

설치 안내서

ProObject 7 Fix#1

TMAXSOFT

저작권 공지

Copyright 2024 TmaxSoft Co., Ltd. All Rights Reserved.

제한된 권리

이 소프트웨어(ProObject®) 사용설명서와 프로그램은 저작권법과 국제 조약에 의해 보호됩니다. 사용설명서와 프로그램은 TmaxSoft Co., Ltd.와의 사용권 계약 하에서만 사용할 수 있으며, 사용설명서는 사용권 계약의 범위 내에서만 배포 또는 복제할 수 있습니다. 이 사용설명서의 전부 또는 일부를 TmaxSoft의 사전 서면 동의 없이 전자, 기계, 녹음 등의 수단으로 전송, 복제, 배포하거나 2차적 저작물을 작성할 수 없습니다.

이 소프트웨어 사용설명서와 프로그램의 사용권 계약은 어떠한 경우에도 사용설명서 및 프로그램과 관련된 지적 재산권(등록 여부를 불문)을 양도하는 것으로 해석되지 않으며, 브랜드나 로고, 상표 등을 사용할 권한을 부여하지 않습니다. 사용설명서는 오로지 정보 제공만을 목적으로 하며, 이로 인한 계약상의 직접적 또는 간접적 책임을 지지 않습니다. 또한 사용설명서 상의 내용이 법적 또는 상업적인 특정 조건을 만족시킬 것을 보장하지 않습니다. 사용설명서는 제품의 업그레이드나 수정에 따라 예고 없이 변경될 수 있으며, 내용상의 오류가 없음을 보장하지 않습니다.

상표 공지

ProObject®는 TmaxSoft Co., Ltd.의 등록 상표입니다. 본 사용설명서에 기재된 모든 제품과 회사 이름은 각각 해당 소유주의 상표로서 참조용으로만 사용되며 반드시 상표 표시 (™, ®)를 하지는 않습니다.

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. 설치 전 준비사항	1
1.1. 개요	1
1.2. 시스템 요구사항	1
1.2.1. 서버	2
1.2.2. ProStudio	2
1.3. 설치 순서	3
2. 런타임 엔진 설치 및 제거	4
2.1. 개요	4
2.2. 설치	4
2.3. 환경설정	7
2.3.1. 기본 설정	7
2.3.2. 런타임 서버 설정	10
2.3.3. 런타임 환경변수 설정	16
2.4. 설치 확인	16
2.5. 제거	17
2.6. 패치	17
3. 개발 서버 설치 및 제거	18
3.1. 설치 개요	18
3.2. 설치 전 준비사항	19
3.2.1. 디렉터리 구조	19
3.2.2. DB 계정 생성 및 JDBC 드라이버 설정	21
3.2.3. 설정 파일 확인	21
3.2.4. 라이브러리 확인	23
3.3. 설치	24
3.3.1. Dev 서버	24
3.3.2. Ops 서버	35
3.4. 설치확인	43
3.4.1. 노드 설정	43
3.4.2. 프로젝트 생성	49
3.4.3. DO, SO 생성	54
3.4.4. HotDeploy 및 Service Test	61
3.4.5. 빌드 및 런타임 서버 배포	66
3.5. 제거	70
4. ProStudio 설치 및 제거	71
4.1. 설치	71
4.2. 설치 확인	71
4.3. 제거	72
Appendix A: 시작하기	73
A.1. DataObject(DO) 생성	73

A.2. DataObject Factory(DOF) 생성	75
A.3. BizObject(BO) 생성	77
A.4. ServiceObject(SO) 생성.....	85
A.5. 서비스 등록(Generate Application DD).....	90
A.6. 서비스 테스트	91

1. 설치 전 준비사항

본 장에서는 ProObject를 설치하기 전에 필요한 준비사항에 대해서 설명한다.

1.1. 개요

ProObject는 차세대 업무시스템이 요구하는 유연성과 재사용성을 극대화하는 아키텍처 기반틀을 제공하는 애플리케이션 프레임워크이며, 이를 위한 다양한 기능들을 제공한다.

ProObject는 런타임 엔진 서버, 통합 개발 서버, ProObject Studio(이하 ProStudio), ProObject Manager(이하 ProManager), ProObject ManaerOps(이하 ProManaerOps) 제품으로 구성된다.

- 런타임 엔진 서버

고속으로 여러 서비스를 처리하면서, 편리하게 서비스를 작성하도록 이벤트 드리븐 처리를 이용한 고속 요청 처리, 다양한 웹 서비스 처리 지원, 데이터 계층을 이용한 편리한 입출력 처리, 유연하고 손쉬운 서비스 연동 등의 기능을 제공한다.

- 통합 개발 서버

미들웨어 기반의 엔진으로 ProStudio에서 개발되는 리소스의 버전 및 형상관리, 리소스 권한관리, 프로젝트 관리, ProMiner Parsing 기능을 지원한다.

- ProStudio

개발자의 개발환경을 지원하는 Eclipse 기반의 툴로 IN /OUT 데이터 정의, DBIO 생성, 단위 테스트 실행, Service Object(SO) 생성, Job Object(JO) 생성 등의 기능을 지원한다.

- ProManager

웹 브라우저를 사용하는 운영관리 툴로 자원 관리, 노드 관리, ProMiner, 시스템 컨텍스트, Resource(Lock&UnLock) 등의 기능을 제공한다.

- ProManagerOPS

웹 브라우저를 사용하는 운영관리 툴로 운영 노드 관리, 리소스 배포, 배치, 센터컷, 스케줄러, 이미지 로그, 후행처리, 운영서버 관리 등의 기능을 제공한다.

ProObject를 설치하기 앞서 다음의 사항을 준비한다.

- ProObject 서버 및 ProStudio 설치를 위한 충분한 시스템 공간 확보
- JDK 8.0 설치
- Tiberio, Oracle 설치 확인

1.2. 시스템 요구사항

본 절에서는 ProObject 설치를 위한 시스템 요구사항에 대해서 설명한다.

1.2.1. 서버

ProObject 서버를 설치하기 위해 필요한 소프트웨어와 하드웨어는 다음과 같다.(최소사양)

- 시스템 요구사항

플랫폼	하드웨어	소프트웨어
Solaris	1GB 이상의 하드디스크 여유 공간	JDK 8.0
HP-UX	메모리 1GB 이상 권장(최소 512MB)	Oracle(10g, 11g, 12c)
AIX		Tibero(Tibero 6 FS03)
Linux		JEUS 8 (b162106)

- 플랫폼 지원 환경

플랫폼	CPU	RAM Memory	HDD Space
Solaris9~11	UltraSPARC 9, 10, 11	1GB	20GB
	Intel X86 Series 10		
HP-UX 11.x, 11i, 11iV2	PA-RISC 11.x(11.11)	1GB	20GB
	Intel Itanium64 11.x(11.23, 11.31)		
AIX 5L, 6L, 7L	RS6000	1GB	20GB
	IBM pSeries(PowerPC)		
Linux 계열(Kernel 2.6이상)	II Intel x86 series k2.6 이상(k2.4 지원)	1GB	20GB
	Intel Itanium Series k2.6 이상		
	IBM pSeries(PowerPC) k2.6 이상		

1.2.2. ProStudio

ProStudio를 설치하기 위해 필요한 소프트웨어와 하드웨어는 다음과 같다.

- 시스템 요구사항

플랫폼	하드웨어	소프트웨어
Windows 7(32/64bit)	1GB 이상의 하드디스크 여유 공간 메모리 2GB 이상 권장(최소 1GB)	JDK 8.0

- 플랫폼 지원 환경

플랫폼	CPU	RAM Memory	HDD Space
Windows 7(32/64bit)	1GHZ 이상(Windows 7 권장사양 이상)	2GB	1GB

1.3. 설치 순서

제품의 설치는 다음의 과정으로 진행된다.

1. 런타임 엔진 설치
2. 개발 서버 설치
3. ProStudio 설치

2. 런타임 엔진 설치 및 제거

본 장에서는 ProObject 런타임 서버를 설치하고 제거하는 방법에 대해서 설명한다.

2.1. 개요

런타임 엔진을 설치할 때에는 아래와 같은 리소스들을 설치가 필요하다.

- 런타임 엔진 바이너리(Runtime Engine Binary)

ProObject 가 수행되기 위한 기본적인 바이너리와 라이브러리가 포함된 파일들을 이르는 말로, WAS(Web Application Server)에서 수행될 수 있도록 WAR(Web Application Resource) 형태로 배포된다.

- 런타임 서버 리소스(ProObject Server Resource)

ProObject가 WAR에 배포되어 수행될 때 이용할 기본적인 서버 설정들과 사용자 애플리케이션들의 리소스를 의미한다.

위의 리소스들은 자동 설치나 수동 설치를 통해 설치를 진행할 수 있으며 원하는 방향에 맞게 설치를 진행하면 된다. 설치가 완료된 이후는 정상적으로 설치되었는지 설치 확인 을 통해 설치 결과를 검증하도록 한다.

자동 설치를 할 수 없는 환경이거나, 수동으로 설치를 진행하기를 원하는 경우에는 아래의 절차에 따라 설치를 진행하도록 한다.

ProObject 런타임 서버의 수동 설치는 다음과 같은 과정으로 진행하면 된다.

1. 런타임 엔진 바이너리 배포

ProObject의 런타임 엔진을 기동하기 위해서는 런타임 엔진 바이너리를 JEUS에 배포하여 이용한다. 런타임 엔진 바이너리를 배포하는 방법에 대하여는 "JEUS Applications & Deployment 안내서"를 참고한다.

2. 런타임 서버 리소스 배포 ([설치 참고](#))

3. 런타임 서버 설정 ([환경설정 참고](#))

4. ProObject 기동 확인 ([설치 확인 참고](#))

2.2. 설치

ProObject 의 런타임 엔진 바이너리가 WAS에 정상적으로 배포되어 기동할 준비를 마쳤다면 런타임 서버 리소스들을 배포해야 한다.

1. ProObject의 기본 디렉터리는 아래와 같다.

```
${PROBJECT_HOME}
|--config
|--application
|--logs
|--application
```



```

|--{APPLICATION}
    |--{SERVICE_GROUP}
|--Monitoring
|--system
|--temp

```

`\${PROBJECT_HOME}`

ProObject 의 최상위 디렉터리로 실제 디렉터리 이름과 위치는 설치할 때 결정된다.

config

ProObject 의 가장 기본적인 설정 파일들을 담고 있으며, 부팅, 기동에 필요한 여러 설정 파일들이 담겨 있다.

다음은 필요한 파일에 대한 설명이다.

파일	설명
proobject.xml	<p>현재 서버에서 기동할 ProObject 런타임 엔진의 설정들이 기술된 설정 파일이다.</p> <p>파일에 기술된 설정들은 런타임 엔진을 초기화할 때 고정적으로 사용할 변수들이 지정되며, 설치가 이루어질 때 외에는 별도로 수정할 필요는 없다.</p> <p>파일의 설정이 정상적으로 되어 있지 않은 경우에는 런타임 엔진의 기동이 실패할 수 있다. 해당 설정 정보에 대한 자세한 내용은 런타임 엔진 설정을 참고한다.</p>
system.properties	<p>ProObject 런타임을 수행하는 서버에 대한 설정들이 기술된 설정 파일이다. ProObject Manager(이하 ProManager)를 이용해서 서버의 설정을 변경할 수 있다.</p>

application

ProStudio를 통해 개발된 애플리케이션이 ProManager를 통해 배포되는 디렉터리로, ProObject에서 사용할 애플리케이션들이 배포되는 디렉터리이다. 해당 폴더에 애플리케이션이 배포되어야 런타임에서 서비스를 제공할 수 있다.

logs

ProObject 의 수행 중 오류 및 수행 정보들을 기록하는 로그 파일들이 담기는 디렉터리로 시스템 로그와 애플리케이션 로그로 분류된다. {LOG_HOME}은 로그가 쌓이는 디렉터리의 최상위 디렉터리로, ProObject 런타임 엔진의 시스템 로그가 저장된다.

다음은 해당 경로에 저장되는 로그에 대한 설명이다.

파일	설명
ProObject.log	런타임 엔진에서 작성하는 모든 로그들이 기록된다.
ChannelEventManager.log	네트워크 I/O 또는 IPC를 처리하는 채널 이벤트 핸들러와 이를 관리하던 도중에 발생하는 로그들이 기록된다.
EventManager.log	이벤트 계층(Event Layer)에서 채널 이외의 경우에 발생하는 로그들이 기록된다.

파일	설명
NodeAddressManager.log	remote_servicegroup.xml과 remote.address.properties 등을 통해 외부 연동을 기술한 경우 연동에 관련된 로그들이 기록된다.
Monitoring.log	모니터링 기능을 활성화한 경우 모니터링과 관련된 로그들이 기록된다.

다음은 하위 디렉터리에 대한 설명이다.

- **logs/application**

애플리케이션별로 기록되는 로그 파일들이 저장된다. 디렉터리 내부에서는 `{APPLICATION_NAME}.log`의 이름으로 로그가 생성된다.

하위 디렉터리	설명
<code>{APPLICATION}</code>	애플리케이션에 속한 서비스그룹별로 기록되는 로그들이 생성되는 디렉터리이다. 디렉터리 내부에서는 <code>{SERVICEGROUP_NAME}.log</code> 의 이름으로 서비스 그룹별 로그가 생성된다.
<code>{APPLICATION}/{SERVICE_GROUP}</code>	서비스 그룹에 속한 서비스 별로 기록되는 로그들이 생성되는 디렉터리이다. 디렉터리 내부에서는 <code>{SERVIC_NAME}.log</code> 의 이름으로 서비스별 로그가 생성된다.

- **logs/Monitoring**

런타임 엔진의 모니터링과 관련된 로그들이 생성되는 디렉터리이다.

system

ProObject의 시스템 라이브러리나 패치 파일들이 위치한다.

