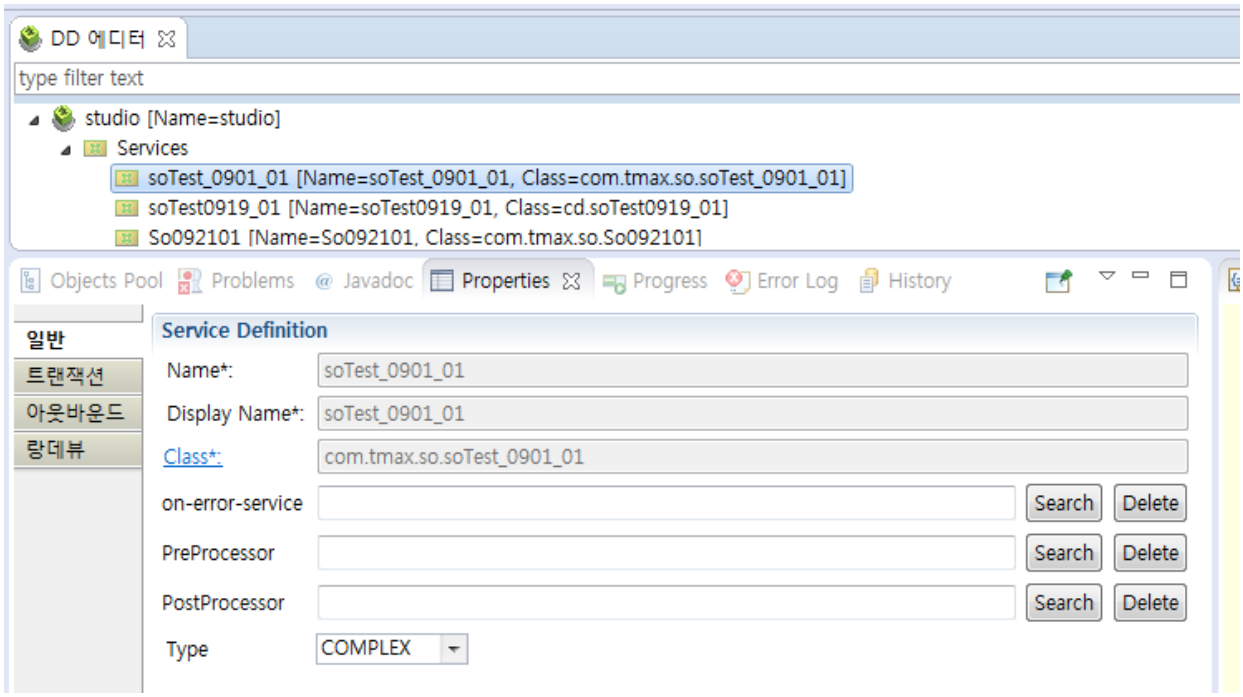
수정할 서비스를 선택한 후 **[Properties]** 탭을 선택하면 서비스의 메타 정보를 수정할 수 있다. 서비스 메타에 대한 자세한 설명은 ProObject 런타임 엔진 개발자 안내서의 "메타 설정"을 참고한다.

- **[일반]** 탭

**[일반]** 탭에서 기본정보 및 선, 후처리 등을 설정한다.

- SO

  다음은 SO의 **[일반]** 탭 화면에 대한 설명이다.

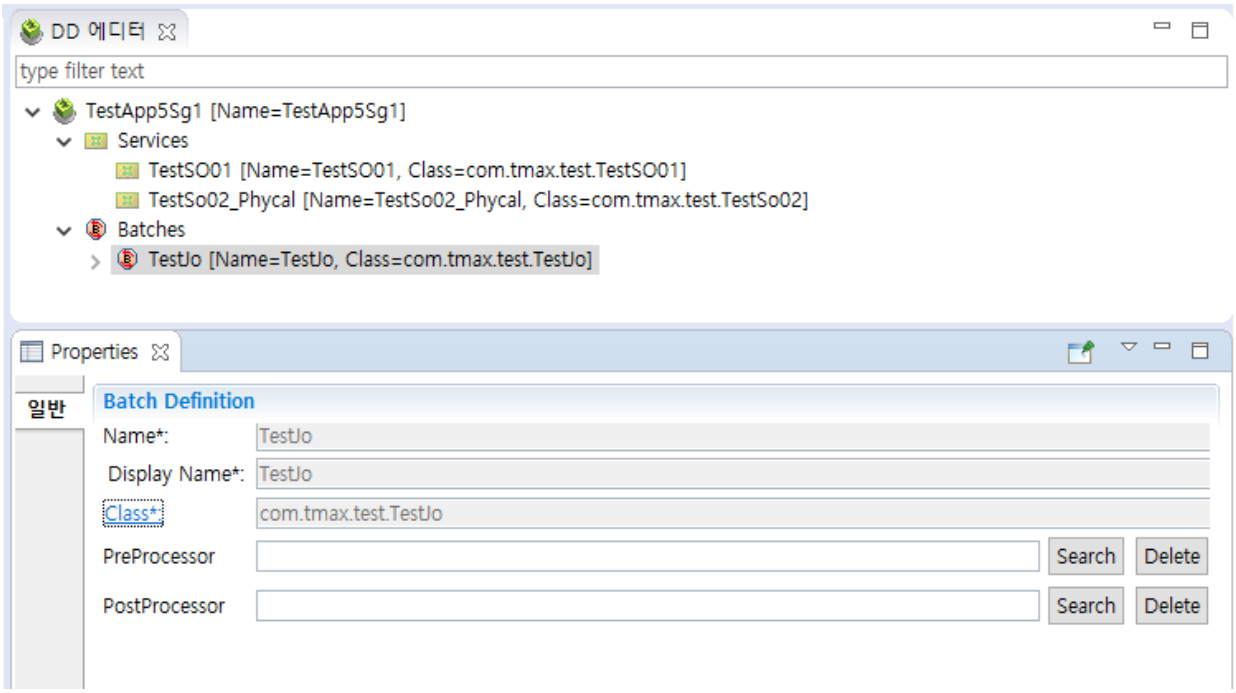



Deploy Descriptor Editor - [일반] 탭 (SO)

| 항목 | 설명 |
|---|---|
| Name/Dasplay Name | 리소스를 생성하는 경우 입력한 이름(물리명)이다. |
| Class | 클래스 이름을 Package Path 포함해서 입력한다. |
| on-error-service | 에러로 리턴되었을 때 수행되는 서비스명을 설정한다.<br><br>SO만 설정 가능하며 **[Search]** 버튼을 클릭해서 설정할 수 있다. **[Delete]** 버튼을 클릭해서 설정된 값을 초기화할 수 있다. |
| PreProcessor | 서비스를 수행하기 전에 수행되어야 하는 클래스를 설정한다.<br><br>**[Search]** 버튼을 클릭해서 PreProcessor를 상속한 BO를 설정한다. **[Delete]** 버튼을 클릭해서 설정된 값을 초기화할 수 있다. |
| PostProcessor | 서비스를 수행한 후에 수행되어야 하는 클래스를 설정한다.<br><br>**[Search]** 버튼을 클릭해서 PostProcessor를 상속한 BO를 설정한다.<br>**[Delete]** 버튼을 클릭해서 설정된 값을 초기화할 수 있다. |
| Type | ◦ COMPLEX : 서비스 그룹의 워커 스레드에서 수행되는 일반적인 서비스의 형태이다.<br>◦ EVENT : 단일 스레드인 이벤트 처리 스레드에서 수행되는 서비스이다.<br>◦ OUTBOUND : Out Bound SO Type을 말하며, Value가 선택될 때 **[아웃바운드]** 탭의 **[설정]** 버튼이 활성화된다. |

◦ JO

다음은 JO의 **[일반]** 탭 화면에 대한 설명이다. 각 항목에 대한 설명은 SO의 **[일반]** 탭 화면(Deploy Descriptor Editor - [일반] 탭 (SO))을 참고한다.

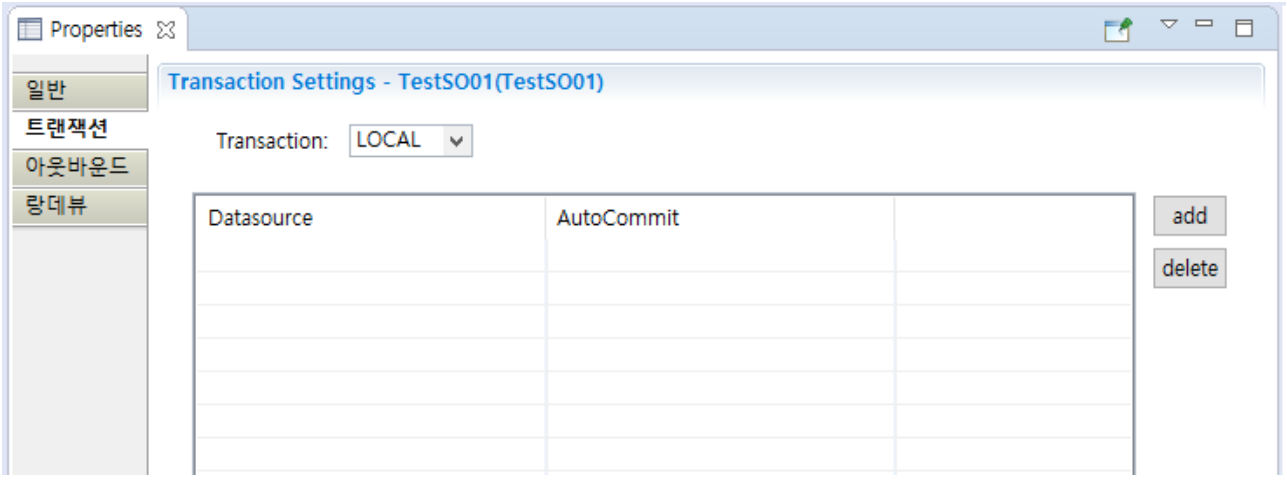



Deploy Descriptor Editor - [일반] 탭 (JO)

- **[Transaction]** 탭

다음은 SO의 트랜잭션 정보를 설정하는 화면에 대한 설명이다.

Global 트랜잭션일 경우에는 **[add]** 버튼을 클릭해서 Row를 추가한 후에 'Datasource', 'AutoCommit' 여부를 설정한다. Row를 선택한 후 **[delete]** 버튼을 클릭하면 해당 Row를 삭제할 수 있다.



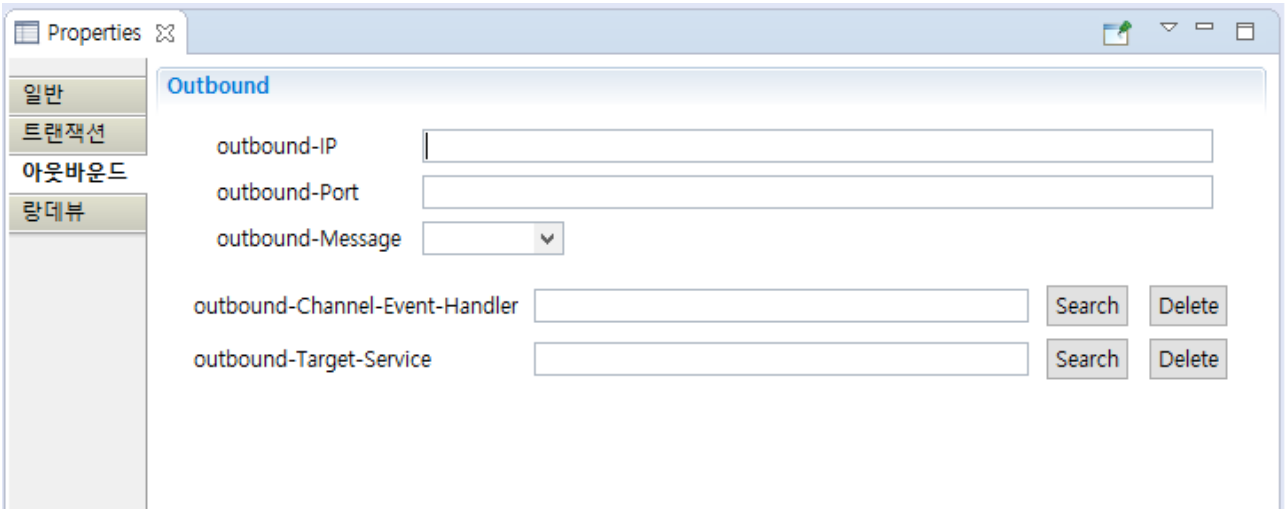

Deploy Descriptor Editor - [Transaction] 탭

| 항목 | 설명 |
|---|---|
| Transaction | LOCAL, GLOBAL 트랜잭션 여부를 설정한다.<br><br>GLOBAL 트랜잭션일 경우에는 Datasource, AutoCommit을 설정해야 한다. |

| 항목 | 설명 |
| --- | --- |
| Datasource | 런타임에서 사용될 데이터소스를 설정한다. |
| AutoCommit | Auto Commit 여부를 설정한다. (true/false) |

- **[아웃바운드]** 탭

  다음은 SO의 OutBound SO Type일 경우 설정이 필요한 화면이다. **[일반]** 탭의 **'Type'** 항목이 'OUTBOUND'로 선택된 경우 입력 가능하도록 활성화된다.

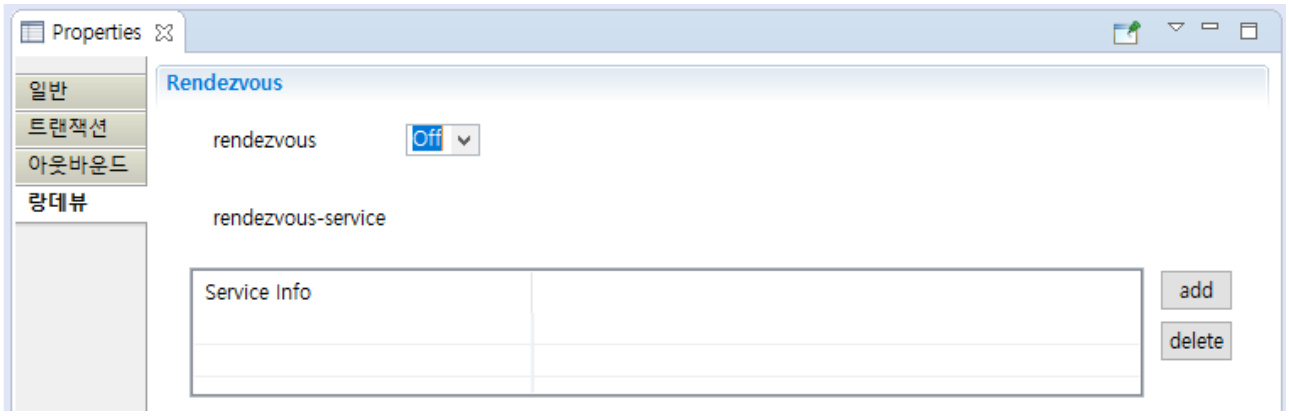



Deploy Descriptor Editor - [아웃바운드] 탭

| 항목 | 설명 |
| --- | --- |
| outbound-IP | 요청할 외부 IP를 설정한다. |
| outbound-Port | 요청할 외부 IP의 Port를 설정한다. |
| outbound-Message | 전송할 메시지 유형을 설정한다.<br>• SERIALIZE<br>• Json Type |
| outbound-Channel-Event-Handler | 채널 핸들러중 현재 서비스에 사용될 Event Handler를 설정한다.<br>**[Search]** 버튼을 클릭해서 ChannelEventHandler를 상속한 BO를 설정한다. **[Delete]** 버튼을 클릭해서 설정된 값을 초기화할 수 있다. |
| outbound-Target-Service | 호출할 서비스의 이름이 ServiceManager를 통해 호출한 호출자의 이름과 동일하지 않은 경우에 설정한다.<br>**[Search]** 버튼을 클릭해서 선택할 SO를 설정한다. **[Delete]** 버튼을 클릭해서 설정된 값을 초기화할 수 있다. |

- **[랑데뷰]** 탭

  아래 SO의 랑데뷰 연동서비스를 설정하는 화면이다.

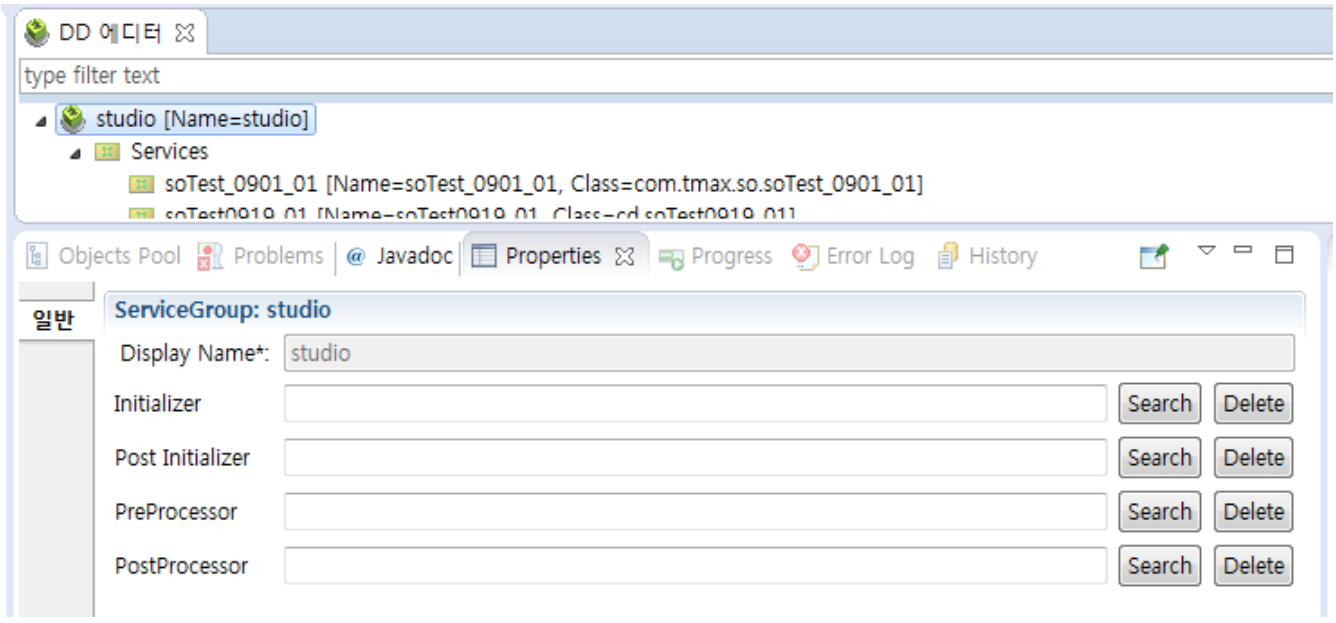


Deploy Descriptor Editor - [랑데뷰] 탭

| 항목 | 설명 |
|---|---|
| rendezvous | rendezvous 설정을 On/Off한다. |
| Service Info | 랑데뷰할 서비스를 설정한다.<br><br>**[add]** 버튼을 클릭해서 "App.SG.serviceName" 형태로 지정한다. **[Delete]** 버튼을 클릭해서 설정된 값을 초기화할 수 있다. |

#### 5.4.1.4. 서비스 그룹 속성 수정

Deployment Descripter Editor를 사용해서 servicegroup.xml에 등록된 서비스 그룹 속성을 수정할 수 있다. 수정할 서비스를 선택한 후 **[properties]** 탭을 선택하면 서비스 그룹의 메타 정보를 수정할 수 있다.



Deploy Descriptor Editor - 서비스 그룹 속성 설정

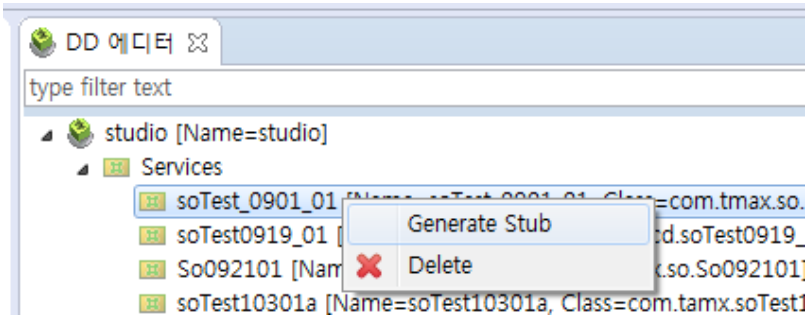
서비스 그룹 메타에 대한 자세한 설명은 ProObject 런타임 엔진 개발자 안내서의 "애플리케이션 개발"을 참고한다.

#### 5.4.1.5. WSDL 생성

웹 서비스로 호출하기 위해 ProStudio에서 WSDL를 생성하는 기능을 제공한다.

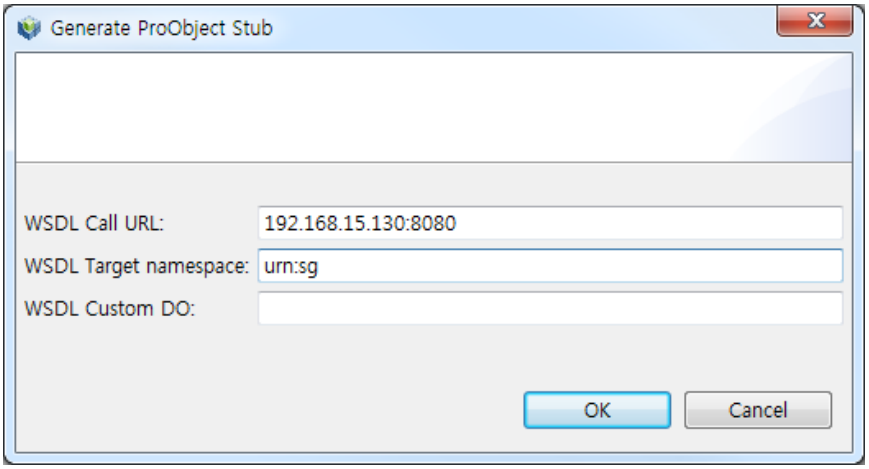
다음은 WSDL 생성 과정에 대한 설명이다.

1. **Package Explorer**에서 servicegroup.xml의 SO를 선택한 후 컨텍스트 메뉴에서 **[Generate Stub]**를 선택한다.




Deploy Descriptor Editor - WSDL 생성 (1)

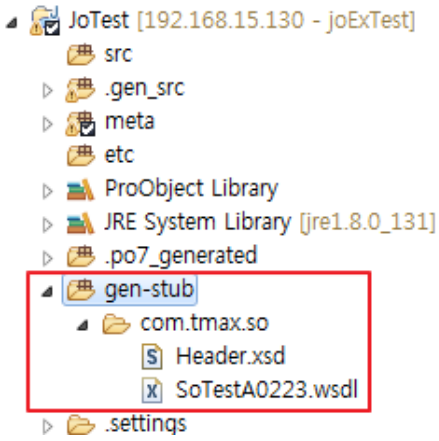
2. **Generate ProObject Stub 화면**에서 각각의 properties 정보를 입력한다. 입력된 내용을 확인하고 **[OK]** 버튼을 클릭한다.




Deploy Descriptor Editor - WSDL 생성 (2)

| 항목 | 설명 |
|---|---|
| WSDL Call URL | 웹 서비스를 호출할 서버의 ip:port 정보를 입력한다. (필수) |
| WSDL Target namespace | WSDL 파일의 네임스페이스를 지정한다. (필수) |
| WSDL Custom DO | WSDL 파일의 커스텀 DO 정보를 입력한다. |

3. 저장이 완료되면 WSDL과 전문 헤더에 대한 xsd가 생성된다.



Deploy Descriptor Editor - WSDL 생성 (3)

## 5.4.2. 메타 관리

본 절에서는 DO 작성 화면에서 메타 관리를 위한 편의 기능에 대해서 설명한다.

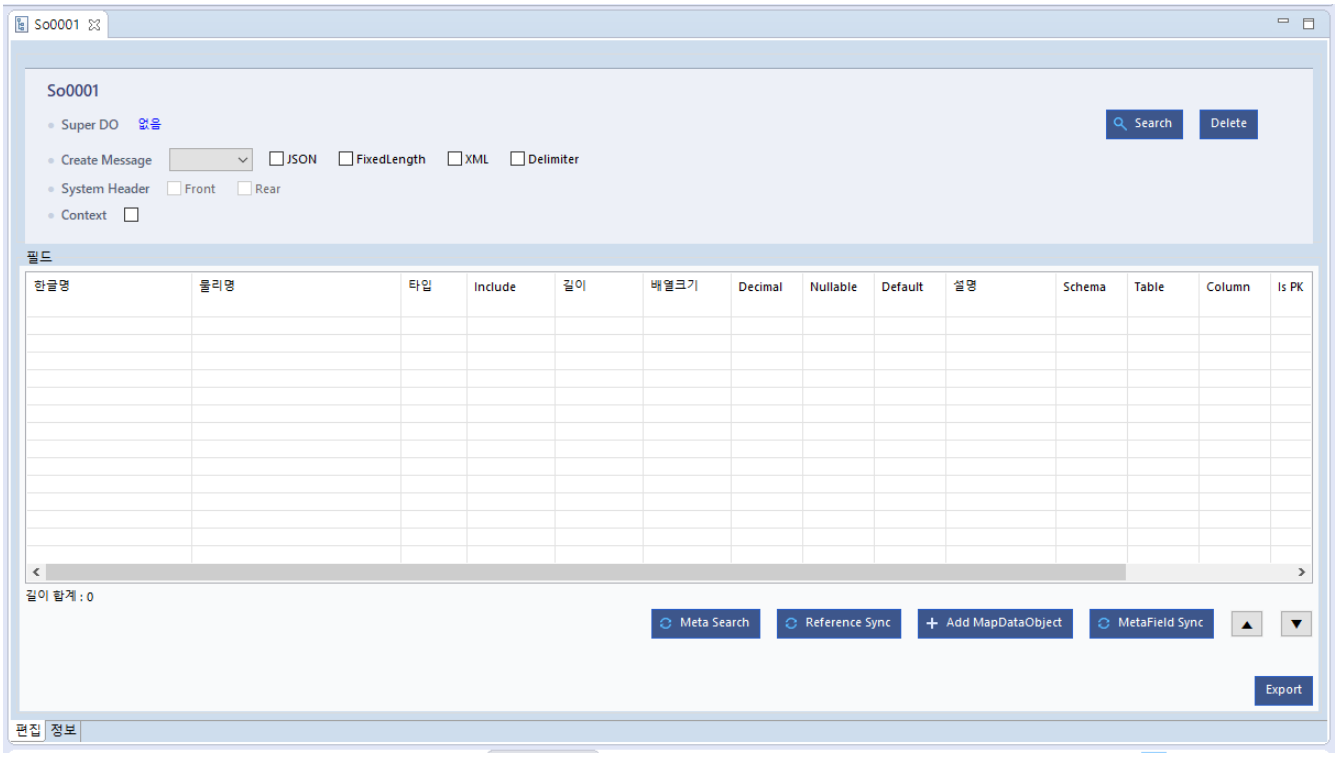
### 5.4.2.1. 메타 조회 기능

DO 작성화면에서 메타를 추가할 수 있는 기능인 메타 조회기능(Meta Search) 기능을 제공하고 있으며, **MetaData View** 화면의 Property를 활용하는 방법 보다 나은 편의성을 제공한다.

DO 작성 화면에서 메타 조회 기능 사용을 위해서는 SiteConfig.xml의 USE_FILED_NAME_META_SEARCH 설정이 true가 되어야 한다.

```
<?xml version="1.0" encoding="EUC-KR" standalone="yes"?>
<siteConfig xmlns="http://www.tmax.co.kr/proobject/siteConfig">
    :
  중략
    :
   <siteElement id="USE_FILED_NAME_META_SEARCH" value="true" type="boolean" xmlns=""/>
</siteConfig>
```

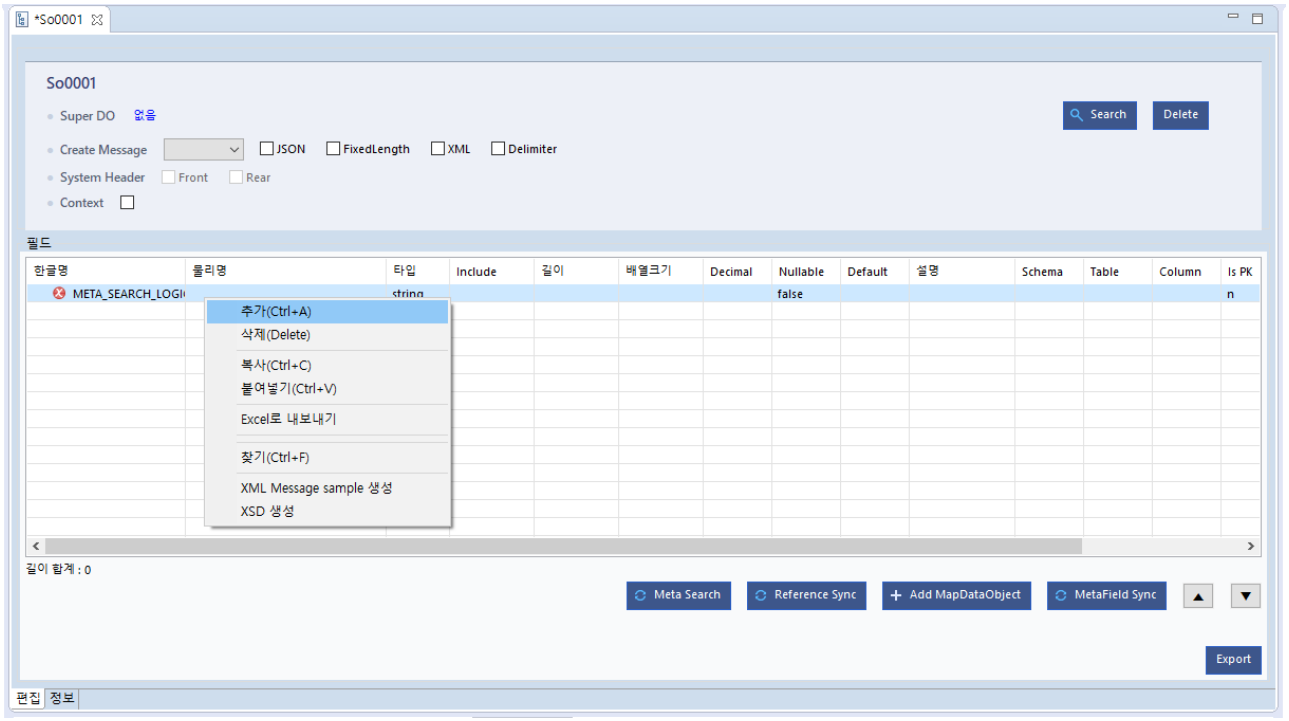
해당 설정이 활성화되면 메타 조회 화면에 **[Meta Search]** 버튼이 활성화된다.




메타 조회 기능 - [Meta Search] 버튼 활성화

다음은 DO 작성화면에서 메타를 추가하는 과정에 대한 설명이다.

1. 메타 추가를 위해서는 DO Grid의 컨텍스트 메뉴에서 **[추가(Ctrl+A)]**를 선택한다. **'한글명'**에 'META_SEARCH_LOGICAL_NAME'이 표시되면 **'물리명'**에 검색할 메타를 입력한다.

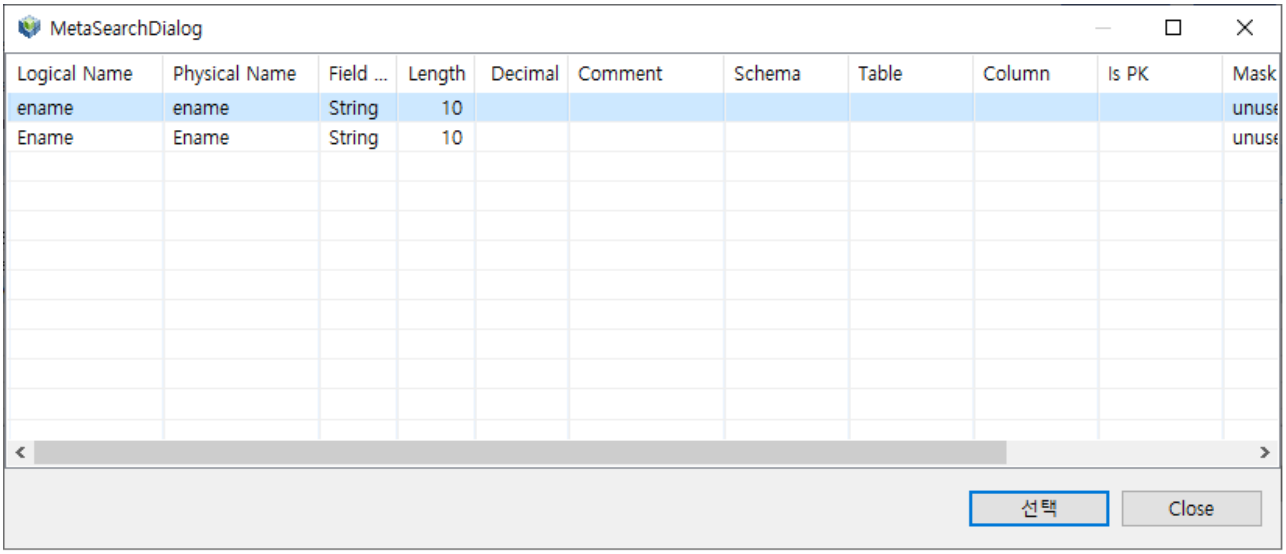



메타 조회 기능 - Dummy Meta Row추가

2. **'물리명'** 컬럼에 추가할 메타의 물리명을 넣고 **[Meta Search]** 버튼을 클릭하면 다음과 같이 문자를 포함하는 메타 내역이 조회된다.

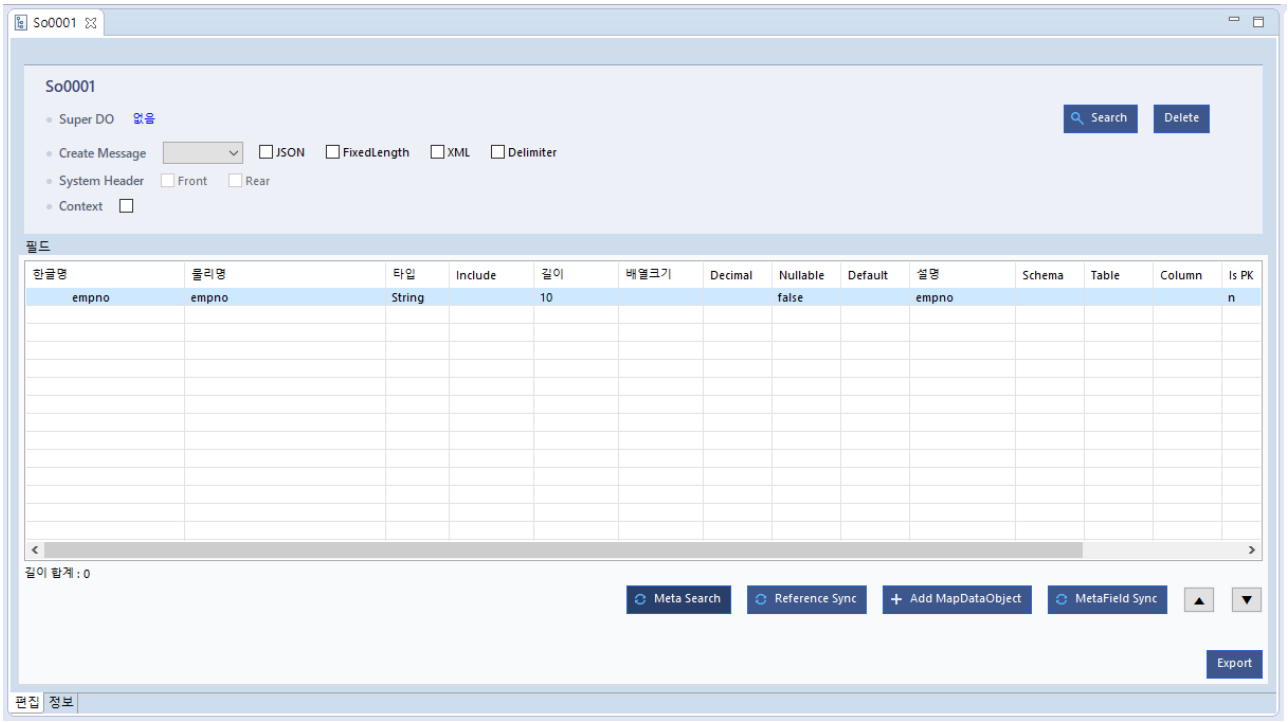
   목록에서 적용할 메타를 선택한 후 **[선택]** 버튼을 클릭하거나 더블클릭하면 DO Grid에 메타가 적용된다. 대상

물리명이 1개일 경우에는 아래 화면이 뜨지않고 DO Grid에 적용된다. **[Close]** 버튼을 클릭하면 DO Grid에 메타를 적용하지 않고 창을 닫는다.




메타 조회 기능 - 검색된 메타 선택

3. 다음은 메타 조회 기능을 사용하여 메타를 추가한 결과이다.



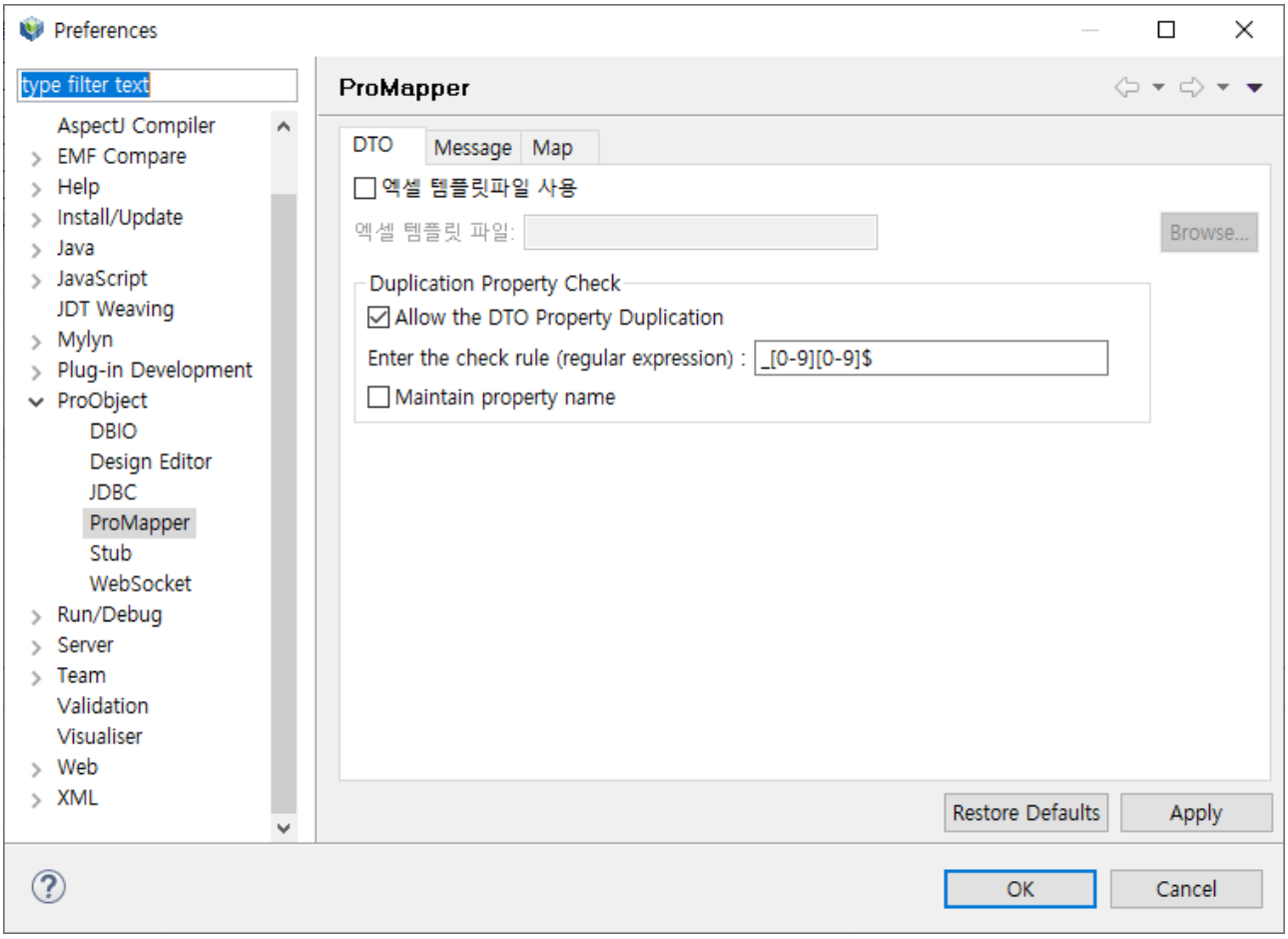

메타 조회 기능 - 결과

### 5.4.2.2. 메타 일련번호 패턴 적용

DO를 작성하는 경우 메타 내역에 대한 일련번호 패턴을 강제할 수 있다. 메타 필드 내역이 개발자별 또는 전체 개발자가 일관된 패턴 적용을 위한 기능이다. 해당 기능을 사용하기 위해서 서버 파일과 ProStudio 설정을 해야 한다.