기본적으로 DevOps와 관련된 라이브러리가 설치되거나, 런타임 엔진의 패치를 제공하기 위해 사용한다. DevOps와 관련된 설치 내용은 [설치](#), 런타임 엔진의 패치는 [패치](#)를 참고한다.

temp

ProObject 런타임 엔진에 파일 업로드를 포함한 임시 파일들이 저장되는 공간이다. 별도로 사용되지 않으며 런타임 엔진이 기동 중이 아닐 경우에는 안의 디렉터리들을 지워도 무방하다.

2. 압축을 해제해서 ProObject 디렉터리 구조를 생성했다면 ProObject의 Home 디렉터리를 환경변수로 설정해야 한다.

환경변수는 시스템 환경변수와 JVM 환경변수를 모두 지원한다. 두 환경변수가 모두 설정된 경우에는 JVM 환경변수가 우선시되어 설정된다.

- OS 환경변수를 이용할 경우 Linux를 기준으로 다음과 같이 ProObject Home을 설정할 수 있다.

```
PROBJECT_HOME = 사용자 지정 디렉터리
export PROBJECT_HOME
```



설정을 변경한 다음에는 `.profile` 또는 `.bash_profile`을 입력하여 설정한 내용을

반영시켜야 한다.

- JVM 옵션을 설정하려면 WAS 서버가 기동될 때 다음과 같이 ProObject Home을 설정할 수 있다.

```
-DPROOBJECT_HOME = 사용자 지정 디렉터리
```

3. ProObject의 바이너리 복사와 ProObject Home 경로가 설정되었다면, ProObject의 환경을 설정해주어야 한다. 각 옵션에 대한 자세한 설명은 [런타임 엔진 설정](#)을 참고한다.

2.3. 환경설정

본 절에서는 ProObject 기동에 필요한 설정들에 대하여 기술한다. 해당 설정들이 제대로 되어 있지 않은 경우에는 ProObject가 정상적으로 부팅될 수 없음을 유의하도록 한다.

2.3.1. 기본 설정

ProObject 엔진의 가장 기본적인 설정은 다음의 파일에 한다.

```
${PROOBJECT_HOME}/config/proobject.xml
```

최상위 요소로는 <ProObjectConfig>에 설정한다.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<ProObjectConfig>
  <!--ProObject 기본 설정들-->
  <application-path>${PROOBJECT_HOME}/application</application-path>
  <locale></locale>
  <adaptive-port>>false</adaptive-port>
  <proobject-port>6776</proobject-port>
  <engine-config>
    <!--ProObject 런타임 엔진 설정 -->
    :
    종략
    :
  </engine-config>
  <schedule-config>
    <!--ProObject 스케줄링 엔진 설정 -->
    :
    종략
    :
  </schedule-config>
  <batch-config>
    <!--ProObject 일괄처리 설정 -->
    :
    종략
    :
```

```
</batch-config>
</ProObjectConfig>
```

항목	설명
<application-path>	<p>런타임 서버가 인식하는 애플리케이션의 디렉터리 위치를 변경할 경우에 사용한다. 설정값은 애플리케이션들을 배포하고자 하는 유효한 디렉터리를 설정한다.</p> <p>JVM 환경변수가 설정된 경우 이 설정이 우선시된다(JVM 환경변수는 -DPROOBJECT_APPLICATION_HOME으로 설정할 수 있다).</p> <p>(기본값: \${PROOBJECT_HOME}/application)</p>
<locale>	<p>런타임 엔진의 기본 언어 정보를 설정한다. 설정에 따라 런타임 엔진의 로그/오류 등의 메시지 언어를 변경할 수 있다.</p> <p>(기본값: EN, 현재 설정값은 EN(영어)만 설정 가능)</p>
<server-name>	<p>런타임 엔진의 서버 이름을 설정한다.</p> <p>JVM 환경변수가 설정된 경우 이 설정이 우선시된다(JVM 환경변수는 -DPROOBJECT_SERVER_NAME으로 설정할 수 있다).</p>
<engine-config>	<p>런타임 엔진에서 사용할 각종 설정들을 설정한다. 자세한 내용은 런타임 엔진 설정을 참고한다.</p>
<adaptive-port>	<p>런타임 엔진이 부팅되며 서버 포트를 바인딩에 실패한 경우 자동으로 가용할 포트를 찾아 바인딩을 시도할지 설정한다.</p> <p>해당 옵션이 비활성화되어 있을 때 바인딩에 실패하면 서버 기동이 실패한다. (기본값: false)</p>
<proobject-port>	<p>런타임 엔진 간의 통신할 때 사용할 기본 TCP 포트를 설정한다.</p> <p>(기본값: 6776)</p>
<proobject-channel-config>	<p>ProObject 간의 통신에 사용되는 채널에 대한 설정을 설정한다. 자세한 내용은 채널 환경설정을 참고한다.</p>
<file-port>	<p>런타임 엔진이 TCP 를 통해 파일 송/수신을 처리할 경우 사용할 TCP 포트를 설정한다. (기본값: 4444)</p>
<engine-config>	<p>ProObject 런타임 기동 및 여러 부가적인 처리를 위해 엔진에서 필요한 여러 설정들을 설정한다. 자세한 내용은 런타임 엔진 설정을 참고한다.</p>
<schedule-config>	<p>ProObject의 스케줄링에 관련된 여러 설정들을 설정한다. 해당 설정에 대한 자세한 내용은 ProObject 런타임 엔진 개발자 안내서의 "스케줄러 서비스 개발"을 참고한다.</p>
<batch-config>	<p>ProObject의 일괄처리에 관련된 여러 설정들을 설정한다. 해당 설정에 대한 자세한 내용은 ProObject 런타임 엔진 개발자 안내서의 "배치 서비스 개발"을 참고한다.</p>

2.3.1.1. 런타임 엔진 설정

런타임이 기동하거나 부가 기능들을 처리함에 있어 공통적으로 필요한 정보들을 설정한다. 해당 설정은 <ProObjectConfig><engine-config>에 설정한다.

```
<ProObjectConfig>
  :
  <engine-config>
    <listener-name>listener_name</listener-name>
    <host-name>host_name</host-name>
    <proObject-datasource>datasource_name</proObject-datasource>
    <initial-context-factory>class_name</initial-context-factory>
    <provider-url>server_address</provider-url>
    <transaction-manager-factory-class>class_name</transaction-manager-factory-class>
  </engine-config>
</ProObjectConfig>
```

항목	설명
<listener-name>	Node.JAVA를 사용하는 경우 사용할 Node.JAVA 스레드를 지니는 웹 컨넥션 이름을 설정한다. JEUS의 리스너 이름이 아니라 웹 컨넥션 이름을 설정해야 한다. 설정이 잘못된 경우 부팅이 되지 않으므로 주의한다.
<proObject-datasource>	ProObject 런타임 엔진이 기본적으로 사용하는 데이터소스의 이름을 설정한다. 런타임 설정 및 여러 기본 정보들을 불러오는데 사용되며, dbio_config에 설정된 alias의 이름을 설정한다.
<service-alias-datasource>	ProObject에서 서비스의 별명을 설정한 데이터소스의 이름이다. dbio_config에 지정된 alias의 이름을 설정한다.
<initial-context-factory>	ProObject에서 JNDI Lookup하는 경우 사용할 ContextFactory의 클래스 이름을 설정한다. 별도의 설정이 없을 경우 ProObject Pub/Sub과 데이터소스 등의 Lookup을 시도하는 가장 기본적인 ContextFactory이므로 Pub/Sub이나 DB I/O가 있는 경우에는 반드시 설정해주어야 한다. 기본값은 JEUS로 설정되며, jeus.jndi.JEUSContextFactory으로 지정된다.
<provider-url>	ProObject의 JNDI Lookup하는 경우 접속할 서버의 주소를 설정한다. 설정하지 않을 경우 현재 서버를 사용한다.
<transaction-manager-factory-class>	ProObject의 트랜잭션을 관리하는 TransactionManager가 사용할 TransactionFactory 클래스를 설정한다. (기본값: com.tmax.proobject.engine.transaction.JeusTransactionManagerFactory)

다음은 기본적으로 배포되는 proobject.xml의 런타임 엔진 설정 예제이다.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<ProObjectConfig>
  <locale>en</locale>
  <server-name>ProObject7</server-name>
  <engine-config>
    <listener-name>http</listener-name>
    <host-name>KuiperUbuntu</host-name>
    <proObject-datasource>tibero6</proObject-datasource>
    <initial-context-factory>jeus.jndi.JEUSContextFactory</initial-context-factory>
    <provider-url>localhost:19736</provider-url>
  </engine-config>
</ProObjectConfig>
```

2.3.1.2. 채널 환경설정

ProObject에서 사용하는 연결들에 대해 설정한다.

```
<ProObjectConfig>
  :
</ProObjectConfig>
```

항목	설명
<on-close-service>	<p>ProObject 간의 통신 중 연결이 끊어진 경우 호출되는 서비스를 설정한다. 서비스의 입력은 다음과 같은 데이터 오브젝트로 제한되며, 출력은 무시된다.</p> <ul style="list-style-type: none"> • 입력 : <ul style="list-style-type: none"> com.tmax.proobject.engine.event.channel.eventhandler.proobject.dto.ProObjectChannelCloseInfo ◦ remoteIp : 연결이 끊어진 상대방의 IP이다. ◦ remotePort : 연결이 끊어진 상대방의 Port이다. ◦ localIp : 연결이 끊어진 자신의 IP이다. ◦ localPort : 연결이 끊어진 자신의 Port이다. ◦ serverName : 연결이 끊어진 서버의 이름이다. ◦ exception : 연결이 끊어질 때 발생한 예외 객체이다.

2.3.2. 런타임 서버 설정

본 절에서는 서버별로 설정하는 각종 설정에 대하여 설명한다.

해당 설정들이 제대로 되어 있지 않은 경우에는 ProObject에서 원하는 동작들이 정상적으로 수행되지 않으므로 유의하도록 한다. ProObject 엔진의 가장 기본적인 설정은 다음이 경로 파일에 Key-Value 방식을 이용해 설정한다.

```
`${PROJECT_HOME}/config/system.properties
```

2.3.2.1. 서버 설정

서버의 가장 기본적인 설정들을 설정한다.

```
SYSTEM_CHARSET = charset_code  
SYSTEM_REPLACE_CHARSET_WITH_REQUEST = [true|false]  
SYSTEM_GLOBAL_DEPLOY_VERSION = deploy_version
```

항목	설명
SYSTEM_CHARSET	서버에서 사용할 기본 인코딩을 설정한다. (기본값: UTF-8)
SYSTEM_REPLACE_CHARSET_WITH_REQUEST	서버 내부 동작은 기본 charset으로 동작하면서 응답을 줄 때만 request에 설정된 charset으로 전환할지 여부를 설정한다. (기본값: false)
SYSTEM_GLOBAL_DEPLOY_VERSION	서버의 전역 배포 버전을 설정한다. 특별한 경우 외에는 설정하지 않는 것을 권장한다. (기본값: 0)

2.3.2.2. 웹 관련 설정

서버의 웹과 관련된 기본적인 설정들을 설정한다.

```
SYSTEM_WEB_ALLOW_HEADER = header1,header2,header3...  
SYSTEM_WEB_ALLOW_ORIGIN = [true|false]  
SYSTEM_WEB_RESPONSE_EXCEPTION_STACKTRACE_EXCLUDE = [true|false]  
SYSTEM_WEB_PARAMETER_CHARSET = charset_code  
SYSTEM_WEB_DOWNLOAD_FILE_SIZE_LIMIT = file_size_limit_value  
SYSTEM_WEB_READ_BUFFER_SIZE = read_buffer_size  
SYSTEM_WEB_WRITE_BUFFER_SIZE = write_buffer_size  
SYSTEM_WEBSOCKET_TEXT_BUFFER_SIZE = buffer_size  
SYSTEM_WEBSOCKET_BINARY_BUFFER_SIZE = buffer_size  
SYSTEM_HTTP_CORS_DISABLE = [true|false]  
SYSTEM_PROOJECT_SESSION_NULL_ENABLE = [true|false]  
SYSTEM_WEB_RESTRICTED_HTTP_METHOD = http_method:http_method:....
```

항목	설명
SYSTEM_WEB_ALLOW_HEADERS	<p>런타임 엔진이 HTTP 에서 허용할 헤더의 종류를 설정한다.</p> <p>별도로 지정하지 않는 경우 지정되지 않은 헤더가 포함된 서비스는 요청이 거절된다.</p> <p>여러 개의 헤더를 설정하려면 콤마(,)로 구분하여 설정한다.</p> <p>(기본값: origin, x-requested-with, content-type, accept, ProObjectWebFileTransfer)</p>
SYSTEM_WEB_ALLOW_ORIGIN	<p>웹 브라우저를 통해 요청을 전달한 경우 요청을 허용할 주소를 설정한다. 주소가 잘못된 경우 웹 브라우저가 요청을 전달하는 것을 거부할 수 있다. (기본값: false, 범위: true false)</p>
SYSTEM_WEB_RESPONSE_EXCEPTION_STACKTRACE_EXCLUDE	<p>런타임 엔진이 웹을 통해 요청을 받은 서비스를 처리를 하던 중 오류가 발생했을 때 오류 스택을 함께 전달할지 여부를 설정한다.</p> <ul style="list-style-type: none"> ◦ true : 오류가 발생했을 때 오류 스택을 함께 전달하지 않는다. ◦ false : 오류가 발생했을 때 오류 스택을 함께 전달한다. (기본값)
SYSTEM_WEB_PARAMETER_CHARSET	<p>인코딩 정보를 설정한다. (기본값: ISO-8859-1)</p> <p>GET으로 데이터를 실어서 오거나 쿼리 스트링으로 추가적인 정보를 보내는 경우 해당 정보들의 문자열 셋 정보를 설정한다.</p> <p>웹 표준 상으로 ISO-8859-1로 지정하도록 정해져있으므로, ISO-8859-1로 들어와 ProObject의 기본 인코딩으로 전환된다. WAS에서 별도의 설정으로 쿼리 스트링의 인코딩을 변경했다면, 그에 맞춰 인코딩을 변경해주어야 한다.</p>
SYSTEM_WEB_DOWNLOAD_FILE_SIZE_LIMIT	<p>웹을 통해 파일을 다운로드하는 경우 다운로드를 허용할 파일의 최대 크기를 원하는 크기를 KB 단위로 설정한다.</p> <p>(기본값: 1024 (1MB))</p>
SYSTEM_WEB_READ_BUFFER_SIZE	<p>HTTP를 통해 요청을 받았을 때 Body를 읽을 때 사용하는 버퍼의 크기를 설정한다. (기본값: 16384, 16KB)</p>
SYSTEM_WEB_WRITE_BUFFER_SIZE	<p>HTTP를 통해 응답을 전달할 때 Body를 전송할 때 사용할 버퍼의 크기를 설정한다. (기본값: 16384, 16KB)</p>
SYSTEM_WEBSOCKET_TEXT_BUFFER_SIZE	<p>웹 소켓을 통해 텍스트 데이터를 송신할 때 사용할 버퍼의 크기를 설정한다. (기본값: 8192, 8KB)</p>
SYSTEM_WEBSOCKET_BINARY_BUFFER_SIZE	<p>웹 소켓을 통해 이진 데이터를 송신할 때 사용할 버퍼의 크기를 설정한다. (기본값: 8192, 8KB)</p>
SYSTEM_HTTP_CORS_DISABLE	<p>Cross-Domain과 관련된 설정을 자동으로 할지 여부를 설정한다. (기본값: false, 범위: true false)</p>
SYSTEM_PROOBJECT_SESSION_NULL_ENABLE	<p>전달되는 SessionID가 NULL인 경우 Session의 생성 여부를 설정한다. (기본값: false, 범위: true false)</p>

항목	설명
SYSTEM_WEB_RESTRICTED_HTTP_METHOD	<p>제한하고 싶은 HTTP 메소드의 이름들을 지정한다. 메소드의 이름은 콜론(:) 을 통해 구분한다.</p> <p>이 때 POST, GET, PUT, DELETE를 제외한 메소드는 PO가 지원하지 않으므로 설정해도 영향을 받지 않는다.</p>

2.3.2.3. 로그 설정

다음은 서버의 로그 관련 설정 항목에 대한 설명이다.

```

SYSTEM_LOG_HOME = log_home_path
SYSTEM_LOG_LEVEL = [SEVERE|WARNING|INFO|CONFIG|FINE|FINER|FINEST]
SYSTEM_LOG_CONSOLE_HANDLER_ENABLE = [TRUE|FALSE]
SYSTEM_LOG_FILEHANDLER_STORE = [AGGREGATE|SEPERATE]

SYSTEM_{MODULE}_LOG_LEVEL = [SEVERE|WARNING|INFO|CONFIG|FINE|FINER|FINEST]
SYSTEM_{MODULE}_LOG_ENCODING = log_encoding_code
SYSTEM_{MODULE}_LOG_ASYNC_HANDLER_BUFFER_SIZE = buffer_size
SYSTEM_{MODULE}_LOG_FILEHANDLER_INTERVAL_TYPE = [NONE|DAY|HOUR]
SYSTEM_{MODULE}_LOG_FILEHANDLER_INTERVAL = log_filehandler_interval
SYSTEM_{MODULE}_LOG_FILEHANDLER_LIMIT = log_filehandler_limit

```

설정항목 중 {MODULE} 항목은 런타임 엔진의 주요 모듈들을 의미하며, 각 모듈의 로그들을 일괄적으로 설정할 수 있다. 각 모듈을 설정할 때에는 다음 중에서 지정해야 한다.

- PROOBJECT : 최상위 모듈로 런타임 엔진의 모든 로그를 기록한다.
- CHANNELEVENTMANAGER : 채널 이벤트 계층을 관장하는 모듈로 채널과 관련된 로그를 기록한다.
- EVENTMANAGER : 이벤트 계층을 관장하는 모듈로 이벤트와 관련된 로그를 기록한다.

항목	설명
SYSTEM_LOG_HOME	<p>런타임 서버의 로그가 생성되는 위치를 설정한다.</p> <p>(기본값: \${PROOBJECT_HOME}/logs/)</p>
SYSTEM_LOG_LEVEL	<p>서버의 기본 로그 레벨을 설정한다.</p> <ul style="list-style-type: none"> ◦ SEVERE ◦ WARNING ◦ INFO (기본값) ◦ CONFIG ◦ FINE ◦ FINER ◦ FINEST

항목	설명
SYSTEM_LOG_CONSOLE_HANDLER_ENABLE	런타임 서버의 로그에 콘솔 핸들러(Console Handler)의 추가 여부를 설정한다. (기본값: false)
SYSTEM_LOG_FILEHANDLER_STORE	런타임 서버의 로그를 기록하는 방식을 설정한다. <ul style="list-style-type: none"> AGGREGATE : 모든 스레드들의 로그들이 이벤트 스레드에서 모여, 이벤트 스레드는 비동기식 파일 기록 스레드로 로그를 전달하여 로그를 기록한다. (기본값) SEPERATE : 로그를 남기는 모든 스레드가 개별적으로 로그를 기록한다.
SYSTEM_{MODULE}_LOG_LEVEL	서버의 모듈별 시스템 로그 레벨을 설정한다(SYSTEM_LOG_LEVEL과 동일). 별도의 설정이 없을 경우에는 SYSTEM_LOG_LEVEL의 설정을 따른다.
SYSTEM_{MODULE}_LOG_ENCODING	서버의 모듈별 인코딩을 설정한다. (기본값: UTF-8)
SYSTEM_{MODULE}_LOG_ASYNC_HANDLER_BUFFER_SIZE	서버의 모듈별 로그 버퍼의 크기를 설정한다. (기본값: 4096) 너무 크게 설정하는 경우에는 메모리를 낭비할 가능성이 높으며, 너무 작게 설정하면 성능이 떨어질 수 있으므로 주의하여 설정하도록 한다. 단, 설정은 반드시 2의 지수의 값들 중 하나로 설정해야 한다.
SYSTEM_{MODULE}_LOG_FILE_HANDLER_INTERVAL_TYPE	서버의 모듈별 로그가 파일에 로그를 기록하는 경우 어떤 주기로 파일을 변경할지를 설정한다. <ul style="list-style-type: none"> NONE : 로그 파일을 변경하지 않는다. DAY : 정해진 날짜 주기마다 로그 파일을 변경한다. (기본값) HOURLY : 00시를 기준으로 정해진 시간 주기마다 로그 파일을 변경한다. 예를 들어 SYSTEM_{MODULE}_LOG_FILEHANDLER_INTERVAL을 6으로 지정한 경우 6시, 12시, 18시, 24시에 파일이 변경된다.
SYSTEM_{MODULE}_LOG_FILE_HANDLER_INTERVAL	서버의 모듈별 로그가 파일에 로그를 기록하는 경우 파일 변경 주기 타입에서 설정한 단위로 얼마나 지나야 파일을 변경할지를 양의 정수로 설정한다. 이때 파일 변경 주기 타입은 SYSTEM_{MODULE}_LOG_FILEHANDLER_INTERVAL_TYPE의 값을 따른다. (기본값: 1, 하루마다 로그 파일이 따로 작성되도록 되어 있다.)
SYSTEM_{MODULE}_LOG_FILE_HANDLER_LIMIT	서버의 모듈별 로그가 파일에 로그를 기록하는 경우 로그 파일을 변경할 때 크기의 제한을 설정한다. 지정한 크기에 근접하는 경우 파일을 변경한다. (기본값: -1) 아래의 타입들 중 동작을 원하는 형태로 값을 지정하여 사용한다. <ul style="list-style-type: none"> 음수 : 파일의 크기를 바탕으로 로그를 나누지 않는다. 양수 : 설정한 값의 KB 단위로 파일이 분할된다.

2.3.2.4. 서비스 설정

서버의 로그 관련 설정에 대해 설명한다.

```
SYSTEM_TIMEOUT = timeout_value
SYSTEM_APPLICATION = application_name1:application_name2.....
SYSTEM_SERVICENAME_CASE = [CAMEL|NATIVE|LOWER|UPPER]
```

항목	설명
SYSTEM_TIMEOUT	<p>서버에 배포된 모든 서비스들에 대한 기본 타임아웃 시간을 밀리초(ms) 단위로 설정한다. (기본값: 60000 (60s, 1분))</p> <p>아래의 타입들 중 동작을 원하는 형태로 값을 지정하여 사용한다.</p> <ul style="list-style-type: none"> ◦ 음수 : 타임아웃이 발생하지 않는다. ◦ 양수 : 설정한 값의 ms 단위로 타임아웃 시간이 설정된다.
SYSTEM_APPLICATION	<p>ProObject에서 수행시킬 애플리케이션의 이름을 설정한다.</p> <p>여러 애플리케이션을 수행시키려면 콤마(,)를 구분자로 사용하며, 단일 애플리케이션 옵션이 활성화된 경우에는 최초에 등장한 애플리케이션만 수행된다.</p> <ul style="list-style-type: none"> • 예 <p>다음은 test 애플리케이션 하나만을 실행하는 경우의 예이다.</p> <pre>SYSTEM_APPLICATION = test</pre> <p>다음은 test, test2 애플리케이션을 배포하는 경우의 예이다.</p> <pre>SYSTEM_APPLICATION = test,test2</pre>
SYSTEM_SERVICENAME_CASE	<p>서비스 이름을 인식하는 방법을 설정한다.</p> <p>다음 중에서 설정한다.</p> <ul style="list-style-type: none"> ◦ CAMEL : 가장 앞의 문자를 대문자로 인식하며, 언더바(_)가 있는 경우 언더바가 서비스 이름에서 제외하고 바로 뒤의 영문자를 대문자로 인식한다. (기본값) ◦ NATIVE : 서비스의 이름을 변형하지 않고 전달된 그대로 인식한다. ◦ LOWER : 서비스의 이름을 항상 소문자로 인식한다. ◦ UPPER : 서비스의 이름을 항상 대문자로 인식한다. <p>해당 설정을 지정한 경우 원하는 형태로 서비스의 이름이 변형되어 서버가 인식되므로 주의하도록 한다.</p>

2.3.2.5. 데이터 오브젝트 설정

데이터 오브젝트의 동작과 관련된 설정들로 해당 항목의 설정들은 ProObject 런타임 엔진 개발자 안내서의 "데이터 오브젝트/데이터 오브젝트 팩토리 개발"을 참고한다.

2.3.2.6. 헬스 체크 설정

연결된 서버들의 alive 여부를 검사하는 헬스 체크관련 설정을 설정한다.

```
SYSTEM_PROOBJECT_HEALTH_CHECK_ENABLE = [true|false]
```

항목	설명
SYSTEM_PROOBJECT_HEALTH_CHECK_ENABLE	연결된 모든 ProObject 서버들의 alive 여부를 검사하기 위한 Heart-Beat 메시지를 송신할지 여부를 설정한다. 해당 옵션이 설정된 후에는 1분에 한 번씩 서비스의 요청을 전달하게 된다.

2.3.3. 런타임 환경변수 설정

런타임 엔진을 기동할 때에 JVM 옵션을 통해 런타임 엔진의 환경변수를 설정이 가능하다.

항목	설명
PROOBJECT_HOME	ProObject 런타임 서버 리소스의 위치를 설정한다.
PROOBJECT_LOG_HOME	ProObject 런타임 서버의 로그가 쌓이는 디렉터리를 설정한다. (기본값: {PROOBJECT_HOME}/logs)
PROOBJECT_SERVER_NAME	ProObject의 서버 이름을 설정한다. proobject.xml에 설정하길 원치 않는 경우에 유효한 옵션이다.

2.4. 설치 확인

ProObject 런타임 바이너리가 배포된 WAS에 서비스를 호출하여 설치가 정상적으로 이루어졌는지를 확인한다. 출력 내용과 \${PROOBJECT_HOME}/logs/ProObject.log를 확인하여 정상적으로 기동되었는지 확인할 수 있다.