- **ProStudio 설정으로 메타 패턴 적용**

ProStudio 설정은 각 개발자별로 설정이 가능하다. **[Window] > [Preferences]** 메뉴를 선택한 후 업무 트리에서 **[ProObject] > [ProMapper]**를 클릭한 후 화면에 각 항목을 설정한다.




메타 일련번호 패턴 - - Studio설정

| 항목 | 설명 |
|---|---|
| Allow the DTO Property Duplication<br><br>Enter thr check rule | 정규식으로 패턴으로 변수명의 뒷부분을 체크한다. 예를 들어 ename이 변수명이고 "_[0-9][0-9]$"를 중복가능 변수로 추가하면 'ename_00(00은 숫자)'를 메타 필드로 추가할 수 있다.<br><br>아래는 설정 예제이다.<br><br>• [0-9][0-9]$<br>• _[0-9][0-9]$<br>• _[a-z][A-Z][0-9][0-9]$ |
| Maintain property name | 메타에 등록된 내역만 사용할지 여부를 설정한다.<br><br>체크되어 있을때 메타에 등록된 내역만 사용한다. 따라서 하나의 메타 필드에 일련번호 패턴 적용 기능의 사용을 위해서는 체크가 해제되어 있어야 한다. |

- **서버 설정으로 메타 패턴 적용**

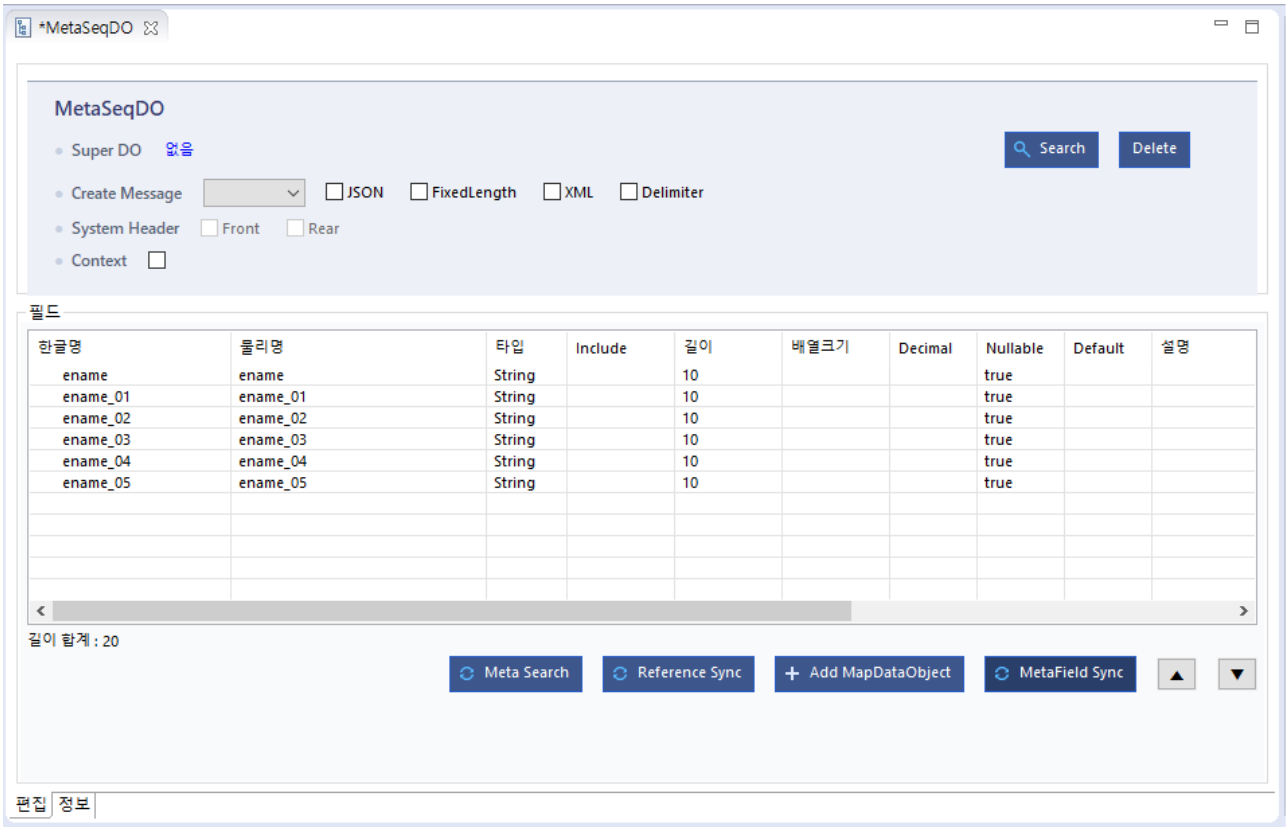
서버 설정으로 메타 패턴 적용하기 위해서 $PROOBJECT_HOME/system/config/ProbuilderConfig.xml에 정규식이 지정되어 있어야 한다.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<probuilder_config xmlns="http://www.tmax.co.kr/proobject/probuilder_config">
  <default-dto-names dtoPrefix="Pre" dtoSuffix="Dto" inDtoPrefix="InPre" inDtoSuffix="ListDto"
xmlns=""/>
  <real-type xmlns="">true</real-type>
  <all-property-from-server xmlns="">false</all-property-from-server>
  <proMapperConfig xmlns="">
          :
        중략
          :
    <dtoDuplicatePropertyNamingConstraintRegex>_[0-9][0-
9]$</dtoDuplicatePropertyNamingConstraintRegex>
  </proMapperConfig>
</probuilder_config>
```

해당 설정은 ProStudio 설정보다 우선시 된다. 따라서 ProStudio 설정이 되어 있더라도 dtoDuplicatePropertyNamingConstraintRegex 설정이 되어 있으면, ProStudio의 "Allow the DTO Property Duplication" 설정은 무시된다.

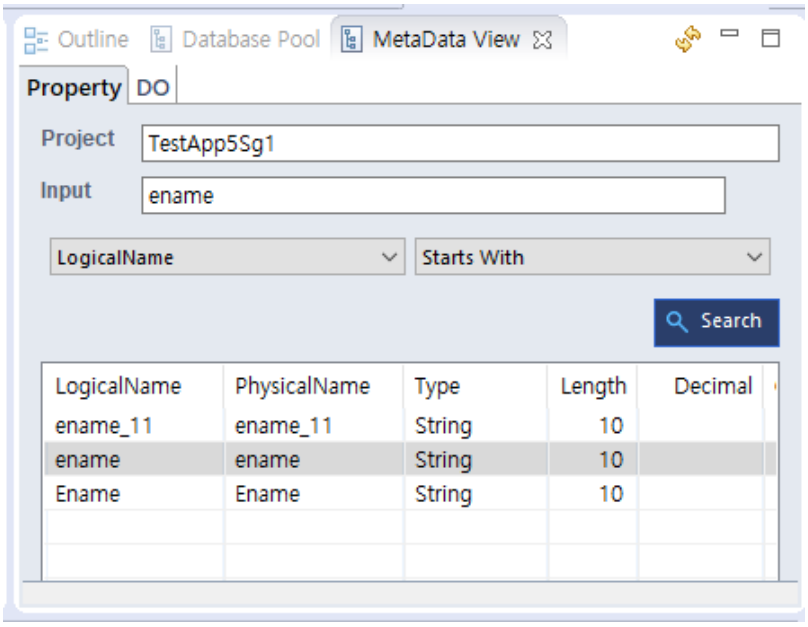
다음은 중복되는 컬럼 **'ename'**에 '_[0-9][0-9] Pattern'을 설정해서 추가하는 과정에 대한 설명이다.

1. **MetaSeqDO 화면**에서 메타 일련번호를 조회한다.



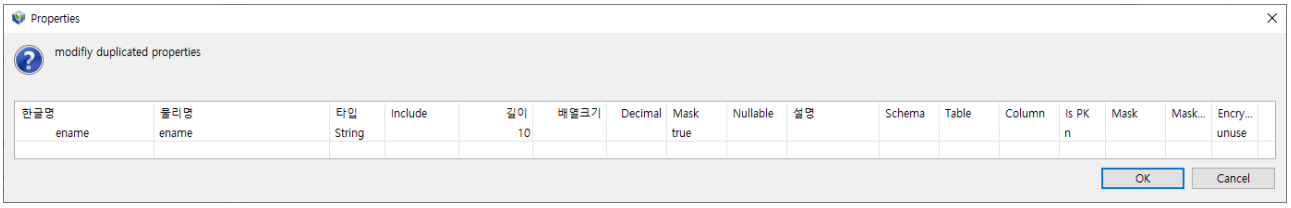

메타 일련번호 조회

2. 중복 메타인 'ename_06'을 다음과 같이 추가한다. 'ename'을 조회 후 중복 추가할 메타 필드인 ename을 더블클릭하여 선택한다.



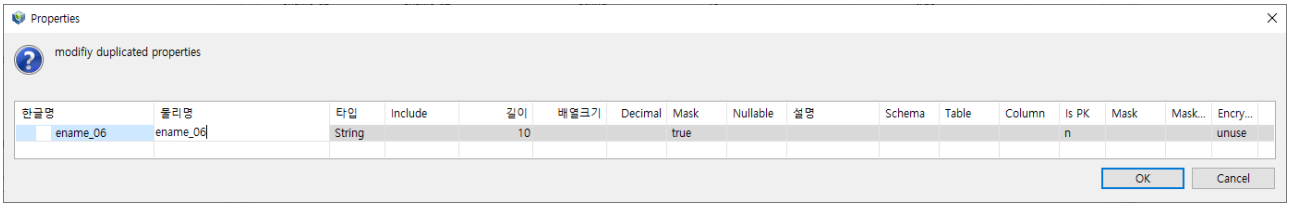

메타 일련번호 패턴 - 메타 조회

선택된 메타 필드인 ename이 표시된다.



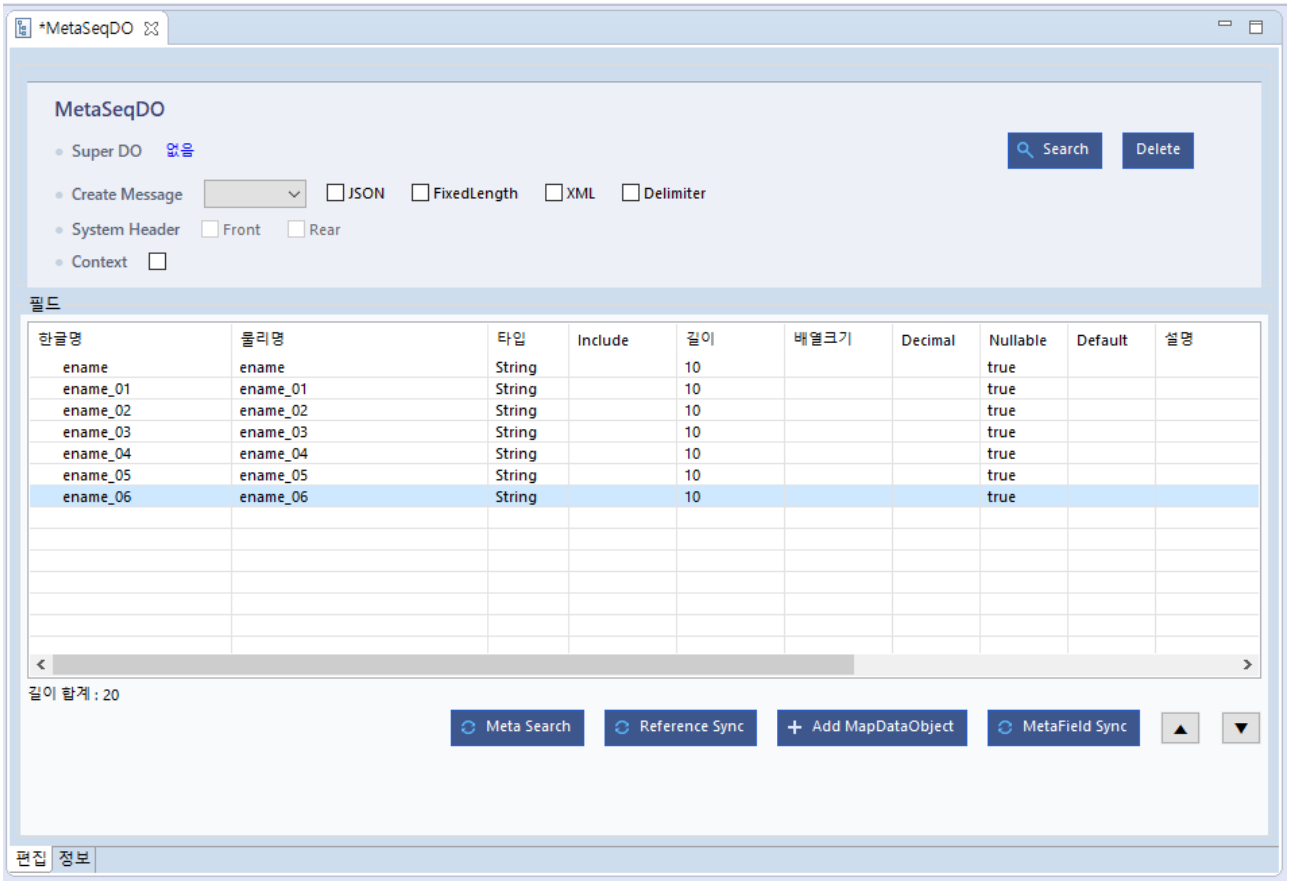

메타 일련번호 패턴 - 중복확인

한글명 및 물리명에 설정된 정규식 패턴인 "_[0-9][0-9]"에 맞게 'ename_06'을 넣는다.




메타 일련번호 패턴 - 이름변경

3. 다음은 일련번호를 적용해서 메타를 추가한 결과화면이다.

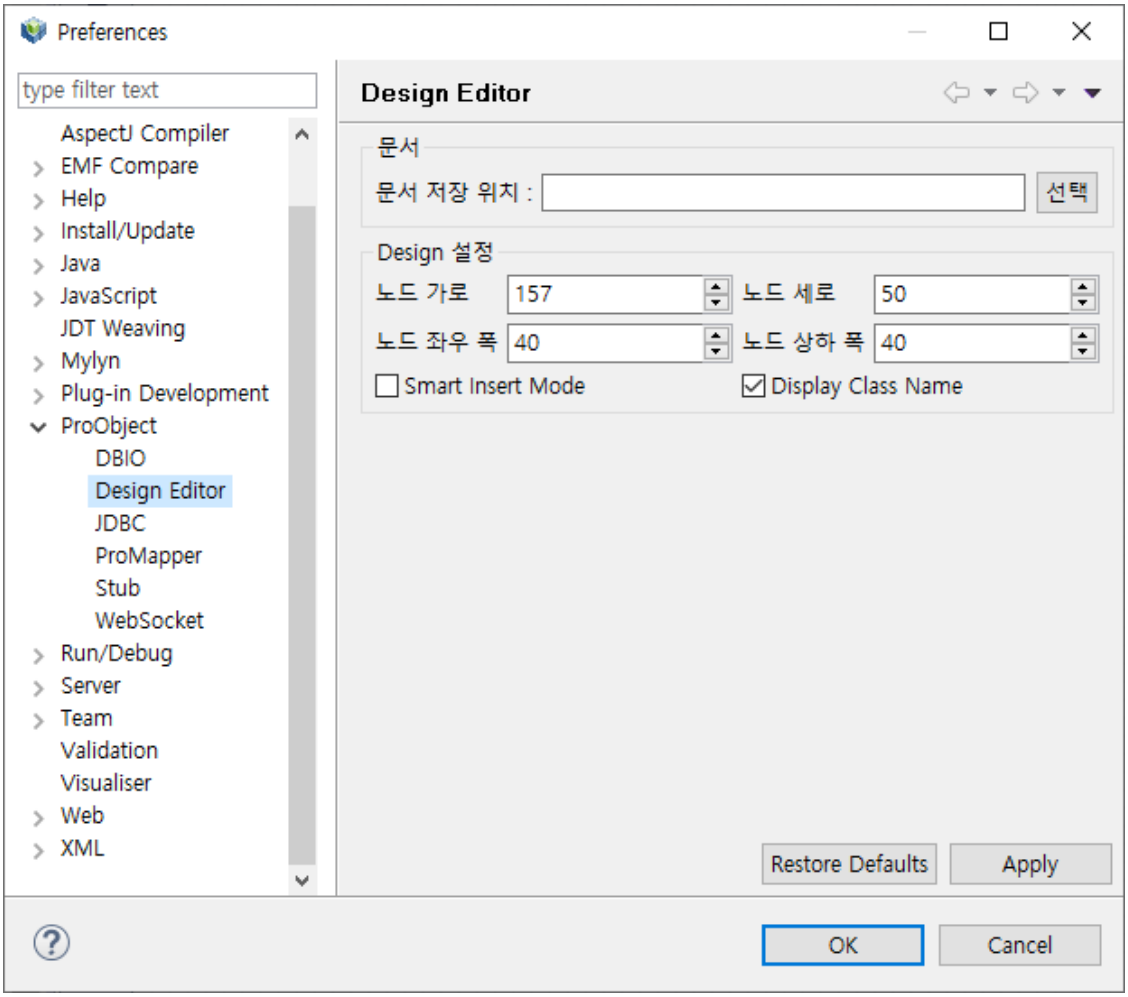


메타 일련번호 패턴 - 일련번호 메타 추가 완료

## 5.4.3. 모듈 상세정보 표시

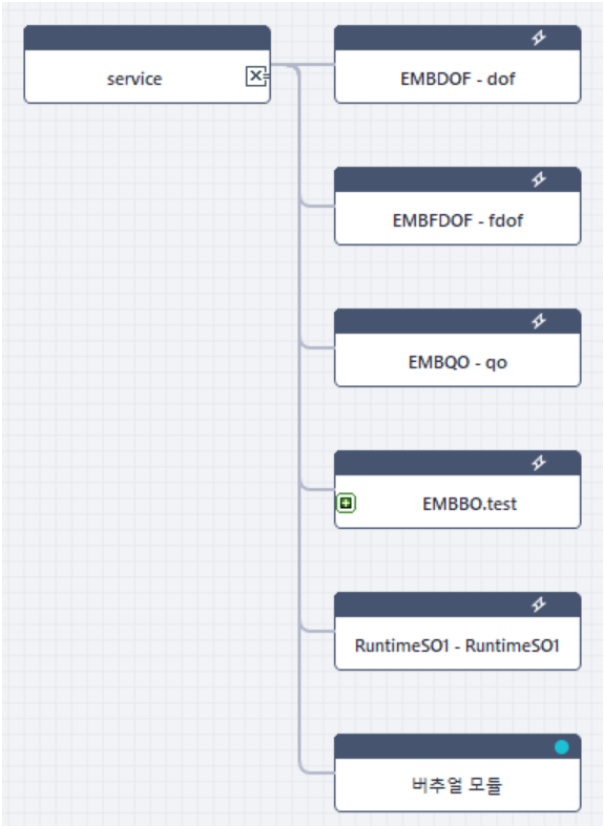
**[Window] > [Preferenecs]** 메뉴를 선택한 후 업무 트리에서 **[ProObject] > [Design Editor]**를 선택하면 EMB에 Flow에 노드애 상세한 정보를 표시할 수 있다. **Sesign Editor 영역**에서 **'Display Class Name'** 항목을 체크한다.

BO의 경우는 'BO_Class_Name.function'으로 표시하며, 나머지 모듈은 'Class_Name - Node Name' 형태로 표시한다.



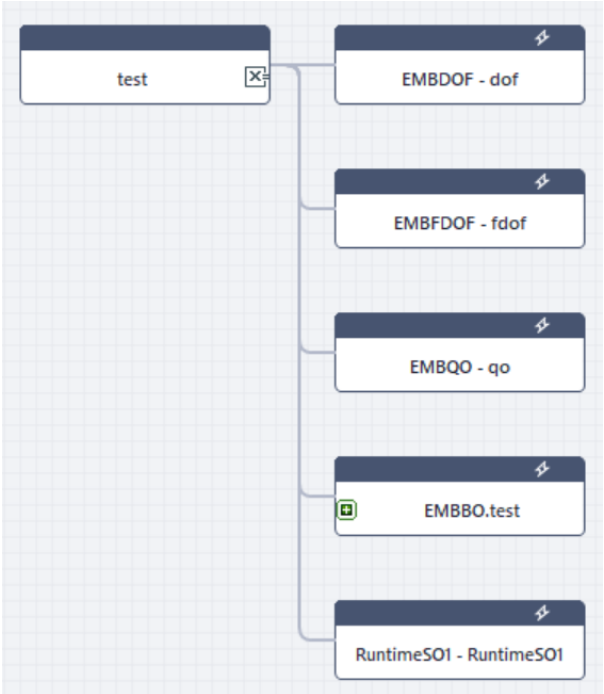
모듈 정보 상세 - 설정화면

- SO에서 모듈 유형별 상세 정보



모듈 정보 상세 - SO의 예시

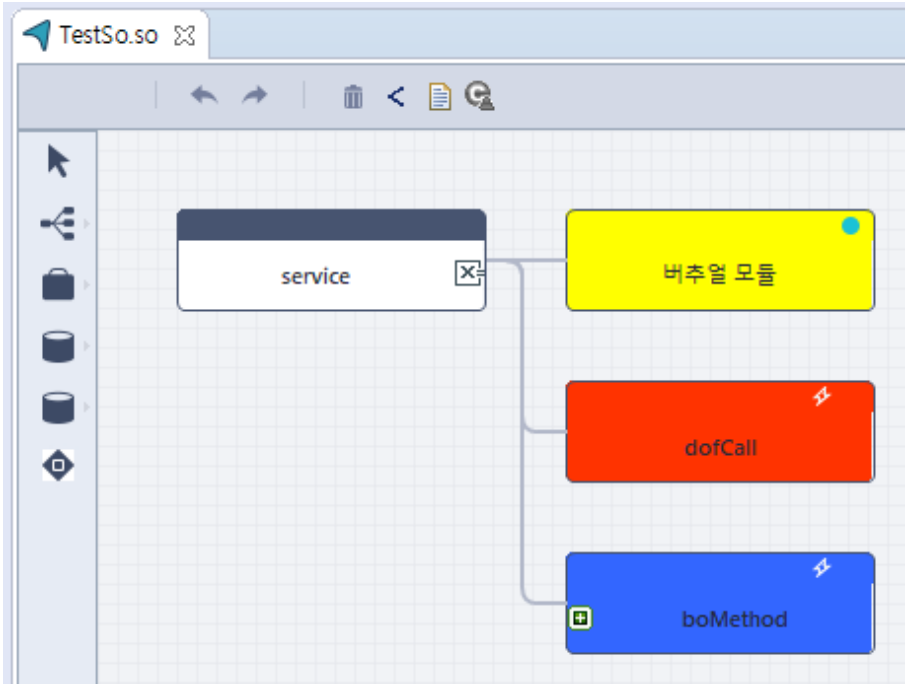
- BI에서 모듈 유형별 상세 정보



모듈 정보 상세 - BO의 예시

## 5.4.4. 모듈 색상 지정

**EMB Designer**에서 다음과 같이 버추얼 모듈, DO Factory 모듈, BO 모듈에 대해 생상을 지정할 수 있다.



EMB Designer 모듈 색상 지정

색상 지정을 위해서는 다음과 같이 SiteConfig.xml 설정 파일의 siteEmbColor 설정이 필요하다. 각 모듈별 설정은 virtual(버추얼 모듈), doCall(DO Factory 모듈), bizMethodCall(BO 모듈)별 iconPaneColor, backgroundColor를 지정한다.

```
<?xml version="1.0" encoding="EUC-KR" standalone="yes"?>
<siteConfig xmlns="http://www.tmax.co.kr/proobject/siteConfig">
     :
    중략
    :
  <siteEmbColors xmlns="">
    <!-- 버추얼 모듈     -->
    <module moduleName="virtual" iconPaneColor="FFFF00" backgroundColor="FFFF00"/>
    <!-- DO Factory 모듈 -->
    <module moduleName="doCall" iconPaneColor="FF3300" backgroundColor="FF3300"/>
    <!-- BO 모듈          -->
    <module moduleName="bizMethodCall" iconPaneColor="3366FF" backgroundColor="3366FF"/>
  </siteEmbColors>
</siteConfig>
```

# Part II. 부가 기능 사용법

# 6. GIT 연동

본 장에서는 ProStudio에서 GIT을 연동하여 소스 코드를 형상관리하는 방법에 대해서 설명한다.

## 6.1. 개요

ProStudio는 Git 서버와 연동해서 프로젝트에 대해서 형상관리를 할 수 있다. ProStudio와 Git 서버를 연동하기 전에 클라이언트에 Git이 설치되어 있어야 하고 Git 서버에는 repository가 생성되어 있어야 한다. 본 장에서는 ProStudio는 Git 서버와 연동하는 과정과 Git을 사용하는 과정에 발생할 수 있는 상황에 대해서 설명한다. 본 장의 내용을 이해하기 위해서는 Git의 기본적인 동작(add index, commit, push, fetch, pull, reset)에 대해서 미리 숙지해야 한다. (http://rogerdudler.github.io/git-guide/index.ko.html 참고)

ProStudio와 Git 서버를 연동은 다음의 과정으로 진행한다.

1. 연동 전 준비사항
2. 로컬에 Remote Git Repository 복사
3. 새로운 프로젝트를 Git 서버에 복사
4. Git 서버로부터 프로젝트 import

Git을 사용해서 여러 개발자가 소스코드에 접근하는 경우 다른 개발자의 소스와 자신이 개발한 소스코드의 내용이 Conflict(충돌)해서 문제가 발생하는 경우가 생길 수 있고, 소스코드에 대한 개발 이력을 추적하는 기능을 사용할 수 있다. 해당 기능에 대한 자세한 내용은 부가 기능을 참고한다.

## 6.2. 연동 전 준비사항

ProStudio와 Git 서버를 연동하기 전에 ProStudio를 사용할 시스템의 사양과 Git 클라이언트에 환경설정이 되어 있어야 한다. 본 절에서는 연동 전에 준비사항에 대해서 설명한다.

### 6.2.1. 시스템 확인

ProStudio 구동 전에 제품을 사용하기 위한 시스템의 최소 요구사항을 확인한다.

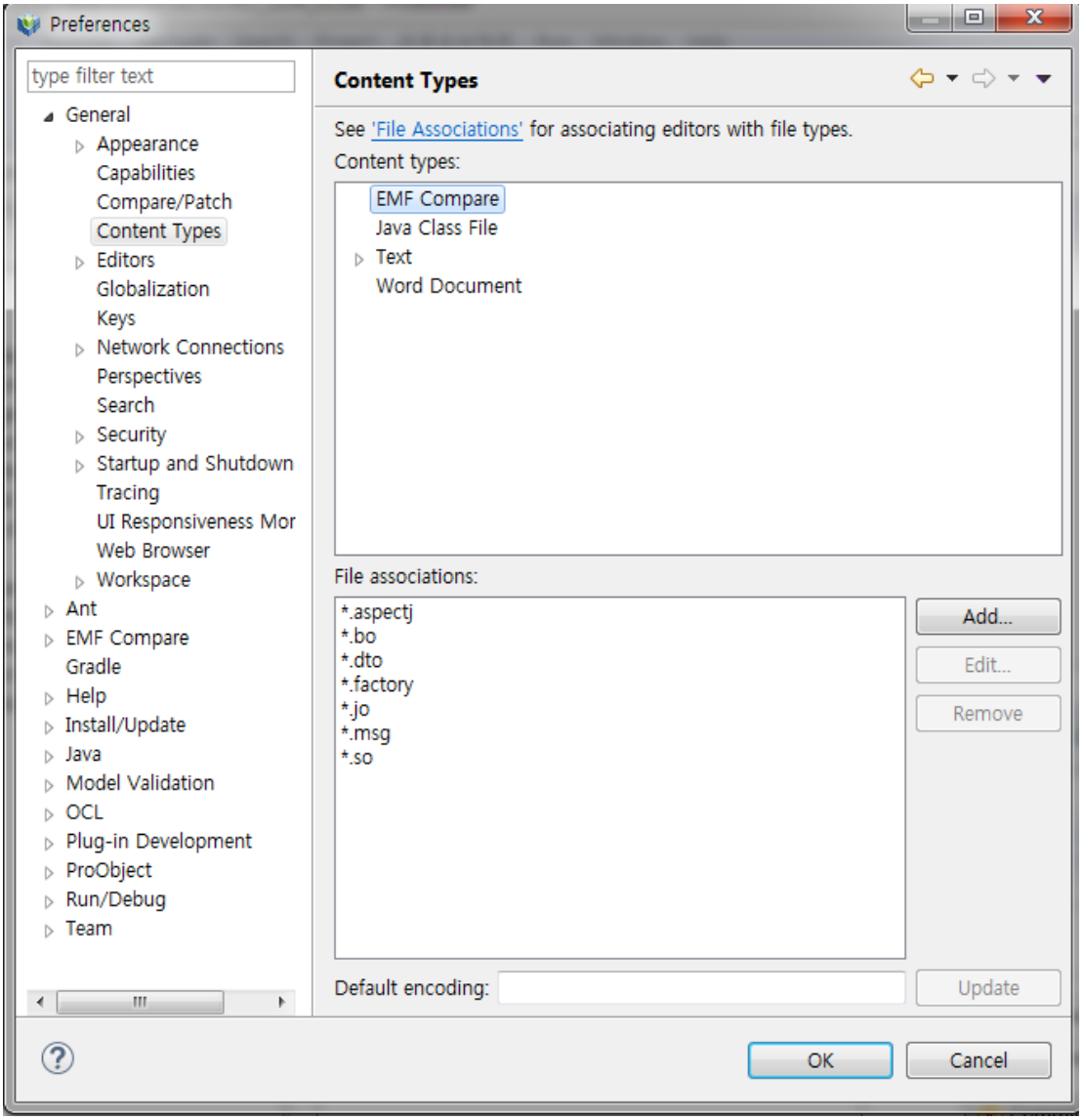
시스템 요구사항을 만족할 경우 ProStudio 구동을 위해 JDK를 설정한다. ProStudio 구동 후 DB Connection을 위한 JDBC 설정, ProStudio 메모리 사이즈 등을 설정하여 ProStudio가 무리없이 동작할 수 있는 환경을 구성한다. 시스템 요구사항은 ProObject 설치 안내서의 "시스템 요구사항"을 참고한다.

### 6.2.2. 환경설정

**[Window] > [Preferences]** 메뉴를 실행해서 Git 관련 환경설정을 한다. **Preferences 화면**에서 Git 클라이언트에 다음의 사항이 설정한다. 설정은 최초 한번만 설정하면 종료했다가 다시 실행해도 유지된다.

- **Content Type 추가**

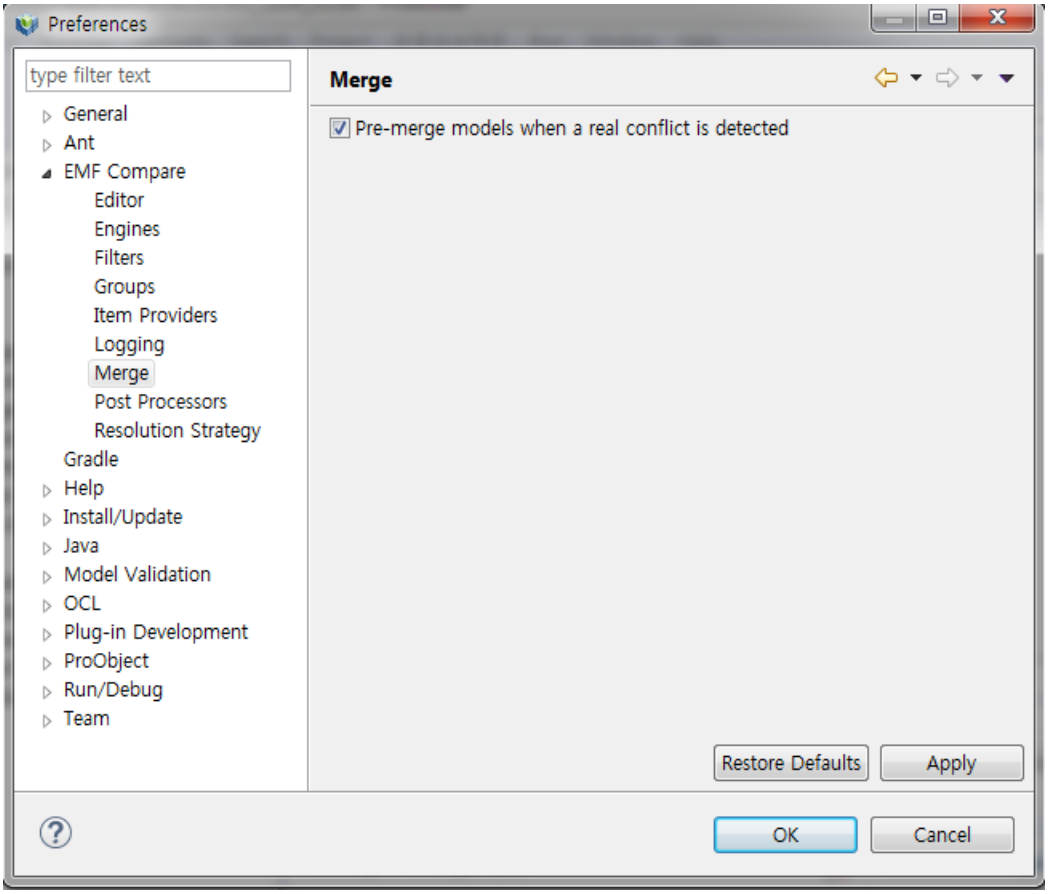
**[General] > [Content Types]**을 선택한 후 Content Type을 추가한다. **EMF Comapre**을 선택한 후 **File associations**에 다음과 같은 항목을 추가한다.



Git 연동 버전 초기 설정 - Content Type 추가

- **Merge 설정**

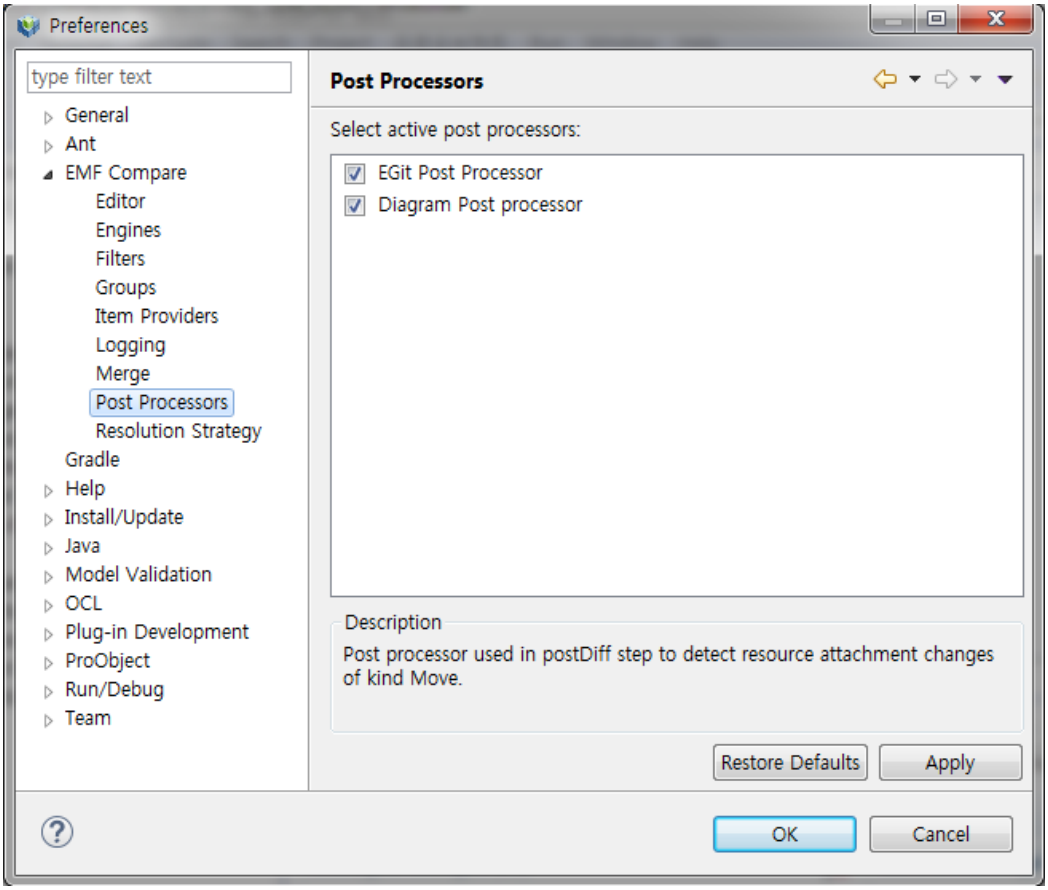
  **[EMF Compare] > [Merge]**을 선택한 후 Merge 옵션을 설정한다. Merge 과정에서 Conflict가 발생하는 경우 파일들의 Conflict나지 않은 부분들은 자동으로 병합하는 기능의 사용 여부를 설정한다.



Git 연동 버전 초기 설정 - Merge 설정

- **Post Processors 설정**

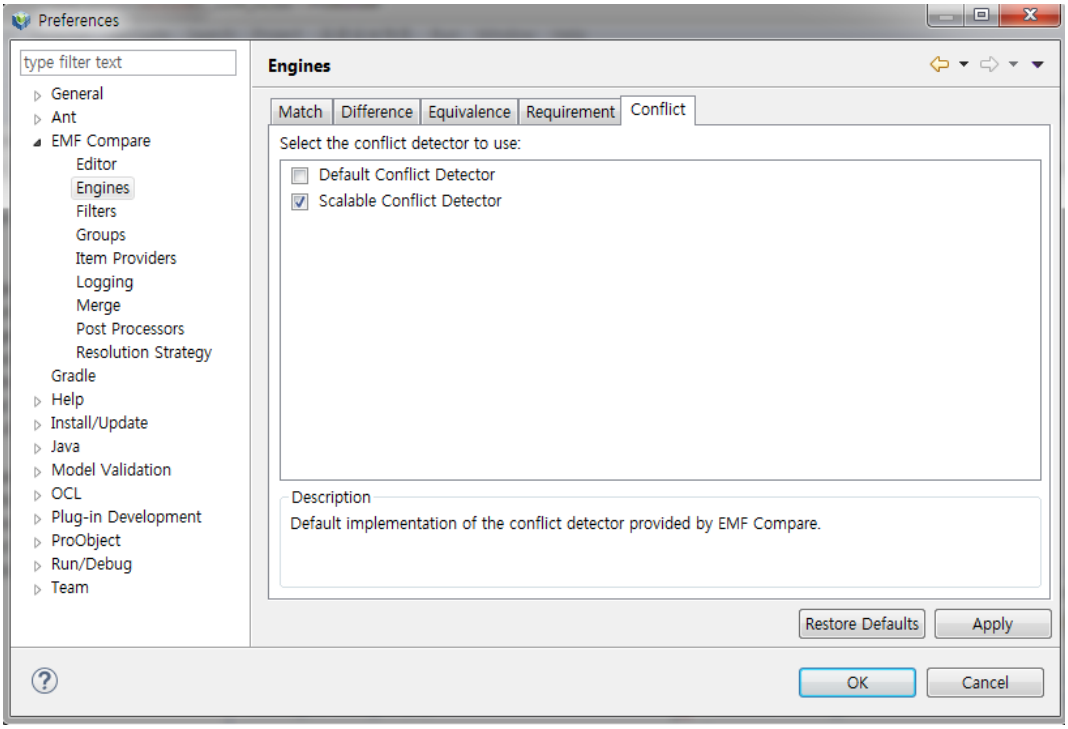
  **[EMF Compare] > [Post Processors]**을 선택한 후 **'EGit Post Processor'**와 **'Diagram Post Processor'** 옵션을 선택한다. 해당 옵션은 Conflict-Merge 과정에서 인덱스 추가까지의 과정을 쉽게 할 수 있도록 지원한다.



Git 연동 버전 초기 설정 - Post Processors 설정

- **Engines 설정**

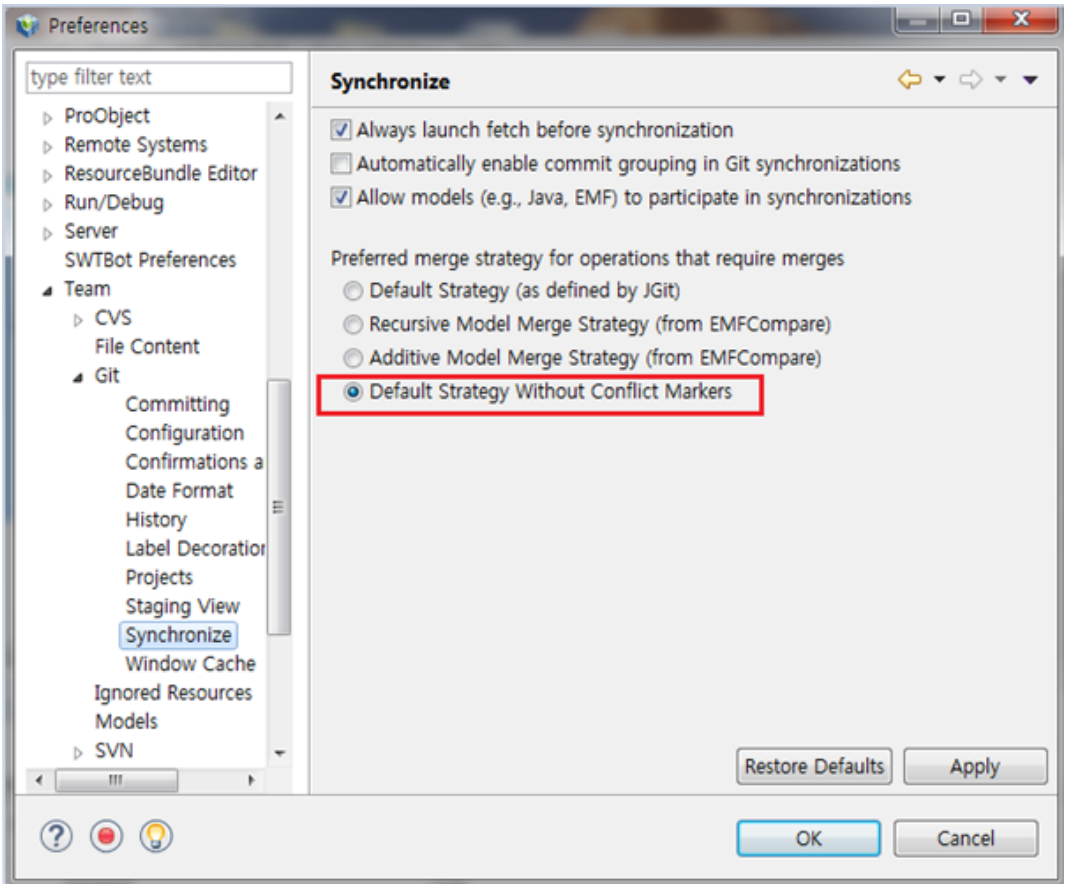
  **[EMF Compare] > [Engines]**을 선택한 후 **[Conflict]** 탭에서 **'Scalable Conflit Detector'** 체크박스를 선택한다.



Git 연동 버전 초기 설정 - Engines 설정

- **Git Synchronize 설정**

**[Team] > [Git]> [Synchronize]**을 선택한 후 옵션들을 다음과 같이 설정한다.



Git 연동 버전 초기 설정 - Git Synchronize 설정

**'Preferred merge strategy ...'** 항목이 Default Strategy로 설정되어 있을 경우 Conflict난 파일에 conflict marker가 표시되어 XML 파일을 에디터로 열 수 없는 문제가 발생하므로 반드시 확인한다.

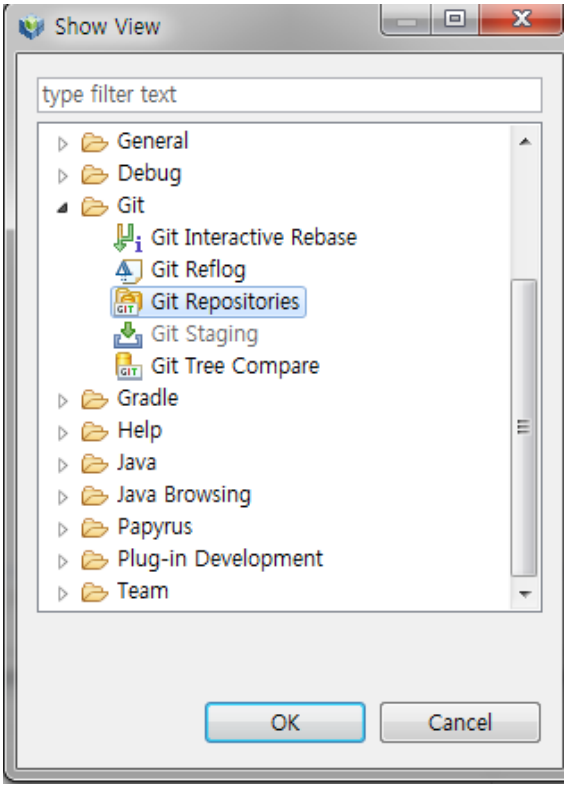
# 6.3. 연동 과정

본 절에서는 ProStudio와 Git 서버를 연동하는 과정에 대해서 설명한다.

## 6.3.1. 로컬에 Remote Git Repository 복사

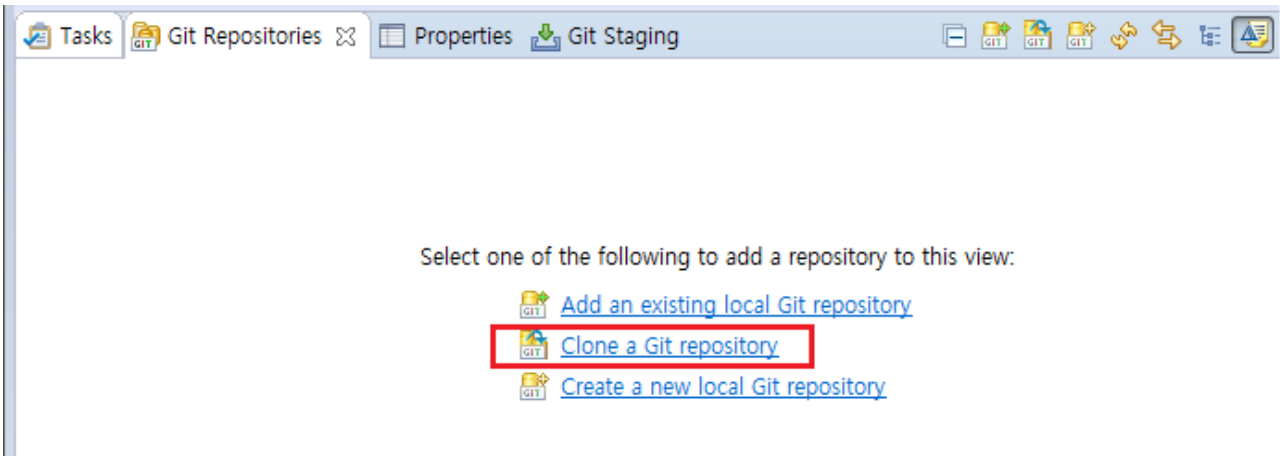
다음의 과정으로 로컬에 Remote Git Repository를 복사한다.

1. ProStudio의 메뉴에서 **[Window] > [Show View] > [Other]**를 선택하면 열리는 **Show View 화면**에서 **[Git] > [Git Repository]**를 선택한 후 **[OK]** 버튼을 클릭한다.



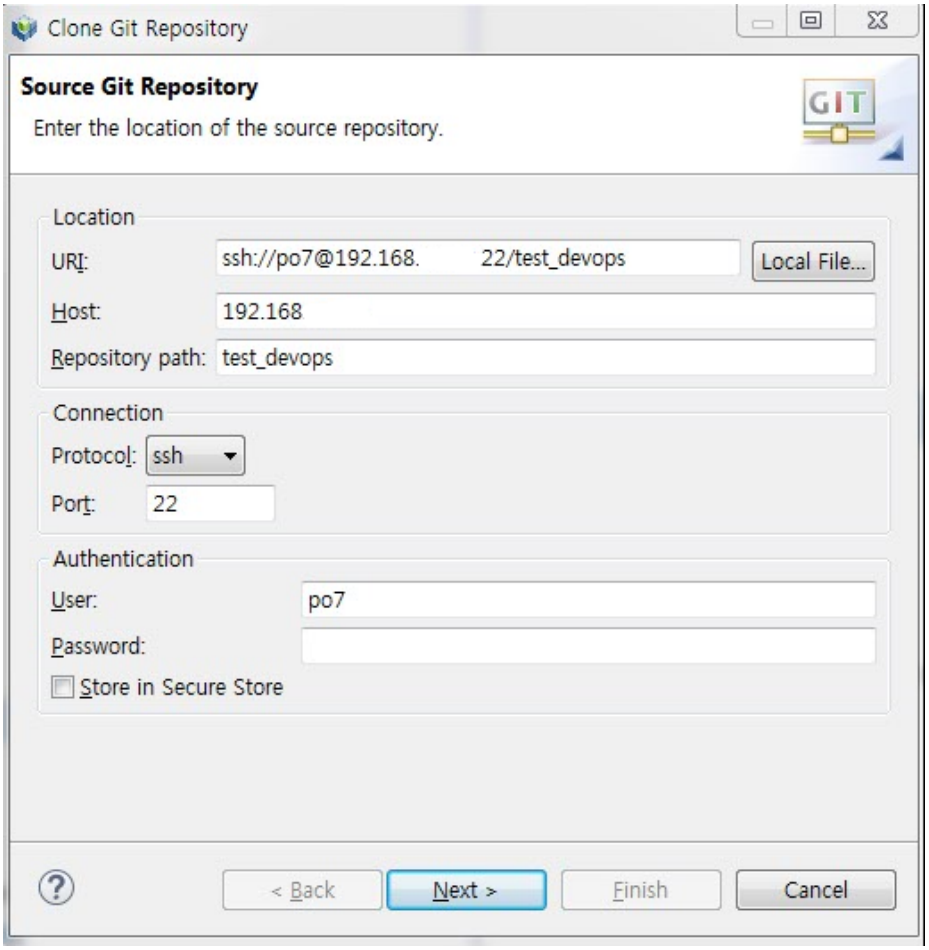

서버에 Git repository 복사 (1)

2. **Git Repository 설정 화면**에서 **'Clone a Git repository'**를 선택한다.




서버에 Git repository 복사 (2)

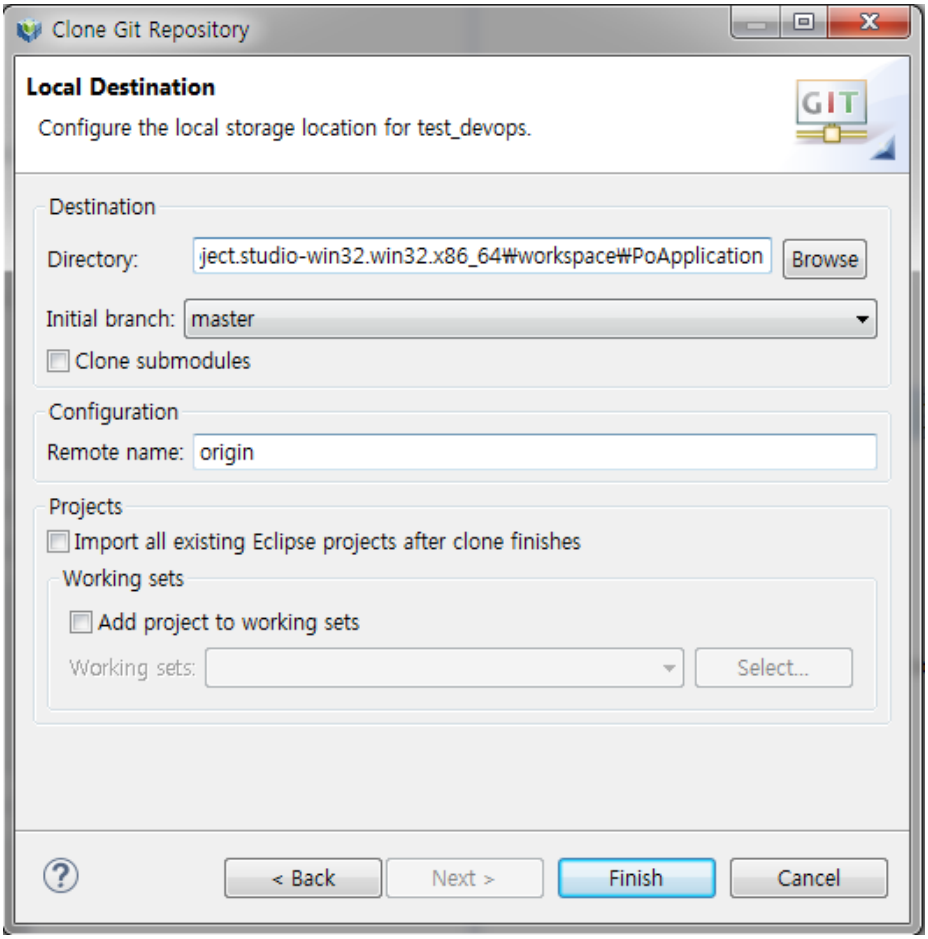
3. **Clone Git Repository 화면**에서 Git Repository 정보를 입력한 후 **[Next]** 버튼을 클릭한다. **'URI'** 항목은 입력하지 않는다.



Clone Git Repository (1)

SSH 프로토콜을 사용해서 Git 서버와 통신하는 경우에는 개인 Key가 서버에 등록되어 있어야 한다. **[Windows] > [Preferences]** 메뉴를 선택한 후 **Preferences 화면**에서 **[General] > [Network Connections] > [SSH2]** 설정에서 private keys에 client id key가 등록되어 있는지 확인할 수 있다.

4. **Clone Git Repository 화면**에서 Local Git 저장소 정보를 입력하고 **[Finish]** 버튼을 클릭하면 저장소가 생성된다. **'Directory'** 항목은 'ProStudio의 WorkSpace 폴더 \application명'으로 입력한다.

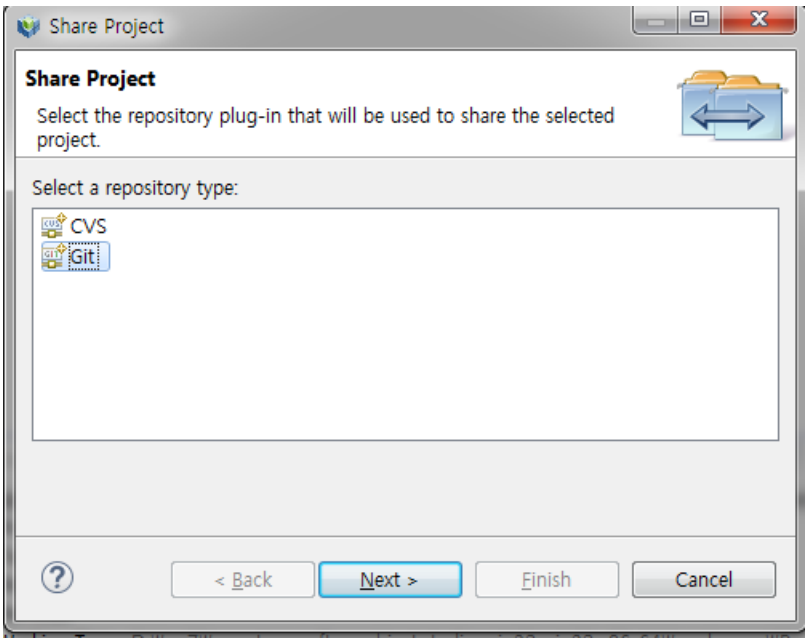



Clone Git Repository (2)

## 6.3.2. 프로젝트 Git 서버에 복사

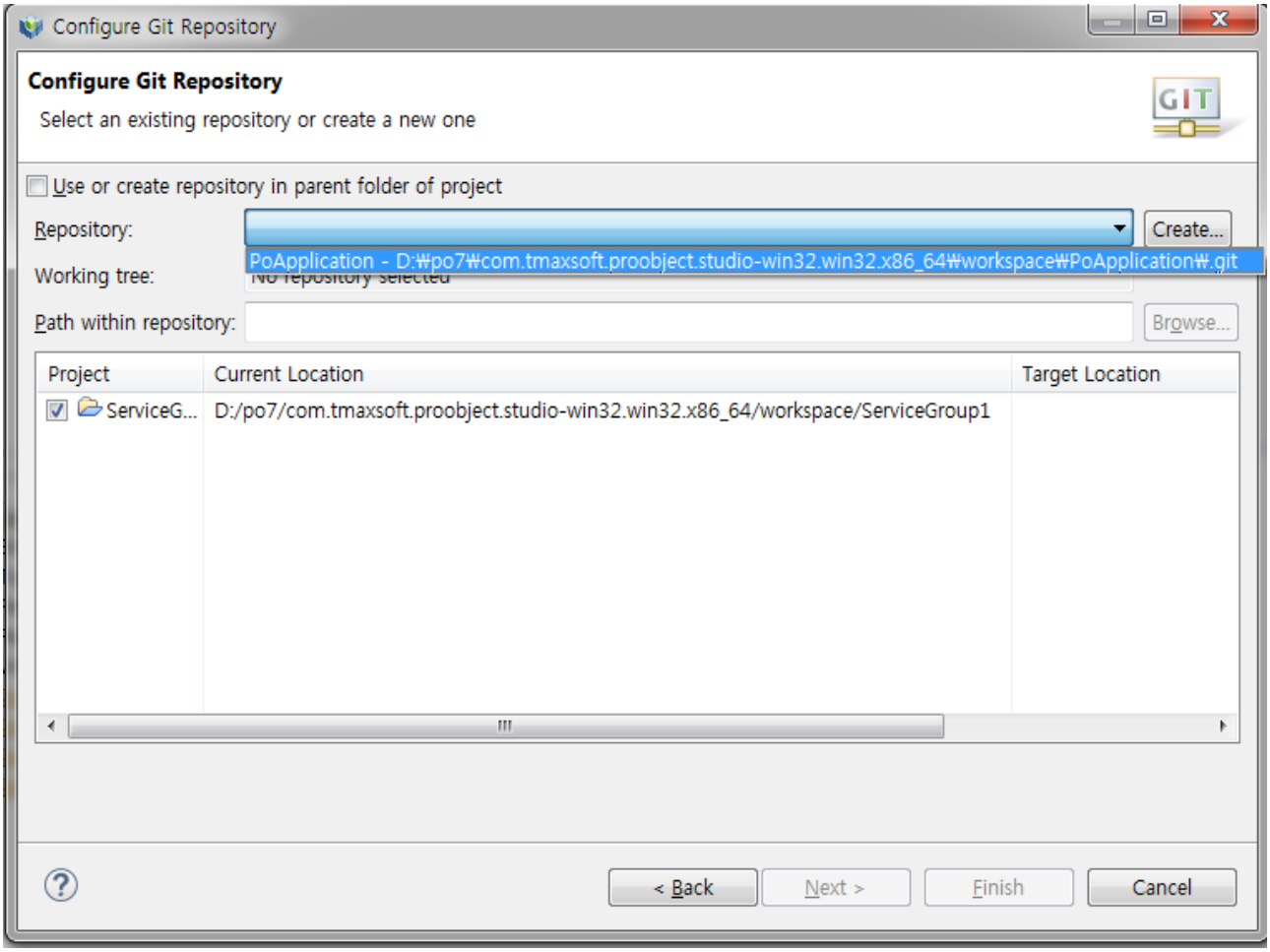
다음의 과정으로 새로 생성한 프로젝트를 Git 서버에 복사한다.

1. 새로운 ProObject 프로젝트를 생성한 후 프로젝트의 컨텍스트 메뉴에서 **[Team] > [Share Project..]**를 선택한 후 **Share Project 화면**에서 **[Git]** 메뉴를 선택한 후 **[Next]** 버튼을 클릭한다.




프로젝트 Git 서버에 복사 (1)

2. **Configure Git Repository 화면**에서 Repository에 clone한 Git repository를 선택하고 **[Finish]** 버튼을 클릭한다.



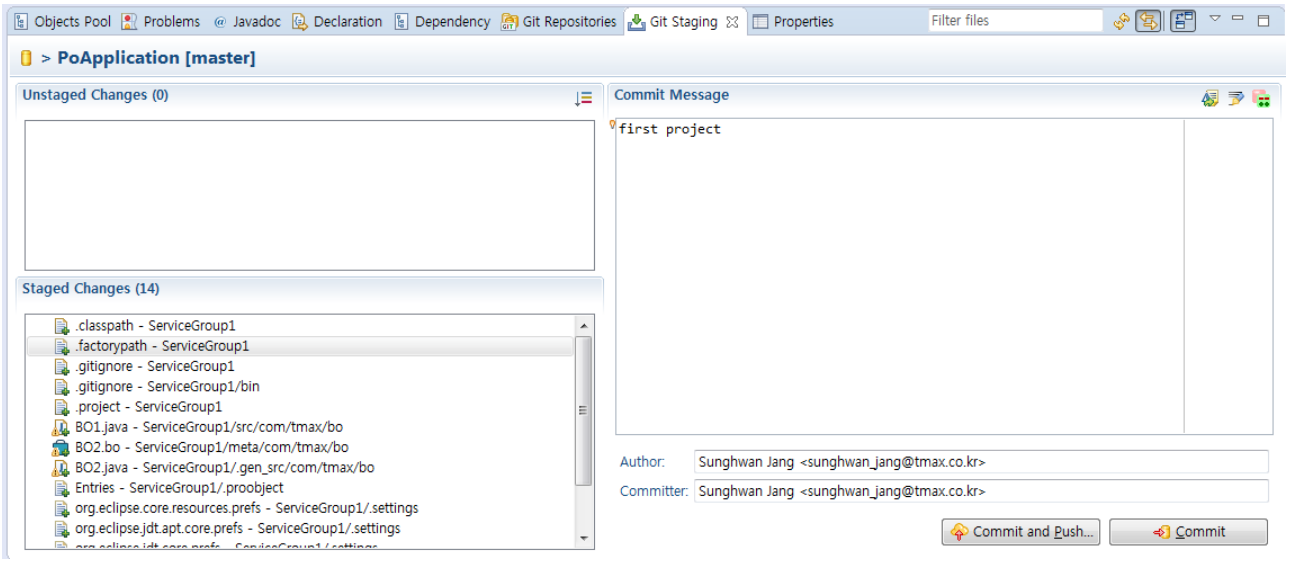

프로젝트 Git 서버에 복사 (2)

3. Git Repository에 프로젝트를 추가한 후 프로젝트의 컨텍스트 메뉴에서 **[Team] > [Add to Index]**를 선택하면 Git이 프로젝트의 변경을 추적할 수 있다.

   Git 추적을 하지 않아도 되는 폴더의 경우 추적에서 제외할 수 있다. 자세한 내용은 "Git 추적 제외"를 참고한다.

4. 등록한 정보는 ProStudio 메뉴의 **[Window] > [Show View] > [Other...]**를 선택한 후 **Show View 화면**에서 **[Git] > [Git Staging]** 메뉴를 선택하면 조회할 수 있다. **Git Staging View 화면**의 Staged Changes에서 Git이 추적하는 파일을 조회할 수 있다.

   또한 **Commit Messgae**를 입력하고 **[Commit and Push]** 버튼을 클릭하면 로컬에서 변경한 내용들이 Git 서버에 반영된다.

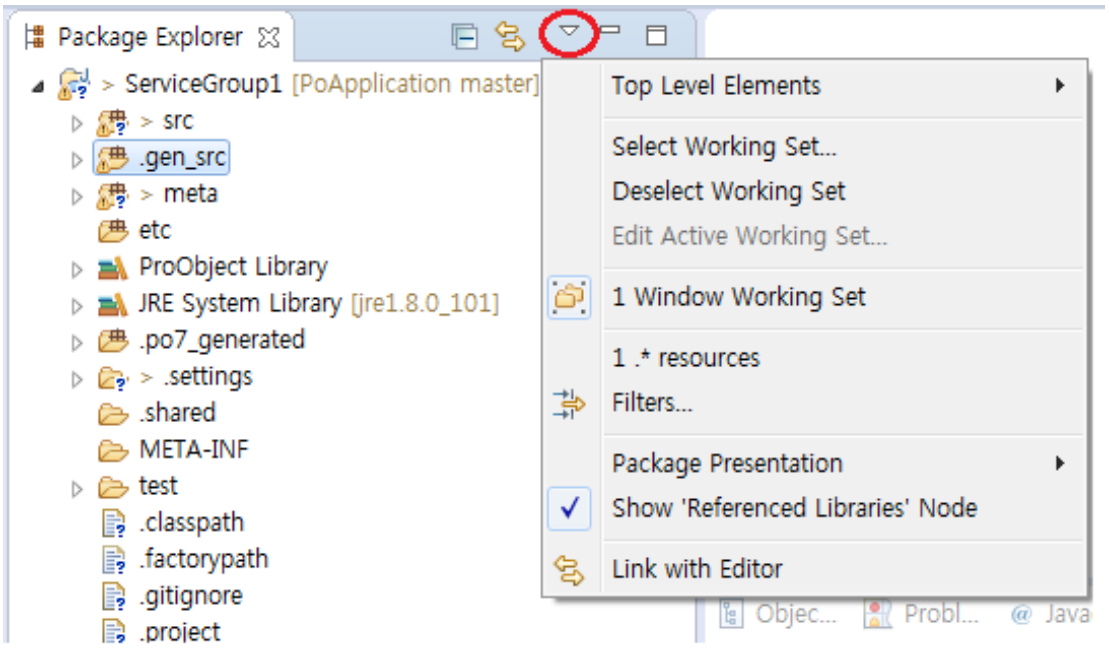



프로젝트 Git 서버에 복사 (3)

## Git 추적 제외

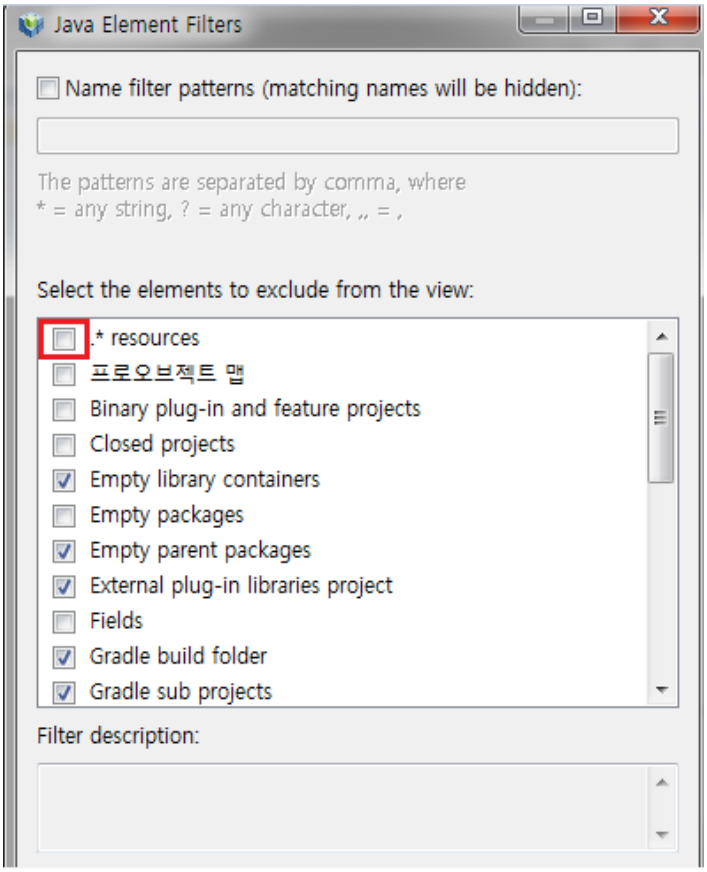
Git이 변경 추적을 하지 않아도 되는 폴더의 경우 다음의 과정으로 예외 처리를 할 수 있다. 다음은 .gen_src 폴더를 추적에서 제외하는 과정에 대한 설명이다.

1. 프로젝트 폴더 중에 추적하지 않아도 되는 폴더를 선택한 후 **Package Exploer**에서 **[▽]**(Filters) 버튼을 클릭한다.




Git 추적 제외 (1)

2. **Java Element Filters 화면**에서 필터 조건을 설정한다. 예제에서는 **'.resources'**를 체크박스를 해제한다.




Git 추적 제외 (2)

3. **Package Exploer**의 프로젝트 목록에서 .gen_src 폴더를 선택한 후 컨텍스트 메뉴에서 **[Team] > [Ignore]**를 선택한다. 그러면 .gen_src 폴더는 추적에서 제외된다.

   참고로 그 외 Git 추적에서 제외해야 하는 파일 및 폴더들 목록은 다음과 같다.

   - .gradle
   - .proobject/Entries
   - .settings
   - bin
   - .shared
   - download
   - etc
   - build.gradle
   - gradle.properties

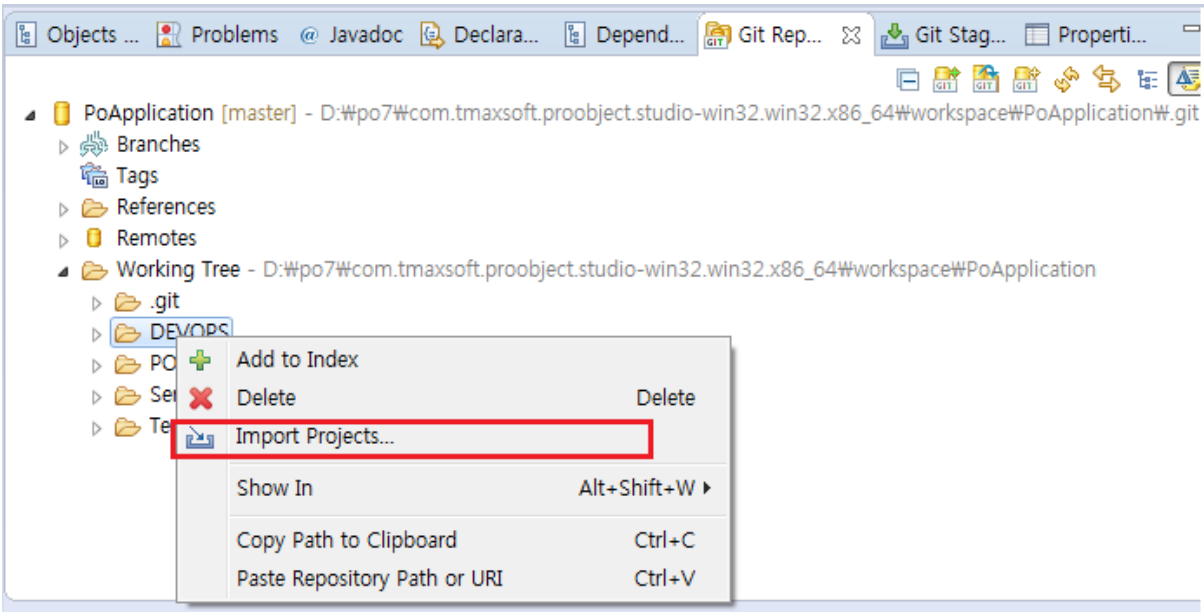
   반대로 최초 프로젝트 생성자가 Git에 push해서 관리해야 한다. 프로젝트 하위의 다음의 파일은 git에 push해서 사용한다.

   - .proobject/Repository
   - .gitignore
   - .project

### 6.3.3. Git 서버로부터 프로젝트 import

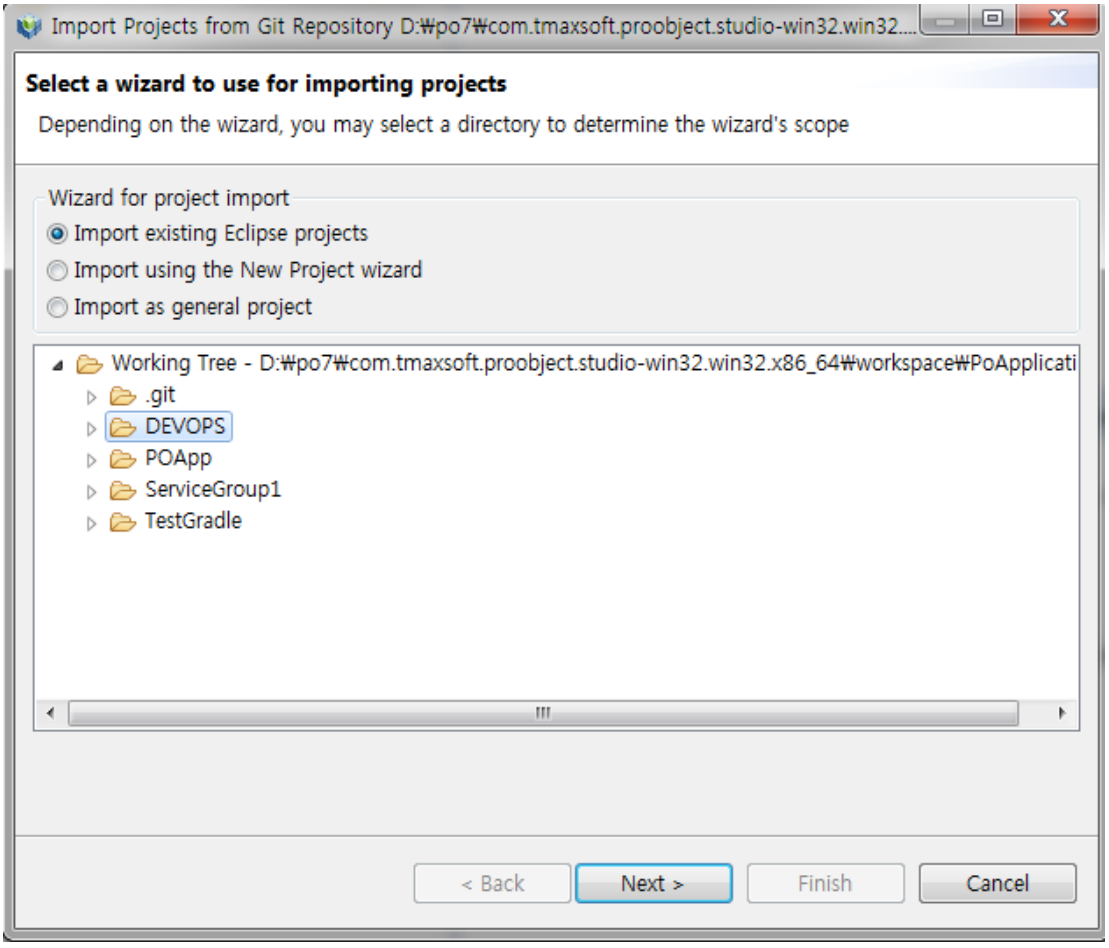
다음은 Git 서버에 복사한 프로젝트를 Import하는 과정에 대한 설명이다. Git 서버에 프로젝트를 복사하는 방법은 로컬에 Remote Git Repository 복사를 참고한다.

1. Git repository Working Tree에서 Import할 폴더를 선택한 후 컨텍스트 메뉴에서 **[Import Projects]**를 선택한다.




Git 서버로부터 프로젝트 import (1)

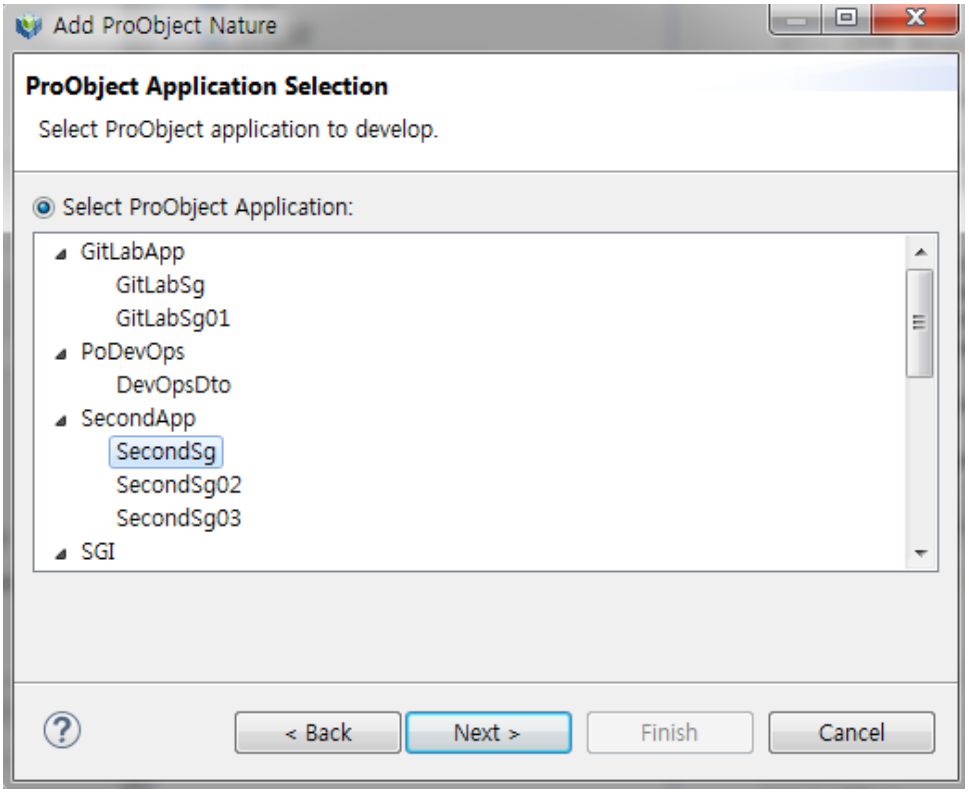
2. 화면에서 Import할 프로젝트를 선택하고 **[Next]**을 클릭한다.



Git 서버로부터 프로젝트 import (2)

3. Git으로부터 최초로 import된 프로젝트는 ProObject 프로젝트가 되기 위한 속성이 없으므로 에러가 발생한다. 따라서 ProObject 속성을 추가해야 한다.

   프로젝트 컨텍스트 메뉴에서 **[Configure] > [Add ProObject Nature]**를 선택한 후 애플리케이션과 서비스 그룹을 선택한 후 **[Finish]** 버튼을 클릭한다.



Add ProObject Nature 화면

# 6.4. 부가 기능

본 절에서는 Git에 커밋된 리소스들을 조회하는 방법과 변경 내역을 서버로 저장하는 Push 방법, 소스 내역을 병합하는 방법, 리소스의 이력들을 비교하는 방법에 대해서 설명한다.

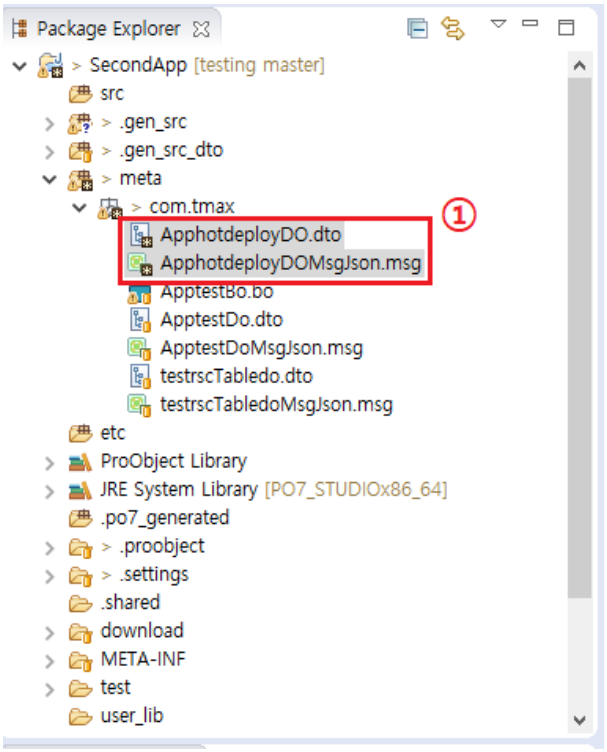
## 6.4.1. Push

ProStudio 내에서의 변경 내용을 서버에 저장하는 Push할 수 있고 Push 성공 또는 실패(Confilct)된 경우 작업에 대한 결과를 확인할 수 있다. 현재의 변경 내용은 아직 로컬 저장소에 커밋되어 HEAD 안에 머물고 있다면 원격 서버로 올리는 Push 작업을 해야 한다.

다음의 3가지 방법으로 ProStudio에서 소스나 리소스를 Push할 수 있다.

- **① Package Explorer 영역**

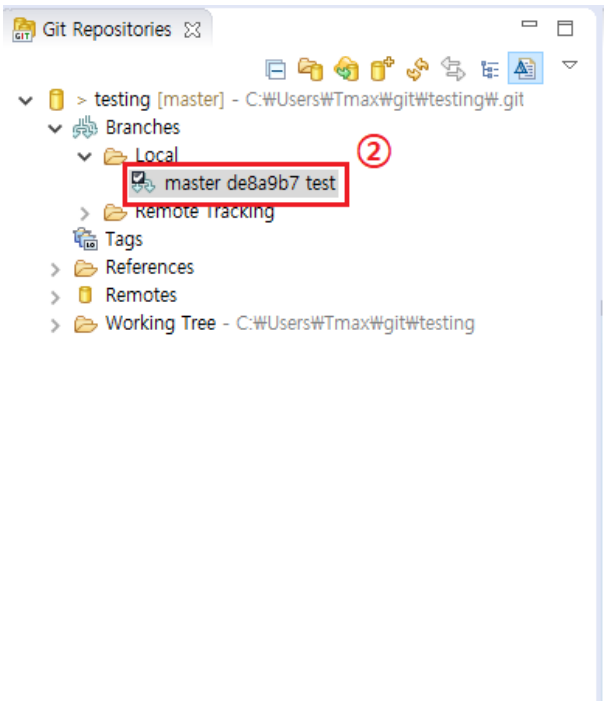
  **Package Explorer 영역**에서 Push할 파일을 선택 후 컨텍스트 메뉴에서 **[Team] > [Push to Upsteam]**를 선택한다.




ProStudio 내에서 Git Push 방법 (1)

- **② Git Repositiries 영역**

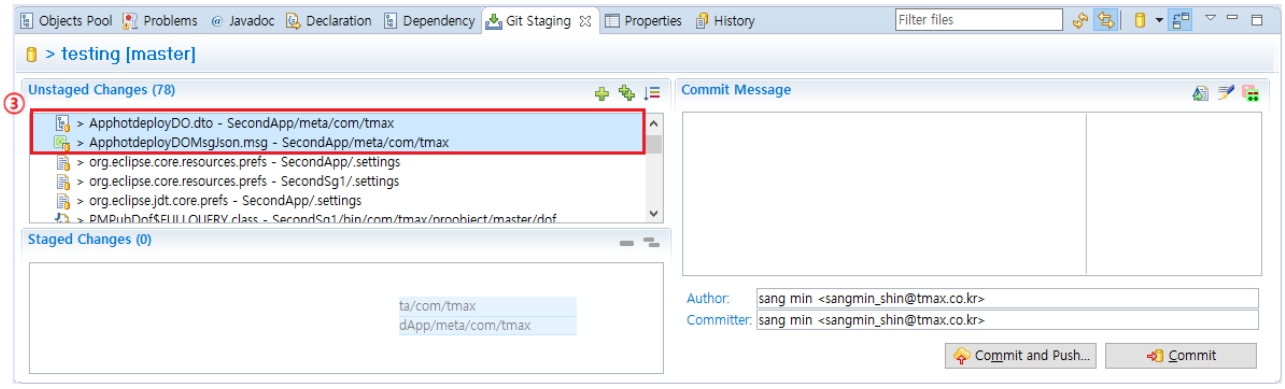
  **Git Repositiries 영역**에서 Push할 Local Branch 선택한 후 컨텍스트 메뉴에서 **[Push to branch]**를 선택한다.




ProStudio 내에서 Git Push 방법 (2)

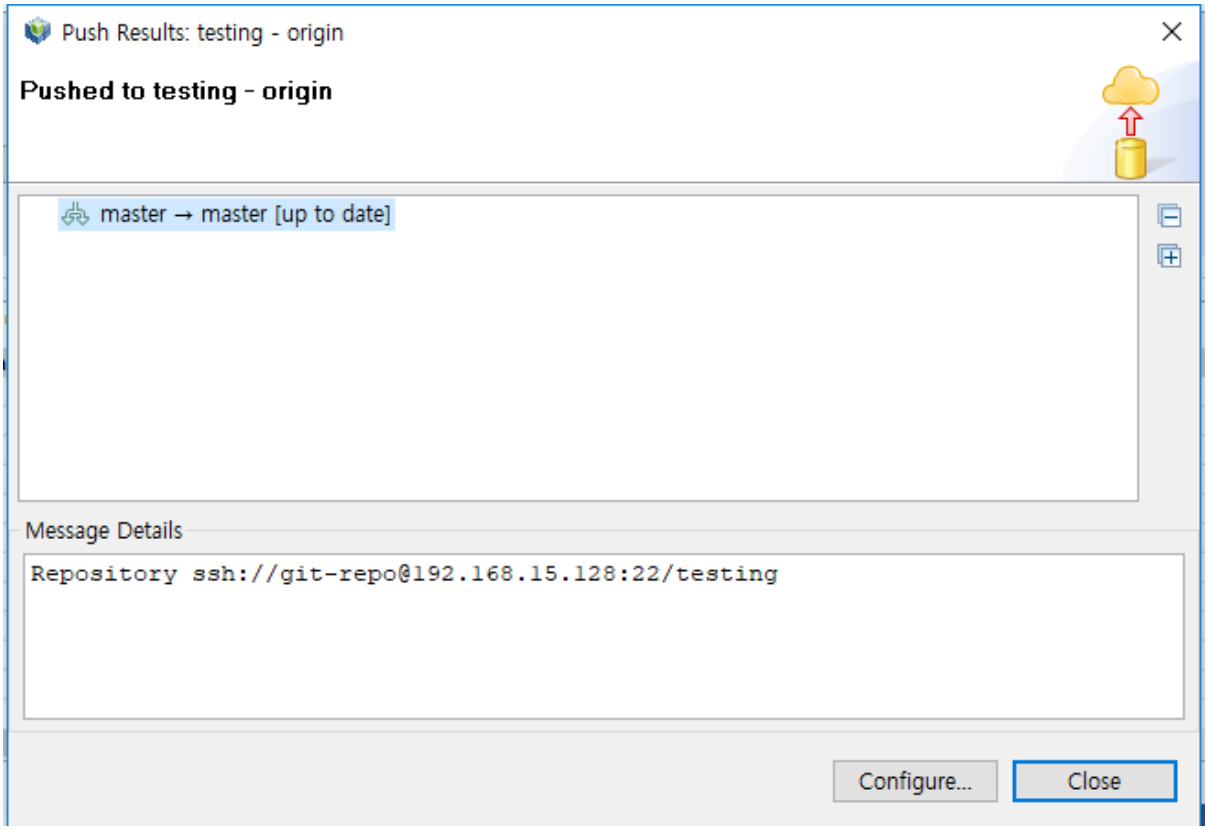
- **③ Git Staging 영역**

  **[Git Staging]** 탭의 **Unstaged Changes**에서 Push할 파일을 찾아 **Staged Changes**로 드래그 앤드 드롭한 후 메시지를 작성한 후 **[Commit and Push]** 버튼 클릭한다.



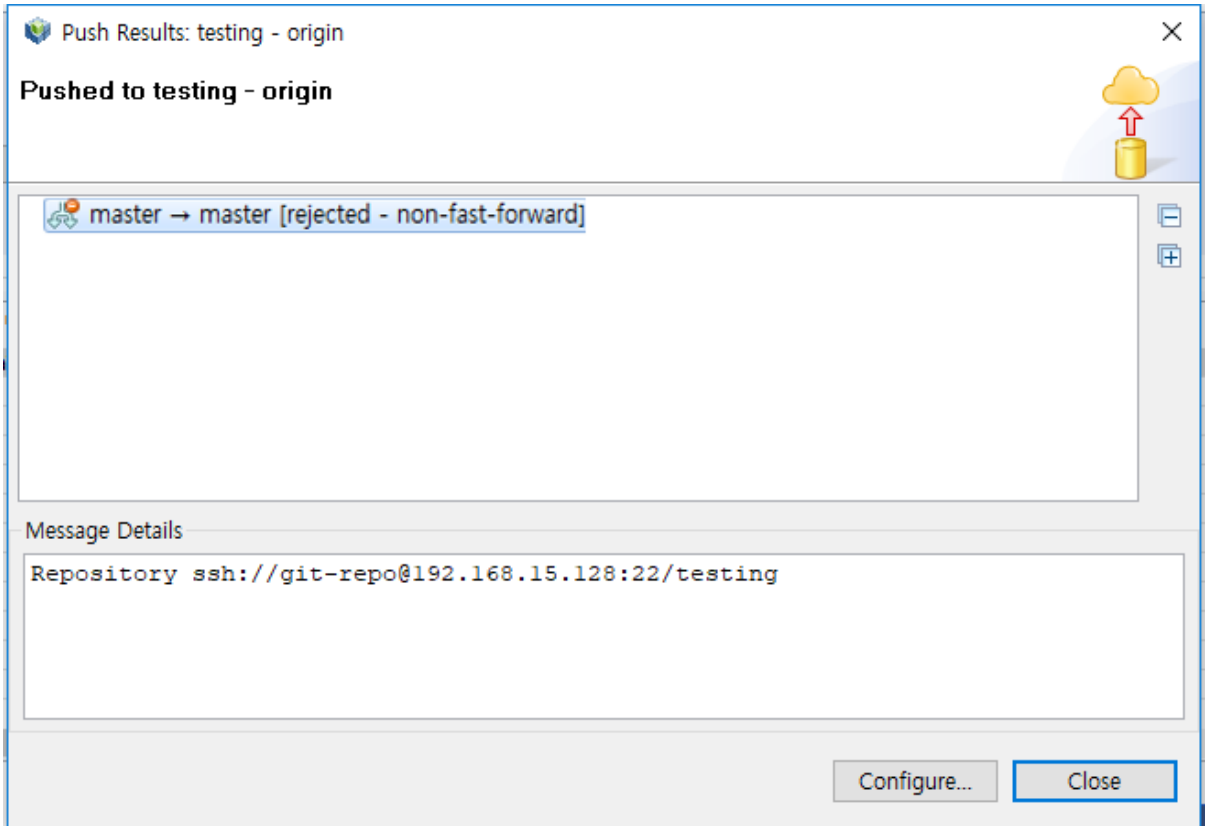

ProStudio 내에서 Git Push 방법 (3)

충돌이 없거나 Push가 정상적으로 될 경우 Pushed 메시지를 확인할 수 있다.




Git Push 성공 메시지

기존에 있던 Git 서버의 리소스를 최신으로 Pull을 하지 않고 변경을 하고 Push하는 경우 Conflict(충돌)이 발생해서 rejected 메시지가 화면에 표시된다. 이 경우의 다른 사람의 수정 내역과 자신의 수정 내역을 비교해서 소스를 통합해야 한다. 자세한 내용은 Conflict된 소스 통합을 참고한다.