웹에서 ProObject가 설치된 서버의 IP:Port로 웹으로 요청을 보내 다음과 같은 메시지를 받았다면 정상적으로 ProObject가 기동된 것이다.

```

{"header":{"service":null,"inputMsgType":null,"outputMsgType":null,"transferKey":null,"fileBaseDirectory":null,"files":[],"responseCode":"RTE-0110","responseMsg":null,"responseMsgDetails":null,"length":0,"guid":null,"tenantId":null},"exception":{"code":"RTE-0110","name":"ApplicationNotFound","message":"[RTE-0110] : ApplicationNotFound - Application null is not deployed. Please check application name you tried to execute or intending to use single application only mode.\"", "stackTrace":"com.tmax.proobject.engine.exception.service.nonexist.ApplicationNotFoundException: [RTE-0110] : ApplicationNotFound - Application null is not deployed. Please check application name you tried to execute or intending to use single application only mode.\n\tat com.tmax.proobject.engine.application.ApplicationManager.getManager(ApplicationManager.java:258)\n\tat com.tmax.proobject.engine.event.channel.eventhandler.httpservlet.HttpServletEventHandler.parseHeader(HttpServletEventHandler.java:523)\n\tat com.tmax.proobject.engine.event.channel.eventhandler.httpservlet.HttpServletEventHandler.handleHttpRequest(HttpServletEventHandler.java:285)\n\tat com.tmax.proobject.engine.event.channel.eventhandler.httpservlet.HttpServletEventHandler.onRead(HttpServletEventHandler.java:266)\n\tat com.tmax.proobject.engine.event.channel.eventhandler.httpservlet.HttpServletEventHandler.prepareHandlingHttpRequest(HttpServletEventHandler.java:235)\n\tat com.tmax.proobject.endpoint.ProObjectHttpServlet.service(ProObjectHttpServlet.java:102)\n\tat javax.servlet.http.HttpServlet.service(HttpServlet.java:683)\n\tat com.tmax.proobject.endpoint.ProObjectHttpServlet.service(ProObjectHttpServlet.java:94)\n\tat javax.servlet.http.HttpServlet.service(HttpServlet.java:786)\n\tat jeus.servlet.engine.ServletWrapper.executeServlet(ServletWrapper.java:170)\n\tat jeus.servlet.filter.FilterChainImpl.internalDoFilter(FilterChainImpl.java:112)\n\tat jeus.servlet.filter.FilterChainImpl.doFilter(FilterChainImpl.java:86)\n\tat jeus.websocket.server.WebSocketFilter.doFilter(WebSocketFilter.java:42)\n\tat jeus.servlet.filter.FilterChainImpl.internalDoFilter(FilterChainImpl.java:98)\n\tat jeus.servlet.filter.FilterChainImpl.doFilter(FilterChainImpl.java:86)\n\tat com.tmax.proobject.engine.event.channel.eventhandler.httpservlet.WebSocketUpgraderFilter.doFilter(WebSocketUpgraderFilter.java:105)\n\tat com.tmax.proobject.endpoint.ProObjectHttpFilter.doFilter(ProObjectHttpFilter.java:51)\n\tat jeus.servlet.filter.FilterChainImpl.internalDoFilter(FilterChainImpl.java:98)\n\tat jeus.servlet.filter.FilterChainImpl.doFilter(FilterChainImpl.java:86)\n\tat jeus.servlet.engine.ServletWrapper.execute(ServletWrapper.java:145)\n\tat jeus.servlet.engine.RequestProcessor.run(RequestProcessor.java:247)\n\tat jeus.io.connection.unified.UnifiedConnector.receiveContent(UnifiedConnector.java:233)\n\tat jeus.io.handler.StreamHandler.dispatchMessage(StreamHandler.java:420)\n\tat jeus.io.impl.nio.handler.NIOStreamHandler.onReadEvent(NIOStreamHandler.java:189)\n\tat jeus.io.impl.nio.JeusSocketChannel.fillReadBuffer(JeusSocketChannel.java:103)\n\tat jeus.io.impl.nio.handler.NIOStreamHandler.waked(NIOStreamHandler.java:108)\n\tat jeus.io.impl.nio.NIOSelector.runWaked(NIOSelector.java:725)\n\tat jeus.io.impl.nio.NIOSelector.processStreamHandlers(NIOSelector.java:611)\n\tat jeus.io.impl.nio.NIOSelector.run(NIOSelector.java:528)\n\tat java.lang.Thread.run(Thread.java:748)"}

```

ProObject 기동 확인

2.5. 제거

설치된 ProObject를 제거하기 위해서는 런타임 엔진 바이너리와 서버 리소스를 모두 제거해주어야 한다.

- 런타임 엔진 바이너리

WAS에서 배포해제(Undeploy)를 통해 제거가 가능하다.

- 서버 리소스

`{PROOBJECT_HOME}` 디렉터리를 제거하기만 하면 되나, 애플리케이션이나 로그의 위치 등을 변경한 경우에는 해당 디렉터리도 함께 제거하도록 한다.

2.6. 패치

필요한 경우에는 설치된 ProObject 런타임 엔진에 패치가 이루어질 수 있다. 기본적으로는 기본 제공되는 WAR 파일을 WAS에 재배포하여 패치를 처리할 수도 있으나, ProObject에는 기존의 바이너리를 변경하지 않고 패치를 적용하는 방법을 제공하고 있다.

런타임 엔진은 `{PROOBJECT_HOME}/system/patch` 디렉터리 하위에 패치 파일들을 배포해서 패치를 적용한다. 패치 파일을 배포하면 런타임 엔진이 자동으로 패치 파일들을 먼저 불러들여 런타임 엔진을 기동된다.

```

${PROOBJECT_HOME}
|--system
|--patch

```

3. 개발 서버 설치 및 제거

본 장에서는 ProObject 개발 서버(DevOps) 모듈을 설치하고 제거하는 방법에 대해서 설명한다.



본 안내서에서 설명하는 내용은 Ubuntu 환경에서 작성되었으며, CentOS의 경우 기반 프로그램들의 설치 명령어가 다를 수 있으므로 주의한다.

3.1. 설치 개요

ProObject의 개발 서버(DevOps)는 개발 전반적인 과정을 통합적으로 지원하고, 관리하는 서버로 ProObject 사상에 맞게 서비스 아키텍처로 설계되었다. 개발 서버에서 제공하는 모든 서비스는 연관된 모듈끼리 서비스 그룹으로 묶여 ProObject 엔진 위에서 수행되도록 구현되었다. 따라서 개발 서버는 ProObject 애플리케이션(ProObject Runtime War)이 배포된 JEUS의 MS(Managed Server) 컨테이너에 설치된다.

개발 서버는 ProObject Studio(이하 ProStudio)에서 개발한 리소스에 대하여 형상관리, 빌드, 테스트, 배포 기능을 제공한다. 또한, ProObject 리소스의 개발 환경을 제어 및 관리하는 ProObject Manager(이하 ProManager), ProObject ManagerOps(이하 ProManagerOps)의 백엔드 서비스 또한 개발 서버에 속한다.

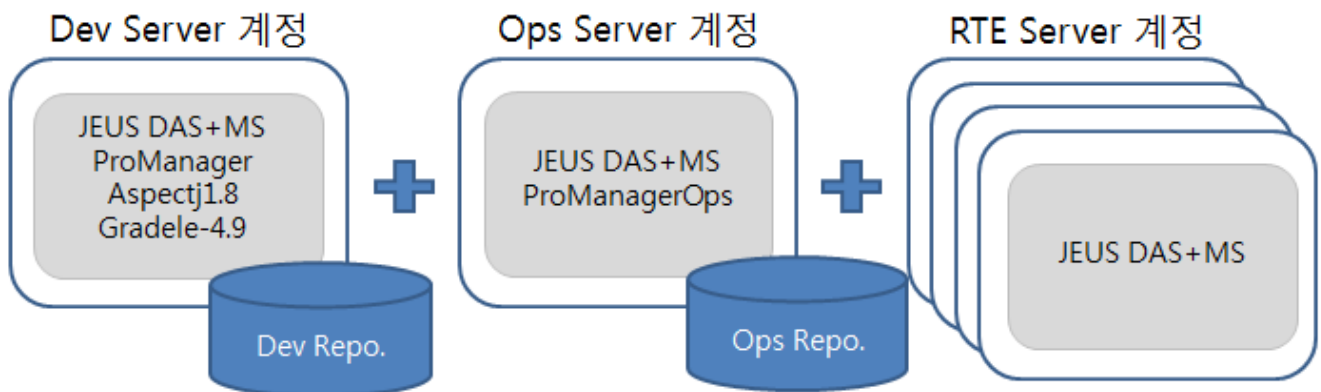
DevOps 개발 환경은 클라이언트 툴인 ProStudio와 매니저를 지원하는 서비스들과 ProObject 런타임 서버에 배포하는 서비스를 제공하는 모듈과, ProObject 리소스로 개발되었으며, 비슷한 서비스들로 구성된 여러 서비스 그룹으로 구성되어 있다.

ProObject 개발 서버는 형상관리와 빌드할 때 제품에서 제공하는 기능을 이용하는 형태의 환경구성 외에 Git/Jenkins를 이용하는 환경구성이 가능하다.



본 장에서는 형상관리와 빌드하는 경우 제품에서 제공하는 기능 이용하는 형태의 환경 구성에 대한 설치만을 기술한다. GIT과 Jenkins 환경에 대한 설치 는 별도의 TmaxSoft 제품설치 지원 담당자에게 문의하도록 한다.

ProObject를 사용하기 위해 필요한 기반 제품들의 전반적인 설치 과정과 ProObject 서버 설치 그리고 연동 가이드를 제공한다. 각 계정에 JEUS를 우선 설치한 후 ProObject 서버별 설치 과정을 설명한다.



설치 구성도

DevOps 모듈은 ProObject의 시스템 프로그램으로 수행된다. 이를 위해 DevOps 바이너리는 `${PROOJECT_HOME}/system/devops`에 위치한다.

전반적인 설치 순서는 다음과 같다.

1. JEUS, DB, 런타임 서버를 설치한다.

JESUS와 PO 런타임 war가 배포된 서버에 개발 서버를 구성하는 바이너리를 배포한다. JEUS, DB, 런타임 서버를 설치했다고 가정하고 설명한다. 런타임 서버 설치에 대한 자세한 내용은 [런타임 엔진 설치 및 제거](#)를 참고한다.

2. 설치전 준비사항을 확인한다.

- system 디렉터리에 DevOps 모듈을 구성하는 서비스 그룹 바이너리를 배포한다. ([디렉터리 구조](#) 참고)
- DB 계정 생성 및 JDBC 드라이버 설정한다. ([DB 계정 생성 및 JDBC 드라이버 설정](#) 참고)
- 서버 설정 정보를 수정한다. ([설정 파일 확인](#) 참고)
- DevOps 환경을 구성하는데 필요한 라이브러리를 확인한다. ([라이브러리 확인](#) 참고)

3. Dev/Ops 서버를 설치한다. 서버 설치에 대한 자세한 내용은 [설치](#)를 참고한다.

4. Dev/Ops 서버의 정상적인 설치 여부를 확인한다. 설치 확인에 대한 자세한 내용은 [설치확인](#)을 참고한다.

3.2. 설치 전 준비사항

본 절에서는 Dev/Ops 서버를 설치하기 전에 확인이 필요한 사항에 대해서 설명한다.

3.2.1. 디렉터리 구조

DevOps 모듈은 ProObject의 시스템 프로그램으로 ProObject를 기반으로 개발되었으나 일반 사용자의 애플리케이션과 다른 디렉터리에 배포된다.

다음은 system 디렉터리 구조에 대한 설명이다.

```
{PROOJECT_HOME}
|--application
|--config
|--bin
|--logs
|--(resource)
|--system
    |--patch
    |--config
    |--devops
        |--aj
        |--binary
        |--jenkins
            |--devclient
            |--script
                |--3rdPartyLib
            |--lib
```

```
|--event
|--metaSample
```

(resource)

Jenkins에서 빌드가 이루어지면서 프로그램에서 자동 생성한다.

system

- system/patch

런타임 엔진의 클래스 패치시에 해당 클래스가 위치하는 디렉터리이다.

- system/config

런타임 엔진에서 보는 DevOps의 application.xml, 서비스 그룹별 properties 파일이 위치한다.

- system/devops

다음은 하위 폴더에 대한 설명이다.

파일	설명
aj	빌드할 때 참조하는 aj 파일(memory calculator 등)이 위치한다.
binary	<p>DevOps 프로그램(서비스 그룹 jar, dto jar)이 위치한다.</p> <p>서비스 그룹은 SO, BO, DOF로 묶은 <i>{servicegroup-name}.jar</i>와 DO, JSON 메시지로 묶은 <i>{servicegroup-name}-dto.jar</i>로 구성된다. 해당 바이너리들이 위치하는 디렉터리는 <code>\${PROJECT_HOME}/system/devops/binary</code>이다.</p> <p>DevOps 모듈을 구성하는 서비스 그룹은 다음과 같다.</p> <ul style="list-style-type: none"> • proobject-devserver : ProStudio에서 호출하는 서비스(메타 검색, app/sg 조회 등), Jenkins 빌드 후 실행되는 배포 요청 서비스 등을 담당한다. • proobject-manager : 개발계 ProManager에서 호출하는 서비스를 담당한다. • proobject-managerops : 운영계 ProManager에서 호출하는 서비스를 담당한다. • proobject-deployserver : 사용자 애플리케이션을 테스트 노드로 배포하는 서비스를 담당한다. (추후 master로 통합 예정) • proobject-master : 노드관리, 사용자 애플리케이션을 운영 노드로 배포하는 서비스를 담당한다. • proobject-devclient : Jenkins에서 구성한 빌드 프로세스 중 스크립트를 이용해 ProObject 서비스를 호출할 때 사용하는 클라이언트 모듈이다. 다른 서비스 그룹과 다르게 dto는 없으며, <code>\${PROJECT_HOME}/system/devops/devclient</code>에 위치한다.

파일	설명
jenkins	<p>Jenkins 빌드할 때 스크립트를 실행하는 프로그램 및 빌드할 때 참조하는 라이브러리 위치한다.</p> <ul style="list-style-type: none"> • devclient : Jenkins 빌드 중 ProObject 서비스 호출할 때 사용되는 클라이언트 모듈 위치 • script/3rdPartyLib : Jenkins 빌드 중 사용하는 라이브러리 위치 • lib : DevOps 서비스 그룹이 참조하는 라이브러리들 위치 • event : DevOps에서 사용하는 이벤트 관련 바이너리 • metaSample : ProManager에서 사용하는 메타 샘플 파일이 위치

3.2.2. DB 계정 생성 및 JDBC 드라이버 설정

개발서버(DevOps) 환경구성을 위해 우선적으로 제품에서 사용하는 Repository DB를 구성한다.

다음은 Dev 계정과 Ops 계정에서 사용할 DB 계정을 2개를 생성하는 과정에 대한 설명이다.



생성 과정은 DB가 설치되어 있다는 전제하에 진행한다. 설치과정에 실행하는 DB 스크립트는 신규 설치를 위한 Full DB 스크립트이다.

1. DEV DB 계정에 접속하여 DB_Script/개발 폴더 안에 있는 Create_PO7_Table.sql, Initialize_PO7_Table.sql 실행한다.
2. OPS DB 계정에 접속하여 DB_Script/운영 폴더 안에 있는 Create_PO7_Table-operation.sql, Initialize_PO7_Table-operation.sql, Create_PO7_Table-runtime.sql 실행한다.
3. JEUS 서버의 데이터소스 구성을 위해 사용하는 DB의 JDBC 드라이버를 각 환경에 맞게 넣어준다.

```

${JEUS_HOME}/lib/datasource

```

3.2.3. 설정 파일 확인

설치에 필요한 설정 파일의 설정 값을 확인한다.