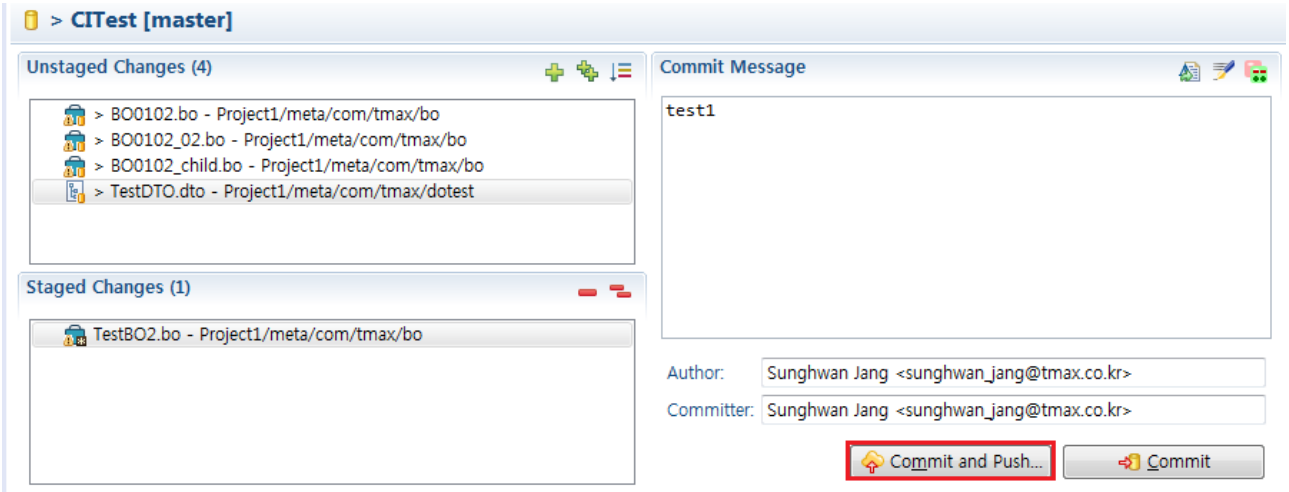

Git Push 실패 메시지

## 6.4.2. Conflict된 소스 통합

동일한 소스를 여러 개발자가 수정한 경우 각 개발자가 수정한 내역이 Conflict하는 경우가 발생한다. 소스의 수정 내역이 Conflict된 경우 다른 사람의 수정 내역과 자신의 수정 내역을 비교해서 소스를 통합해야 한다.

다음은 소스에 Conflict이 발생한 경우 Conflict 발생한 부분을 비교하고 merge하는 방법에 대한 설명이다.

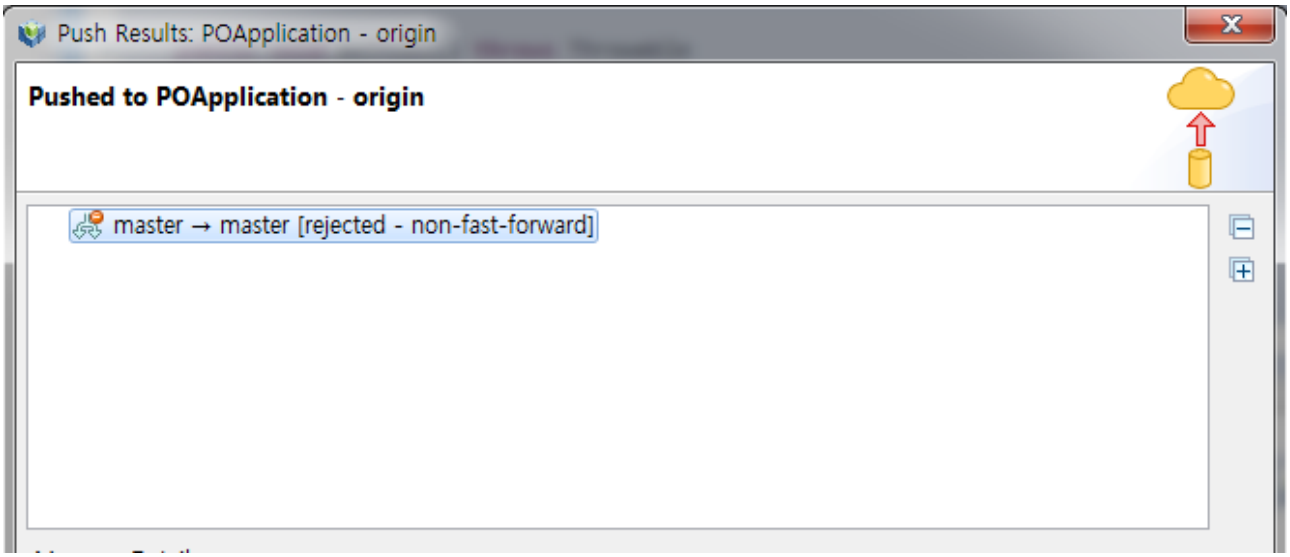
1. 테스트할 A WorkSpace와 연결된 Git 클라이언트가 아닌 다른 Git 클라이언트에 연결된 B WorkSpace에서 비즈니스 오브젝트를 변경한 후에 컨텍스트 메뉴에서 **[Team] > [Commit]**을 선택한다.
2. **Git Staging View 화면**에서 커밋할 내용을 확인하고 **[Commit and Push]** 버튼을 클릭한다.




Git push - 소스 Conflict 발생 Merge 방법 (1)

3. A WorkSpace에서도 비즈니스 오브젝트를 변경한 후 Git 서버에 커밋한다(위의 1, 2번 과정 참고).

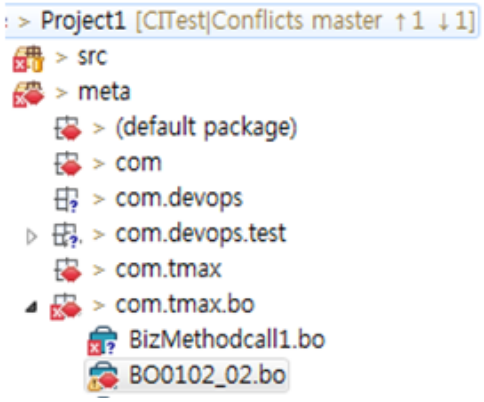
다음은 **Git 서버에서 Pull을 하지 않은 경우**의 예로 pull을 받을 파일이 있는 상황에서 push를 하면 push가 되지 않으므로 **Package Exploer**에서 프로젝트의 컨텍스트 메뉴에서 **[Team] > [Pull]**을 선택해서 Pull을 먼저 수행한다.




Git push - 소스 Conflict 발생 Merge 방법 (2)

4. 서로 다른 Git 클라이언트에서 동시에 변경된 파일이 있는 경우 다음과 같이 Pull 과정에서 에러가 발생한다.

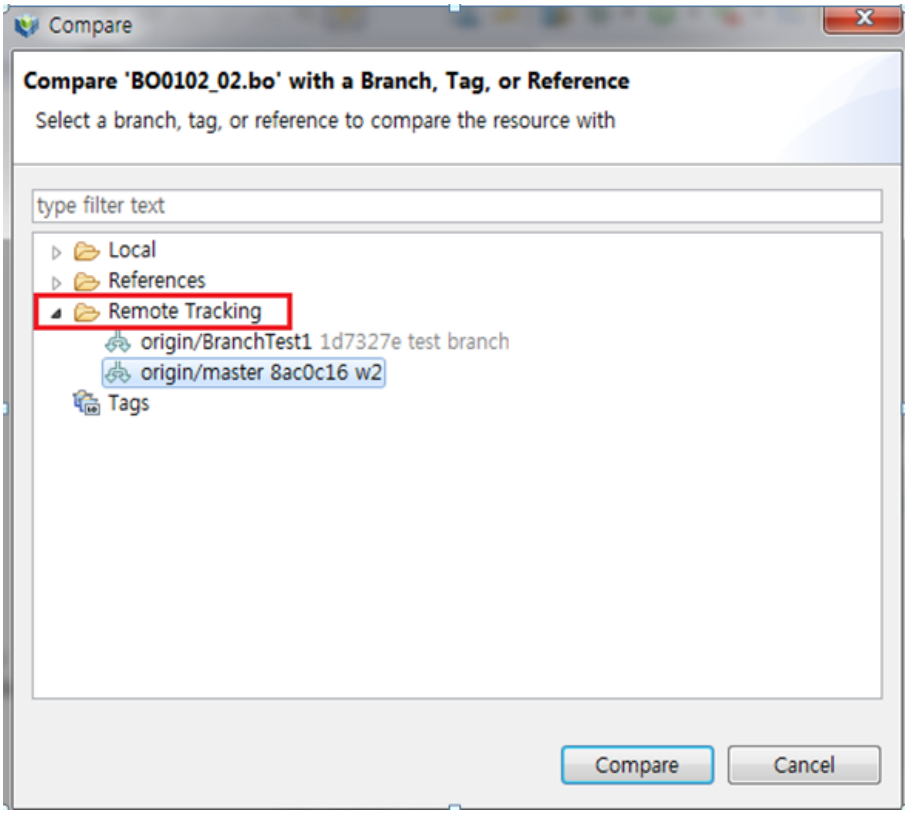
   다음은 **Conflict(충돌)이 발생하여 에러가 발생하는 경우**의 예로 Conflict 상태의 리소스는 **Package Exploer** 또는 **Synchronize View**에서 확인이 가능하다.




Git push - 소스 Conflict 발생 Merge 방법 (3)

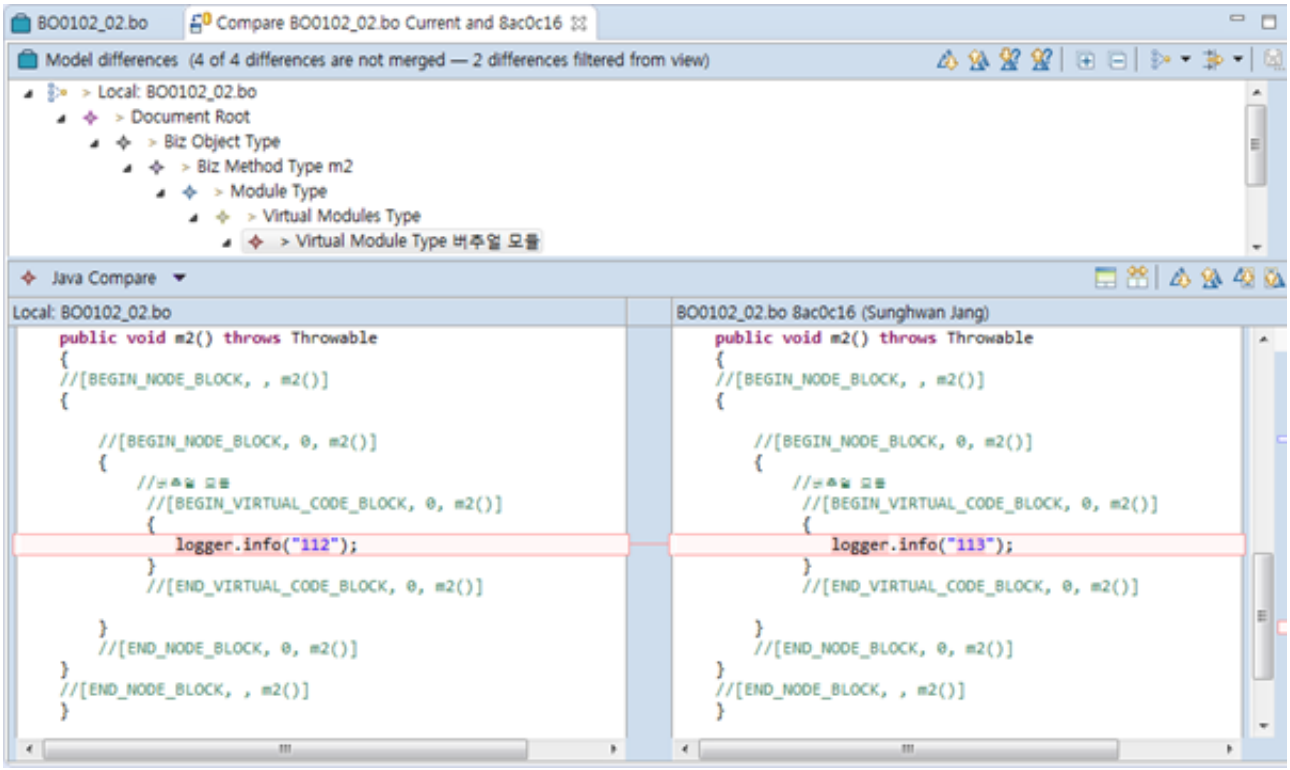
5. **Package explorer**에서 Conflict 상태의 리소스를 선택한 후 컨텍스트 메뉴에서 **[Branch, Tag, or Reference] > [Compare With]**를 클릭한다.

   **Compare 화면**의 메뉴에서 **[Remote Tracking]**을 선택하면 서버의 리소스와 비교할 수 있다.



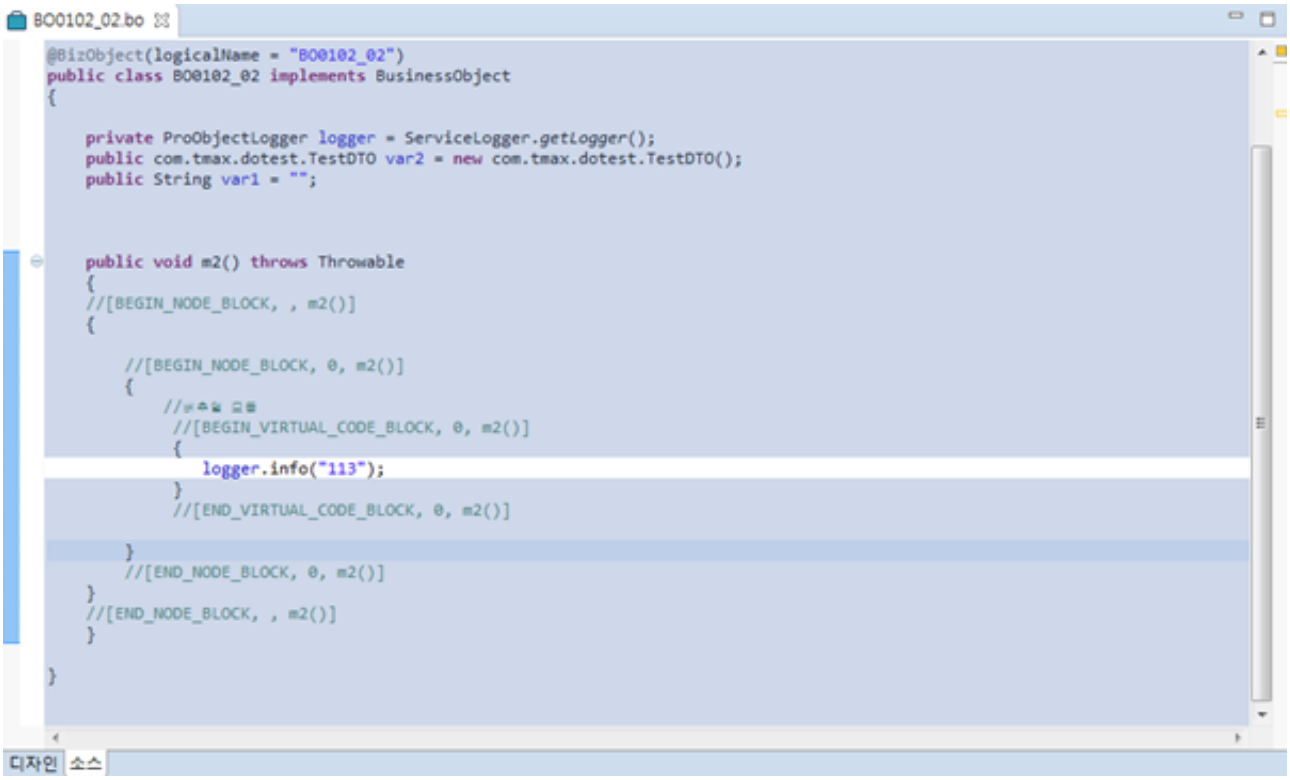

Git push - 소스 Conflict 발생 Merge 방법 (4)

6. 서버의 리소스와 Java 소스 기준으로 비교된 상태이다.



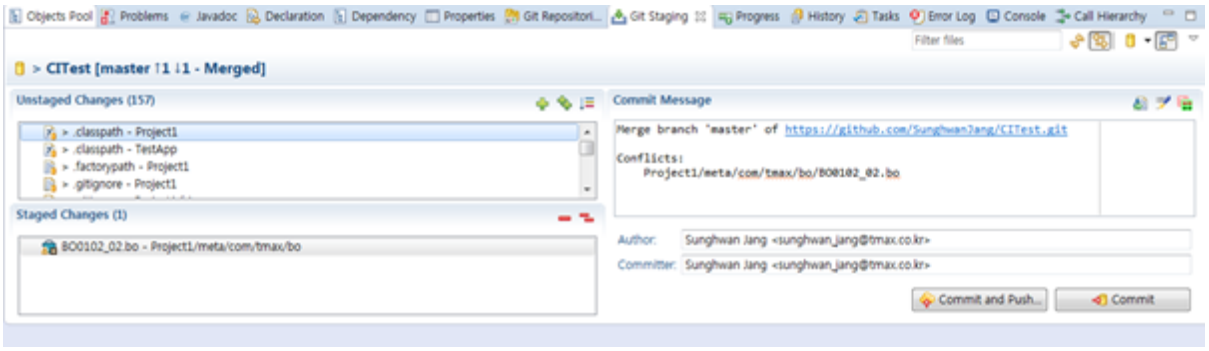

Git push - 소스 Conflict 발생 Merge 방법 (5)

비교한 소스의 내용 중에 Conflict된 부분을 에디터에서 수정하고 저장한다.



Git push - 소스 Conflict 발생 Merge 방법 (6)

7. Conflict이 발생된 부분을 수정한 후 파일을 **[Commit and Push]** 버튼을 클릭해서 서버에 Push한다.



Git push - 소스 Conflict 발생 Merge 방법 (7)

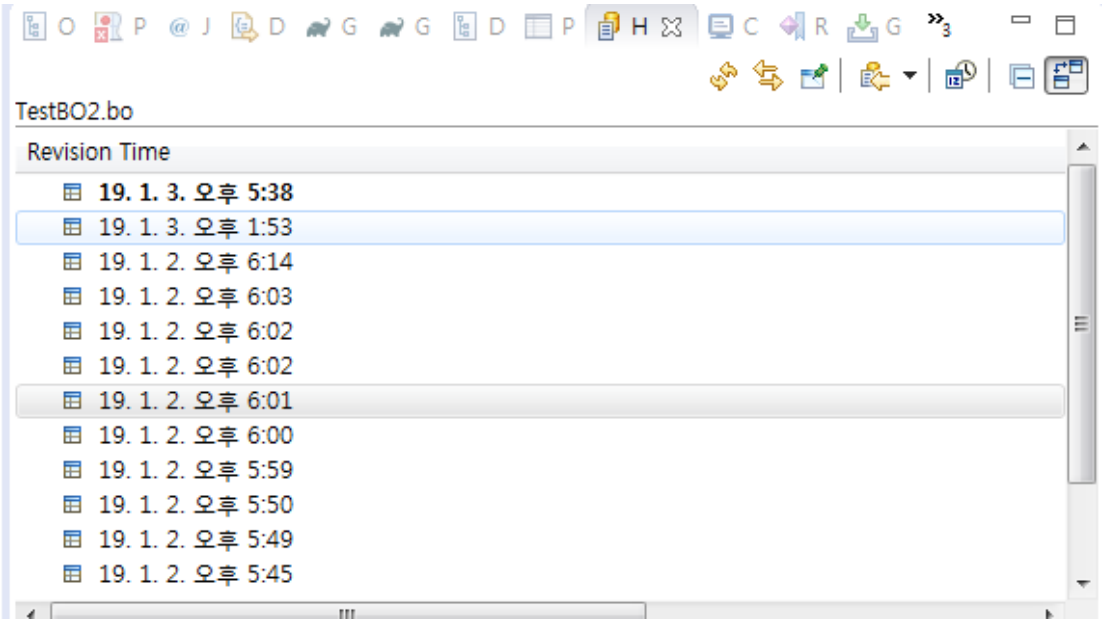
## 6.4.3. 이력 조회

**Package Exploer**에서 파일을 선택한 후 컨텍스트 메뉴에서 **[Compare With] > [Local history]**에서 자신이 이전에 수정한 버전과 비교할 수도 있고 Git과 연계하여 서버 리소스의 변경된 부분을 비교할 수 있다.

### 6.4.3.1. Local History와 현재 WorkSpace 파일 비교

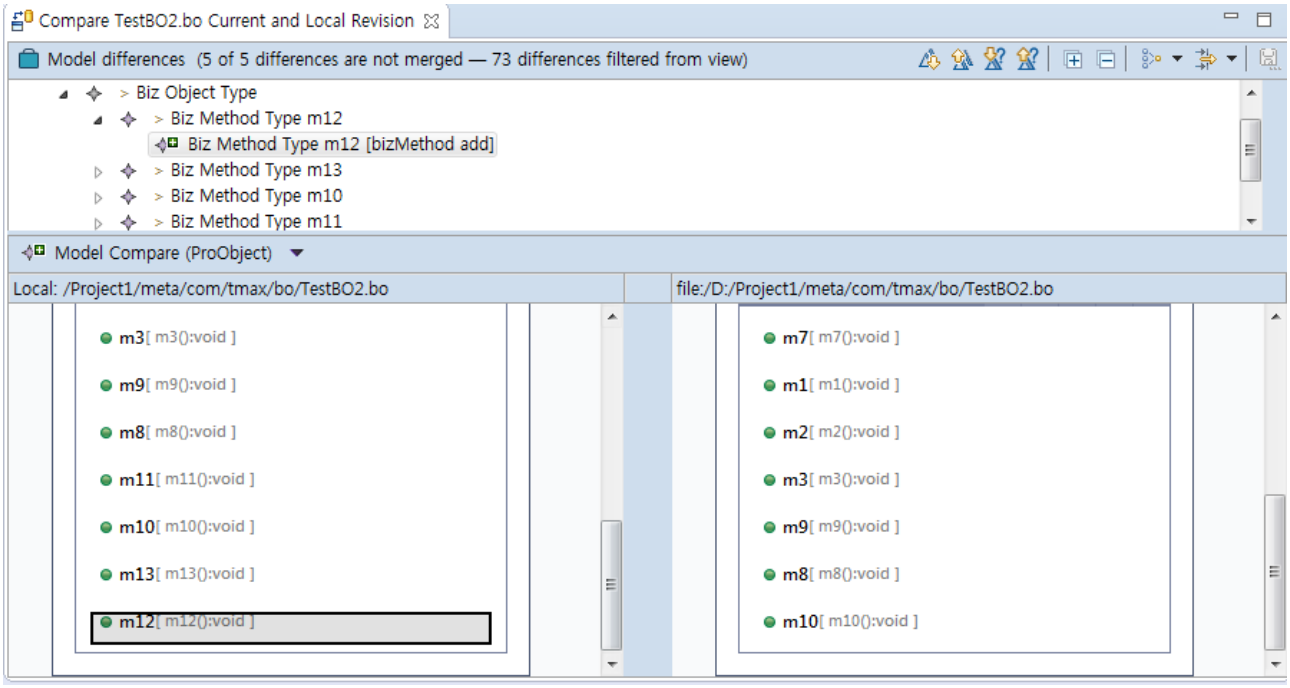
다음은 Local History와 현재 WorkSpace 파일을 비교하는 방법이다.

1. **Package Exploer**에서 파일을 선택한 후 컨텍스트 메뉴에서 **[Compare With] > [Local History]**를 선택한 후 Local History 목록에서 비교할 파일을 더블클릭한다.




Local History 소스 비교 (1)

2. 다음 화면에서 현재 Local의 소스와 특정 commit된 버전을 비교할 수 있다.

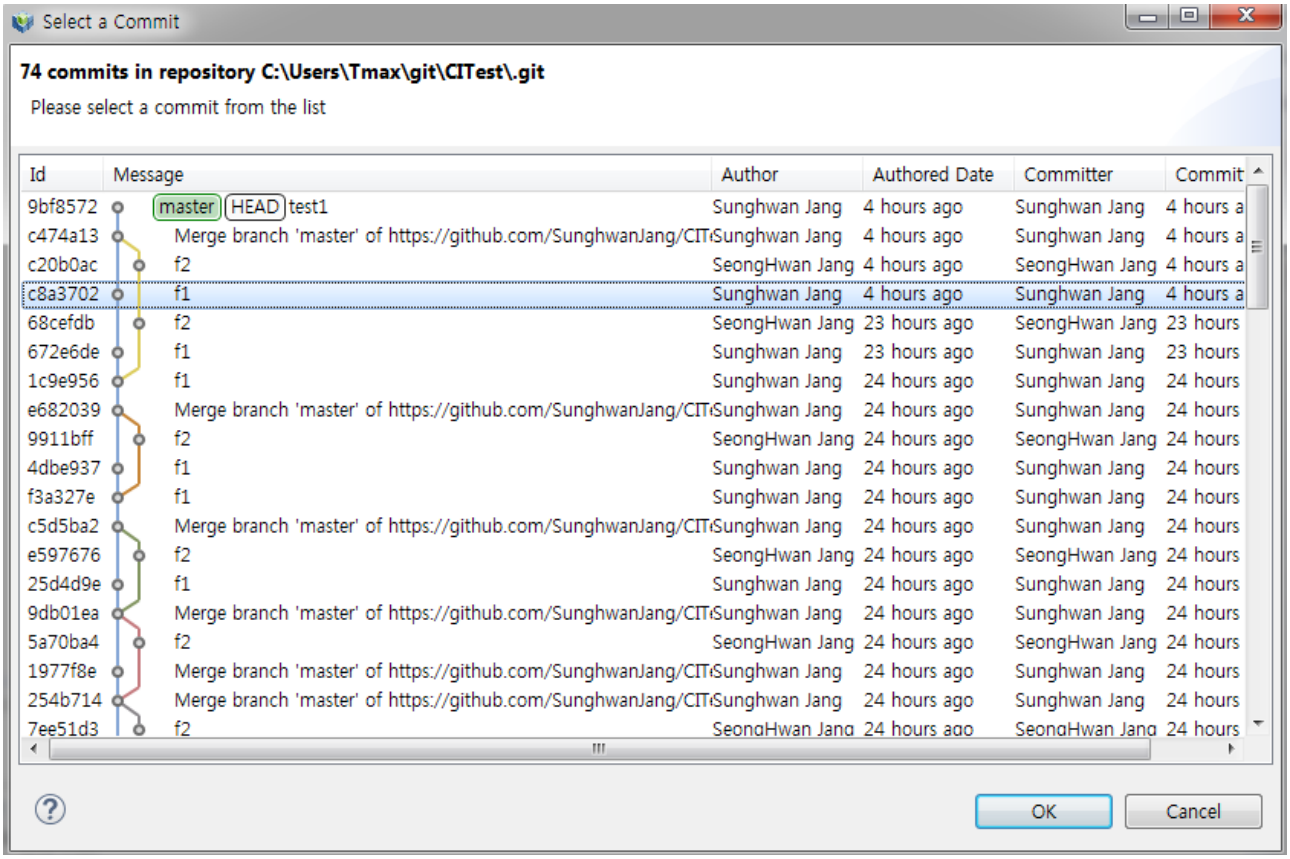



Local History 소스 비교 (2)

### 6.4.3.2. 커밋된 소스와 현재 WorkSpace 파일 비교

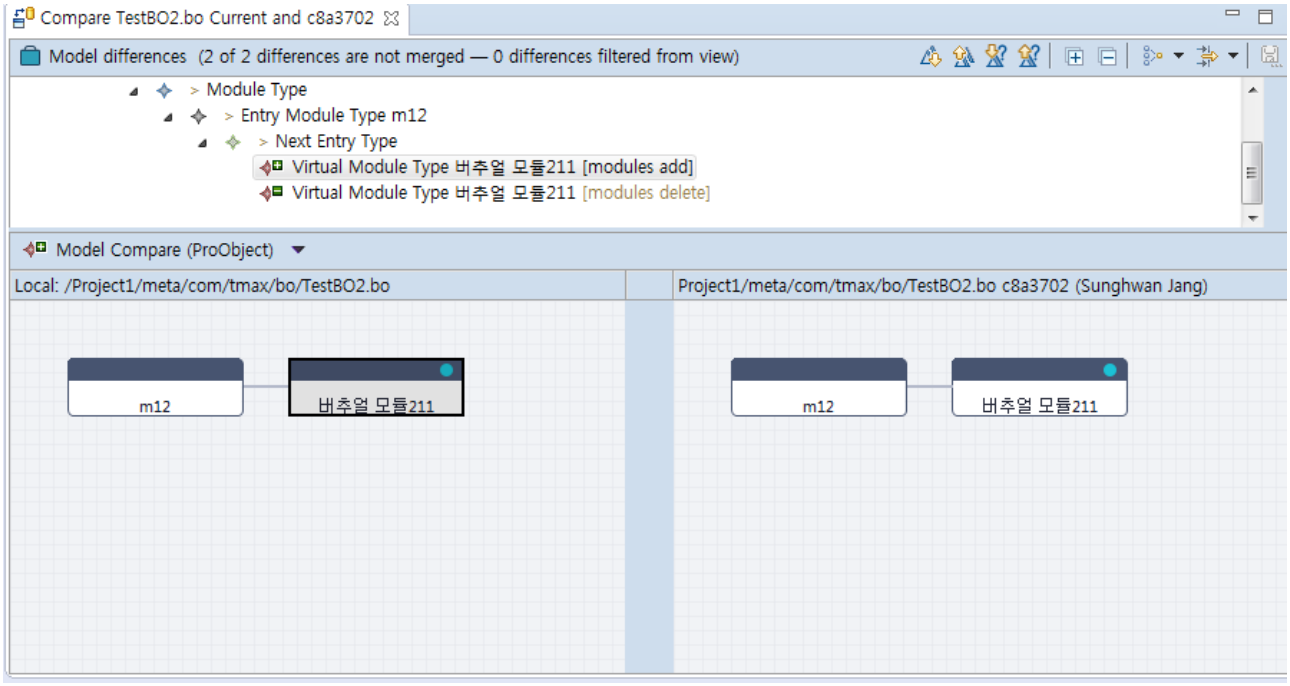
다음은 커밋된 소스와 현재 WorkSpace 파일을 비교하는 방법에 대한 설명이다.

1. **Package Exploer**에서 파일을 선택한 후 컨텍스트 메뉴에서 **[Compare With] > [Commit...]**을 선택한다. **Select a Commit 화면**의 커밋된 목록에서 비교할 특정 커밋을 선택한 후 **[OK]** 버튼을 클릭한다.




서버의 Master Branch 소스 비교 (1)

2. 다음 화면에서 특정 커밋과 현재 리소스를 비교할 수 있다.




서버의 Master Branch 소스 비교 (2)

# 7. Jenkins 사용법

본 장에서는 Jenkins 웹 브라우저의 접속 방법, Jenkins 프로젝트 생성 및 Jenkins 기능에 대해 설명한다.

## 7.1. 개요

Jenkins에서는 Git과 연동해서 ProObject의 개발 및 운영을 위하여 형상관리, Source Generation, Build, Deploy 등의 다양한 기능들을 제공한다. Git 연동에 대한 자세한 내용은 GIT 연동을 참고한다.

다음은 Jenkins 기능에 대한 설명이다.

| 기능 | 설명 |
|---|---|
| **형상관리** | ◦ Git의 원격/로컬 리파지토리, 형상관리, 이력관리, 브랜치, 머지 기능 |
| **Source Generation** | ◦ ProObject의 메타 타입별 자동 Source Generation(DTO, DOF, BO, SO, JO)<br>◦ Source Generation 상황 로그 기록 |
| **Build** | ◦ 서비스 그룹별 Full Build<br>◦ Add, Modified, Deleted Commit된 리소스 Partial Build<br>◦ Build Task(Full Build, Partial Build) 선택 실행<br>◦ Build Version별 Build Binary 관리<br>◦ Trunk에 최신 Build Binary 관리<br>◦ Pojo java Build 및 Binary 관리 |
| **Deploy** | ◦ On-premise 환경 배포 요청<br>◦ Cloud(ProZone) 환경 배포 |

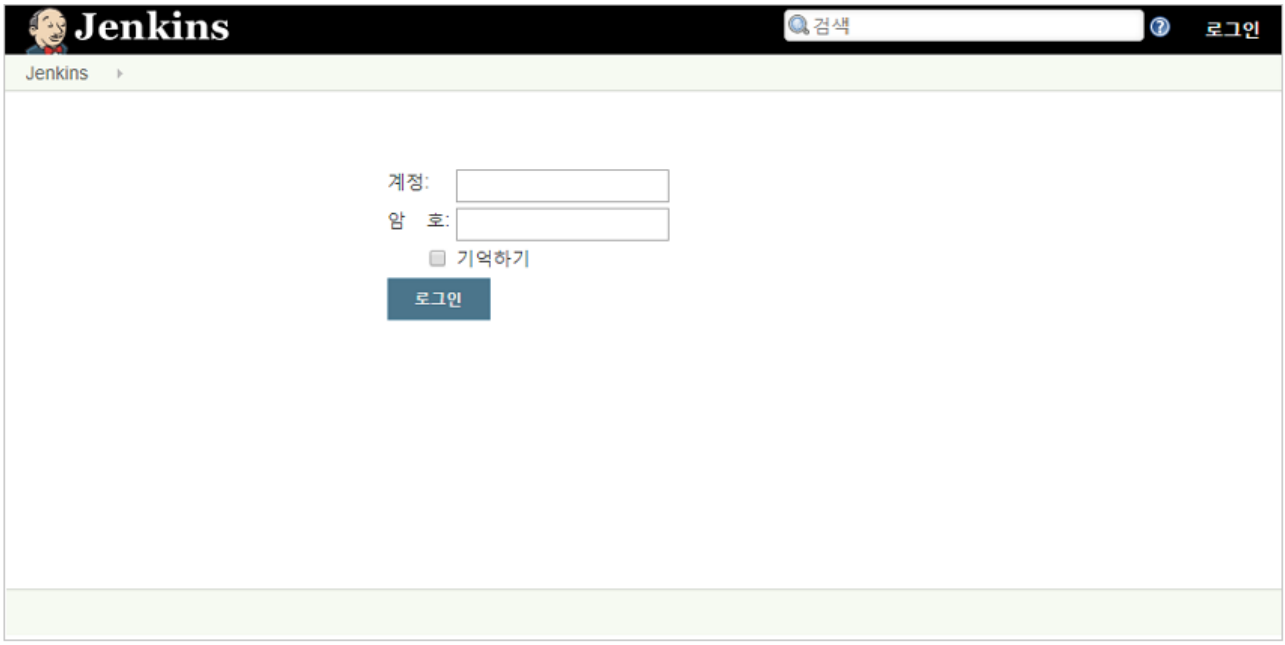
## 7.2. 시작하기

다음은 Jenkins에 시작하는 과정에 대한 설명이다.

1. 웹 브라우저에서 아래와 같이 주소를 입력한다.

   ```
   http://{서버IP 주소:포트}/{Jenkins Name}/login
   ```

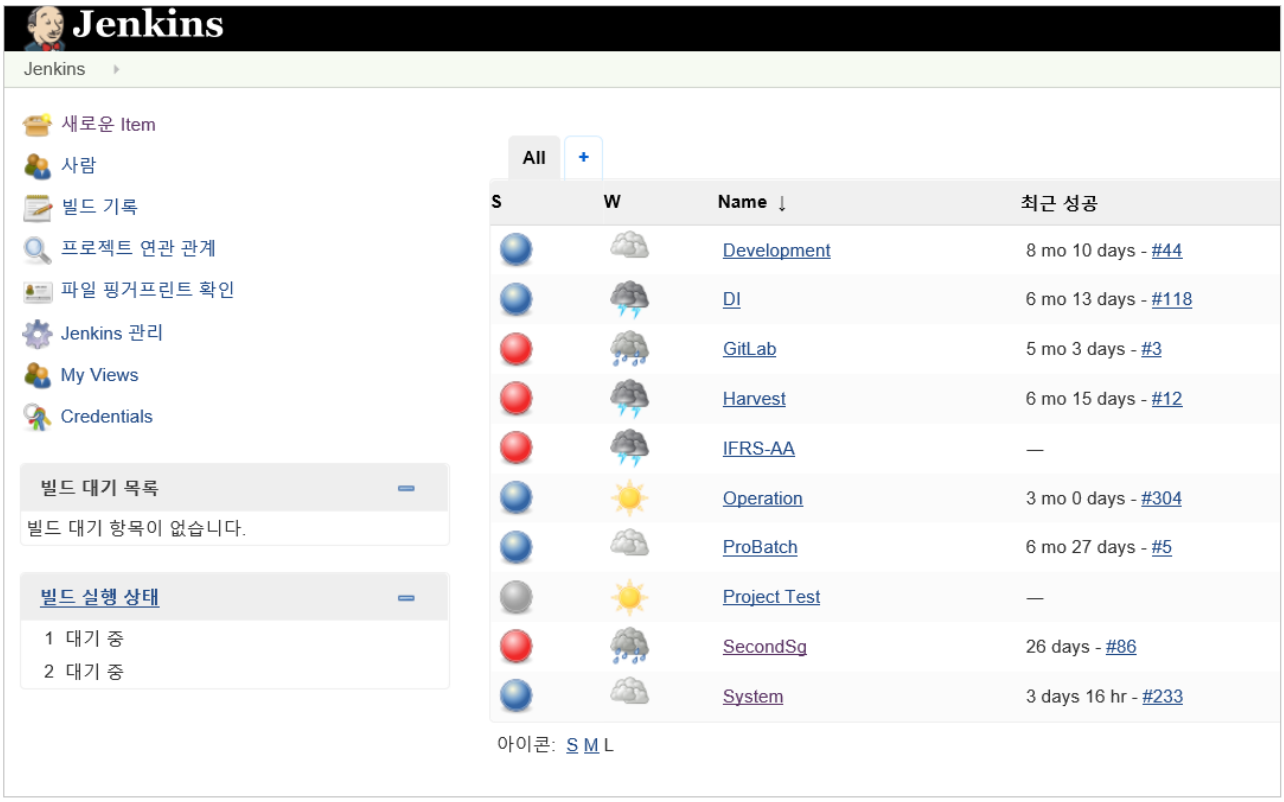
2. 주소를 입력한 후 서버에 정상적으로 Jenkins가 설치된 경우 로그인 페이지가 나타난다. 사용자 ID와 패스워드를 패스워드를 입력한 후 **[로그인]** 버튼을 클릭한다. 잘못된 ID나 패스워드를 입력하는 경우 경고 문구가 나타난다.




로그인 화면

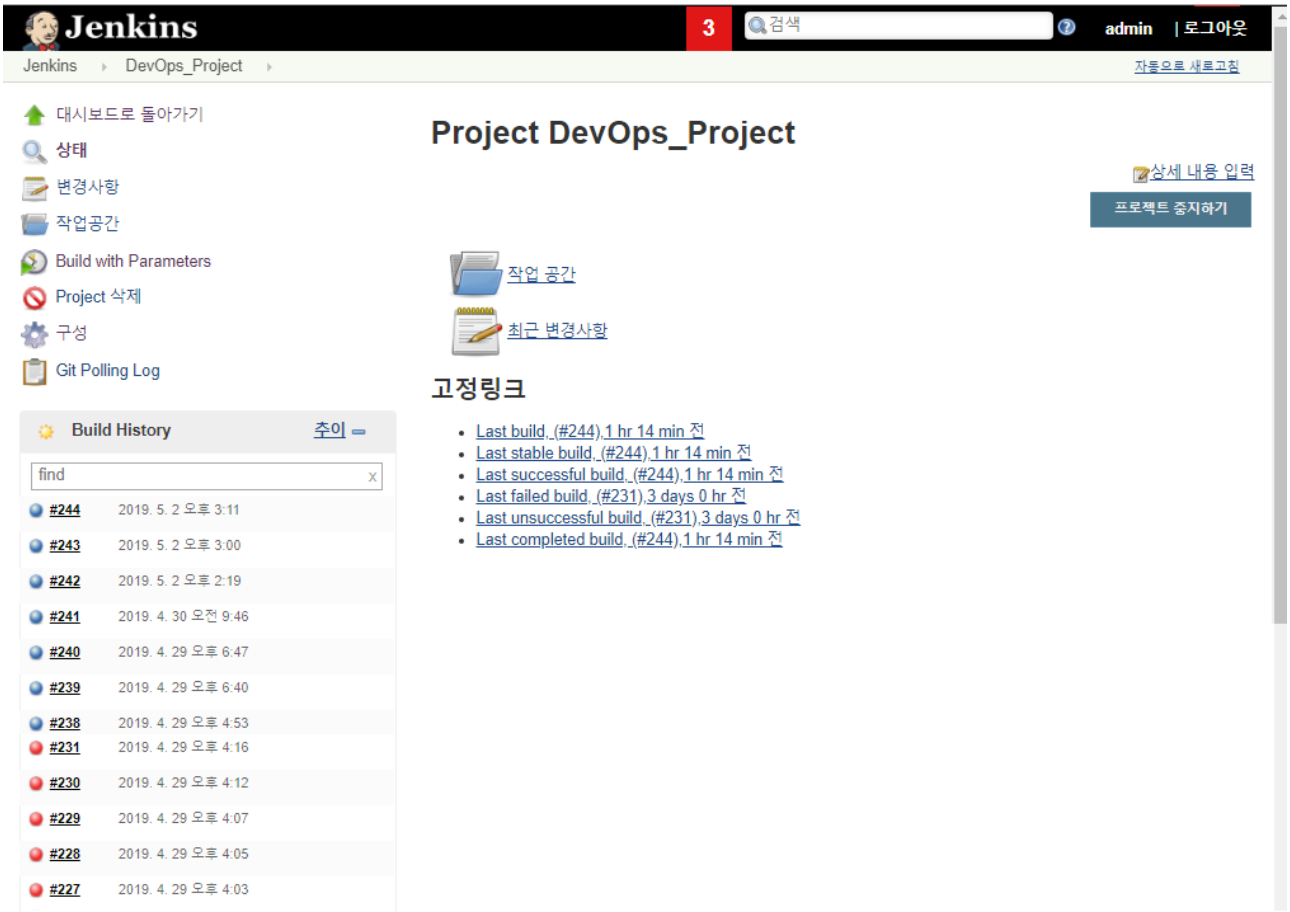
3. 로그인에 성공하면 **대시 보드**으로 이동한다. 대시보드에서는 생성한 프로젝트의 목록과 빌드 성공 여부를 확인할 수 있다.

   **[새로운 Itme]** 메뉴를 클릭하면 신규 프로젝트를 생성할 수 있다. 자세한 내용은 프로젝트 생성을 참고한다.




Jenkins 대시보드

4. 프로젝트를 생성한 후 **대시보드**(Jenkins 프로젝트 메인 화면)에서 해당 프로젝트를 클릭하면 **Jenkins 프로젝트 메인 화면**으로 이동한다. 각 화면의 구성 요소에 대한 자세한 내용은 화면 구성을 참고한다.



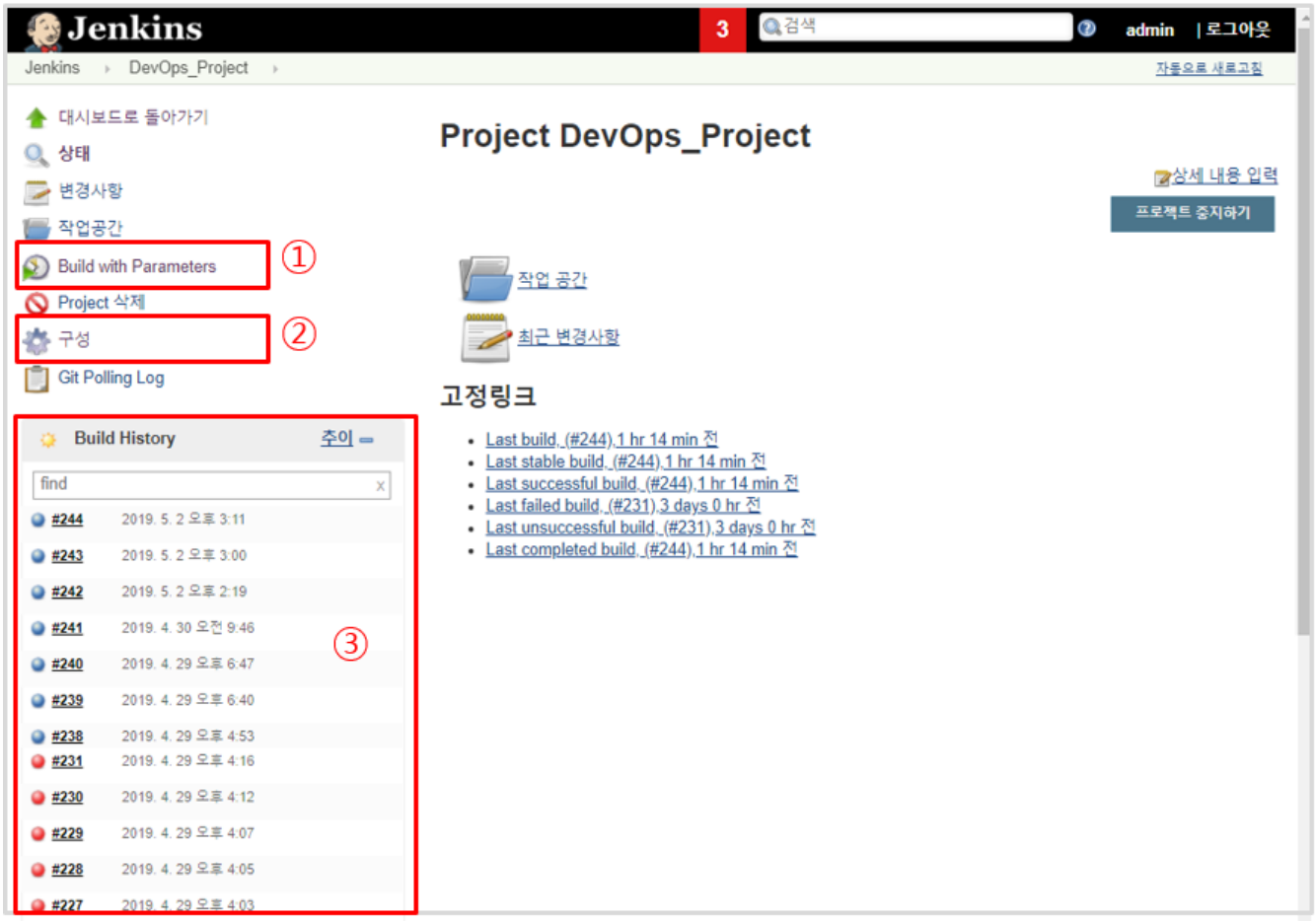
Jenkins 프로젝트 메인 화면

5. 메인 화면의 오른쪽 상단에 **[로그아웃]** 버튼을 클릭하면 로그아웃되며 로그인 초기 화면(로그인 화면)으로 이동한다.

## 7.3. 화면 구성

프로젝트 생성 후 **대시보드**(Jenkins 대시보드)에서 해당 프로젝트를 클릭하면 **Jenkins 프로젝트 메인 화면**으로 이동한다.

다음은 Jenkins 프로젝트 메인 화면 메뉴에 대한 설명이다.

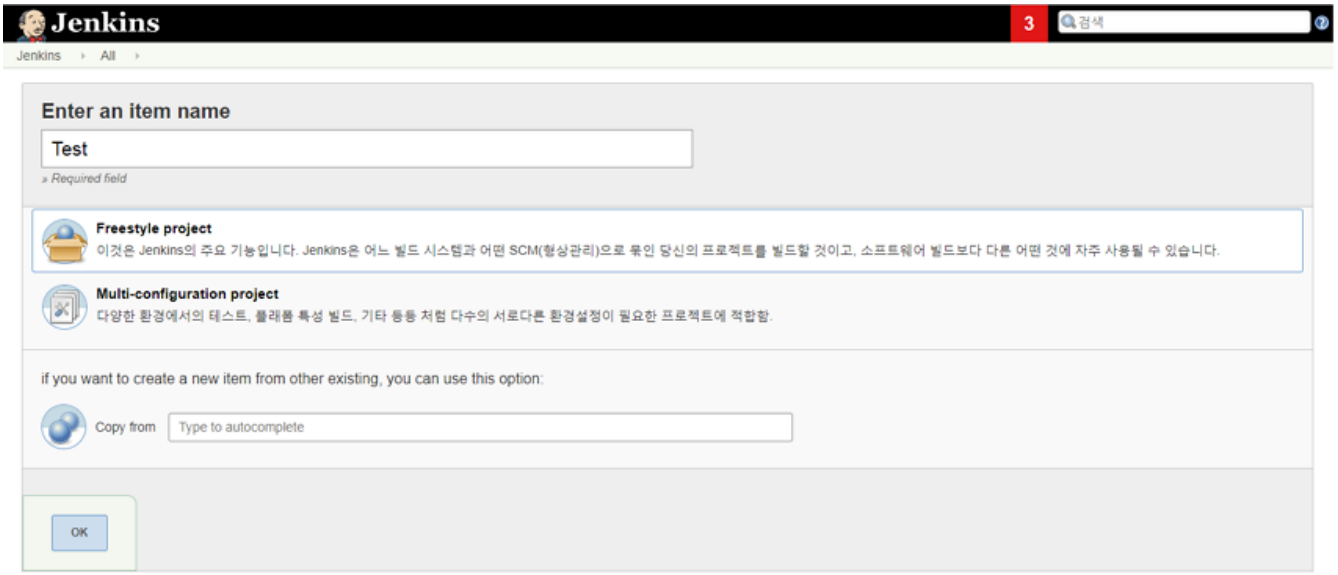



Jenkins 프로젝트 메인 화면 구성

| 메뉴 | 설명 |
|---|---|
| **[Build with Parameters]** | 구성 페이지에 설정한 플로우에 따라 Build job을 수행한다. |
| **[구성]** | Jenkins 프로젝트별로 빌드할 때 수행할 job을 설정하거나 구성한다. 프로젝트 구성에 대한 자세한 내용은 프로젝트 구성의 설명을 참고한다. |
| **[Build History]** | Build Job별로 빌드할 때마다 "#+Build Number"를 부여한다.<br><br>Build Number 앞의 아이콘으로 빌드의 수행 결과를 확인할 수 있다.<br><br>◦ 파랑 : 빌드 성공<br>◦ 빨강 : 빌드 실패<br><br>자세한 내용은 프로젝트 로그 조회를 참고한다. |

# 7.4. 프로젝트 생성

새로운 프로젝트를 생성해야 Jenkins에서 제공하는 기능을 사용할 수 있다. **대시보드**(Jenkins 대시보드)에서 **[새로운 Itme]** 메뉴를 클릭한다. Jenkins의 프로젝트 이름은 입력한 후 **'Freestyle project'**를 선택하고 **[OK]** 버튼을 클릭한다.




새로운 Jenkins 프로젝트 생성 화면

# 7.5. 프로젝트 구성

본 절에서는 System 프로젝트와 DevOps 프로젝트의 구성에 대해서 설명한다.

## 7.5.1. System 프로젝트

System 프로젝트는 ProStudo IDE에서 설정한 GIt으로 메타를 Push할 경우 Post-Checkout hook을 실행 ProMinor의 정적 분석과 Test 서버의 관리 테이블에 메타 정보를 설정한다.

Post-Checkout은 다음의 경로에 위치한다.

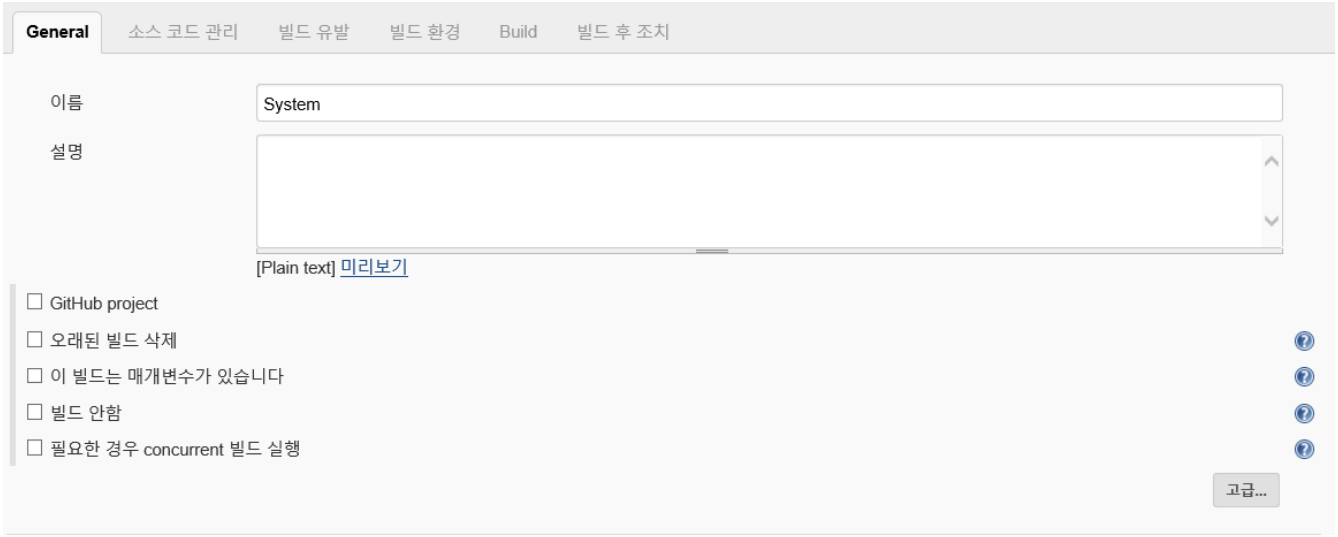
```
${WORKSPACE}/${project name}/.git/hooks/post-checkout
```

Jenkins 프로젝트 메인 화면에서 Jenkins 프로젝트의 **[구성]** 메뉴를 선택한 후 각 탭 화면의 항목을 설정한다.

- [General] 탭
- [소스 코드 관리] 탭
- [빌드 유발] 탭

### [General] 탭

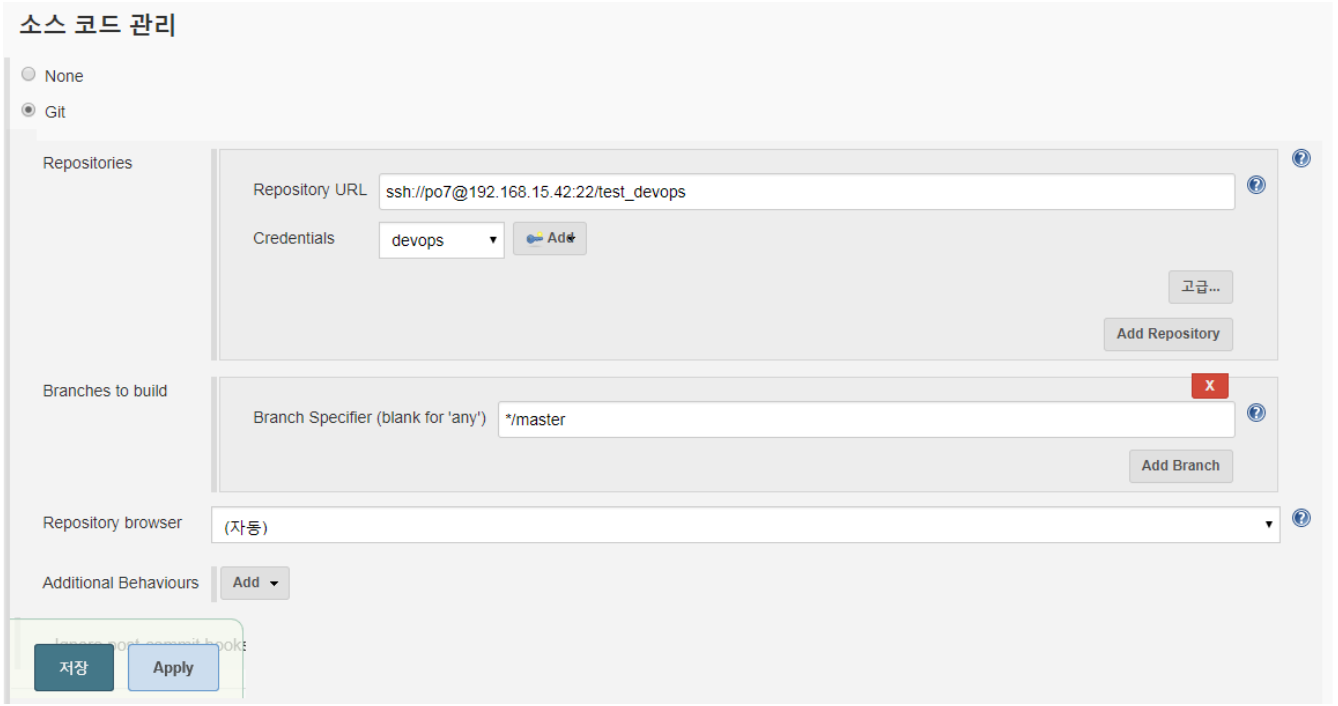
소스 코드의 기본적인 정보를 설정한 후 **[저장]** 버튼을 클릭한다.




System 프로젝트 구성 - [General] 탭

## [소스 코드 관리] 탭

소스 코드를 관리하는 저장소에 관련한 정보를 설정한 후 **[저장]** 버튼을 클릭한다.




System 프로젝트 구성 - [소스 코드 관리] 탭

| 항목 | 설명 |
|---|---|
| None/Git | 소스 코드를 관리할 대상을 선택한다. (**'Git'**을 선택) |
| Repository URL | **'Git'**을 선택한 경우 Git Repository Url을 설정한다. |
| Credentials | **[Add]** 버튼을 클릭해서 사용자 Credential을 추가한다. |
| Branch Specifier | 연결 및 build할 branch를 설정한다. |
| Repository browser | '(자동)'을 선택한다. |

## [빌드 유발] 탭

빌드를 하는 다양한 방식을 설정한다. 예제에서는 **'Poll SCM'** 항목을 선택한 후 **[저장]** 버튼을 클릭한다.


프로젝트 구성 - [빌드 유발] 탭

| 항목 | 설명 |
|---|---|
| 빌드를 원격으로 유발 | 외부에서 URL을 통해 빌드를 진행할 수 있도록 한다. |
| Build after other project are built | 다른 프로젝트를 빌드한 후 이어서 현재 프로젝트를 빌드한다. |
| Build periodically | 빌드를 배치 프로세스처럼 설정해 놓은 시간에 자동으로 빌드를 진행하도록 한다. |
| Poll SCM | 형상관리 서버에를 주기적으로 감시하여 변경된 사항이 존재할 때 빌드를 수행한다. (옵션항목)<br><br>**'Poll SCM'**을 선택하면 Git에 push 후 빌드를 trigger하여 자동으로 Build job을 수행할 수 있다. 선택하지 않을 경우 Jenkins 프로젝트 메인 화면의 **[Build with Parameters]** 메뉴로 Build job을 수행한다. |

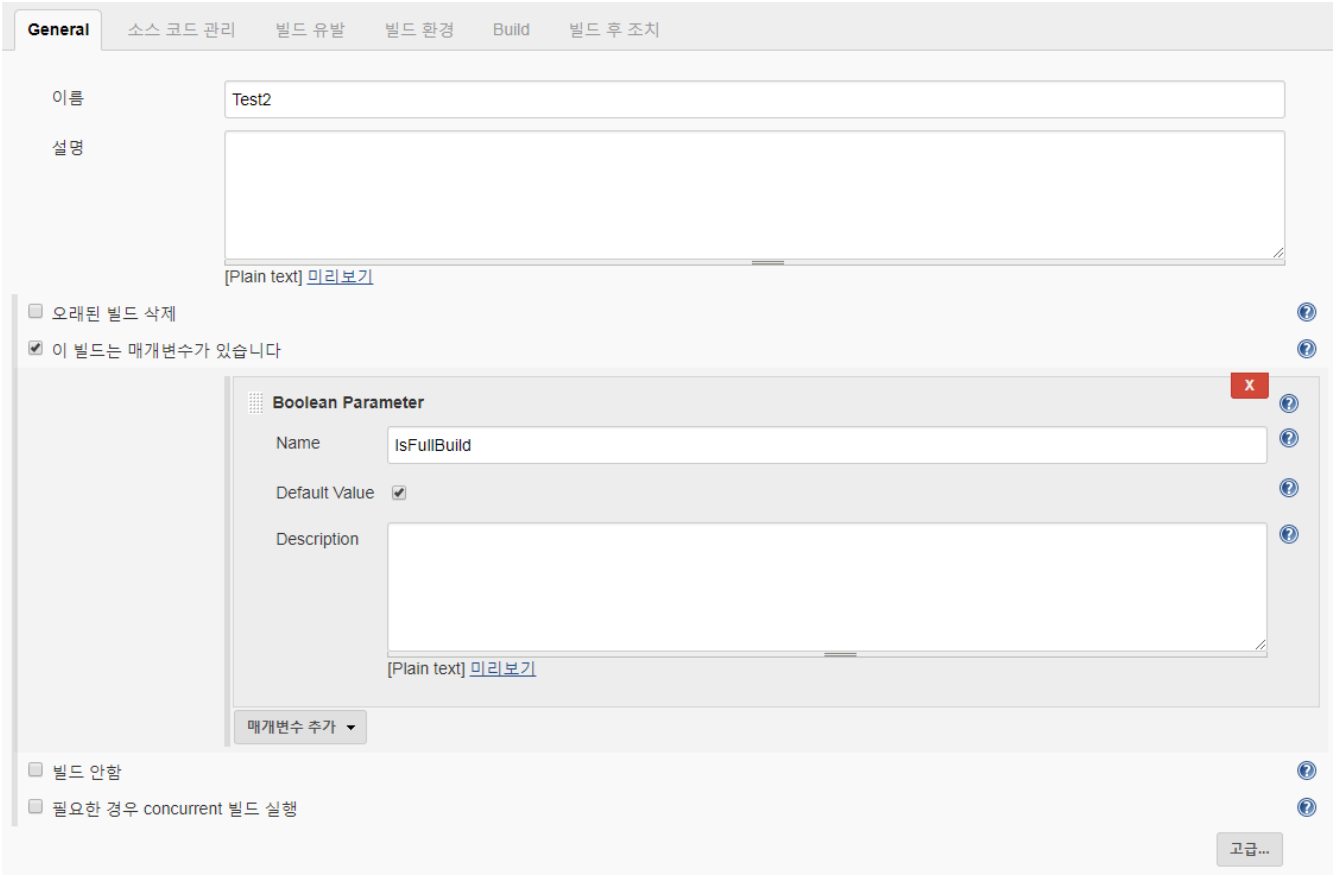
## 7.5.2. DevOps 프로젝트

Jenkins 프로젝트 메인 화면에서 Jenkins 프로젝트의 **[구성]** 메뉴를 선택한 후 각 탭 화면의 항목을 설정한다.

- [General] 탭
- [소스 코드 관리] 탭 (System 프로젝트의 설명을 참고한다.)
- [Build] 탭

### [General] 탭

소스 코드의 기본 적인 정보를 설정한다. **'이 빌드는 매개변수가 있습니다'** 체크박스를 선택한 후 **Boolean Parameter**에 항목을 추가한다. **'Name'**에 항목을 'IsFullBuild'로 설정하고 **'Default Value'** 항목을 선택힌다.

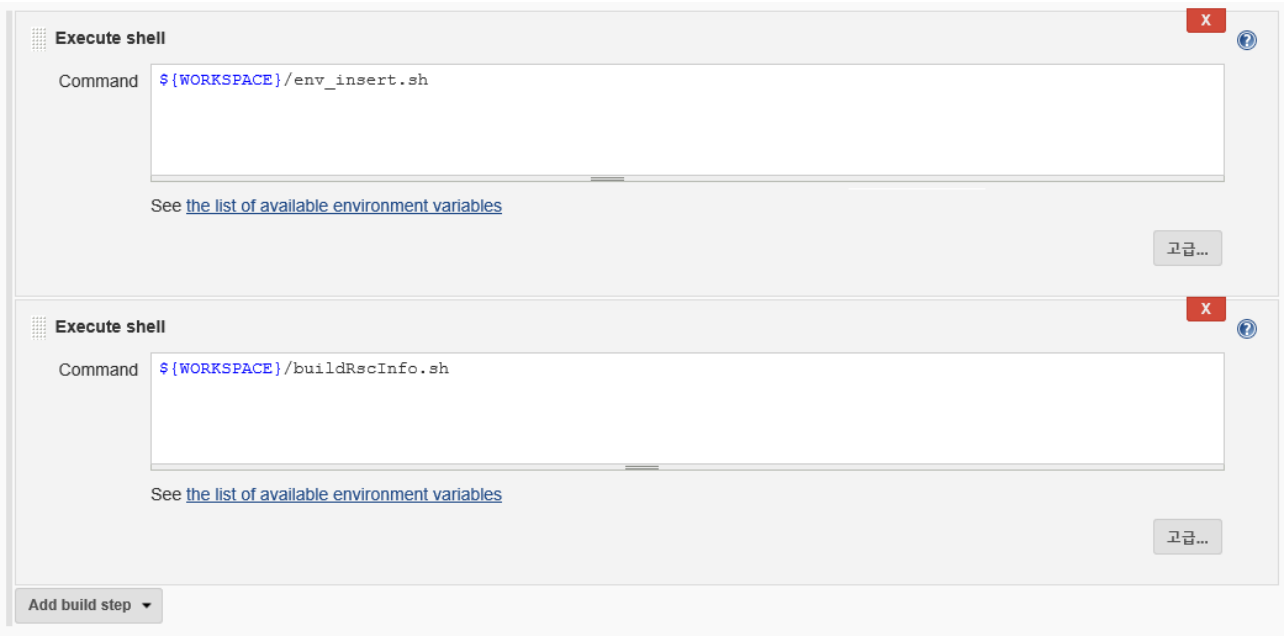



DevOps 프로젝트 구성 - [General] 탭

## [Build] 탭

다음은 빌드를 진행하는 과정에 대한 설명이다.

1. **[Add build Step]** 버튼을 클릭해서 **'Execute shell'**을 추가하고 **'Command'** 항목에 '${WORKSPACE}/env_insert.sh'와 '${WORKSPACE}/buildRscInfo.sh'를 각각 설정한다.

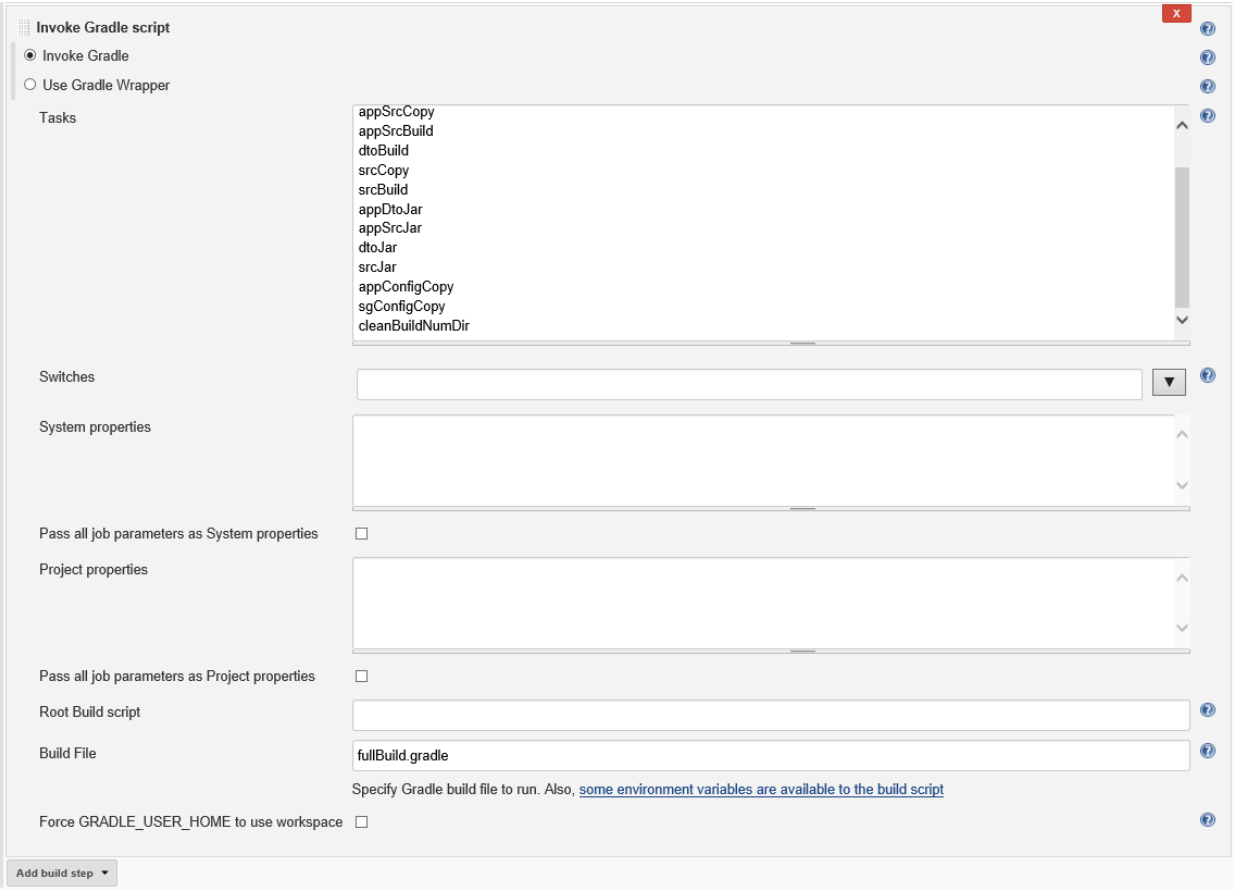



DevOps 프로젝트 구성 - [Build] 탭

다음은 추가한 Execute shell의 설명이다.

| 파일 | 설명 |
|---|---|
| env_insert.sh | Jenkins 빌드를 위한 여러 환경 변수들에 관한 설정을 위한 shell script이다. |
| buildRscInfo.sh | Jenkins 빌드를 위한 데이터를 DB에 저장하기 위한 shell script이다. |

2. **[Add build Step]** 버튼을 클릭해서 **Invoke Gradle script**를 추가한 후 Full Build와 Partial Build 설정을 한다.
   - **Full Build 설정**




DevOps 프로젝트 구성 - [Build] 탭 - Invoke Gradle script 설정 (Full Build)

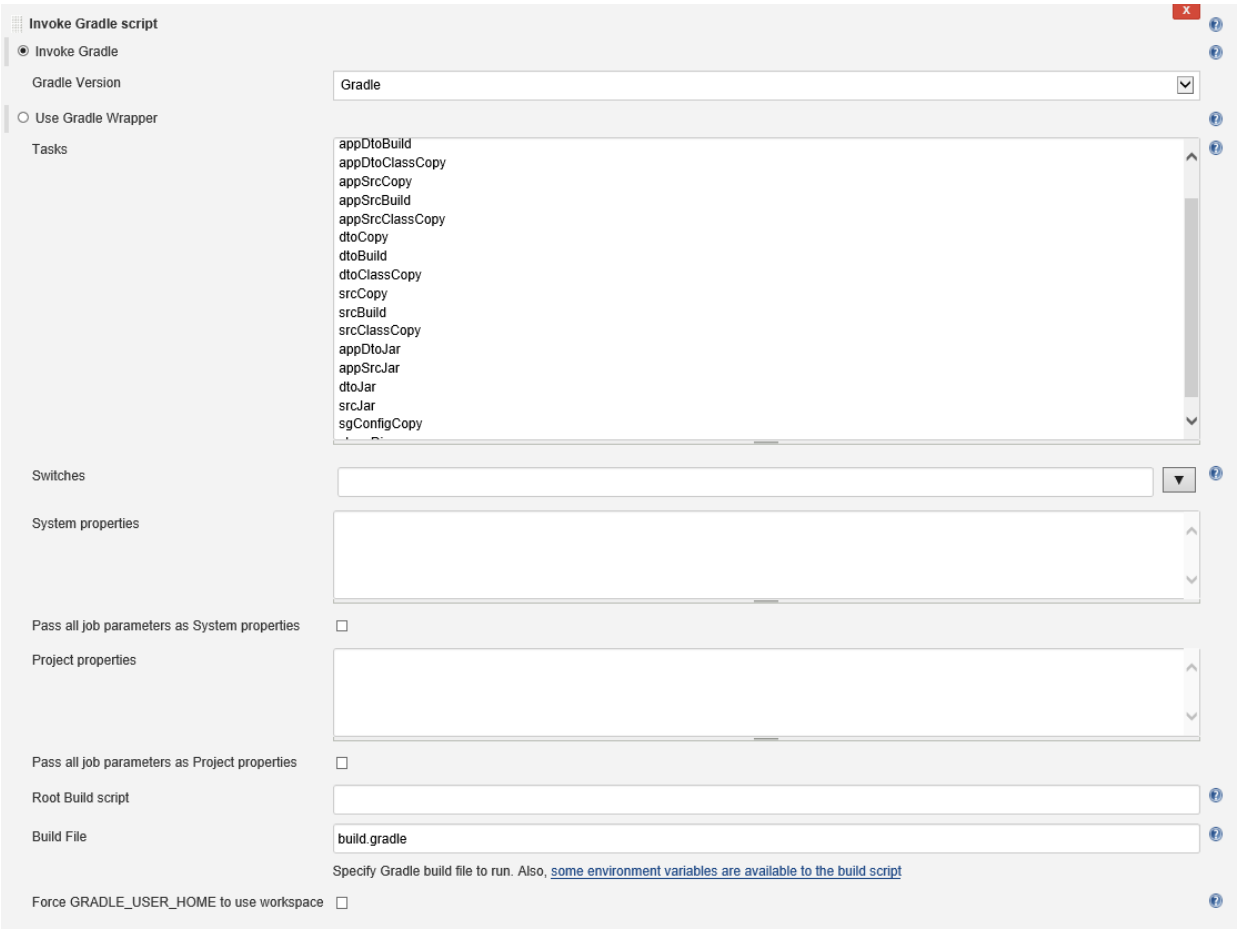
   a. **'Invoke Gradle'** 항목을 선택하고 **'Gradle Version'**을 'Gradle'로 선택한다.

   b. 빌드 옵션에 따라 **'Tasks'** 항목에 다음을 설정한다(설정 값을 확인하려면 **'Tasks'** 항목 끝 화살표를 클릭한다).

```
cleanDir, appDtoBuild, appSrcCopy, appSrcBuild, dtoBuild, srcCopy,
srcBuild, appDtoJar, appSrcJar, dtoJar, srcJar, sgConfigCopy,
cleanBuildNumDir
```

   c. **'Tasks'** 아래 **[고급]** 버튼을 클릭해서 화면을 확장한 후 **'Build File'** 항목을 'fullBuild.gradle'로 설정한다.

   - **Partial Build 설정**




DevOps 프로젝트 구성 - [Build] 탭 - Invoke Gradle script 설정 (Partial Build)

a. **'Invoke Gradle'** 항목을 선택하고 **'Gradle Version'**을 선택한다.

b. 빌드 옵션에 따라 **'Tasks'** 항목에 다음을 설정한다(설정 값을 확인하려면 **'Tasks'** 항목 끝 화살표를 클릭한다).

```
makeEmptyDir, appDtoCopy, appDtoBuild, appDtoClassCopy, appSrcCopy,
appSrcBuild, appSrcClassCopy, dtoCopy, dtoBuild, dtoClassCopy,
srcCopy, srcBuild, srcClassCopy, appDtoJar, appSrcJar, dtoJar, srcJar,
sgConfigCopy, cleanDir
```

c. **'Tasks'** 아래 **[고급]** 버튼을 클릭해서 화면을 확장한 후 **'Build File'** 항목을 'Build.gradle'로 설정한다.

3. **'Execute shell'**을 추가하고 Command에 ${WORKSPACE}/deploy.sh를 설정한 후 **[저장]** 버튼을 클릭한다.

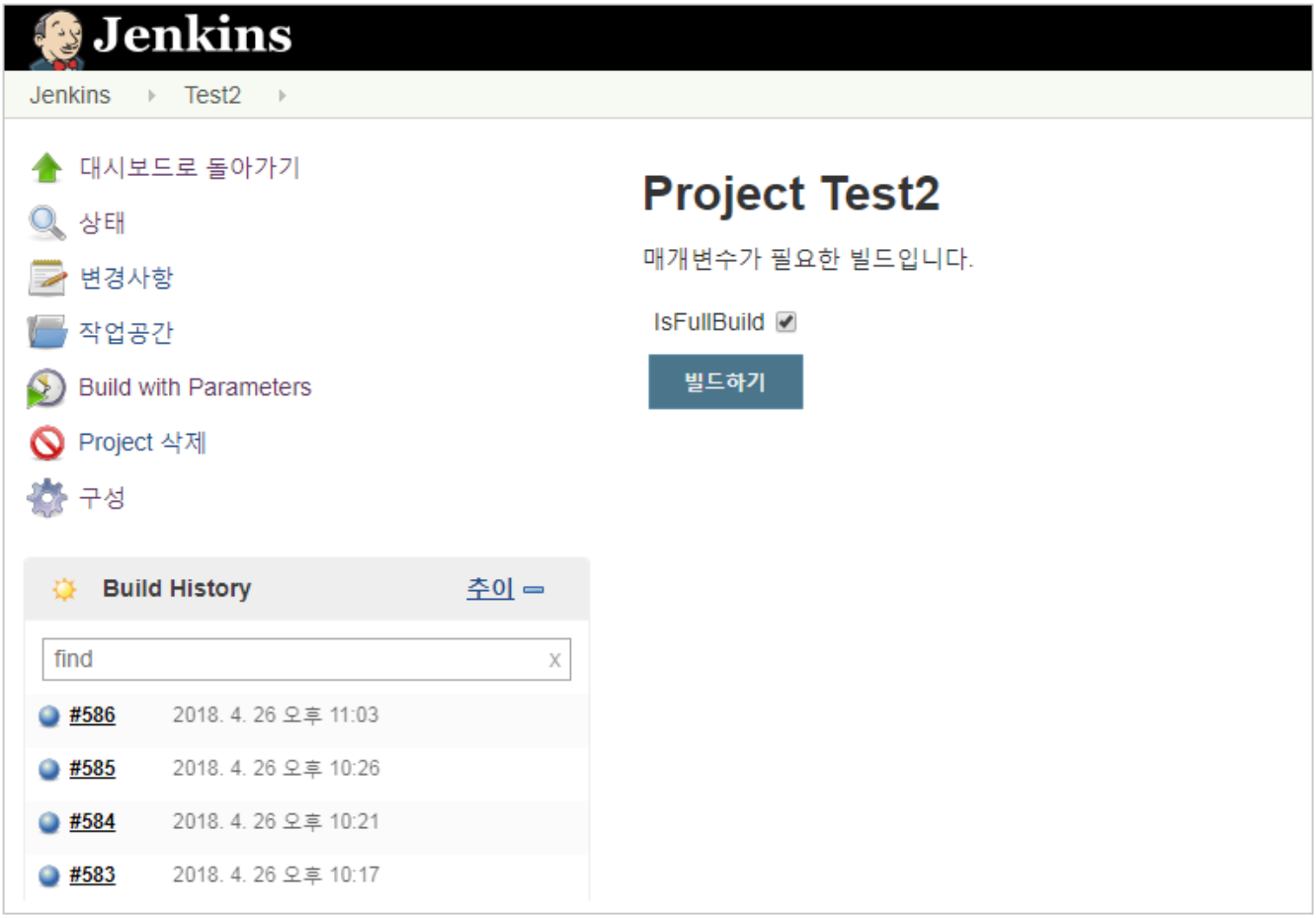

DevOps 프로젝트 구성 - [Build] 탭 - Execute shell 설정

다음은 추가한 Execute shell의 설명이다.

| 파일 | 설명 |
|---|---|
| deploy.sh | 빌드된 리소스를 Deploy Call하기 위한 shell script이다. |

## 7.6. 프로젝트 Build

설정한 Jenkins 프로젝트에서 Build Job Process는 **Jenkins 프로젝트 메인 화면**의 **[Build with Parameters]** 메뉴에서 **'IsFullBuild'** 선택 여부에 따라 'Partial Build'와 'Full Build'로 나뉜다.




프로젝트 Build

다음은 Full Build와 Partial Build에 따른 설정 값에 대한 설명이다.

- Full Build

| 구분 | 설명 |
|---|---|
| 소스젠 | 기존 리소스 + push 리소스 |
| Compile | 전체 범위 |
| jar packaging | 전체 범위 |

- Partial Build

| 구분 | 설명 |
| --- | --- |
| 소스젠 | 기존 리소스 + push 리소스 |
| Compile | push 리소스 |
| jar packaging | push 리소스 |

다음은 Jenkins 프로젝트의 Full Build 옵션으로 Job을 빌드하는 과정에 대한 설명이다.

1. **Source Generation**

   Add, Modified Commit된 ProObject의 메타 유형별(dto, dof, bo, so, jo) 소스를 Generation한다.

2. **Source Generation된 java file 관리**
3. **Build**

   모든 리소스 Full Build 및 Jar 파일을 패키징한다.

4. **빌드 결과 바이너리를 빌드 버전별 및 Trunk에 관리**

   다음은 빌드 결과 바이너리를 관리하는 위치에 대한 설명이다.

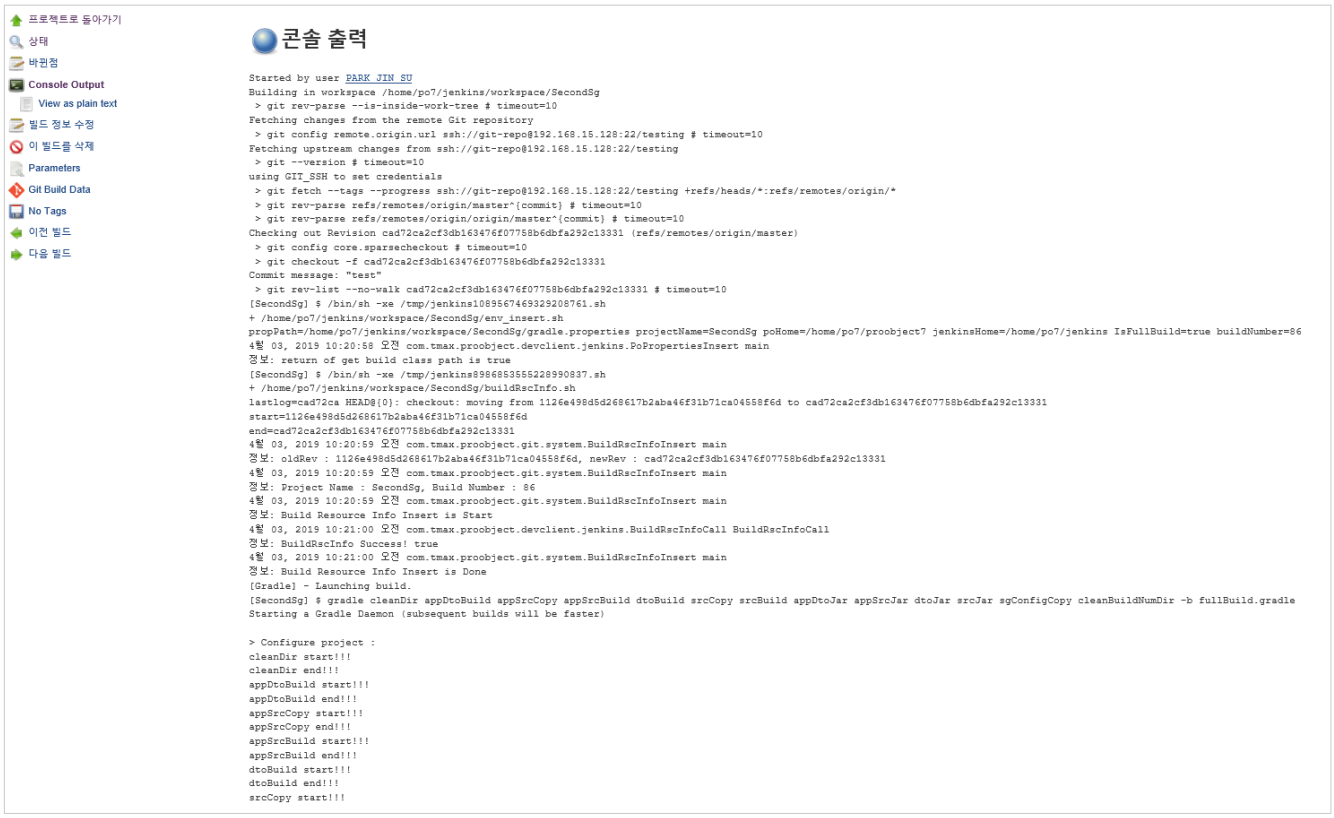
| 종류 | 관리 위치 |
| --- | --- |
| java file | ◦ Do (App) : ${WORKSPACE}/${AppName}/.gen_src/dto/ 하위<br>◦ Do (Sg) : ${WORKSPACE}/${sgName}/.gen_src/dto/ 하위<br>◦ Other Type (App) : ${WORKSPACE}/${AppName}/.gen_src/ 하위<br>◦ Other Type (Sg) : ${WORKSPACE}/${sgName}/.gen_src/ 하위 |
| pojo java file | ◦ ${WORKSPACE}/${sgName}/src/ 하위 |
| meta file | ◦ App : ${WORKSPACE}/${AppName}/meta/ 하위<br>◦ Sg : ${WORKSPACE}/${sgName}/meta/ 하위 |
| class file | ◦ App Do(Trunk) : ${poHome}/resources/classes/_build/${projectName}/${appName}/application/${appName}-dto/ 하위<br>◦ Sg Do(Trunk) : ${poHome}/resources/classes/_build/${projectName}/${appName}/${sgName}-dto/ 하위<br>◦ App Other Type(Trunk) : ${poHome}/resources/classes/_build/${projectName}/${appName}/application/${appName}/ 하위<br>◦ Sg Other Type(Trunk) : ${poHome}/resources/classes/_build/${projectName}/${appName}/${sgName}/ 하위 |

| 종류 | 관리 위치 |
| --- | --- |
| jar file | ◦ App : ${poHome}/resources/classes/_build/_jar/${projectName}/${buildNum}/${appName}/ 하위<br>◦ Sg : ${poHome}/resources/classes/_build/_jar/${projectName}/${buildNum}/${appName}/${sgName}/ 하위 |

# 7.7. 프로젝트 로그 조회

다음은 Build Job Process의 결과 로그에 대한 설명이다.

Jenkins 프로젝트 메인 화면의 **[Build History]** 메뉴에서 Build Job별로 수행된 Build Number의 **[Console Output]** 메뉴에서 Build Job Process의 결과 log를 확인할 수 있다.




프로젝트 로그 조회

# 7.8. 프로젝트 Deploy

Jenkins 프로젝트를 디플로이하는 과정에 대해서 설명한다.

1. **디플로이 설정**

   Jenkins 프로젝트에서 배포할 수 있는 서버의 타입은 'Runtime Server'과 'Test Server'이다. 이 둘의 설정은 **${WORKSPACE}/gradle.properties**의 *'isDeployRequest' 옵션으로 설정할 수 있다.