- **application.xml**

PO 런타임이 DevOps 모듈을 애플리케이션으로 인식하기 위해 필요한 application.xml 설정 파일은 `${PROOBJECT_HOME}/system/config`에 위치한다.

- **사용자 설정이 필요한 환경파일**

사용자 설정 파일은 `${PROOBJECT_HOME}/config`에 위치한다.

- dbio_config.xml

ProStudio에서 데이터 오브젝트 팩토리(DataObjectFactory)를 구현할 때 후에 운영노드에서 쿼리를 수행하는 경우 접속할 데이터베이스 정보를 얻어 입력하는데 사용한다.

- PoDevSvr.xml

ProObject DEV에서 관리하는 DB와 서비스 수행하면서 필요한 값을 설정한다.

```
DB Setting:

DB_TYPE=db_type
DB_USER_ID=db_user_id
DB_PASSWD=db_passwd
DATA_SOURCE=datasource_name
DB_PLUG_NAME=db_plug_name
```

항목	설명
DB_TYPE	DB 종류를 설정한다. (예: TIBERO, ORACLE)
DB_USER_ID	DB에 접속할 수 있는 사용자 계정 ID를 설정한다.
DB_PASSWD	'DB_USER_ID'에 입력한 사용자 계정 ID에 접속할 수 있는 비밀번호를 설정한다.
DATA_SOURCE	JEUS에 등록된 데이터소스 이름을 설정한다.
DB_PLUG_NAME	현재 기본값은 com.tmax.proobject.commonbo.util.db.DataSourcePlug이다.

- PoOpsSvr.xml

ProObject Master(Ops)에서 서비스 수행하면서 필요한 값을 설정한다. DB Setting에 해당 항목에 대한 설명은 [PoDevSvr.xml](#)를 참고한다.

```
DB_TYPE=db_type
DB_USER_ID=db_user_id
DB_PASSWD=db_passwd
DATA_SOURCE=datasource_name
DB_PLUG_NAME=db_plug_name

DEPLOY_BASE_HOME=deploy_base_home
PO_CONFIG_PATH=po_config_path
```

항목	설명
DEPLOY_BASE_HOME	deploy된 바이너리들이 저장되는 파일 디렉터리 장소를 설정한다.
PO_CONFIG_PATH	Default Config(PoOpsSvr.xml) 경로를 설정한다.

- ProbuilderConfig.xml

ProStudio에서 DTO를 생성하는 경우 필드 타입 관련 설정을 한다.

- SiteConfig.xml

ProStudio의 각종 옵션을 설정한다.

• ProManager 설정

- 서버 설정

아래 파일은 \${PROOBJECT_HOME}/config에 위치한다.

파일	설명
ProManager.properties	개발계 ProManager에서 사용하는 데이터소스 이름을 설정한다.
ProManagerOps.properties	운영계 ProManager에서 사용하는 데이터소스 이름을 설정한다.

- 웹 애플리케이션 설정

파일	설명
proobject-manager.war	하위 setting.js 파일에 개발계 PO 서버의 ip, http port를 설정한다.
proobject-managerops.war	하위 setting.js 파일에 운영계 PO 서버의 ip, http port를 설정한다.

3.2.4. 라이브러리 확인

DevOps 환경을 구성하는데 필요한 라이브러리 목록이다. .

• 서비스 그룹이 참조하는 라이브러리

다음은 \${PROOBJECT_HOME}/system/devops/lib에 위치하는 DevOps를 구성하는 서비스 그룹들이 참조하는 라이브러리 목록이다.

```
commons-dbutils-1.5.jar
commons-io-2.4.jar
commons-lang-2.6.jar
freemarker-2.3.20.jar
guava-17.0.jar
javapoet-1.7.0.jar
jsr173_api-1.0.jar
light-prominer-0.0.1-SNAPSHOT.jar
org.aspectj.ajde_1.8.9.201604061446.jar
org.eclipse.core.commands-3.6.0.jar
org.eclipse.core.contenttype-3.4.100.jar
org.eclipse.core.expressions-3.4.300.jar
org.eclipse.core.filesystem-1.3.100.jar
org.eclipse.core.jobs-3.5.100.jar
org.eclipse.core.resources_3.10.1.v20150725-1910.jar
org.eclipse.core.runtime-3.7.0.jar
org.eclipse.equinox.app-1.3.100.jar
```

```
org.eclipse.equinox.common-3.6.0.jar
org.eclipse.equinox.preferences-3.4.1.jar
org.eclipse.equinox.registry-3.5.101.jar
org.eclipse.jdt.core-3.10.0.jar
org.eclipse.osgi-3.7.1.jar
org.eclipse.text-3.5.101.jar
org.eclipse.xtext.xbase.lib-2.10.0.jar
proobject-client-7.0.0.0.jar
proobject-compiler.jar
proobject-devutil-7.0.0.0.jar
proobject-opsutil-7.0.0.0.jar
proobject-prominer-7.0.0.0.jar
proobject-prominer-dto-7.0.0.0.jar
proobject-prominer-service-7.0.0.0.jar
proobject-srcgen-7.0.0.0.jar
proobject-srcgen-dto-7.0.0.0.jar
stax2-api-3.1.4.jar
woodstox-core-asl-4.4.1.jar
poi-3.15.jar
```

• Git/Jenkins를 이용하는 환경 구성 시 필요한 라이브러리

다음은 `${PROOBJECT_HOME}/system/devops/jenkins/script/3rdPartyLib`에 위치하는 Jenkins를 빌드하는 경우 참조하는 라이브러리 목록이다.

```
aspectj-1.8.10.jar
aspectjrt-1.8.9.jar
aspectjtools-1.8.9.jar
aspectjweaver.jar
org.aspectj.matcher.jar
proobject-client-7.0.0.0-jar-with-dependencies.jar
```

3.3. 설치

본 절에서는 Dev 서버와 Ops 서버의 설치를 하는 과정에 대해서 설명한다.

3.3.1. Dev 서버

다음의 과정으로 Dev 서버 환경을 구성한다.

1. ProObject7\개발\proobject7 폴더를 Dev 계정의 홈에 넣고 `.bash_profile(.profile)`을 다음과 같이 설정한다.

```
export PROOBJECT_HOME=/home/{계정명}/proobject7
```

2. `aspectj1.8` 폴더를 Dev 계정의 홈에 넣고 `/home/{계정명}/aspectj1.8/bin`에 있는 `ajc` 파일에 755 권한을 부여하고, `gradle-4.9` 폴더를 Dev 계정의 홈에 넣고 `/home/{계정명}/gradle-4.9/bin`에 있는 `gradle` 파일에 755 권한을 부여한다.

3. 프로파일개발\profile을 참고하여 아래와 같이 설정한다.

```
export ASPECTJ_HOME=/home/{계정명}/aspectj1.8
export GRADLE_HOME=/home/{계정명}/gradle-4.9
export PATH=$ASPECTJ_HOME/bin:$GRADLE_HOME/bin:$PATH
```

4. JEUS\{개발}\domain.xml 파일의 설정을 수정한다(굵은 글씨).

a. domain Id, port 값은 한 장비 내에 다른 계정과 중복으로 사용하여 충돌되지만 않으면 된다.

<jvm-option> 값은 계정이 다를 경우 사용자 홈 계정에 맞게 설정한다. 해당 리스너 이름은 추후에 proobject.xml의 <http-listener>와 같게 설정해야 한다. Dev 서버는 <datasource>를 사용해야 한다.

```
<!-- DevServer -->
<server>
  <name>DevServer</name>
  <listeners>
    <base>base</base>
    <listener>
      <name>base</name>
      <listen-address>0.0.0.0</listen-address>
      <listen-port>13000</listen-port>
    </listener>
    <listener>
      <name>http-server</name>
      <listen-address>0.0.0.0</listen-address>
      <listen-port>14000</listen-port>
      <keep-alive-timeout>600000</keep-alive-timeout>
    </listener>
  </listeners>
  <jvm-config>
    <jvm-option>-Xmx256m -XX:MaxPermSize=128m</jvm-option>
    <jvm-option>-DPROOJECT_HOME=/home/po7dev/proobject7</jvm-option>
  </jvm-config>
  <web-engine>
    :
    :
    <web-connections>
      <http-listener>
        <name>httpdev</name>
        <server-listener-ref>http-server</server-listener-ref>
        <thread-pool>
          <min>10</min>
          <max>20</max>
        </thread-pool>
      </http-listener>
    </web-connections>
    :
    :
    <data-sources>
      <data-source>tibero6_dev</data-source>
    </data-sources>
    :
    :
  </web-engine>
```

- b. 계정명을 다르게 생성하였을 때만 홈 계정 경로에 맞게 수정한다. <target-server>의 <name>은 생성한 서버의 이름이다.

```
<!-- Promanager Application Deploy -->
<deployed-application>
  <id>promanager.war</id>
  <path>/home/po7dev/proobject7/_for_jeus/proobject-manager-war-7.0.0.0.war</path>
  <type>WAR</type>
  <target-server>
    <name>DevServer</name>
  </target-server>
  <classloading>ISOLATED</classloading>
  <use-fast-deploy>>false</use-fast-deploy>
  <keep-generated>>false</keep-generated>
  <shared>>false</shared>
  <node-java-context>>false</node-java-context>
</deployed-application>

<!-- Runtime Application Deploy -->
<deployed-application>
  <id>proobject-runtime.war</id>
  <path>/home/po7dev/proobject7/_for_jeus/proobject-runtime.war</path>
  <type>WAR</type>
  <target-server>
    <name>DevServer</name>
  </target-server>
  <classloading>ISOLATED</classloading>
  <use-fast-deploy>>false</use-fast-deploy>
  <keep-generated>>false</keep-generated>
  <shared>>false</shared>
  <node-java-context>>false</node-java-context>
</deployed-application>
</deployed-applications>
```

- c. <server>에 설정한 <data-source> 값과 같게 설정해야 한다(<data-source-id>, <export-name>). 사용자 환경의 IP, DB port, DB 계정에 맞게 설정한다.

```
<data-source>
  <database>
    <data-source-id>tibero6_dev</data-source-id>
    <export-name>tibero6_dev</export-name>
    <data-source-class-name>com.tmax.tibero.jdbc.ext.TbConnectionPoolDataSource</data-
source-class-name>
    <data-source-type>ConnectionPoolDataSource</data-source-type>
    <vendor>tibero</vendor>
    <server-name>192.168.3.38</server-name>
    <port-number>8629</port-number>
    <database-name>tibero</database-name>
    <user>po7devdb</user>
    <password>po7devdb</password>
    <login-timeout>0</login-timeout>
    <auto-commit>DRIVER</auto-commit>
    <stmt-query-timeout>0</stmt-query-timeout>
    <pool-destroy-timeout>10000</pool-destroy-timeout>
    <property>
      <name>driverType</name>
```

```

    <value>thin</value>
    <type>java.lang.String</type>
  </property>
  <support-xa-emulation>>false</support-xa-emulation>
  <connection-pool>
    :
    :
  </data-source>

```

5. proobject-manager-war-7.0.0.0.war에 \${PROOBJECT_HOME}/_for_jeus에서 setting.js 파일을 아래와 같이 수정한 후 반영한다.

```

var poDevSrvInfo = Top.Data.create({
  ip: "192.168.105.111",
  port: "14000",
  App: "/proobject",
  SG: "/proobject-manager",
  Service: ""
});

var testSrvInfo = Top.Data.create({
  ip: "192.168.105.111",
  port: "14000",
  App: "/proobject",
  SG: "/proobject-testserver",
  Service: ""
});

var DEVOPS_version = "DB";

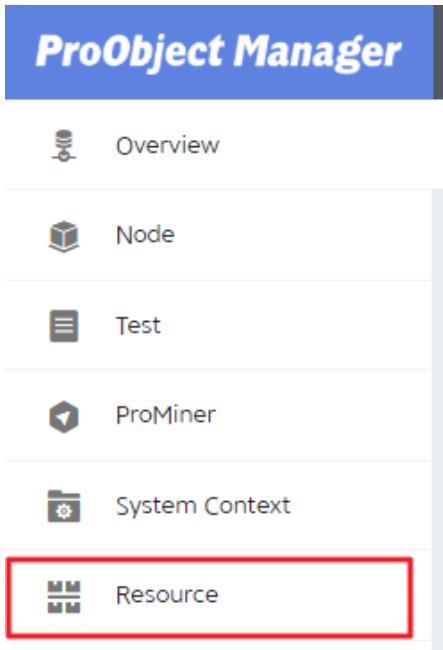
```

다음의 설정을 입력하면 **ProManager**에 **[Resource]** 탭이 추가된다. (DB 버전에서 반드시 필요한 옵션)

```

var DEVOPS_version = "DB";