```
isDeployRequest = true : [Runtime Server]
isDeployRequest = false : [Test Server]
```

Runtime Server는 사이트의 프로그램이 수행됨으로써 고객들에게 서비스를 제공하는 운영 환경이다. Runtime Server에 디플로이를 위해서는 Jenkins 프로젝트에서 Build Process 후에 Shell Script를 수행해 배포 서비스를 호출하게 된다. Test Server는 개발자가 Test 할 수 있는 개발용 서버환경이다.

2. **애플리케이션 및 서비스 그룹 디플로이**

   Jenkins 프로젝트에서 빌드하는 경우 **[IsFullBuild]** 메뉴를 선택한 후 빌드한다. Build Job Process 이후 IsFullBuild가 선택됨을 확인하여 개발 서버의 서비스 그룹 배포 서비스 호출 및 수행한다.

3. **클래스 디플로이**

   Jenkins 프로젝트에서 빌드를 하는 경우 **[IsFullBuild]** 메뉴 선택하지 않고 빌드한다. Build Job Process 이후 IsFullBuild가 해제됨을 확인하여 개발 서버의 클래스 배포 서비스 호출 및 수행한다.

4. **로그 조회**

   Jenkins 프로젝트 메인 화면의 **[Build History]** 메뉴에서 Build Job별로 수행된 Build Number의 Console Output 메뉴에서 Build Job Process 이후의 디플로이 결과 로그를 확인할 수 있다.

Jenkins ▸ Test2 ▸ #450

```
  :
  :
  :
BUILD SUCCESSFUL in 0s
[Test2] $ /bin/sh -xe /tmp/jenkins5066578219145850719.sh
+ /home/devops/tkins/workspace/Test2/deploy.sh
Apr 17, 2018 4:24:31 PM com.tmax.proobject.devclient.jenkins.SgDeployCall main
INFO: ERROR:1
Apr 17, 2018 4:24:31 PM com.tmax.proobject.devclient.jenkins.SgDeployCall main
INFO: propPath: /home/devops/tkins/workspace/Test2/gradle.properties
Apr 17, 2018 4:24:31 PM com.tmax.proobject.devclient.jenkins.SgDeployCall main
INFO: ERROR:2
Apr 17, 2018 4:24:31 PM com.tmax.proobject.devclient.jenkins.SgDeployCall main
INFO: ERROR:3
Apr 17, 2018 4:24:31 PM com.tmax.proobject.devclient.jenkins.SgDeployCall main
INFO: ERROR:4
Apr 17, 2018 4:24:31 PM com.tmax.proobject.devclient.jenkins.SgDeployCall main
INFO: Deploy Success! true
Finished: SUCCESS
```

프로젝트 Deploy 결과 log

# 8. DB 형상관리

본 장에서는 ProStudio에서 제공하는 DB 형상관리의 주요 기능에 대해 설명한다.

## 8.1. 개요

ProStudio는 Git 및 Db의 형상관리 방식을 지원한다. DB 형상관리는 개발자가 개발한 리소스(SO, BO, DO 등)가 DB에 바로 저장되기 때문에 Git처럼 Local로 리소스 등을 Pull하는 작업을 하는 단계가 줄어든다. 따라서 리소스의 관리가 직관적이다는 장점이 있다. 하지만 Git의 Branch 등을 관리하는 Git의 고유 기능은 이용하지 못한다.

## 8.2. Project 생성

DB 형상관리도 Git과 마찬가지로 프로젝트를 먼저 생성해야 한다. 애플리케이션과 서비스 그룹 프로젝트 생성에 대한 자세한 내용은 프로젝트 생성을 참고한다.

## 8.3. Project 리소스 관리

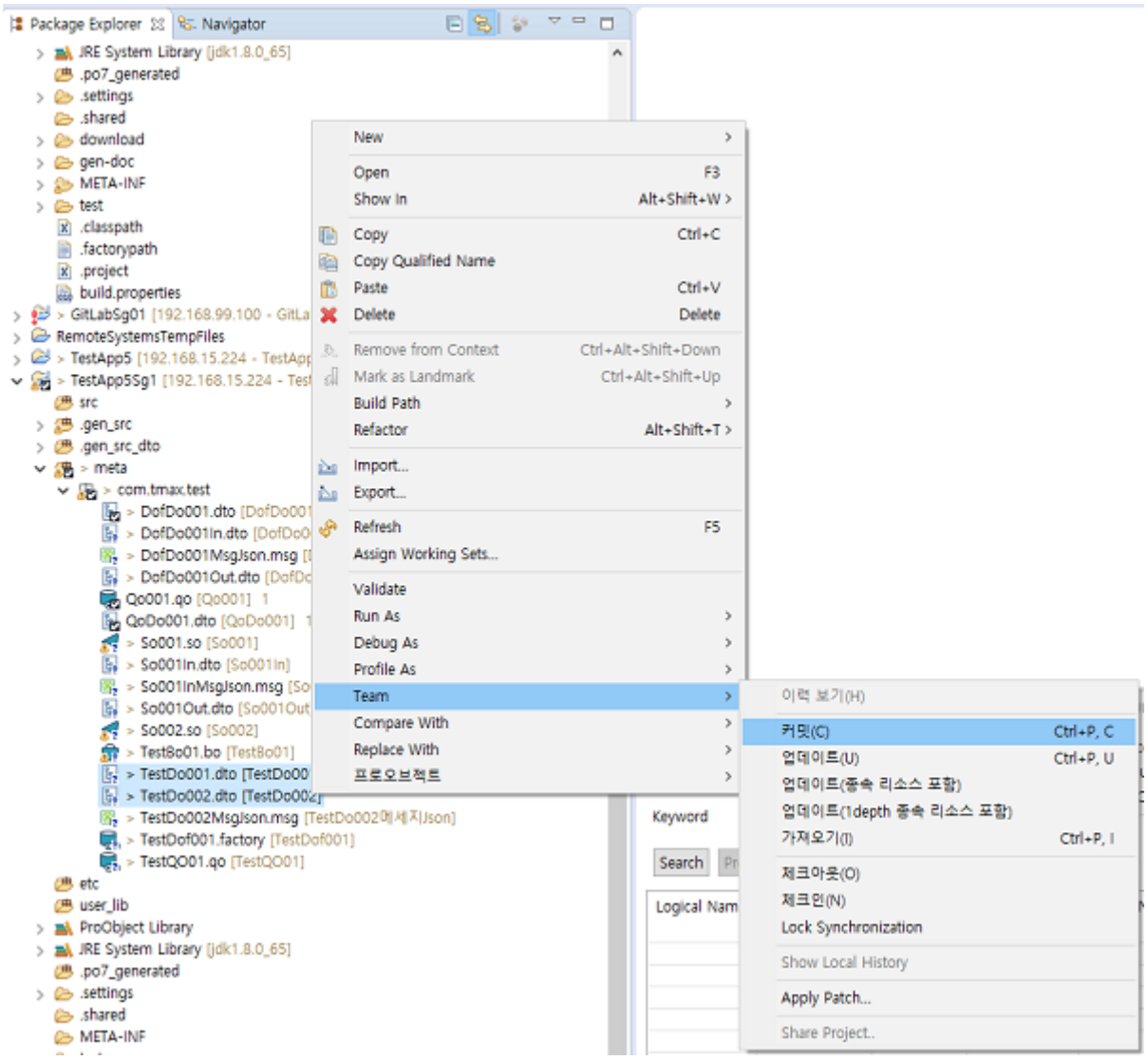
본 절에서는 리소스를 저장하는 커밋, 리소스의 편집 권한을 가져오는 체크아웃, 리소스의 편집 권한을 해제하는 체크인, Lock의 현행화를 위한 Lock Synchronization에 대해서 설명한다.

### 8.3.1. 커밋

리소스를 DB에 저장 하는 단계이다. 이때 서버에 리소스 정보 및 ProMiner 정보 그리고 컴파일 단계등을 거친다. 커밋된 이후에도 체크인 이전 까지는 편집 및 커밋이 가능하다.

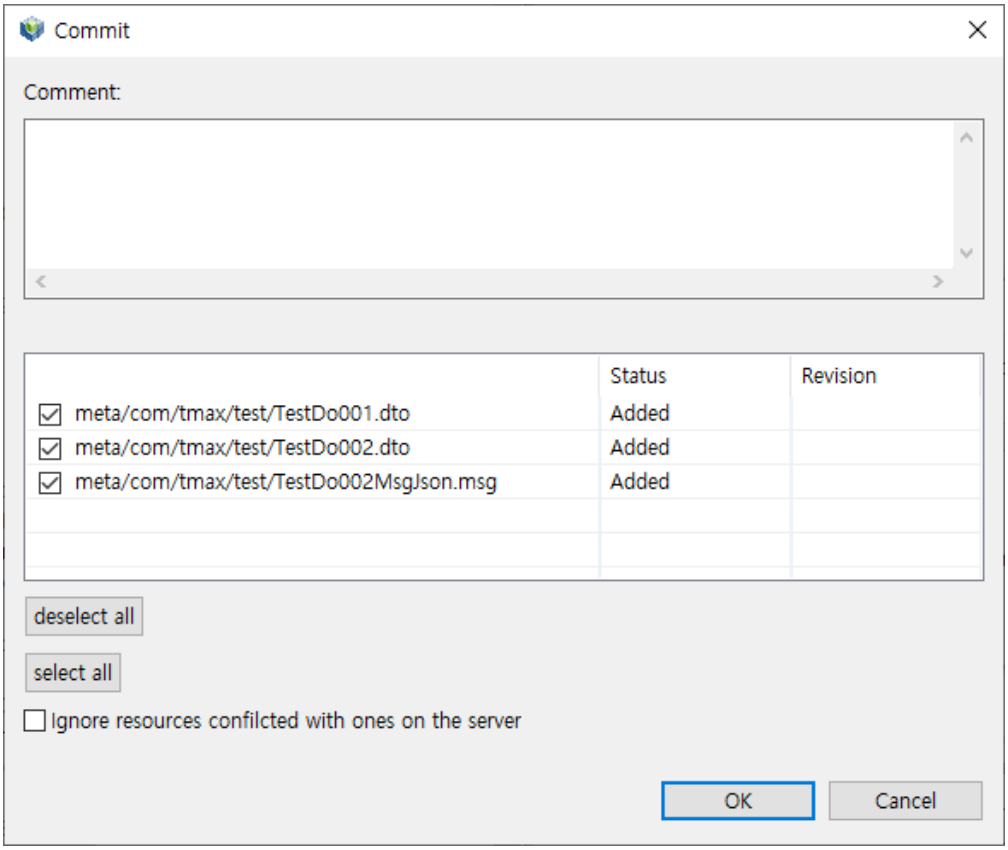
다음은 커밋하는 과정에 대한 설명이다.

1. DB에 저장 할 리소스를 선택(여러 개 선택 가능)한 후 컨텍스트 메뉴에서 **[Team] > [커밋(C)]**를 선택한다. 이때 커밋 가능한 리소스는 체크아웃된 리소스 및 신규 작성한 리소스이다.



리소스 커밋 - Commit 리소스 선택 및 처리

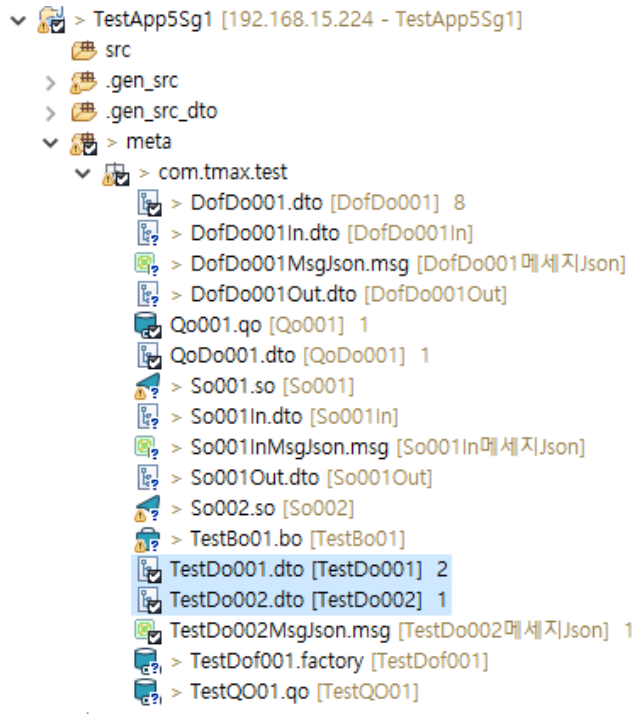
2. 커밋할 대상이 되는 리소스 내역이 표시되는 화면에서 **[OK]** 버튼을 클릭하면 커밋 작업을 수행한다.




리소스 커밋 - 커밋 처리

| 항목 | 설명 |
|---|---|
| Comment | 커밋할 리소스의 설명을 입력한다. |
| deselect all | 커밋할 리소스 대상을 모두 제외한다. |
| select all | 커밋할 리소스 대상을 모두 선택한다. |
| Ignore resources | 서버에 리소스가 충돌하더라도 무시한다. |

3. 정상적으로 Commit되면 아래와 같이 네비게이터의 리소스에 **[V]** 표시가 된다. 또한 리소스의 오른쪽에 커밋 리비전이 1만큼 증가한다.



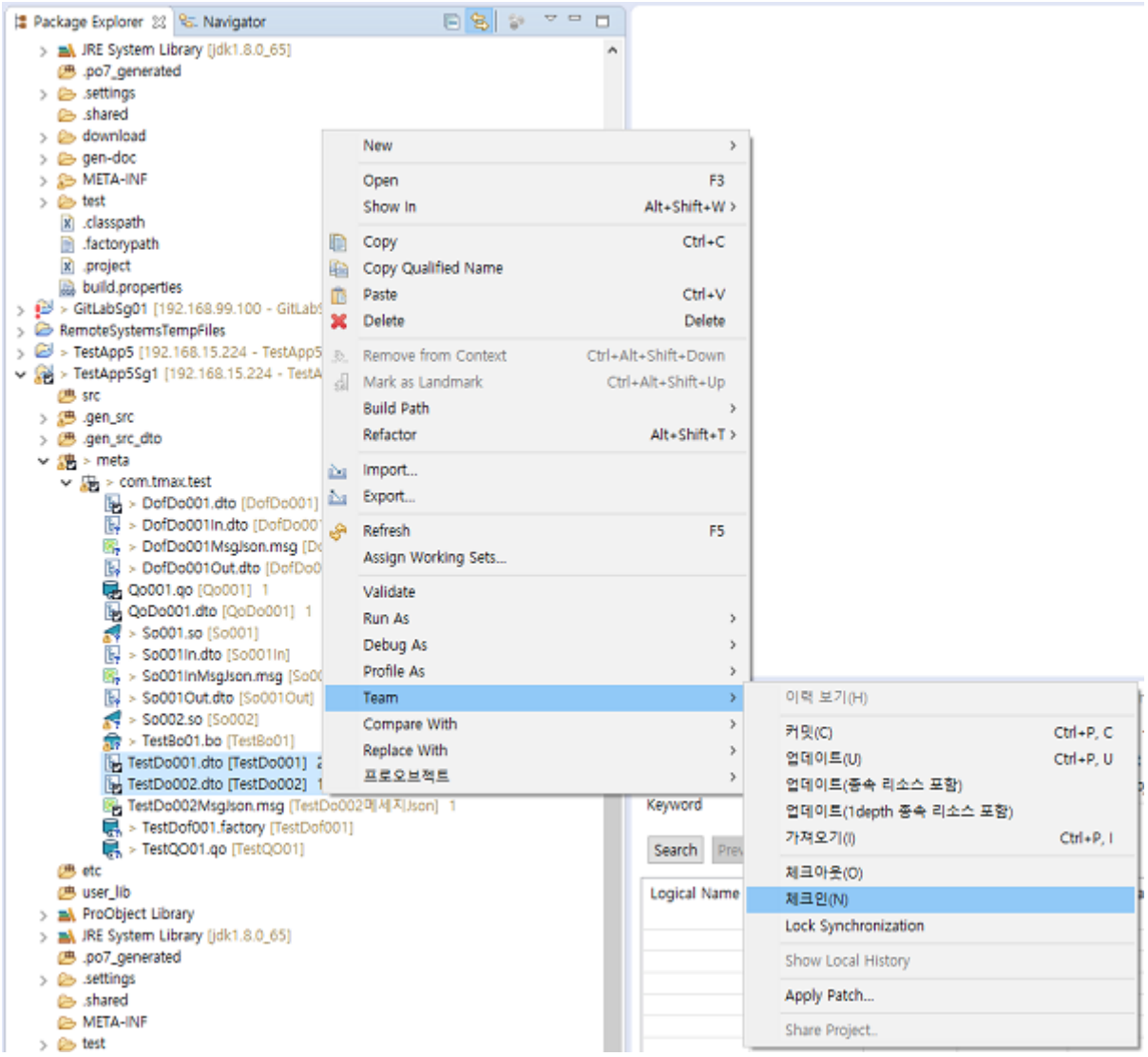

리소스 커밋 - 커밋 완료

## 8.3.2. 체크인

리소스에 대한 수정이 끝나면 체크인 과정을 거쳐야 한다. 체크인이 완료된 상태에서는 편집 권한이 없어진다. 추후 **[체크아웃]**하여 수정 권한을 획득할 수 있다. 체크인의 대상의 리소스는 우측에 숫자(1이상)가 있어야 하며, **[V]** 표시가 있는 것들이다. 이러한 리소스가 체크인 대상이 된다.

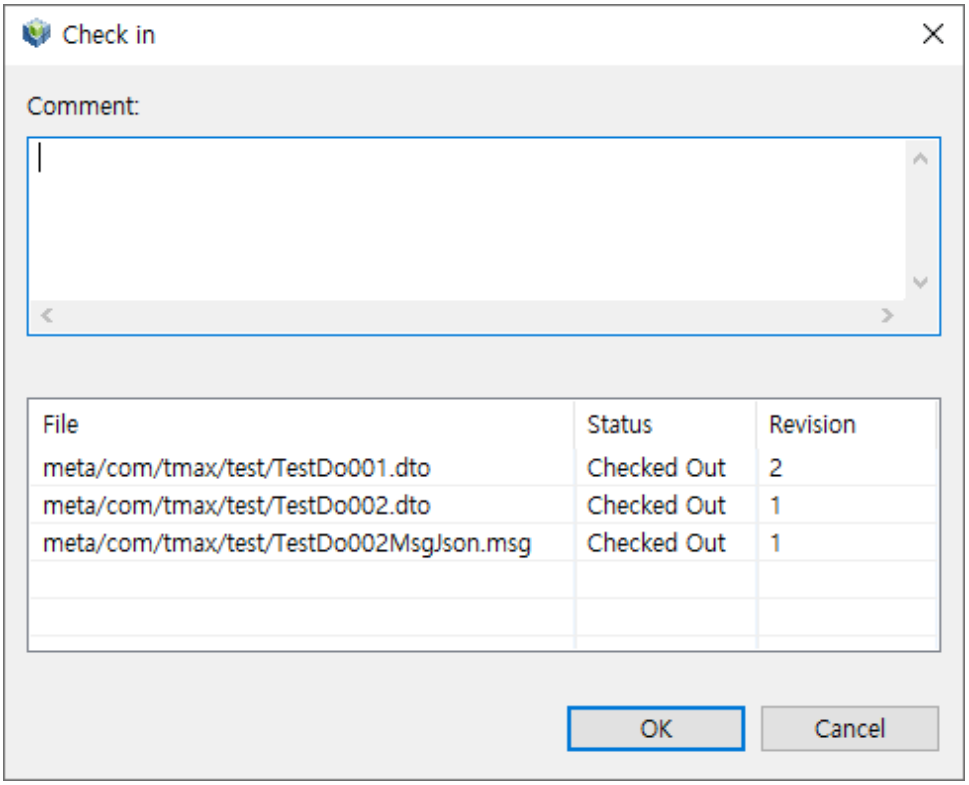
다음은 체크인하는 과정에 대한 설명이다.

1. 체크인할 리소스를 선택(복수 선택 가능)한 후 컨텍스트 메뉴에서 **[Team] > [체크인(N)]**를 선택한다.



리소스 체크인 - 리소스 선택 및 처리

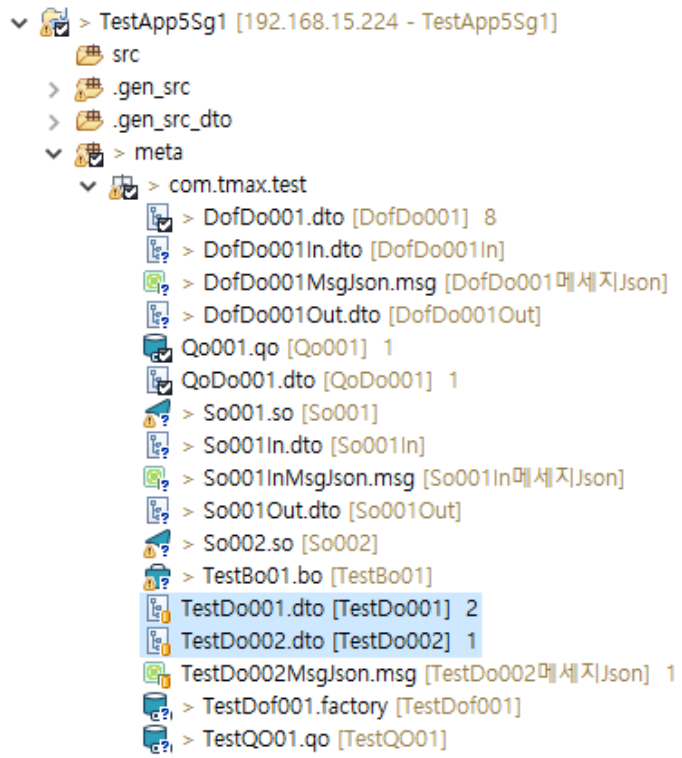
2. Comment를 입력하고 **[OK]** 버튼을 클릭하면 체크인이 수행된다.




리소스 체크인 - 체크인 처리

| 항목 | 설명 |
|---|---|
| Comment | 체크인 설명을 입력한다. |
| File | Package path 및 리소스명을 나타낸다. |
| Status | 현재 상태가 Check Out, Check In 상태인지를 나타낸다. |
| Revision | 현재 리소스의 리비전을 나타낸다. 커밋 할때마다 1씩 증가한다. |

3. 정상적으로 체크인되면, 아래와 같이 네비게이터의 리소스에 **[V]** 표시가 없어진다.



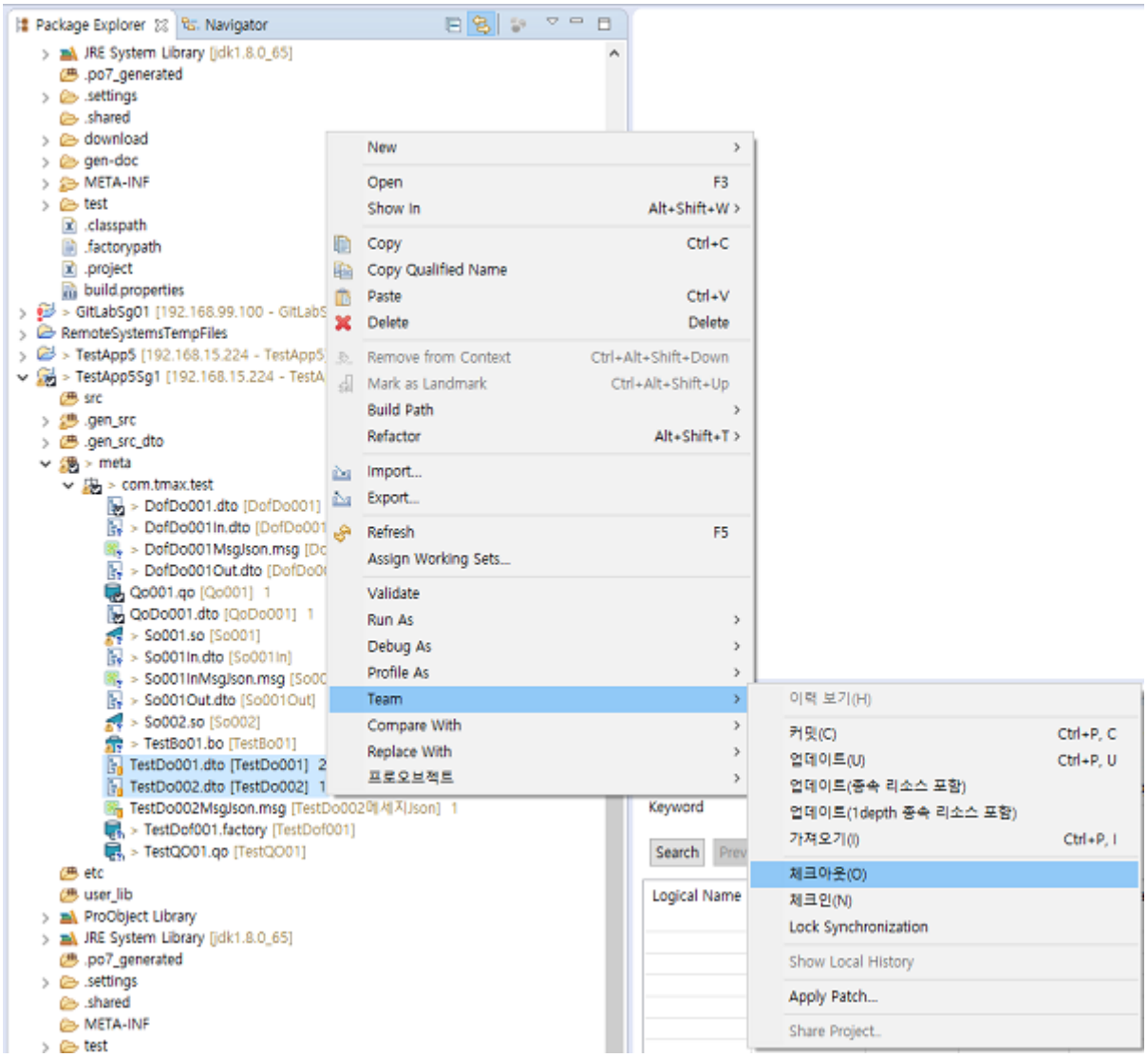

리소스 커밋 - 커밋 완료

## 8.3.3. 체크아웃

체크아웃은 리소스의 편집 권한을 획득 할 수 있는 작업이다. 체크인된 리소스는 모두 체크아웃 작업을 진행해야 편집 권한을 가진다. 체크아웃 대상의 리소스는 우측에 숫자(1이상)가 있어야 하며, **[V]** 표시가 없어야 한다. 이러한 리소스가 체크아웃 대상이 된다.

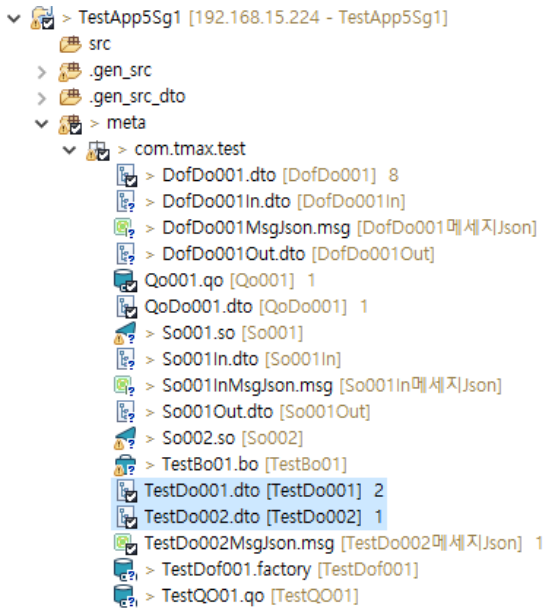
다음은 체크아웃하는 과정에 대한 설명이다.

1. 체크아웃할 리소스를 선택(복수 선택 가능)한 후 컨텍스트 메뉴에서 **[Team] > [체크아웃(O)]**를 선택한다.



리소스 체크아웃 - 리소스 선택 및 처리

2. 정상적으로 체크아웃되면 아래와 같이 네비게이터의 리소스에 **[V]** 표시가 된다.



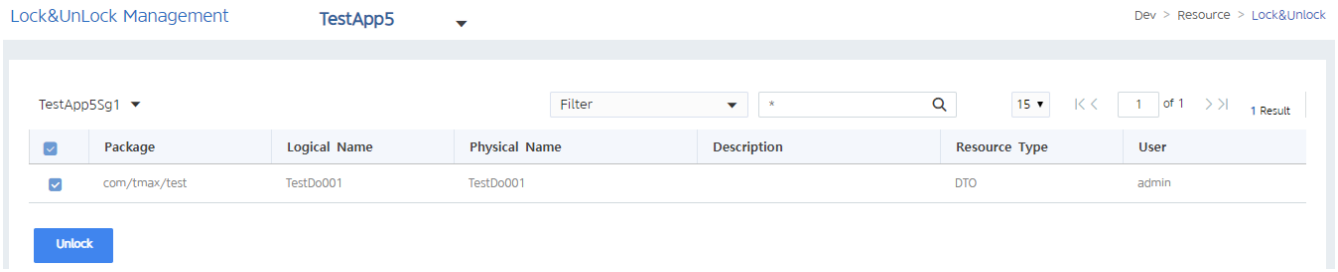
리소스 체크아웃 - 체크아웃 완료

## 8.3.4. Lock Synchronization

Lock Synchronization 기능은 ProManager에 의해 실제로는 Check In(Unlock) 상태이지만, Resource Tree에 보이는 내용은 CheckOut(상태일 경우)에 Sync를 맞추기 위해서 제공하는 기능이다.

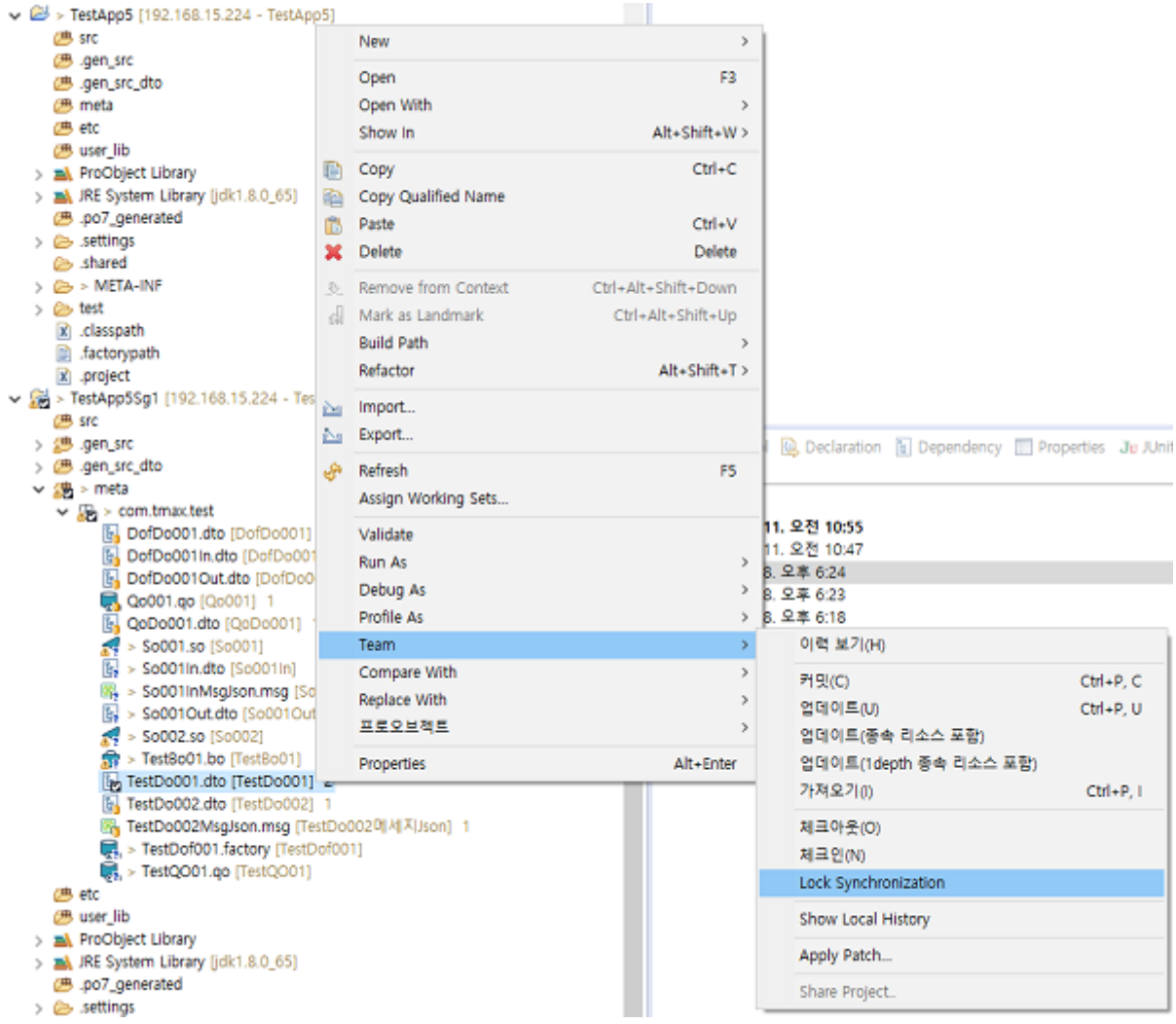
예를 들어 체크아웃한 상태로 장기간 자리 비움 상태일 경우에는 해당 리소스의 편집을 할 수 없다. 이를 보완하기 위해 강제로 체크아웃(Lock)상태를 해제할 수 있다.

**ProManager**에서 체크아웃(Lock)상태를 강제로 해제한다.



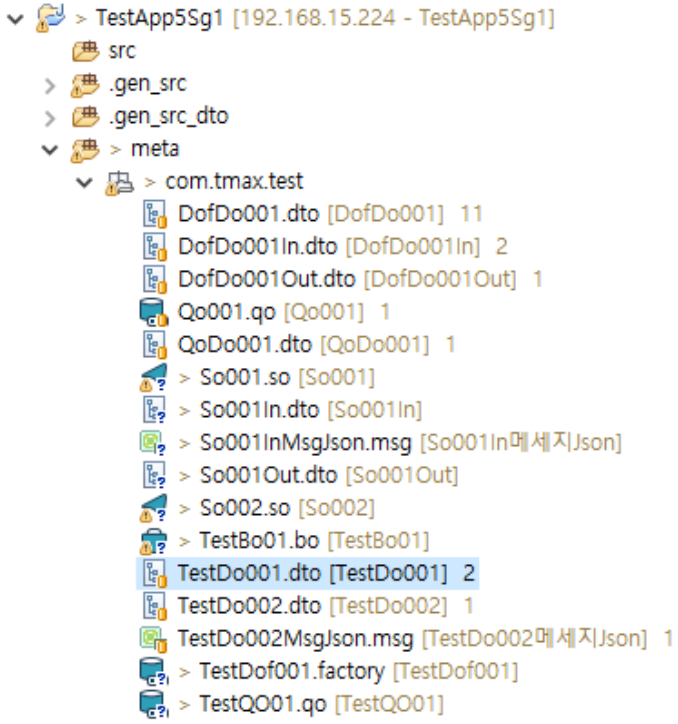

Lock Sync - Lock 해제

**ProStudio**의 리소스는 체크아웃 상태이며, **[Team] > [Lock Synchronization]**를 클릭한다.



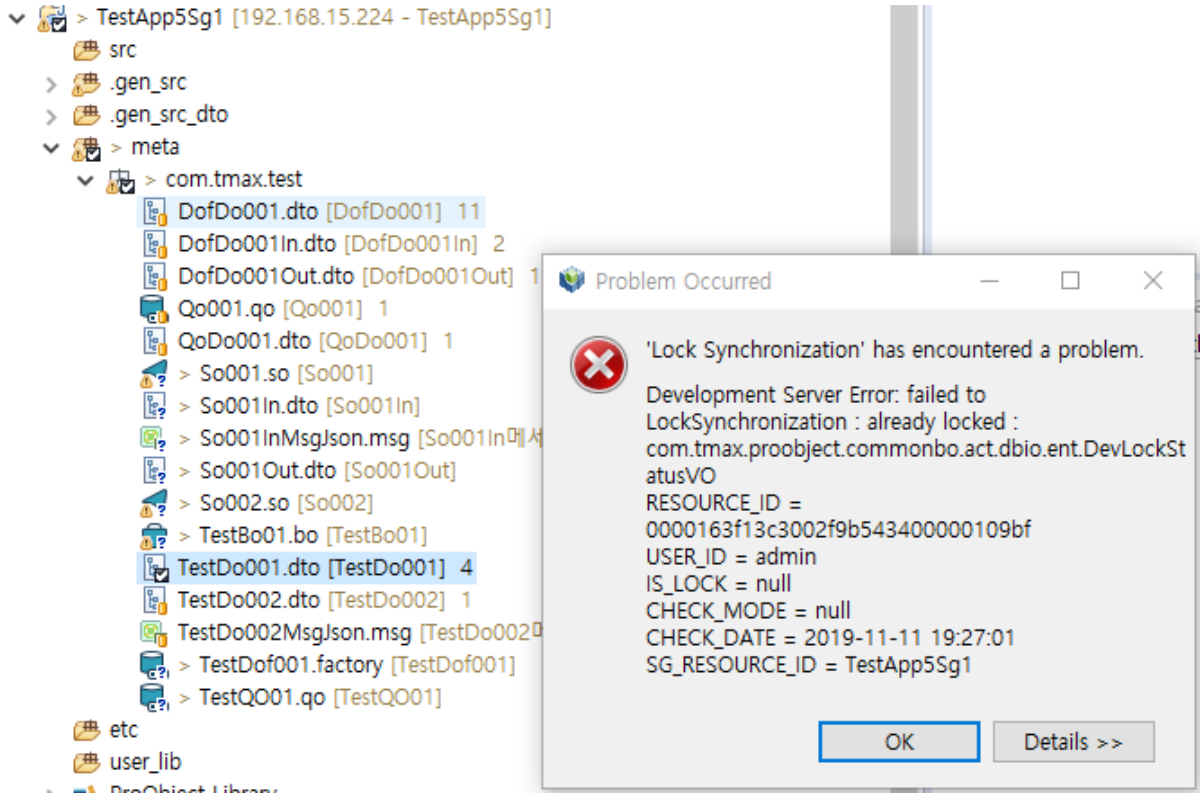

Lock Sync - Lock Sync 작업 수행

아래와 같이 체크아웃(Lock)이 해제되어 체크인 상태가 되었다는 것을 확인할 수 있다.




Lock Sync - Lock이 해제된 상태

현재 리소스의 Lock(체크아웃) 상태인 경우 상세 정보 확인을 위해서도 사용된다.




Lock Sync - 상태정보 표시

# 8.4. Project 버전관리

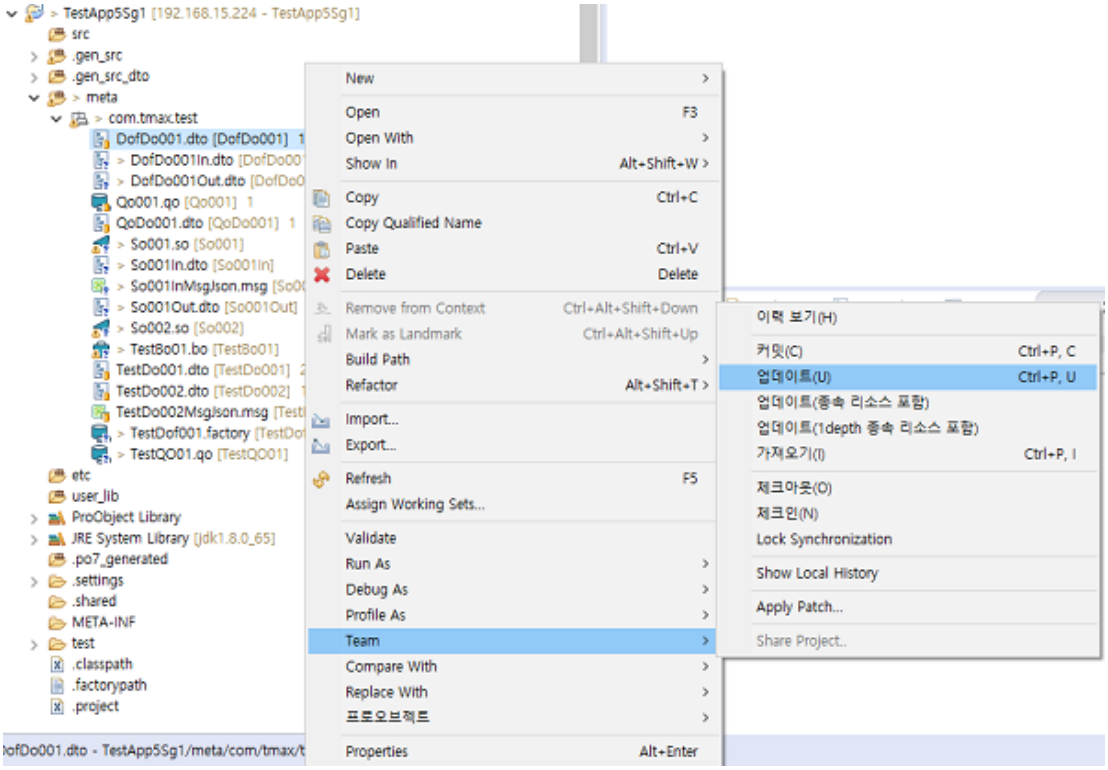
**[Resource 가져오기]** 기능을 사용하면 프로젝트의 최신 리소스를 DB(Server)에서 가져 올 수 있다. 또한 요건이 변경된 리소스의 편집을 위해서는 ProStudio의 Local 리소스를 DB(Server)의 리소스로 맞추고 시작해야 하는데 이

경우 업데이트 기능을 사용한다.

본 절에서는 DB에 저장된 리소스를 Local로 가져오거나 Local에 있는 리소스를 현행화하는 기능에 대해 설명한다.

## 8.4.1. 업데이트

현행화할 리소스를 선택한 후 컨텍스트 메뉴에서 **[Team] > [업데이트]**를 선택하면 DB(Server)에 있는 리소스를 Local로 현행화한다.

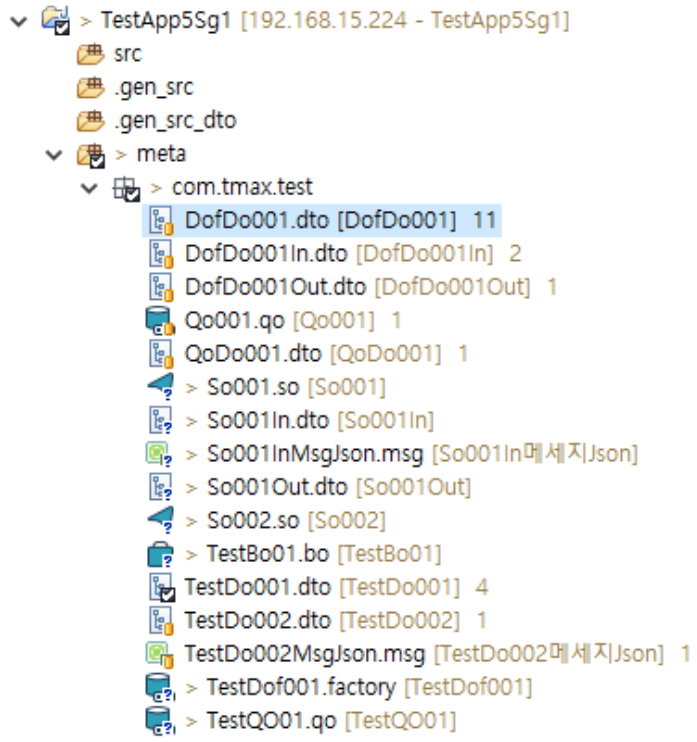


리소스 업데이트 - 업데이트 실행

업데이트의 유형은 다음과 같이 3가지가 있다.

| 구분 | 설명 |
|---|---|
| 업데이트(U) | 선택된 리소스만 현행화한다. |
| 업데이트(종속 리소스 포함) | 종속성이 있는 리소스 내역은 모두 현행화한다. |
| 업데이트(1depth 종속 리소스 포함) | 종속성이 있는 1Depth의 리소스만 현행화한다. |

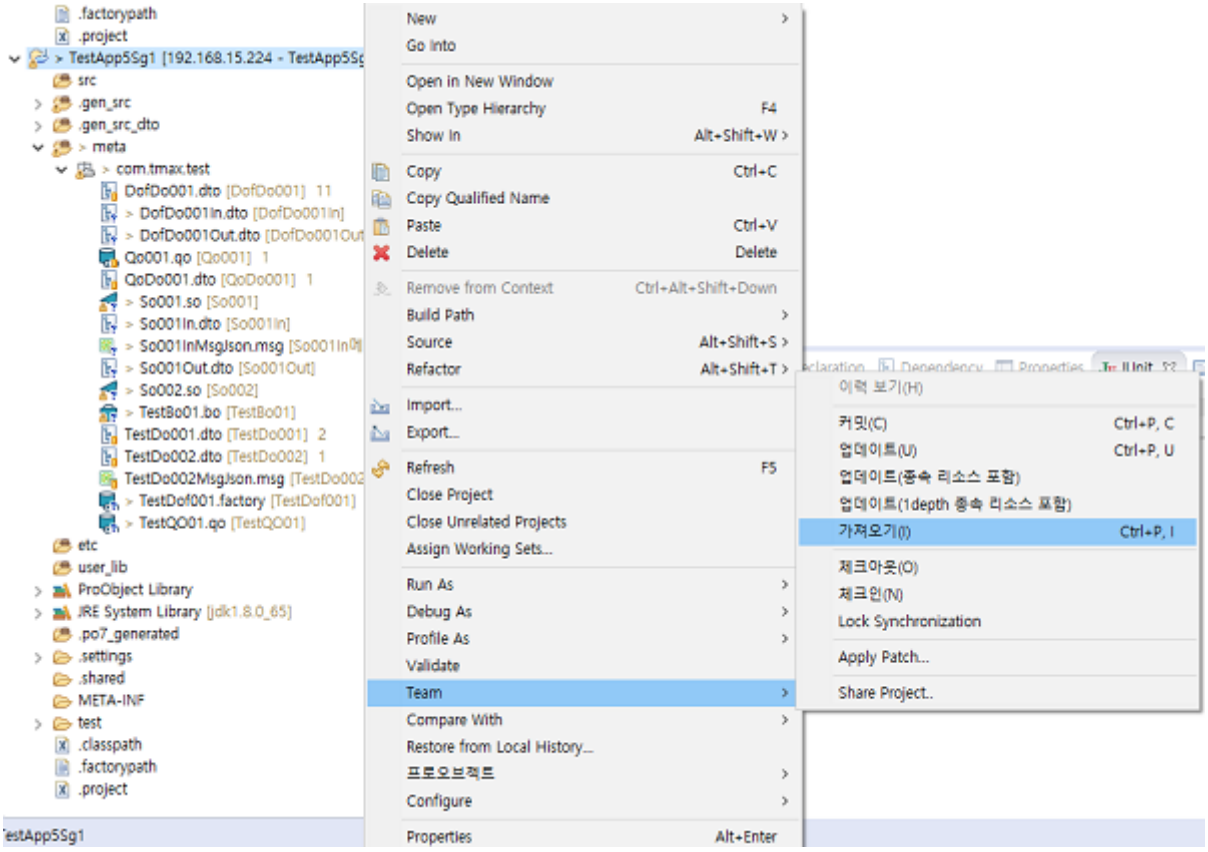
업데이트가 완료되면 최신 리비전으로 현행화된다.




리소스 업데이트 - 현행화 결과

## 8.4.2. 리소스 가져오기

프로젝트를 선택한 후 컨텍스트 메뉴에서 **[Team] > [가져오기(I)]**를 기능을 사용하면 프로젝트의 최신 리소스를 DB(Server)에서 가져 올 수 있다.




Resource 가져오기

# 8.5. 기타 기능

본 절에서는 DB 형상관리의 부가 기능에 대해 알아본다.

## 8.5.1. 커밋 시 컴파일 Skip기능

다량의 리소스를 한번에 커밋할 때 컴파일을 Skip하여 커밋 시간을 절약할 수 있다. 추후 일괄 빌드 등의 작업은 필요하다.

- 방법 1)

  $PROOBJECT_HOME/system/PoDevSvr.xml에 SKIP_COMPILE_IN_COMMIT를 TRUE로 설정한다.

```
<?xml version="1.0" encoding="EUC-KR" standalone="yes"?>
  <serverConfig xmlns="http://www.tmax.co.kr/proobject/serverConfig">
    :
   중략
    :
  <configField id="SKIP_COMPILE_IN_COMMIT" value="TRUE" type="String" xmlns=""/>
</serverConfig>
```

- 방법 2)

  $PROOBJECT_HOME/system/config/ProbuilderConfig.xml의
  commitActionWithoutCompileOption을 true로 설정한다.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
   <probuilder_config xmlns= .. 중략 ..
   <commitActionWithoutCompileOption xmlns="">true</commitActionWithoutCompileOption>
</probuilder_config>
```

# 9. AOP 개발 방법

본 장에서는 ProStudio에서 AOP(aspect oriented programming)의 aspect code를 작성하는 방법에 대하여 설명한다.

## 9.1. 개요

ProStudio에서 aspectj code를 작성을 위해 다음의 과정이 필요하다.

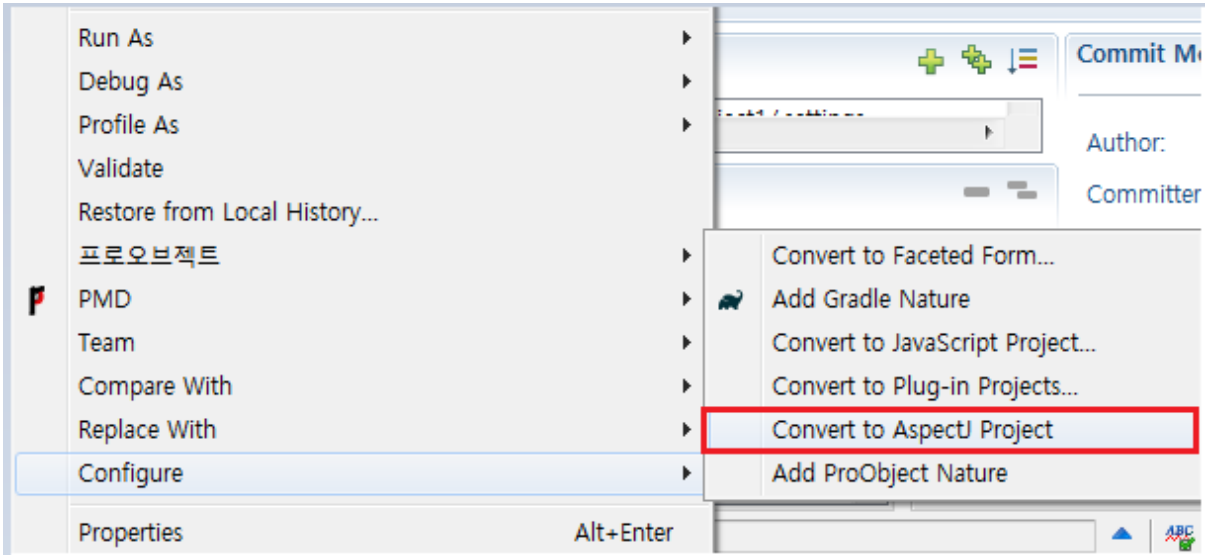
1. aspect 코드 환경설정
2. aspect code 작성
3. aspect 적용 확인

## 9.2. 환경설정

본 절에서는 aspectj code를 작성하기 위해 필요한 준비사항에 대해서 설명한다.

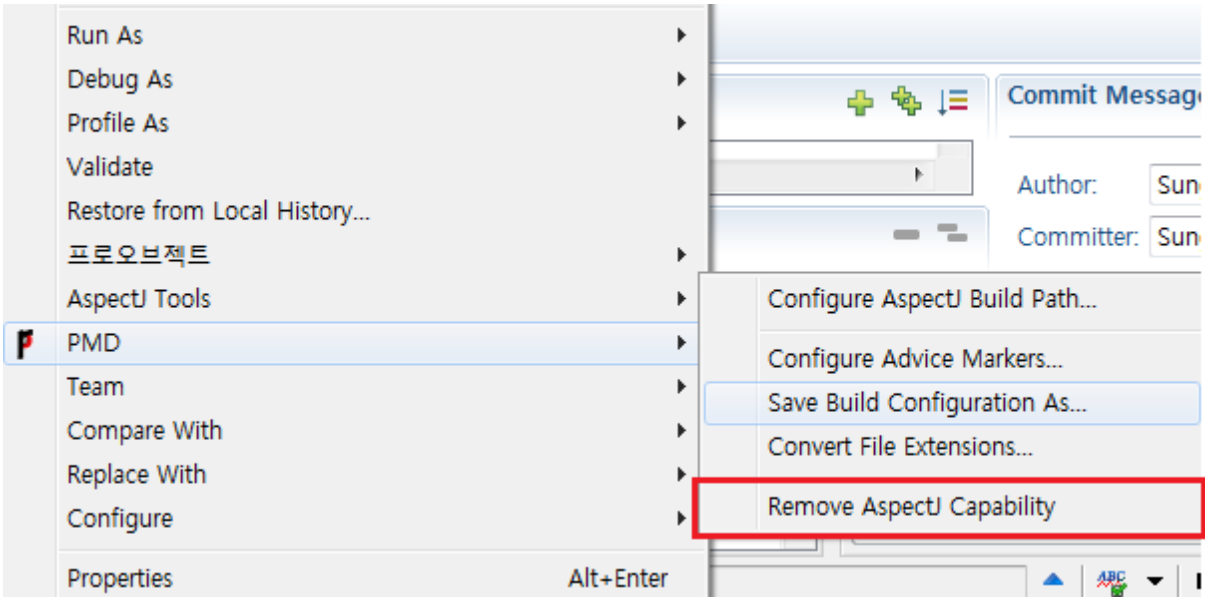
ProStudio에서 aspectj code를 작성하기 위해서는 ajdt(aspectj development tool) 플러그인이 설치되어 있어야 한다. 만약 ProStudio에 ajdt 플러그인 설치되어 있지 않은 경우 ajdt가 설치된 ProStudio를 받아서 진행해야 한다.

Project를 선택한 후 컨텍스트 메뉴에서 **[Configure] > [Convert to AspectJ Project]** 메뉴를 실행해서 aop code를 컴파일할 수 있는 프로젝트로 변경한다.




Convert to AspectJ Project

이후에 다시 일반 java 컴파일러를 사용하는 프로젝트로 변경하려면 Project를 선택한 후 컨텍스트 메뉴에서 **[Configure] > [Remove AspectJ Capability]** 메뉴를 실행해서 일반 java 컴파일러를 사용하는 프로젝트로 변경한다.



Remove AspectJ Capability

# 9.3. aspect code 작성

aspect code 작성을 위한 준비가 완료된 후엔 aspect code를 작성하는 방법, aspect code가 적용되는 target code를 작성한다. 본 절에서는 간단한 aspect 예제 코드를 통해 aop의 개념에 대해 설명한다.

다음은 aspect code를 작성하기 위해서 알아야 하는 AOP 용어들에 대한 정의이다.

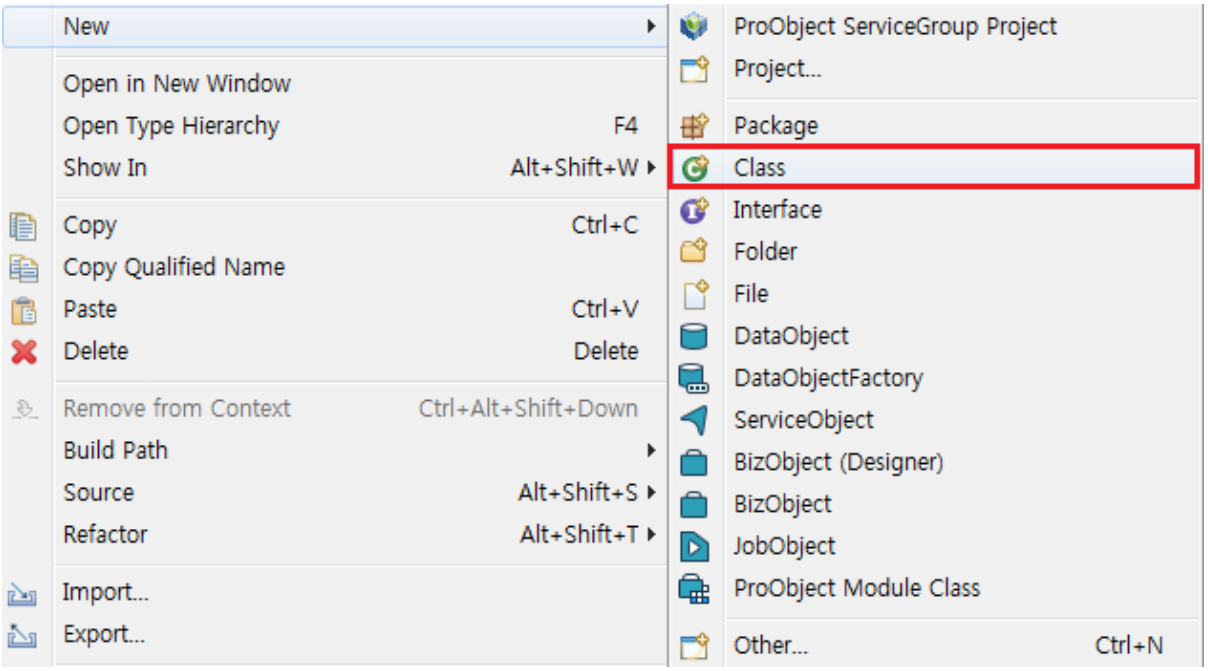
| 용어 | 설명 |
|---|---|
| Base code | aspect가 적용되지 않은 소스 코드로 aop 기능을 사용하지 않고 개발되는 모든 java 소스 코드이다. |
| Aspect code | base code에 적용될 코드이며 pointcut, advice를 포함한다. |
| Pointcut | base code에 aspect code가 적용되는 조건이다. 예를 들어 '특정 패키지에 속하는 클래스', '특정 method 호출' 등이 된다. |
| Advice | 특정 pointcut 조건이 만족되었을 때 실제 실행되는 코드들이며 실행 시점 정보까지 포함한다. |

### aspect code 예제

다음은 모든 BO에서 add method를 호출하기 전에 호출하는 객체가 List인지 확인 후에 size check하는 로직을 수행하는 aspect code 예제이다.

1. Aspect code 생성

   일반 java class를 생성한 후에 aspect code를 작성한다.



일반 java class 생성

일반 java class에 @Aspect annotation을 붙여서 aspect code를 만든다.

```
import java.util.List;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class SampleAspect {
```

2. Pointcut 작성

```
@Pointcut("within(com.tmax.proobject.model.business.BusinessObject+)")
public void BO() {

}

@Pointcut("call(* add*(..))")
public void add() {

}
```

첫 번째 pointcut은 BusinessObject의 구현체 클래스만 대상으로 하는 조건이고, 두 번째 pointcut은 add method를 호출하는 모든 클래스를 대상으로 하는 조건이다. pointcut을 작성하는 구체적인 방법 및 다른 pointcut을 사용하고자 한다면 AspectJ 프로그래밍 가이드를 참고한다.

3. Advice 작성

Advice에는 method annotation으로 실행 위치를 표시한다. @Before, @Around, @After 등... 이 adviec에는 before가 사용되었으므로 pointcut 대상 조건의 실행 전 advice 내부 코드를 실행하게 된다.

@Before 내부의 value에는 pointcut을 명시한다. 이 예에서는 위에서 만든 pointcut들이 and 조건으로 연결되어 있다. 따라서 이 advice는 BusinessObject의 구현체 클래스에서 add method를 호출하기 이전에 실행된다.

advice에 구현된 코드를 자세히 살펴보면 다음과 같다.

```
@Before(value = "BO() && add()")
public void beforeMethod(JoinPoint joinPoint) throws Exception {
    Object ret = null;
    System.out.println("before advice" + joinPoint.getTarget());

    if(joinPoint.getTarget() instanceof List) {
        List thisList = (List)joinPoint.getTarget();
    System.out.println("before add method, ArrayList size: " + thisList.size());

    if(thisList.size() > 10000)
        throw new Exception();
   }
}
```

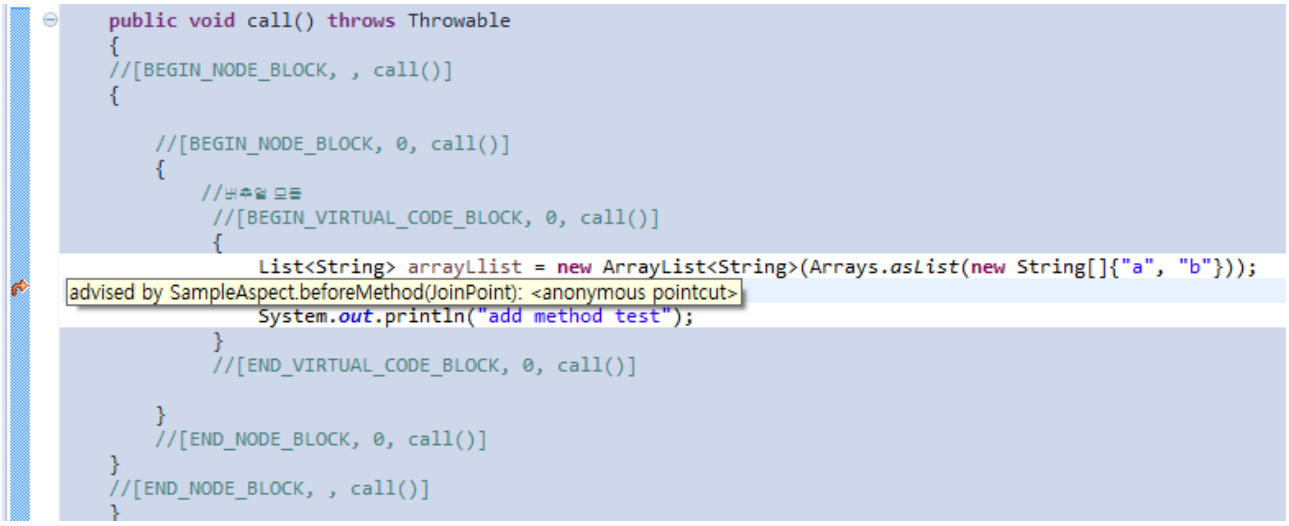
joinPoint라는 객체에서 aop가 적용된 target 객체를 보고 이 target 객체가 java.util.List의 형태이면 target List의 size를 확인 후에 size가 10000보다 크면 exception을 낸다.

4. target code에서 AOP 적용 확인

만약 Aspect code가 정상적으로 작성되었다면 AOP가 적용될 대상 코드에서 확인이 가능하다.

```
@BizObject(logicalName = "TestBO")
public class TestBO implements BusinessObject
{

    private ProObjectLogger logger = ServiceLogger.getLogger();



    public void call() throws Throwable
    {
    //[BEGIN_NODE_BLOCK, , call()]
    {

        //[BEGIN_NODE_BLOCK, 0, call()]
        {
            //부추일 모름
             //[BEGIN_VIRTUAL_CODE_BLOCK, 0, call()]
             {
                 List<String> arrayLlist = new ArrayList<String>(Arrays.asList(new String[]{"a", "b"}));
                 arrayLlist.add("c");
                 System.out.println("add method test");
             }
             //[END_VIRTUAL_CODE_BLOCK, 0, call()]

        }
        //[END_NODE_BLOCK, 0, call()]
    }
    //[END_NODE_BLOCK, , call()]
    }

}
```

aspect code 예제 (1)



aspect code 예제 (2)

### aspect code가 적용되는 target code 작성 예제

다음은 com.tmax.bo 패키지에 속하는 모든 클래스들의 모든 method들 중에서 m으로 시작하는 method 호출 전, 후에 로그를 찍고 exception 처리를 하는 aspect code 예제이다.

1. Aspect code 생성

   일반 java class에 @Aspect annotation을 붙여서 aspect code를 만든다.

```
import java.util.List;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class SampleAspect {
```

2. Pointcut 작성

   첫 번째 pointcut은 com.tmax.bo 패키지에 속하는 모든 클래스의 모든 메소드들을 대상으로 하는 조건이다.

   pointcut value에 대하여 좀더 설명하자면 다음과 같다. execution(returnType pacakge.class.methodName(arguments)) return type, class, method 등에 '*'으로 표시가 된 것은 모든 것을 대상으로 삼겠다는 것이고, argument type에서는 '..'이 모든 것을 대상으로 한다는 것을 의미한다.