```



ProManager 메인 화면 메뉴

6. `${PROOBJECT_HOME}/config`에 있는 `proobject.xml` 파일을 수정한다.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE xml>
<ProObjectConfig>
  <single-application>>false</single-application>
  <file-port>4445</file-port>
  <proobject-port>6777</proobject-port>
  <server-name>DevServer</server-name>
  <engine-config>
    <listener-name>httpdev</listener-name>
    <host-name>ns.test.local</host-name>
    <!--container-name>ProObject7</container-name-->
    <proObject-datasource>tibero6_dev</proObject-datasource>
    <initial-context-factory>jeus.jndi.JEUSContextFactory</initial-context-factory>
    <provider-url>192.168.105.111:13000</provider-url>
  </engine-config>
</ProObjectConfig>
```

항목	설명
<file-port>	파일 포트 정보를 설정한다. (다른 계정과 충돌하지 않도록 설정해야 함)
<proobject-port>	PO 포트 정보를 설정한다. (다른 계정과 충돌하지 않도록 설정해야 함)
<server-name>	JEUS의 domain.xml에 설정한 Dev(Ops, RTE) 서버의 서버명을 설정한다.
<listener-name>	JEUS의 domain.xml에 설정한 Dev(Ops, RTE) 서버의 http-listener name을 설정한다.
<host-name>	사용자 장비의 hostname 정보를 설정한다.
<proObject-datasource>	JEUS의 domain.xml에 설정한 데이터소스 정보를 설정한다.
<provider-url>	JNDI Lookup하는 경우 접속할 서버의 주소를 설정한다. 설정하지 않을 경우 현재 서버를 사용한다.

7. \${PROOJECT_HOME}/system/config에 있는 dbio_config.xml 파일을 domain.xml에 있는 <datasource> 부분과 동일하게 설정한다.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dbio-config xmlns="http://www.tmax.co.kr/proobject/dbio-config">
  <connection-info>
    <datasources>
      <pairDataSource alias="tiber06_dev">
        <non-XA-datasource jndi_name="tiber06_dev" />
        <XA-datasource jndi_name="tiber06_dev" />
      </pairDataSource>
    </datasources>
    <async-jdbc conn_name="tiber06_dev" dbname="tiber0" userid="po7devdb"
      passwd="po7devdb" ip="192.168.3.38" port="8629" />

    <!-- for Studio DO Factory target DB -->
    <studio-jdbc conn_name="tiber06_dev" username="po7devdb" passwd="po7devdb"
      driver="com.tmax.tiber0.jdbc.TbDriver" pool_size=""
      url="jdbc:tiber0:thin:@192.168.3.38:8629:tiber0" />
  </connection-info>
```

8. \${PROOJECT_HOME}/system/config에 있는 PoDevSvr.xml 파일을 domain.xml에 있는 <datasource> 부분과 동일하게 설정한다.

```
<!-- DB Setting -->
<configField id="DB_TYPE" value="TIBERO" type="String" xmlns="" />
<configField id="DB_USER_ID" value="po7devdb" type="String" xmlns="" />
<configField id="DB_PASSWD" value="po7devdb" type="String" xmlns="" />
<configField id="DATA_SOURCE" value="tiber06_dev" type="String" xmlns="" />
<configField id="DB_PLUG_NAME" value="com.tmax.proobject.commonbo.util.db.DataSourcePlug"
  type="String" xmlns="" />
```

DB Setting 이외 나머지 모두 Dev 서버, 계정명, 경로로 설정한다.

```
<!-- Directory Setting -->
<configField id="BASE_HOME" value="/home/po7dev/proobject7/resources/srcs" type="String"
  xmlns="" />
<configField id="PO_APP_HOME" value="/home/po7dev/proobject7/resources/deploy_service_group"
  type="String" xmlns="" />
<configField id="CLASS_HOME" value="/home/po7dev/proobject7/resources/classes" type="String"
  xmlns="" />
<configField id="METAS_HOME" value="/home/po7dev/proobject7/resources/metass" type="String"
  xmlns="" />
<configField id="TEMP_HOME" value="/home/po7dev/proobject7/resources" type="String" xmlns="" />
<configField id="ETC_HOME" value="/home/po7dev/proobject7/resources/etcs" type="String"
  xmlns="" />
<configField id="EXTRA_LIBRARY_DIR" value="/home/po7dev/proobject7/lib/extra" type="String"
  xmlns="" />
<configField id="EXTRA_LIBRARY" value="" type="String" xmlns="" />
<configField id="TRANSFER_DOWNLOAD_HOME" value="/home/po7dev/proobject7/resources" type="String"
  xmlns="" />

<!-- Config directory setting -->
```

```

<!-- 20180622 path edit -->
<configField id="PO_CONFIG_PATH" value="/home/po7dev/proobject7/system/config" type="String"
xmlns=""/>

<!-- System Setting -->
<configField id="ETC_HOME" value="/home/po7dev/proobject7/resources/etcs" type="String"
xmlns=""/>
<configField id="EXTRA_LIBRARY_DIR" value="/home/po7dev/proobject7/lib/extra" type="String"
xmlns=""/>
<configField id="EXTRA_LIBRARY" value="" type="String" xmlns=""/>
<configField id="TRANSFER_DOWNLOAD_HOME" value="/home/po7dev/proobject7/resources" type="String"
xmlns=""/>

<!-- Config directory setting -->
<!-- 20180622 path edit -->
<configField id="PO_CONFIG_PATH" value="/home/po7dev/proobject7/system/config" type="String"
xmlns=""/>

<!-- System Setting -->
<configField id="JDK_VERSION" value="1.7" type="String" xmlns=""/>
<configField id="CLASSPATH_HOME"
value="/home/po7dev/proobject7/resources/classes:
/home/po7dev/proobject7/application/system/devops/lib/guava-17.0.jar:
/home/po7dev/proobject7/application/system/devops/lib/proobject-compiler.jar:
/home/po7dev/proobject7/application/system/devops/lib/javapoet-1.7.0.jar:
/home/po7dev/proobject7/application/system/devops/lib/org.eclipse.text_3.5.101.jar:
/home/po7dev/proobject7/application/system/devops/lib/org.eclipse.core.resources_3.10.1.v20150725
-1910.jar:
/home/po7dev/proobject7/application/system/devops/lib/org.aspectj.ajde_1.8.9.201604061446.jar:
/home/po7dev/proobject7/application/system/devops/lib/commons-dbutils-1.5.jar" type="String"
xmlns=""/>

<!-- <configField id="DD_HOME" value="home/po7rel/proobject7/resources/metasploit/" type="STRING"
xmlns=""/> -->

<!-- Service Setting -->

<!-- Common -->
<configField id="IS_CHK_RESOURCE_ID" value="TRUE" type="String" xmlns=""/>
<configField id="DELETE_DEPLOYED_RESOURCE" value="TRUE" type="STRING" xmlns=""/>

<!-- ResourcesDelete/ResourcesUpdate service -->
<configField id="IS_DBIO_DEP" value="TRUE" type="String" xmlns=""/>
<configField id="IS_DELETE_TEMP_WORK_DIR" value="TRUE" type="String" xmlns=""/>
<configField id="UNIQUE_PHYSICAL" value="TRUE" type="String" xmlns=""/>

<!-- Source Gen Setting -->
<configField id="PROMAPPER_SOURCE_GENERATE_ON_SERVER" value="TRUE" type="String" xmlns=""/>

<!-- Build Server -->
<configField id="BUILD_BASE_HOME" value="/home/po7dev/proobject7/resources/srcs/_build"
type="String" xmlns=""/>
<!-- if BUILD_BASE_HOME does not exist, default directory is
${PROBJECT_HOME}/resources/srcs/_build .-->
<configField id="BUILD_CLASS_HOME" value="/home/po7dev/proobject7/resources/classes/_build"
type="String" xmlns=""/>

<!-- if BUILD_CLASS_HOME does not exist, default directory is
${PROBJECT_HOME}resources/classes/_build .-->

```

```
<!-- Deploy Server -->
<configField id="DEPLOY_BASE_HOME" value="/home/po7dev/proobject7/resources/classes/_deploy"
type="String" xmlns=""/>
<!-- if DEPLOY_BASE_HOME does not exist, default directory is
${PROOJECT_HOME}/resources/classes/_deploy .-->
```

9. \${PROOJECT_HOME}/system/config에 있는 proManager.properties 파일을 domain.xml에 있는 <datasource> 부분과 동일하게 설정한다.

```
DataSource=tibero6_dev
```

10. .bash_profile(.profile)에 alias 설정한 후 정상 기동 여부를 확인한다.

다음은 dasboot > devboot 순서대로 기동하도록 설정한 내용이다. (프로파일개발\profile 참고).

```
alias dasboot='startDomainAdminServer -u jeus -p jeus'
alias dasdown='stopServer -host 127.0.0.1:9736 -u jeus -p jeus -verbose'
alias devboot='startManagedServer -domain domain1 -server DevServer -u jeus -p jeus'
alias devdown='stopServer -host 127.0.0.1:13000 -u jeus -p jeus'
```

\${PROOJECT_HOME}/logs/ProObject.log에 다음과 같이 출력되는지 확인한다.

```
[2019.11.27 13:39:16][INFO] ===== ProObject Booting =====
[2019.11.27 13:39:16][INFO] version : 7.0.0.1.244
[2019.11.27 13:39:16][INFO] Global Deploy Version : 0
[2019.11.27 13:39:16][INFO] License : standard
[2019.11.27 13:39:16][INFO] Installed Path : /home/po7dev/proobject7/
[2019.11.27 13:39:16][INFO] Application Path : /home/po7dev/proobject7/application/
[2019.11.27 13:39:16][INFO] =====
[2019.11.27 13:39:16][INFO] Starting Booting sequence.
[2019.11.27 13:39:16][INFO] [BootLoader][ChannelManager] Starting initialing.
[2019.11.27 13:39:16] [INFO] [ChannelManager][NodeJAVA] NodeJAVA is detected. Trying to
initialize with web-connection "httpdev"
[2019.11.27 13:39:16] [INFO] [BootLoader][ChannelManager] Initialized successfully.
[2019.11.27 13:39:16] [INFO] [BootLoader][EventManager] Starting initialing.
[2019.11.27 13:39:16] [INFO] [BootLoader][EventManager] Initialized successfully.
[2019.11.27 13:39:16] [INFO] [BootLoader][Application] Starting initialing.
[2019.11.27 13:39:16] [INFO] SYSTEM_QOS_BOOT_ENABLE : false
[2019.11.27 13:39:16] [INFO] -- [APPLICATION : proobject] initializing
[2019.11.27 13:39:17] [INFO] -- Initializing ServiceGroup [proobject - system]
[2019.11.27 13:39:17] [INFO] -- Initializing ServiceGroup [proobject - monitoring]
[2019.11.27 13:39:17] [INFO] -- Initializing ServiceGroup [proobject - proobject-master]
[2019.11.27 13:39:18] [INFO] -- Initializing ServiceGroup [proobject - proobject-deployserver]
[2019.11.27 13:39:18] [INFO] -- Initializing ServiceGroup [proobject - proobject-manager]
[2019.11.27 13:39:18] [INFO] -- Initializing ServiceGroup [proobject - proobject-devserver]
[2019.11.27 13:39:18] [INFO] -- Initializing ServiceGroup [proobject - proobject-managerops]
[2019.11.27 13:39:18] [INFO] -- Initializing ServiceGroup [proobject - proobject-testserver]
[2019.11.27 13:39:18] [WARNING] [NodeAddressManager] Config File(remote_servicegroup.xml)
is missing, remote address manager is disabled.
[2019.11.27 13:39:18] [INFO] [BootLoader][Application] Initialized successfully.
[2019.11.27 13:39:18] [INFO] -- ProObject Port 6777 is bound ...
```

```

[2019.11.27 13:39:18] [INFO] -- File Port 4445 is bound ...
[2019.11.27 13:39:18] [INFO] Properties Validation :
/home/po7dev/proobject7/config/system.properties
[2019.11.27 13:39:18] [INFO] Properties File Last Modified Date : 2019/11/27/13:26:00
[2019.11.27 13:39:18] [INFO]
| Property | Value | Default Value |
|-----|-----|-----|
| Property | Value | Default Value |
|-----|-----|-----|
| SYSTEM_APPLICATION | | |
| false | - |
| SYSTEM_CHARSET | UTF-8 | UTF-8 |
| false | - |
| SYSTEM_GLOBAL_DEPLOY_VERSION | 0 | 0 |
| false | - |
| SYSTEM_SERVICENAME_CASE | CAMEL | CAMEL |
| false | - |
| SYSTEM_TIMEOUT | 60000 | 60000 |
| false | - |
| SYSTEM_WEB_PARAMETER_CHARSET | ISO-8859-1 | ISO-8859-1 |
| false | - |
| SYSTEM_WEB_READ_BUFFER_SIZE | 50000 | 16384 |
| false | - |
| SYSTEM_WEB_WRITE_BUFFER_SIZE | 50000 | 16384 |
| false | - |
| SYSTEM_EVENTMANAGER_LOG_FILEHANDLER_REMOVAL_PERIOD | 60 | 1 |
| false | - |
| SYSTEM_CHANNELEVENTMANAGER_LOG_FILEHANDLER_REMOVAL_PERIOD | 60 | 1 |
| false | - |
| SYSTEM_PROOBJECT_LOG_FILEHANDLER_REMOVAL_PERIOD | 60 | 1 |
| false | - |
| SYSTEM_LOG_LEVEL | INFO | INFO |
| false | - |

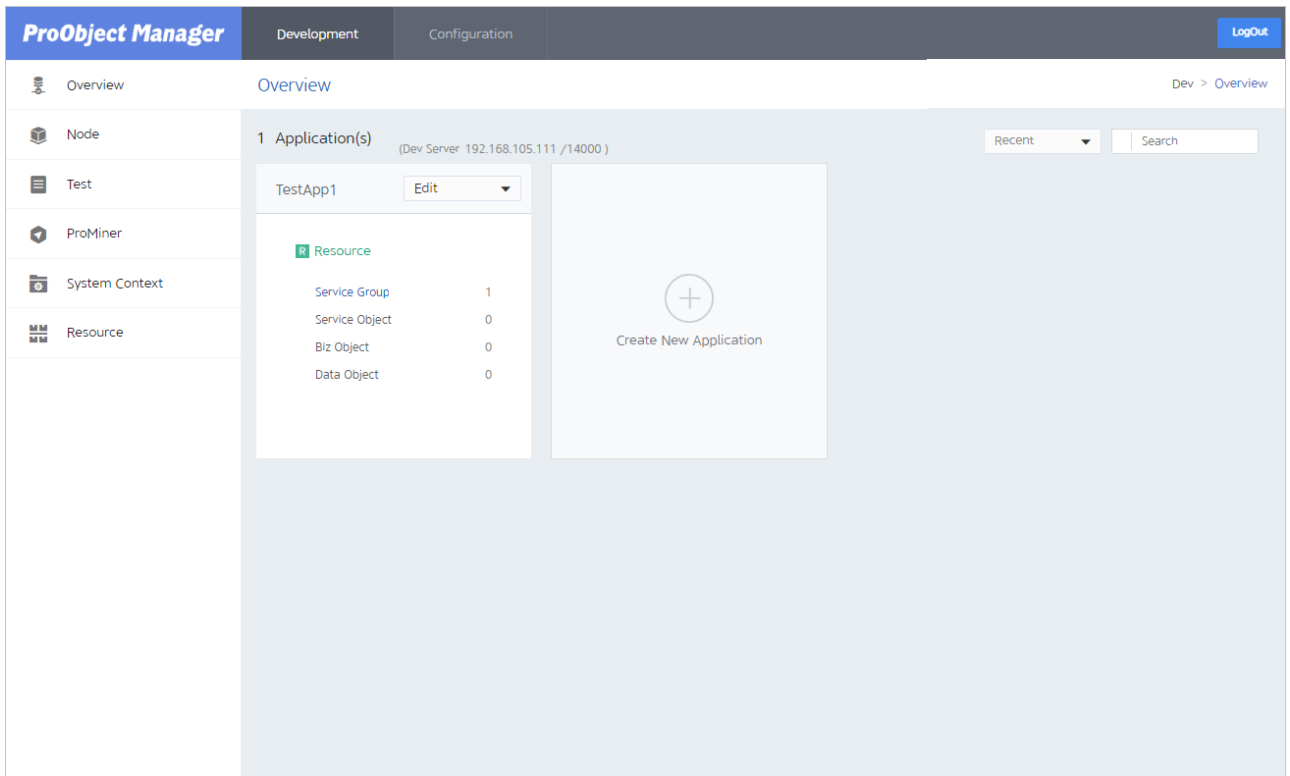
[2019.11.27 13:39:18] [INFO] FAIL MESSAGE
[2019.11.27 13:39:18] [INFO] SYSTEM_GLOBAL_BUFFER_SIZE : Wrong Property
[2019.11.27 13:39:18] [INFO] SYSTEM_HOTDEPLOY_VERSION_CONSISTENCY_POLICY : Wrong Property
[2019.11.27 13:39:18] [INFO] SYSTEM_DATAOBJECT_QUERY_LOGGING_LEVEL : Wrong Property
[2019.11.27 13:39:18] [INFO] SYSTEM_APPLICATION : Split error on property SYSTEM_APPLICATION
[2019.11.27 13:39:18] [INFO]

[2019.11.27 13:39:18] [INFO] [ChannelManager] Activating Configured Channels.
[2019.11.27 13:39:18] [INFO] [ChannelManager] Configured channels are activated.
[2019.11.27 13:39:18] [INFO] Booting is completed. ProObject is ready.
[2019.11.27 13:39:18] [INFO] [GUID-ServerReady]<<proobject.system.FileSequencerUpdateService>>
is ready for execution : Method - service, WaitObject - null
[2019.11.27 13:39:18] [INFO] [GUID-ServerReady]<<proobject.system.FileSequencerUpdateService>>
done for executing : Method - service
[2019.11.27 13:39:18] [INFO] [GUID-ServerReady]<<proobject.system.FileSequencerUpdateService>>
is done!