   두 번째 pointcut은 이름이 'm'으로 시작하는 모든 method를 대상으로 삼는 조건이다.

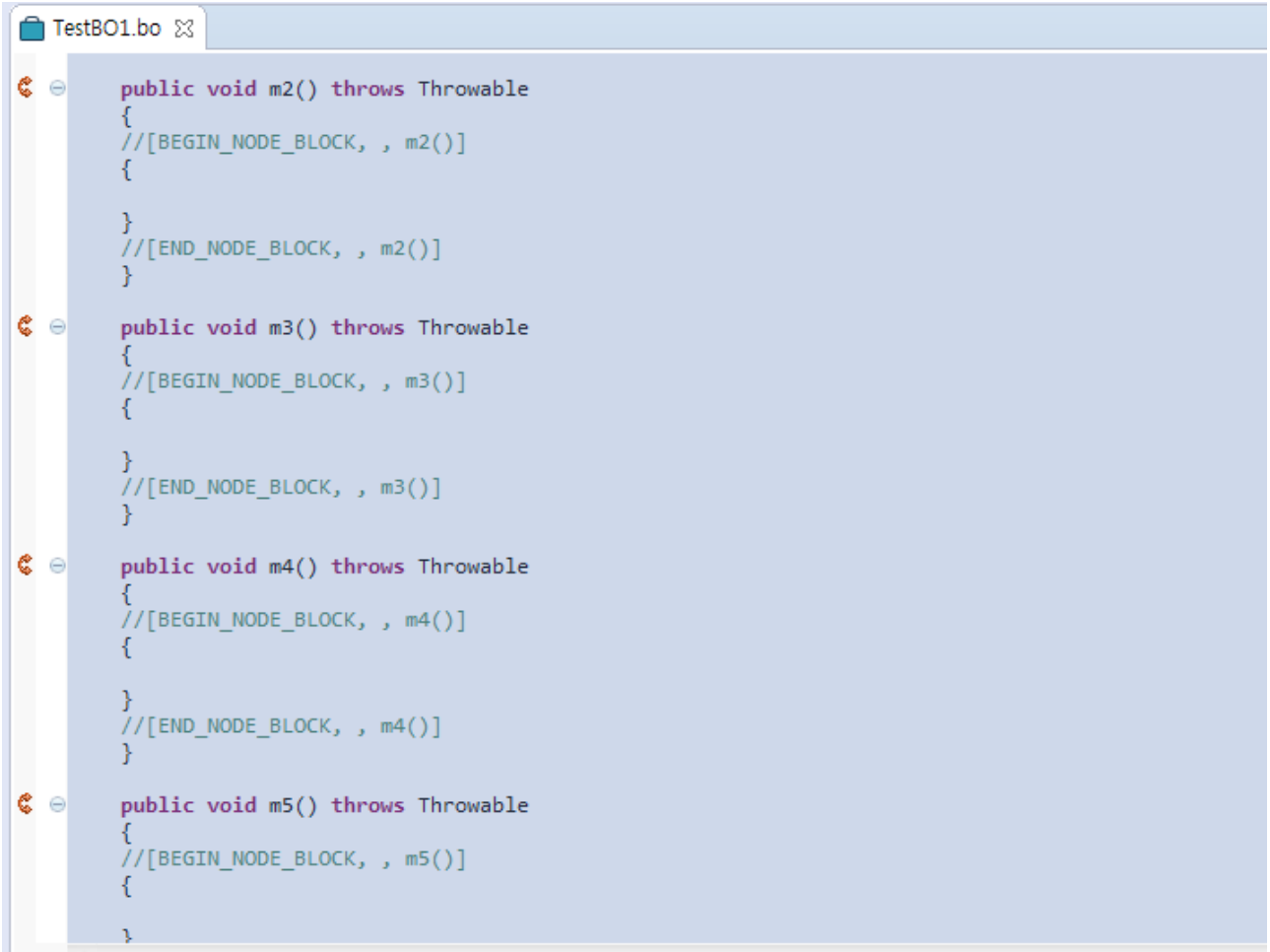
```
@Pointcut("execution(* com.tmax.bo.*.*(..))")
public void anyBizOperation() {
}

@Pointcut("execution(* m*(..))")
public void startsWithM() {
}
```

3. Advice 작성

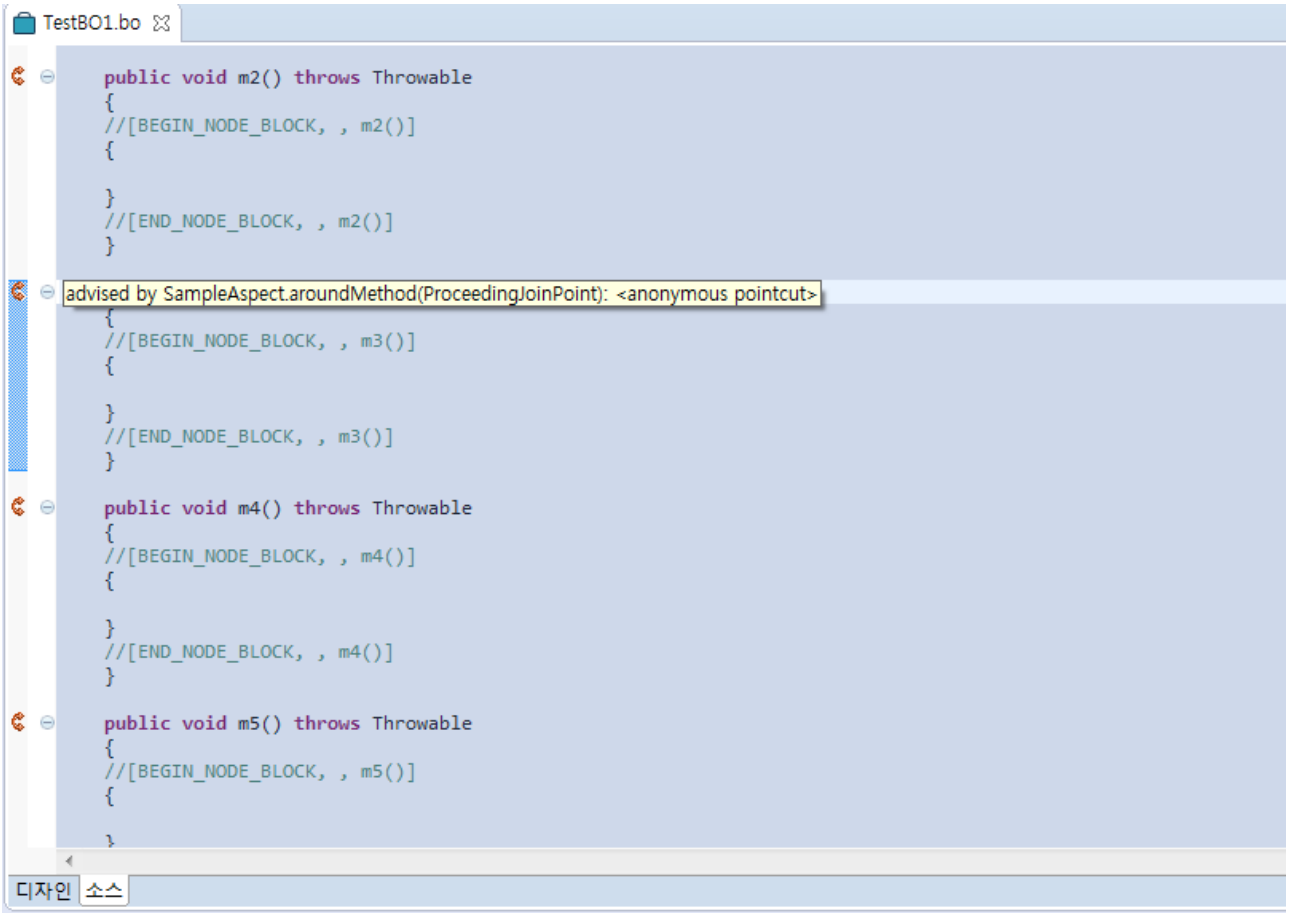
   Around advice에서는 advice code 내부에서 실행 위치를 결정한다. 실제 target method가 실행되는 위치가 thisJoinPoint.proceed()가 호출되는 시점이다. 따라서 target method 호출 전 후로 로그를 찍을 수 있고 exception 처리도 가능하다.

```
@Around(value = "anyBizOperation() && startsWithM()")
public Object aroundMethod(ProceedingJoinPoint thisJoinPoint){
    Object ret = null;
    try{
    System.out.println("around advice before: " + thisJoinPoint.getSignature());
    ret = thisJoinPoint.proceed();

    System.out.println("around advice after: " + thisJoinPoint.getSignature());
    }catch(Throwable e){
        System.out.println("catch Throwble in Aspect: " + e.getMessage());
    }
  return ret;
}
```

4. target code에서 AOP 적용 확인




target code에서 AOP 적용 확인 (1)



target code에서 AOP 적용 확인 (2)

## 9.4. 서버 빌드과정에서 AOP 적용

본 절에서는 target code에 aspect code가 정상적으로 적용되었는지 확인하는 방법을 설명한다.

gradle script file의 ajDir로 지정된 폴더에 작성된 aspect code를 위치시키고 빌드를 한다.

```
17 // variables
18 ext {
19
20     //
21     javaDtoDir = "${sgName}/.gen_src/dto"
22     javaDir = "${sgName}/.gen_src"
23     pojoSrcDir = "${sgName}/src"
24
25     appJavaDtoDir = "${appName}/.gen_src/dto"
26     appJavaDir = "${appName}/.gen_src"
27     appPojoSrcDir = "${appName}/src"
28
29     //
30     ajDir = "${poHome}/system/devops/aj"
31
```

build.gradle

# Appendix A: 확정점 플러그인 설치 및 구현

본 부록에서는 확정점 플러그인 설치 및 구현 방법에 대해서 설명한다.

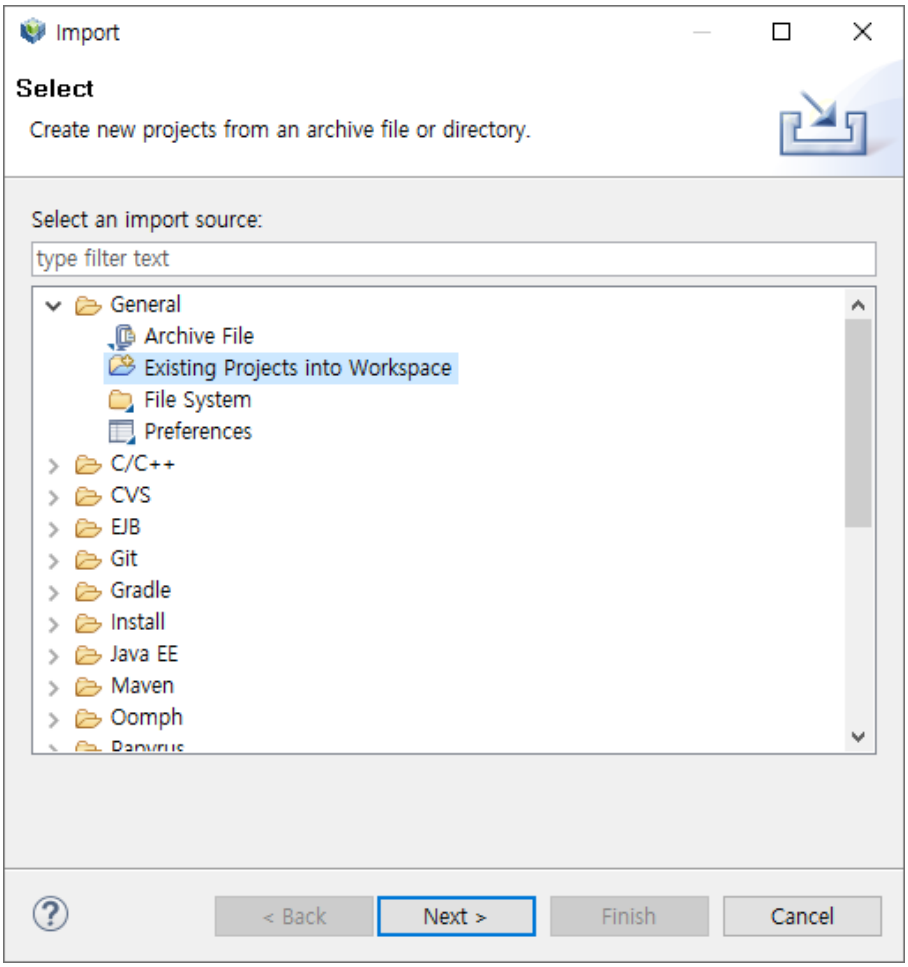
## A.1. Extension 등록 방법

본 절에는 Extension을 등록하는 방법에 대한 설명이다.

### A.1.1. Document project 지정

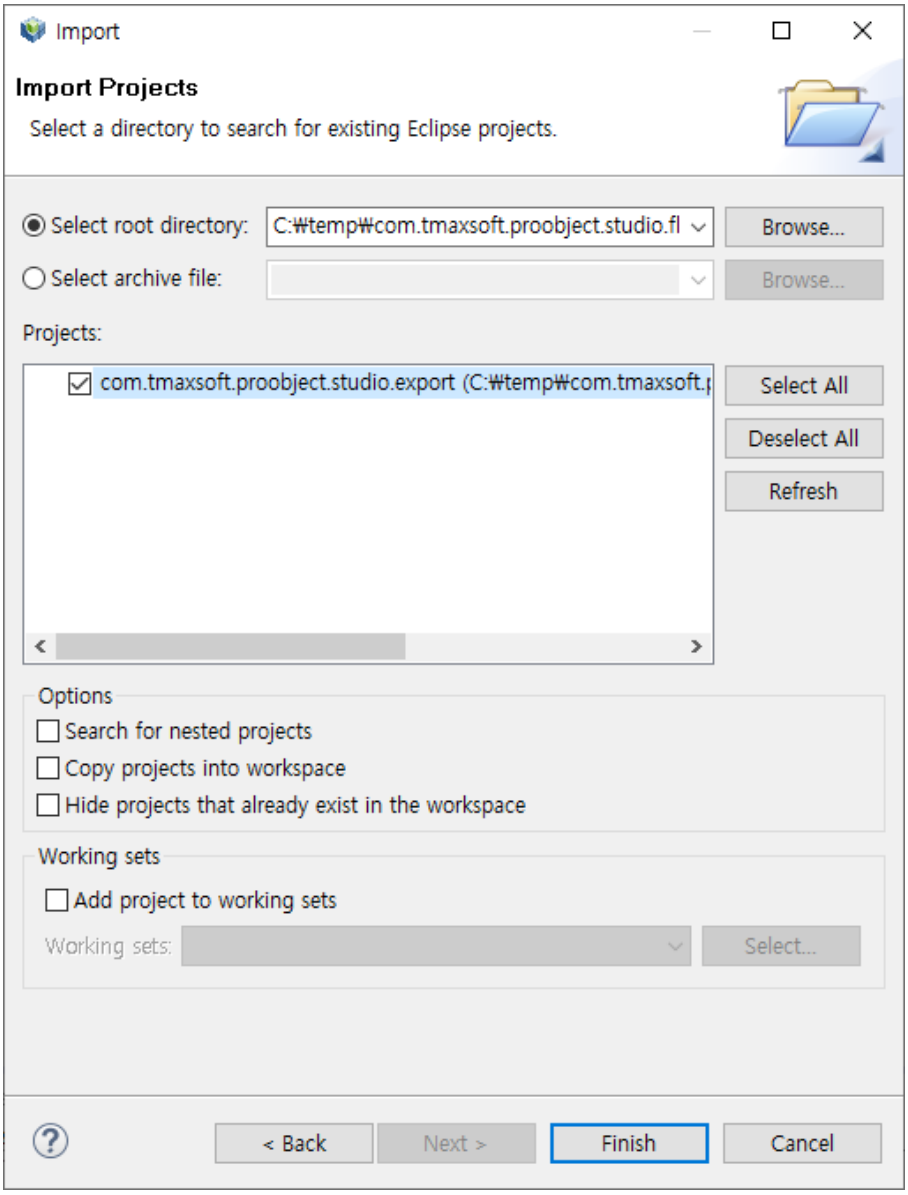
sample로 제공한 com.tmaxsoft.proobject.studio.flow.exporter.sample_src 폴더에 Document project를 지정한다.

1. **Package Explorer**의 컨텍스트 메뉴에서 **[Import]**를 선택한다.
2. **'Existing Projects into Workspace'**를 선택한 후 **[Next]** 버튼을 클릭한다.




Document project 지정 (1)

3. 제공된 샘플 플러그인 프로젝트 선택한 후 원하는 포맷에 맞게 코드를 작성한 후 **[Finish]** 버튼을 클릭한다.

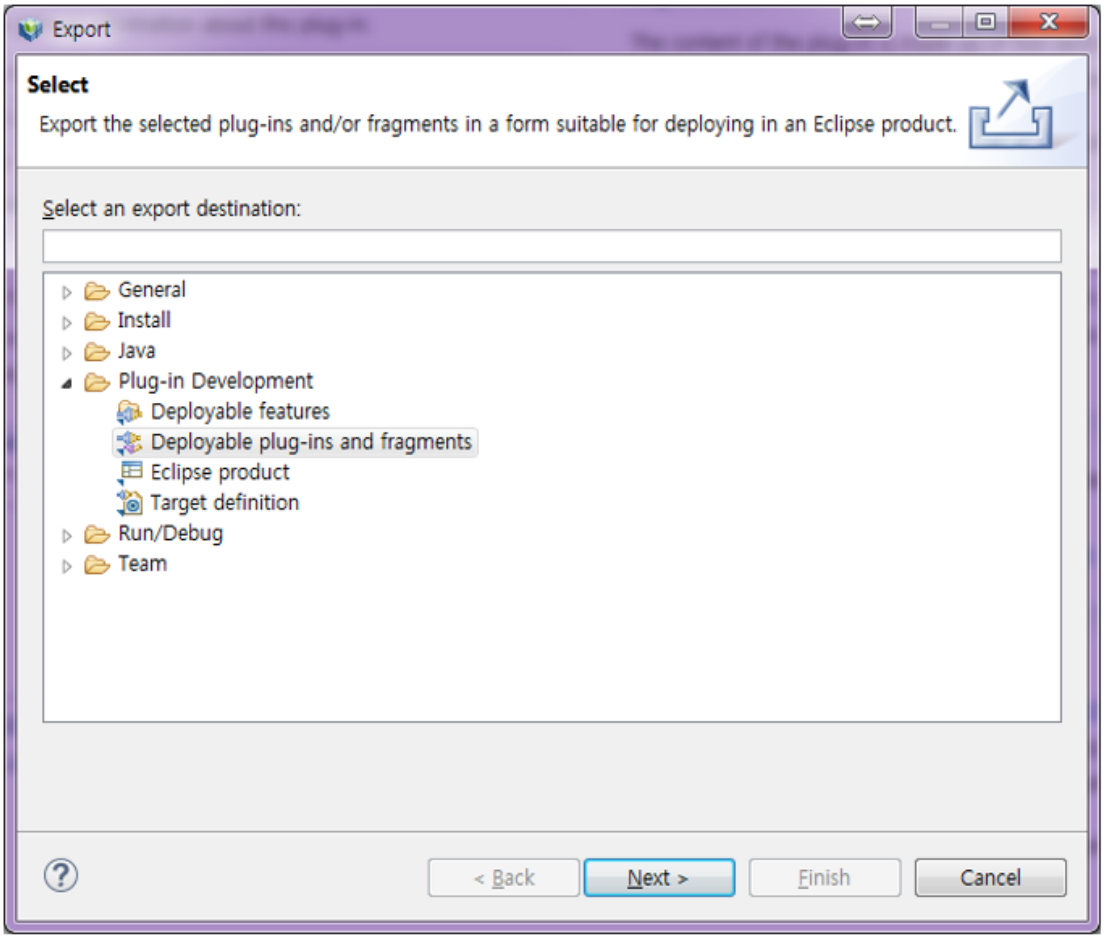



Document project 지정 (2)

## A.1.2. 플러그인 설치 및 배포

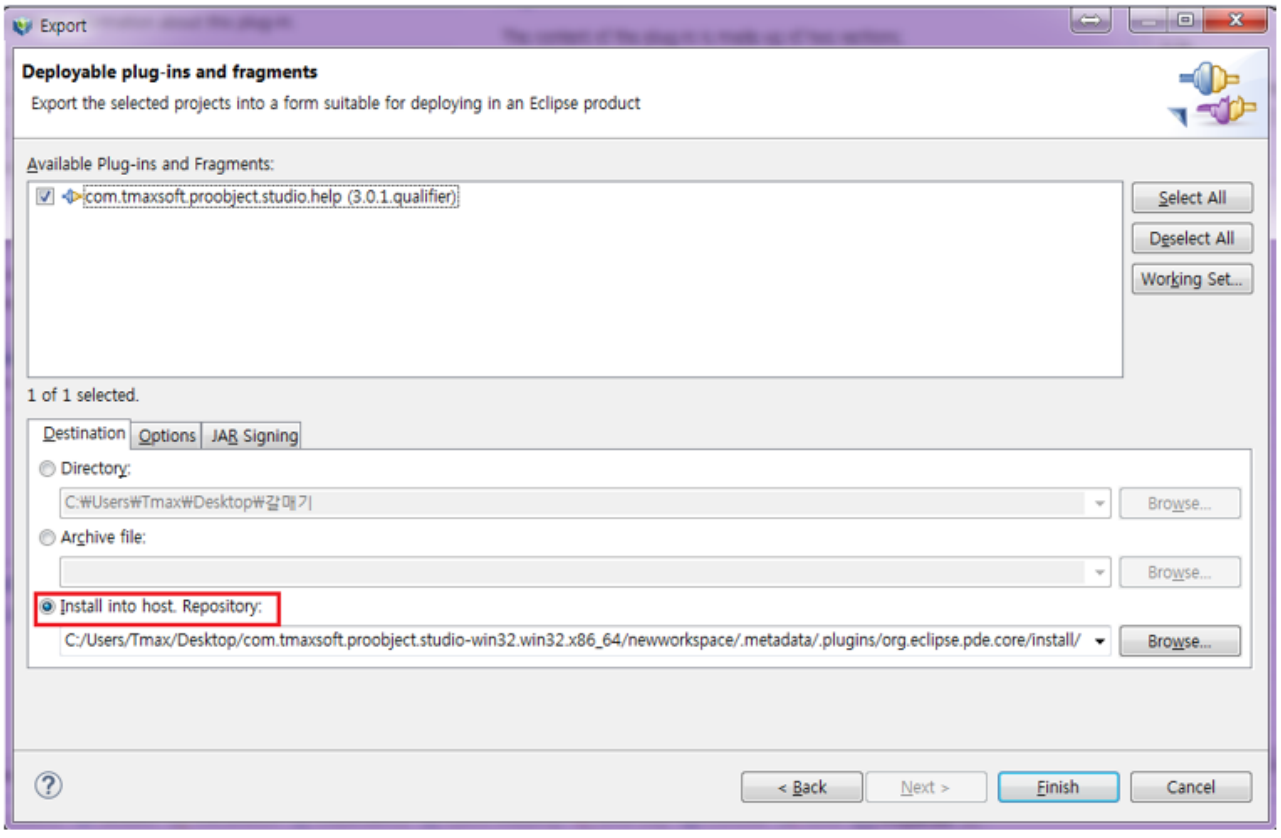
다음은 플러그인을 설치 및 배포하는 과정에 대한 설명이다.

1. **Package Explorer**의 컨텍스트 메뉴에서 **[Export]**를 선택한다.
2. **'Deployable plug-ins and fragments'**를 선택한 후 **[Next]** 버튼을 클릭한다.



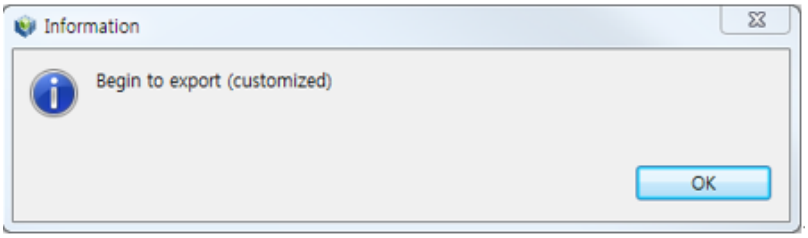

플러그인 설치 및 배포 (1)

3. 'com.tmaxsoft.proobject.studio.export'를 체크한 후 'install into host, Repository'를 선택하고 **[Finish]** 버튼을 클릭한다.




플러그인 설치 및 배포 (2)

4. 설정 후 ProStudio를 재시작한다.
5. **[산출물 내보내기]** 아이콘을 클릭한 후 다음과 같은 다이얼로그가 나오면 설정이 완료된다.



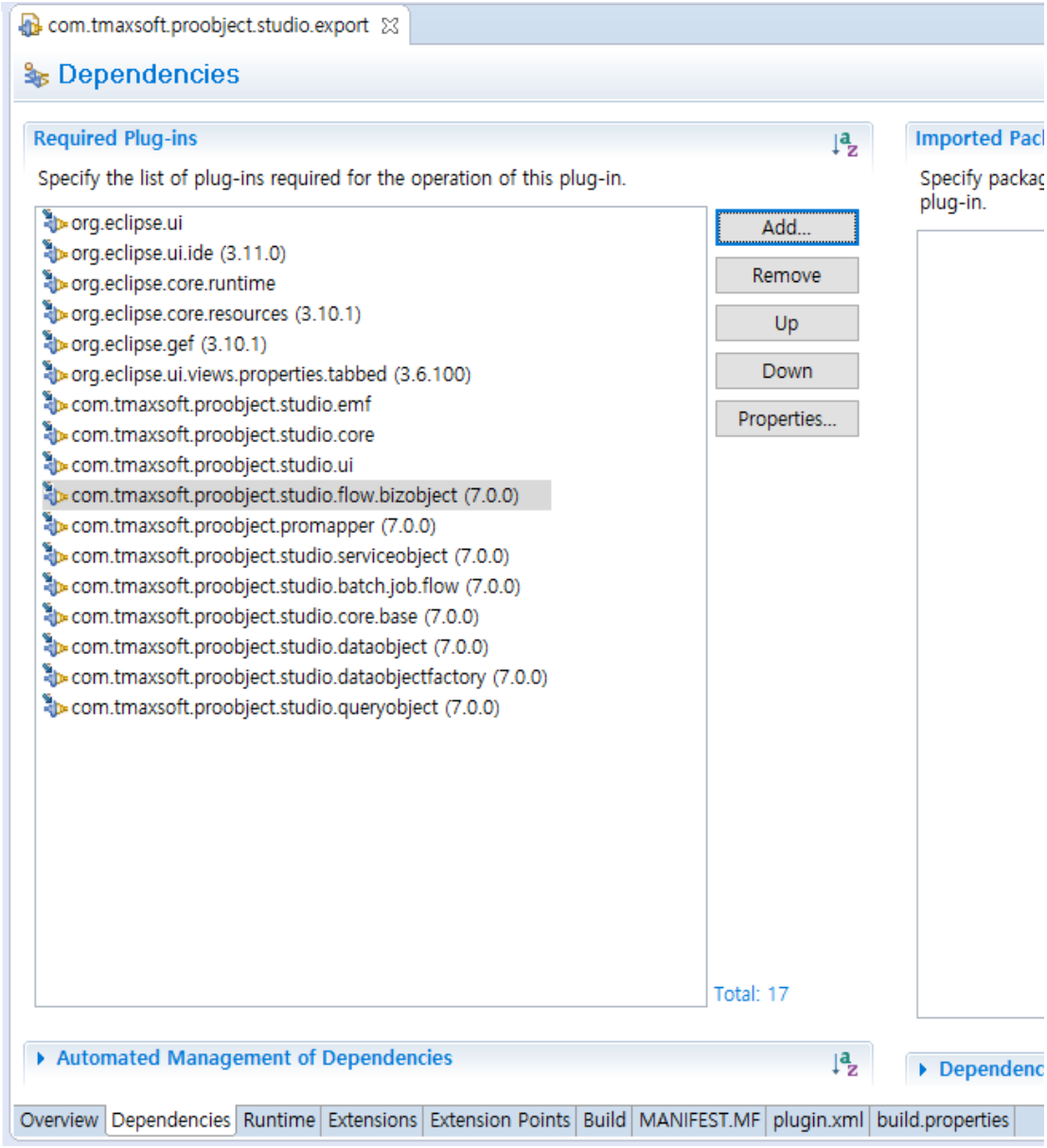

플러그인 설치 및 배포 (3)

# A.2. 문제해결

다음과 같이 Import project 시 문제가 에러가 발생하는 경우 해결방법에 대해서 설명한다.

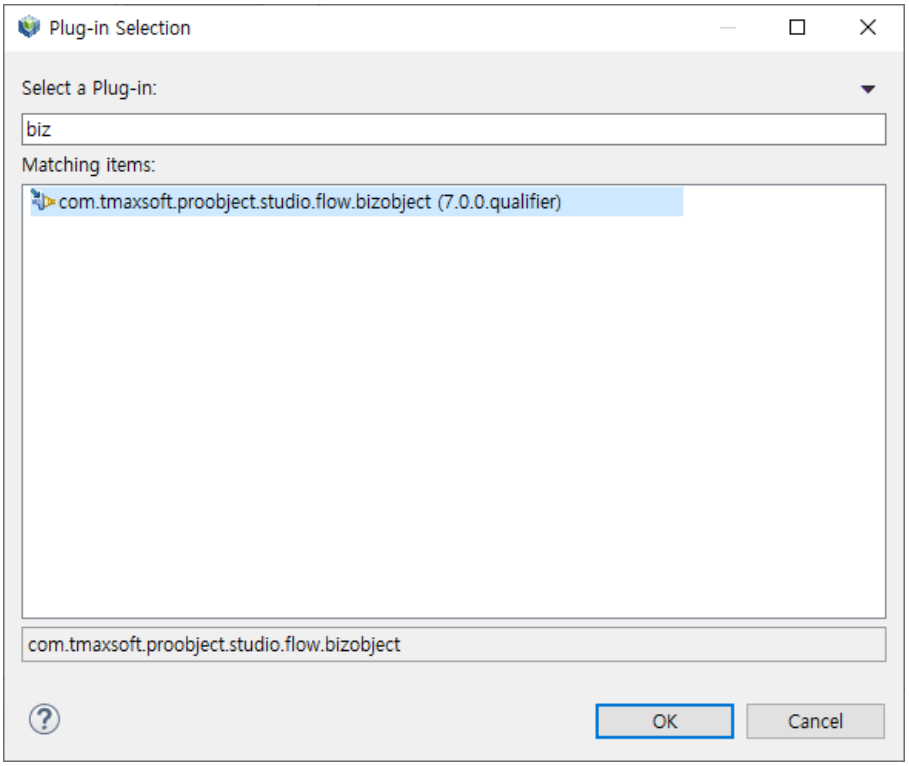
```
10 import com.tmaxsoft.proobject.studio.flow.bizboject.action.export.IDocumentBuilder;
11 import com.tmaxsoft.proobject.studio.flow.bizboject.action.export.visitor.BizObjectVisitor;
12 import com.tmaxsoft.proobject.studio.flow.bizboject.action.export.visitor.DataObjectFactoryVisitor;
13 import com.tmaxsoft.proobject.studio.flow.bizboject.action.export.visitor.DataObjectVisitor;
14 import com.tmaxsoft.proobject.studio.flow.bizboject.action.export.visitor.JobObjectVisitor;
15 import com.tmaxsoft.proobject.studio.flow.bizboject.action.export.visitor.QueryObjectVisitor;
16 import com.tmaxsoft.proobject.studio.flow.bizboject.action.export.visitor.ServiceObjectVisitor;
17
18 public class WordDocumentBuilder implements IDocumentBuilder {
19
20     public WordDocumentBuilder() {
21         System.out.println("initialize WordDocumentBuilder");
22     }
23
24     @Override
25     public ServiceObjectVisitor buildSo(DocumentContext context) {
26         return new WordSoVisitor();
27     }
28
29     @Override
30     public BizObjectVisitor buildBo(DocumentContext context) {
31         return new WordBoVisitor();
32     }
33
34     @Override
35     public JobObjectVisitor buildJo(DocumentContext context) {
36         return new WordJoVisitor();
37     }
38
39     @Override
40     public DataObjectVisitor buildDo(DocumentContext context) {
41         return new WordDoVisitor();
42     }
```

Plugin.xml 파일의 dependencies tab의 com.tmaxsoft.proobject.studio.flow.bizobject를 지운다.



플러그인 설치 문제해결 (1)

아래의 플러그인을 새로 추가한 후 <Ctrl>+S를 누른다.



플러그인 설치 문제해결 (2)

# Appendix B: Property 검색 커스터마이징

본 부록에서는 ProStudio의 Property 검색 커스터마이징 기능에 대해서 설명한다. 해당 기능을 사용하기 위한 환경 구성은 TmaxSoft의 기술지원에 문의한다.

커스터마이징 Property 검색기능을 사용하기위해서는 별도 커스터마이징 가능한 애플리케이션을 PO에 세팅하고, 사용여부 설정이 필요하다. 배포된 애플리케이션의 SO는 Site 요건에 맞게 커스터마이징 가능하다.

적용 방법 및 활성화/비활성화 방법은 아래 내용을 참고한다.

1. 애플리케이션에서 사용하는 DB 스키마는 EXT_DEV_RSC_PROP 테이블을 생성한다.

```
CREATE TABLE EXT_DEV_RSC_PROP (
   RESOURCE_ID VARCHAR(64) NOT NULL,
   SEQ VARCHAR(10) NOT NULL,
   FIELD_TYPE VARCHAR(16),
   PHYSICAL_NAME VARCHAR(128),
   LOGICAL_NAME VARCHAR(128),
   RESOURCE_GROUP VARCHAR(128),
   INCLUDE VARCHAR(128),
   NODE_TYPE VARCHAR(32),
   META_ID VARCHAR(512),
   REFERENCE_ID VARCHAR(512),
   REVISION VARCHAR(8),
   LENGTH VARCHAR(128),
   ARRAY VARCHAR(128),
   COMMENTS VARCHAR(512),
   IS_PK VARCHAR(1),
   IS_KEY VARCHAR(1),
   IS_INDEX VARCHAR(1),
   ALLOW_NULL VARCHAR(1),
   DB_TYPE VARCHAR(16),
   DEFAULT_VALUE VARCHAR(128),
   SCHEMA VARCHAR(256),
   META_TYPE VARCHAR(32),
   TABLE_NAME VARCHAR(512),
   COLUMN_NAME VARCHAR(512),
   MASKING VARCHAR(128),
   MASKING_RANGE VARCHAR(128),
   ENCRYPT VARCHAR(128),
   DECIMAL_SIZE VARCHAR(128) DEFAULT '0',
   CREATE_TIME VARCHAR(32),
   UPDATE_TIME VARCHAR(32),
   CREATOR VARCHAR(128),
   MODIFIER VARCHAR(128),
   IS_USE VARCHAR(8) DEFAULT 'y'
)
```

2. PO 애플리케이션을 추가한다.

   $PROOBJECT_HOME/application Directory에 ExtApp.tar를 압축 해제하여, 커스터마이징 Property 조회한 후 애플리케이션을 설치한다.

```
[po7@pas99:/home/po7/proobject7/application]$tar -xvf ExtApp.tar
ExtApp/
ExtApp/common/
ExtApp/common/lib
ExtApp/servicegroup/
ExtApp/servicegroup/ExtSg/
ExtApp/servicegroup/ExtSg/ExtSg.jar
ExtApp/servicegroup/ExtSg/config/
ExtApp/servicegroup/ExtSg/config/servicegroup.properites.log
ExtApp/servicegroup/ExtSg/config/servicegroup.xml
ExtApp/servicegroup/ExtSg/config/servicegroup.properties
ExtApp/servicegroup/ExtSg/dto/
ExtApp/config/
ExtApp/config/application.properties
ExtApp/config/application.xml
ExtApp/config/application.properties.log
ExtApp/config/dbio_config.xml
```

기본애플리케이션을 수정해서 사용해야 할 경우에는 아래 Service Group의 리소스 중 PropertyGetList를 수정하여 재배포한다. 해당애플리케이션에서 사용하는 DO는 Default Property 조회에서 사용하는 것과 동일하다.

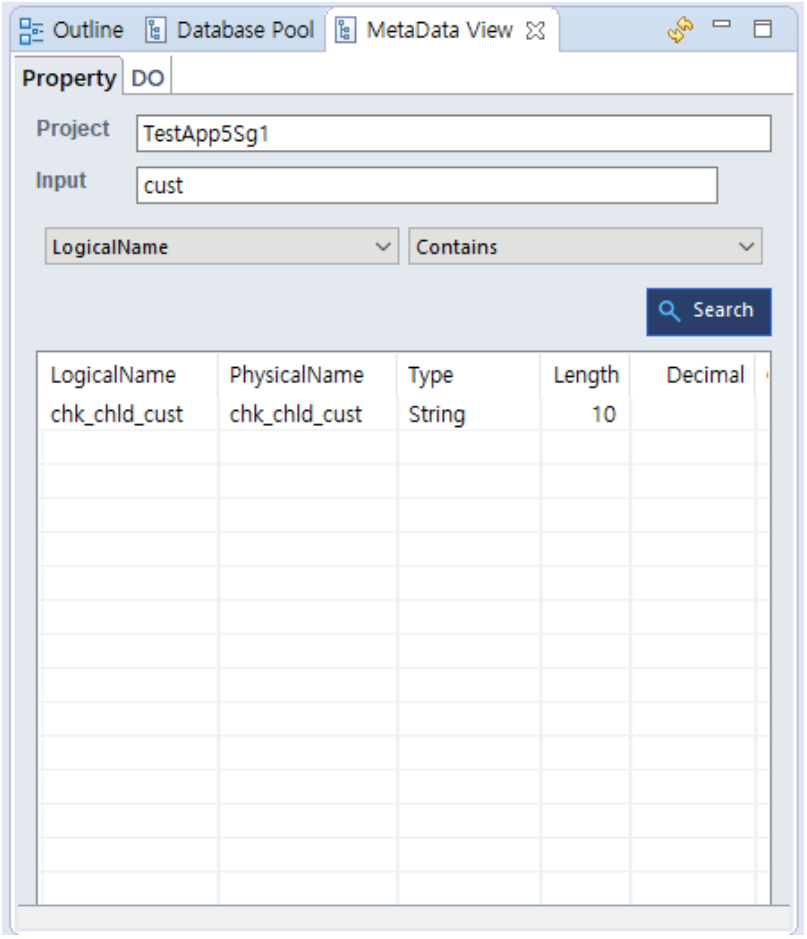
```
[po7@pas99:/home/po7/proobject7/application]$tar -xvf ExtApp/servicegroup/ExtSg/ExtSg.jar
    0 Mon Jul 15 14:29:20 KST 2019 META-INF/
   69 Mon Jul 15 14:29:20 KST 2019 META-INF/MANIFEST.MF
    0 Mon Jul 15 14:29:20 KST 2019 ext/
    0 Mon Jul 15 14:21:08 KST 2019 ext/meta/
 3858 Mon Jul 15 14:20:52 KST 2019 ext/meta/MetaInfoDof$FULLQUERY.class
 9584 Mon Jul 15 14:20:52 KST 2019 ext/meta/MetaInfoDof.class
 3067 Mon Jul 15 14:28:44 KST 2019 ext/meta/PropertyGetList.class
 1019 Mon Jul 15 14:28:42 KST 2019 ext/meta/PropertyGetListExecutor.class
```

3. $PROOBJECT_HOME/system/PoDevSvr.xml에 사용 여부를 설정한다.
   USE_EXT_META_SYSTEM_SERVICE가 TRUE로 설정해서 커스터마이징 검색 옵션을 사용하도록 설정한다.

```
<?xml version="1.0" encoding="EUC-KR" standalone="yes"?>
  <serverConfig xmlns="http://www.tmax.co.kr/proobject/serverConfig">
    :
   중략
    :
<configField id="USE_EXT_META_SYSTEM_SERVICE" value="TRUE" type="String" xmlns=""/>
</serverConfig>
```

4. Studio에서 Meta Property가 정상적으로 조회된다.

   다음은 USE_EXT_META_SYSTEM_SERVICE가 TRUE일 경우 커스터마이징 Property 조회 SO인 PropertyGetList에 의해 조회된다.



Property 커스터마이징 - 메타 조회