```

11. 다음 경로에 접속하여 정상적으로 실행되었는지 확인한다.

```
http://192.168.105.111:14000/promanager
```



ProManager 메인화면



팝업과 함께 로그인 안된다면 dbio_config.xml, setting.js, proManager.properties, PoDevSvr.xml을 확인해서 ProObject의 DB 설정을 확인한다.

domain.xml과 proobject.xml 파일 설정 비교

다음은 각 설정된 domain.xml과 proobject.xml 파일의 예이다. 각 설정된 항목을 비교할 수 있다.

- <domain.xml>

```
<servers>
  <server>
    <name>adminServer</name>
    <!-- node-name>ns.test.local</node-name -->
    <listeners>
      <base>base</base>
      <listener>
        <name>base</name>
        <listen-address>0.0.0.0</listen-address>
        <listen-port>9736</listen-port>
      </listener>
      :
      :
    <!-- DevServer -->
    <server>
      <name>DevServer</name>
      <listeners>
        <base>base</base>
        <listener>
          <name>base</name>
```

```

        <listen-address>0.0.0.0</listen-address>
        <listen-port>13000</listen-port>
    </listener>
    <listener>
        <name>http-server</name>
        <listen-address>0.0.0.0</listen-address>
        <listen-port>14000</listen-port>
        <keep-alive-timeout>60000</keep-alive-timeout>
    </listener>
</listeners>
<jvm-config>
    <jvm-option>-Xmx256m -XX:MaxPermSize=128m</jvm-option>
    <jvm-option>-DPROJECT_HOME=/home/po7dev/proobject7</jvm-option>
</jvm-config>
<web-engine>
    :
    :
<web-connections>
    <http-listener>
        <name>httpdev</name>
        <server-listener-ref>http-server</server-listener-ref>
        <thread-pool>
            <min>10</min>
            <max>20</max>
        </thread-pool>
    </http-listener>
</web-connections>
    :
    :
<data-sources>
    <data-source>tibero6_dev</data-source>
</data-sources>
    :
    :
<data-source>
    <database>
        <data-source-id>tibero6_dev</data-source-id>
        <export-name>tibero6_dev</export-name>
        <data-source-class-
name>com.tmax.tibero.jdbc.ext.TbConnectionPoolDataSource</data-source-class-name>
        <data-source-type>ConnectionPoolDataSource</data-source-type>
        <vendor>tibero</vendor>
        <server-name>192.168.3.38</server-name>
        <port-number>8629</port-number>
        <database-name>tibero</database-name>
        <user>po7devdb</user>
        <password>po7devdb</password>
    </data-sources>
    :
    :
</server>
</servers>

```

- <proobject.xml>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE xml>

```

```

<ProObjectConfig>
  <single-application>false</single-application>
  <file-port>4445</file-port>
  <proobject-port>6777</proobject-port>
  <server-name>DevServer</server-name>
  <engine-config>
    <listener-name>httpdev</listener-name>
    <host-name>ns.test.local</host-name>
    <!--container-name>ProObject7</container-name-->
    <proobject-datasource>tibero6_dev</proobject-datasource>
    <initial-context-factory>jeus.jndi.JEUSContextFactory</initial-context-factory>
    <provider-url>192.168.105.111:13000</provider-url>
  </engine-config>

```

3.3.2. Ops 서버

다음은 Ops 서버 환경을 구성하는 방법에 대한 설명이다.

1. ProObject7\운영\proobject7 폴더를 Ops 계정의 홈에 넣어준다.

'bash_profile(.profile)'을 다음과 같이 설정한다.

```
export PROOBJECT_HOME=/home/계정명/proobject7
```

2. JEUS\운영\domain.xml 파일을 설정한다.

- a. 도메인 ID, port 값은 한 장비 내에 다른 계정과 중복으로 사용하며 충돌되지만 않으면 된다. <jvm-option>값은 계정이 다를 경우 사용자 홈 계정에 맞게 설정한다. 해당 리스너 이름은 추후에 proobject.xml의 <http-listener>와 같게 설정해야 한다.

```

<!-- OpsServer -->
<server>
  <name>OpsServer</name>
  <listeners>
    <base>base</base>
    <listener>
      <name>base</name>
      <listen-address>0.0.0.0</listen-address>
      <listen-port>23000</listen-port>
    </listener>
    <listener>
      <name>http-server</name>
      <listen-address>0.0.0.0</listen-address>
      <listen-port>24000</listen-port>
      <keep-alive-timeout>600000</keep-alive-timeout>
    </listener>
  </listeners>
  <jvm-config>
    <jvm-option>-Xmx256m -XX:MaxPermSize=128m</jvm-option>
    <jvm-option>-DPROBJECT_HOME=/home/po7ops/proobject7</jvm-option>
  </jvm-config>
  :
  :

```

```

<web-connections>
  <http-listener>
    <name>https</name>
    <server-listener-ref>http-server</server-listener-ref>
    <thread-pool>
      <min>10</min>
      <max>20</max>
    </thread-pool>
  </http-listener>
</web-connections>
:
:
<data-sources>
  <data-source>tibero6_ops</data-source>
</data-sources>

```

- b. 계정명을 다르게 생성하였을 때만 홈 계정 경로에 맞게 수정한다. <target-server>의 <name>은 생성한 서버의 이름이다.

```

<!-- Promanagerops Application Deploy -->
<deployed-application>
  <id>promanagerops.war</id>
  <path>/home/po7ops/proobject7/_for_jeus/proobject-managerops-war-7.0.0.0.war</path>
  <type>WAR</type>
  <target-server>
    <name>OpsServer</name>
  </target-server>
  <classloading>ISOLATED</classloading>
  <use-fast-deploy>>false</use-fast-deploy>
  <keep-generated>>false</keep-generated>
  <shared>>false</shared>
  <node-java-context>>false</node-java-context>
</deployed-application>

<!-- Runtime Application Deploy -->
<deployed-application>
  <id>proobject-runtime.war</id>
  <path>/home/po7ops/proobject7/_for_jeus/proobject-runtime.war</path>
  <type>WAR</type>
  <target-server>
    <name>OpsServer</name>
  </target-server>
  <classloading>ISOLATED</classloading>
  <use-fast-deploy>>false</use-fast-deploy>
  <keep-generated>>false</keep-generated>
  <shared>>false</shared>
  <node-java-context>>false</node-java-context>
</deployed-application>

```

- c. <server>에 설정한 <data-source> 값과 같게 설정해야 한다(<data-source-id>, <export-name>). 사용자 환경의 IP, DB port, DB 계정에 맞게 설정한다.

```

:
:

```



```

<data-source>
  <database>
    <data-source-id>tibero6_ops</data-source-id>
    <export-name>tibero6_ops</export-name>
    <data-source-class-name>com.tmax.tibero.jdbc.ext.TbConnectionPoolDataSource</data-
source-class-name>
    <data-source-type>ConnectionPoolDataSource</data-source-type>
    <vendor>tibero</vendor>
    <server-name>192.168.3.38</server-name>
    <port-number>8629</port-number>
    <database-name>tibero</database-name>
    <user>po7opsdb</user>
    <password>po7opsdb</password>
  </database>
  :
  :

```

3. proobject-managerops-war-7.0.0.0.war에서 \${PROOBJECT_HOME}/_for_jeus에서 setting.js 파일을 다음과 같이 수정한다.

```

var poOperSrvInfo = Top.Data.create({
  ip: "192.168.105.111", //운영계 ProManager를 배포한 P07 서버 ip (MASTER NODE)
  port: "24000", //운영계 ProManager를 배포한 P07 서버 http port (MASTER NODE)
  App: "/proobject",
  SG: "/proobject-managerops",
  Service: ""
});

```

4. \${PROOBJECT_HOME}/config에 있는 proobject.xml 파일을 수정한다. 포트만 충돌되지 않게 사용하며 나머지는 사용자 환경에 맞게 반드시 변경한다. 각 설정항목에 대한 자세한 내용은 [Dev 서버](#)를 참고한다.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE xml>
<ProObjectConfig>
<single-application>false</single-application>
  <file-port>5445</file-port>
  <proobject-port>7777</proobject-port>
  <server-name>OpsServer</server-name>
  <engine-config>
    <listener-name>httpops</listener-name>
    <host-name>ns.test.local</host-name>
    <!--container-name>ProObject7</container-name-->
    <proobject-datasource>tibero6_ops</proobject-datasource>
    <initial-context-factory>jeus.jndi.JEUSContextFactory</initial-context-factory>
    <provider-url>192.168.105.111:23000</provider-url>
  </engine-config>
  :
  :

```

5. \${PROOBJECT_HOME}/system/config에 있는 dbio_config.xml 파일을 domain.xml에 있는 <datasource> 부분과 동일하게 설정한다.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dbio-config xmlns="http://www.tmax.co.kr/proobject/dbio-config">
  <connection-info>
    <datasources>
      <pairDataSource alias="tiber06_ops">
        <non-XA-datasource jndi_name="tiber06_ops" />
        <XA-datasource jndi_name="tiber06_ops" />
      </pairDataSource>
    </datasources>
    <async-jdbc conn_name="tiber06_ops" dbname="tiber0" userid="po7opsdb"
      passwd="po7opsdb" ip="192.168.3.38" port="8629" />

    <!-- for Studio DO Factory target DB -->
    <studio-jdbc conn_name="tiber06_ops" username="po7opsdb" passwd="po7opsdb"
      driver="com.tmax.tiber0.jdbc.TbDriver"
      pool_size="" url="jdbc:tiber0:thin:@192.168.3.38:8629:tiber0" />
  </connection-info>
  :
  :

```

6. `${PROOJECT_HOME}/system/config`에 있는 `PoOpsSvr.xml` 파일을 `domain.xml`에 있는 `<datasource>` 부분과 동일하게 설정한다. DB Setting 이외 나머지 모두 Ops 서버, 계정명, 경로로 설정한다.

```

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

<!--#####
# DB Setting #
#####-->

  <configField id="DB_TYPE" value="TIBERO" type="String" xmlns="" />
  <configField id="DB_USER_ID" value="po7opsdb" type="String" xmlns="" />
  <configField id="DB_PASSWD" value="po7opsdb" type="String" xmlns="" />
  <configField id="DATA_SOURCE" value="tiber06_ops" type="String" xmlns="" />
  <configField id="DB_PLUG_NAME"
    value="com.tmax.proobject.master.util.DataSourcePlug"
    type="String" xmlns="" />
<!--#####
# Deploy Server #
#####-->

  <configField id="DEPLOY_BASE_HOME"
    value="/home/po7ops/proobject7/resources/classes/_deploy"
    type="String" xmlns="" />
  <!-- if DEPLOY_BASE_HOME does not exist, default directory is
  ${PROOJECT_HOME}/resources/classes/_deploy .-->

<!--#####
# Config Directory Setting #
#####-->

  <configField id="PO_CONFIG_PATH" value="/home/po7ops/proobject7/system/config"
    type="String" xmlns="" />
  <configField id="CLASSPATH_HOME"
    value="/home/po7ops/proobject7/system/devops/lib/proobject-opsutil-7.0.0.0.jar"
    type="String" xmlns="" />

```

```
</serverConfig>
```

7. \${PROOJECT_HOME}/system/config에 있는 proManagerOps.properties 파일을 domain.xml에 있는 데이터소스 부분과 동일하게 설정한다.

```
DataSource=tibero6_ops
```

8. .bash_profile(.profile)에 alias 설정을 한 후 기동을 확인한다.

다음은 dasboot > devboot 순서대로 기동하도록 설정한 내용이다(프로파일\운영\.profile 참고).

```
alias dasboot='startDomainAdminServer -u jeus -p jeus'  
alias dasdown='stopServer -host 127.0.0.1:19736 -u jeus -p jeus -verbose'  
alias opsboot='startManagedServer -domain domain1 -server OpsServer -u jeus -p jeus'  
alias opsdown='stopServer -host 127.0.0.1:23000 -u jeus -p jeus'
```

\${PROOJECT_HOME}/logs/ProObject.log에 다음과 같이 출력되는지 확인한다.

```
[2019.11.27 14:06:46][INFO] ===== ProObject Booting =====  
[2019.11.27 14:06:46][INFO] version : 7.0.0.1.244  
[2019.11.27 14:06:46][INFO] Global Deploy Version : 0  
[2019.11.27 14:06:46][INFO] License : standard  
[2019.11.27 14:06:46][INFO] Installed Path : /home/po7ops/proobject7/  
[2019.11.27 14:06:46][INFO] Application Path : /home/po7ops/proobject7/application/  
[2019.11.27 14:06:46][INFO] =====  
[2019.11.27 14:06:46][INFO] Starting Booting sequence.  
[2019.11.27 14:06:46][INFO] [BootLoader][ChannelManager] Starting initialing.  
[2019.11.27 14:06:46][INFO] [ChannelManager][NodeJAVA] NodeJAVA is detected. Trying to  
initialize with web-connection "httpops"  
[2019.11.27 14:06:47][INFO] [BootLoader][ChannelManager] Initialized successfully.  
[2019.11.27 14:06:47][INFO] [BootLoader][EventManager] Starting initialing.  
[2019.11.27 14:06:47][INFO] [BootLoader][EventManager] Initialized successfully.  
[2019.11.27 14:06:47][INFO] [BootLoader][Application] Starting initialing.  
[2019.11.27 14:06:47][INFO] SYSTEM_QOS_BOOT_ENABLE : false  
[2019.11.27 14:06:47][INFO] -- [APPLICATION : proobject] initializing  
[2019.11.27 14:06:48][INFO] -- Initializing ServiceGroup [proobject - system]  
[2019.11.27 14:06:48][INFO] -- Initializing ServiceGroup [proobject - monitoring]  
[2019.11.27 14:06:49][INFO] -- Initializing ServiceGroup [proobject - proobject-master]  
[2019.11.27 14:06:49][INFO] -- Initializing ServiceGroup [proobject - proobject-managerops]  
[2019.11.27 14:06:49][WARNING] [NodeAddressManager] Config File(remote_servicegroup.xml)  
is missing, remote address manager is disabled.  
[2019.11.27 14:06:49][INFO] [BootLoader][Application] Initialized successfully.  
[2019.11.27 14:06:49][INFO] -- ProObject Port 7777 is bound ...  
[2019.11.27 14:06:49][INFO] -- File Port 5445 is bound ...  
[2019.11.27 14:06:49][INFO] Properties Validation :  
/home/po7ops/proobject7/config/system.properties  
[2019.11.27 14:06:49][INFO] Properties File Last Modified Date : 2019/11/27/14:03:13  
[2019.11.27 14:06:49][INFO]  
| Property | Value | Default Value |  
Deprecated | Changed |  
| SYSTEM_APPLICATION | | | false
```

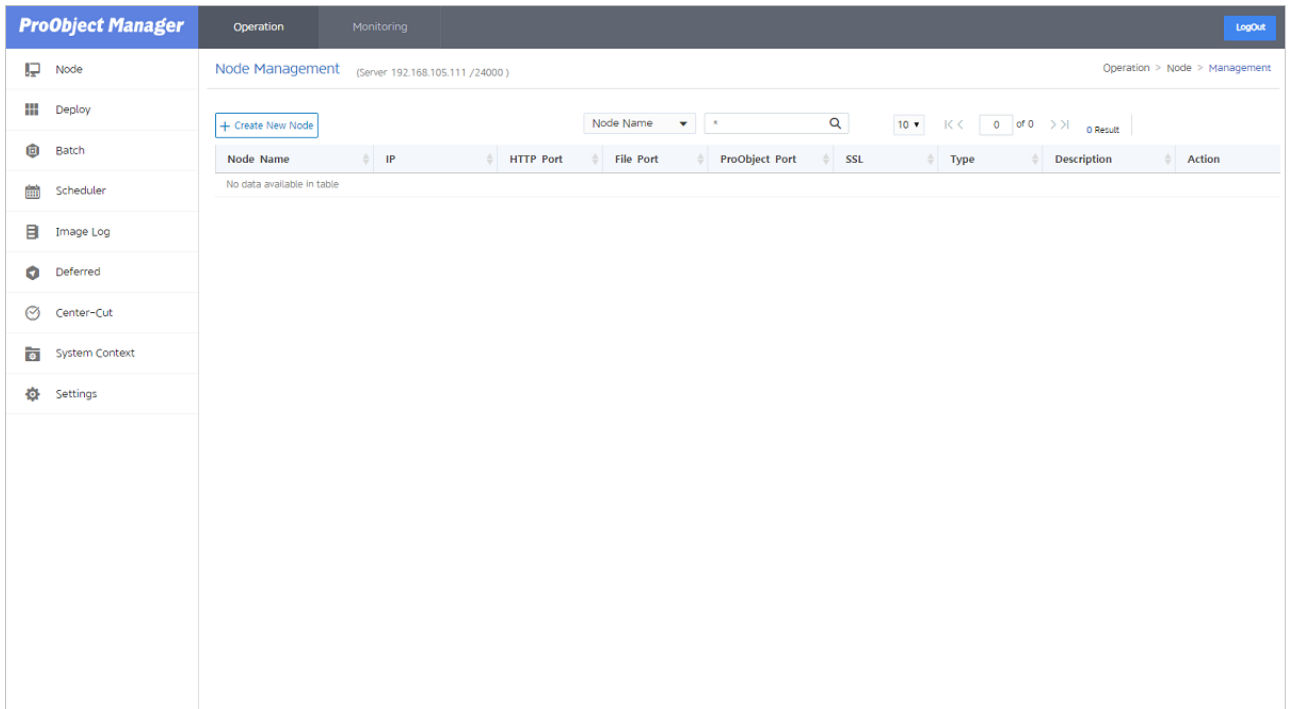
-	SYSTEM_CHARSET	UTF-8	UTF-8	false
-	SYSTEM_GLOBAL_DEPLOY_VERSION	0	0	false
-	SYSTEM_SERVICENAME_CASE	CAMEL	CAMEL	false
-	SYSTEM_TIMEOUT	60000	60000	false
-	SYSTEM_WEB_PARAMETER_CHARSET	ISO-8859-1	ISO-8859-1	false
-	SYSTEM_WEB_READ_BUFFER_SIZE	50000	16384	false
-	SYSTEM_WEB_WRITE_BUFFER_SIZE	50000	16384	false
-	SYSTEM_EVENTMANAGER_LOG_FILEHANDLER_REMOVAL_PERIOD	60	1	false
-	SYSTEM_CHANNELEVENTMANAGER_LOG_FILEHANDLER_REMOVAL_PERIOD	60	1	false
-	SYSTEM_PROOBJECT_LOG_FILEHANDLER_REMOVAL_PERIOD	60	1	false
-	SYSTEM_LOG_LEVEL	INFO	INFO	false

```
[2019.11.27 14:06:49] FAIL MESSAGE
[2019.11.27 14:06:49] SYSTEM_GLOBAL_BUFFER_SIZE : Wrong Property
[2019.11.27 14:06:49] SYSTEM_HOTDEPLOY_VERSION_CONSISTENCY_POLICY : Wrong Property
[2019.11.27 14:06:49] SYSTEM_DATAOBJECT_QUERY_LOGGING_LEVEL : Wrong Property
[2019.11.27 14:06:49] SYSTEM_APPLICATION : Split error on property SYSTEM_APPLICATION
[2019.11.27 14:06:49]
```

```
[2019.11.27 14:06:49] [INFO] [ChannelManager] Activating Configured Channels.
[2019.11.27 14:06:49] [INFO] [ChannelManager] Configured channels are activated.
[2019.11.27 14:06:49] [INFO] Booting is completed. ProObject is ready.
[2019.11.27 14:06:49] [INFO] [GUID-ServerReady]<<proobject.system.FileSequencerUpdateService>>
is ready for execution : Method - service, WaitObject - null
[2019.11.27 14:06:49] [INFO] [GUID-ServerReady]<<proobject.system.FileSequencerUpdateService>>
done for executing : Method - service
[2019.11.27 14:06:49] [INFO] [GUID-ServerReady]<<proobject.system.FileSequencerUpdateService>>
is done!
```

9. 다음 경로에 접속하여 정상적으로 실행되었는지 확인한다.

<http://192.168.105.111:24000/promanagerops>



ProManagerOps 메인화면



팝업과 함께 로그인 안된다면 dbio_config.xml, setting.js, proManagerOps.properties, PoOpsSvr.xml을 확인해서 ProObject의 DB 설정을 확인한다.

domain.xml과 proobject.xml 파일 설정 비교

다음은 각 설정된 domain.xml과 proobject.xml 파일의 예이다. 각 설정된 항목을 비교할 수 있다.

- <domain.xml>

```
<servers>
  <server>
    <name>adminServer</name>
    <listeners>
      <base>base</base>
      <listener>
        <name>base</name>
        <listen-address>0.0.0.0</listen-address>
        <listen-port>19736</listen-port>
      </listener>
      :
      :
    <!-- OpsServer -->
    <server>
      <name>OpsServer</name>
      <listeners>
        <base>base</base>
        <listener>
          <name>base</name>
          <listen-address>0.0.0.0</listen-address>
          <listen-port>23000</listen-port>
        </listener>
```

```

    <listener>
      <name>http-server</name>
      <listen-address>0.0.0.0</listen-address>
      <listen-port>24000</listen-port>
      <keep-alive-timeout>600000</keep-alive-timeout>
    </listener>
  </listeners>
  <jvm-config>
    <jvm-option>-Xmx256m -XX:MaxPermSize=128m</jvm-option>
    <jvm-option>-DPROOJECT_HOME=/home/po7ops/proobject7</jvm-option>
  </jvm-config>
  :
  :
  <web-connections>
    <http-listener>
      <name>httpops</name>
      <server-listener-ref>http-server</server-listener-ref>
      <thread-pool>
        <min>10</min>
        <max>20</max>
      </thread-pool>
    </http-listener>
  </web-connections>
  :
  :
  <data-sources>
    <data-source>tibero6_ops</data-source>
  </data-sources>
  :
  :
  <data-source>
    <database>
      <data-source-id>tibero6_ops</data-source-id>
      <export-name>tibero6_ops</export-name>
      <data-source-class-name>com.tmax.tibero.jdbc.ext.TbConnectionPoolDataSource</data-
source-class-name>
      <data-source-type>ConnectionPoolDataSource</data-source-type>
      <vendor>tibero</vendor>
      <server-name>192.168.3.38</server-name>
      <port-number>8629</port-number>
      <database-name>tibero</database-name>
      <user>po7opsdb</user>
      <password>po7opsdb</password>
    </database>
  </data-sources>
  :
  :

```

- <proobject.xml>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE xml>
<ProObjectConfig>
  <single-application>false</single-application>
  <file-port>5445</file-port>
  <proobject-port>7777</proobject-port>
  <server-name>OpsServer</server-name>

```

```
<engine-config>
  <listener-name>httpops</listener-name>
  <host-name>ns.test.local</host-name>
  <!--container-name>ProObject7</container-name-->
  <proObject-datasource>tibero6_ops</proObject-datasource>
  <initial-context-factory>jeus.jndi.JEUSContextFactory</initial-context-factory>
  <provider-url>192.168.105.111:23000</provider-url>
</engine-config>
```

3.4. 설치확인

개발 서버(DevOps)의 설치 확인은 ProManager와 ProStudio를 사용해서 확인한다. ProManager에서 설정한 노드와 ProStudio의 통신 가능여부를 확인하고 프로젝트 생성, 서비스 등록, 배포 정상동작 여부를 확인하는 과정을 통해서 진행한다.



개발 서버(DevOps)의 설치 확인을 하기 전에 ProStudio가 설치되어 있어야 한다. ProStudio의 설치과정은 [ProStudio 설치 및 제거](#)를 참고한다.

개발 서버(DevOps)의 정상적인 설치 여부는 다음의 과정으로 확인한다.

1. 노드 설정

ProManager를 실행해서 노드와 서비스 그룹을 생성하고 메타 정보를 설정한다.

2. 프로젝트 생성

ProStudio를 실행해서 프로젝트를 생성한다.

3. DO, SO 생성

ProStuido에서 테스트할 DO, SO 리소스를 생성하고 커밋한다.

4. HotDeploy 및 Service Test

ProManager에서 리소스를 HotDeploy하고 Service Test를 진행한다.

5. 런타임 서버에 리소스 배포

ProManagerOps를 실행해서 Runtime 서버에 리소스를 배포한다.

3.4.1. 노드 설정

ProManager를 실행해서 노드와 서비스 그룹을 생성하고 메타 정보를 설정한다.

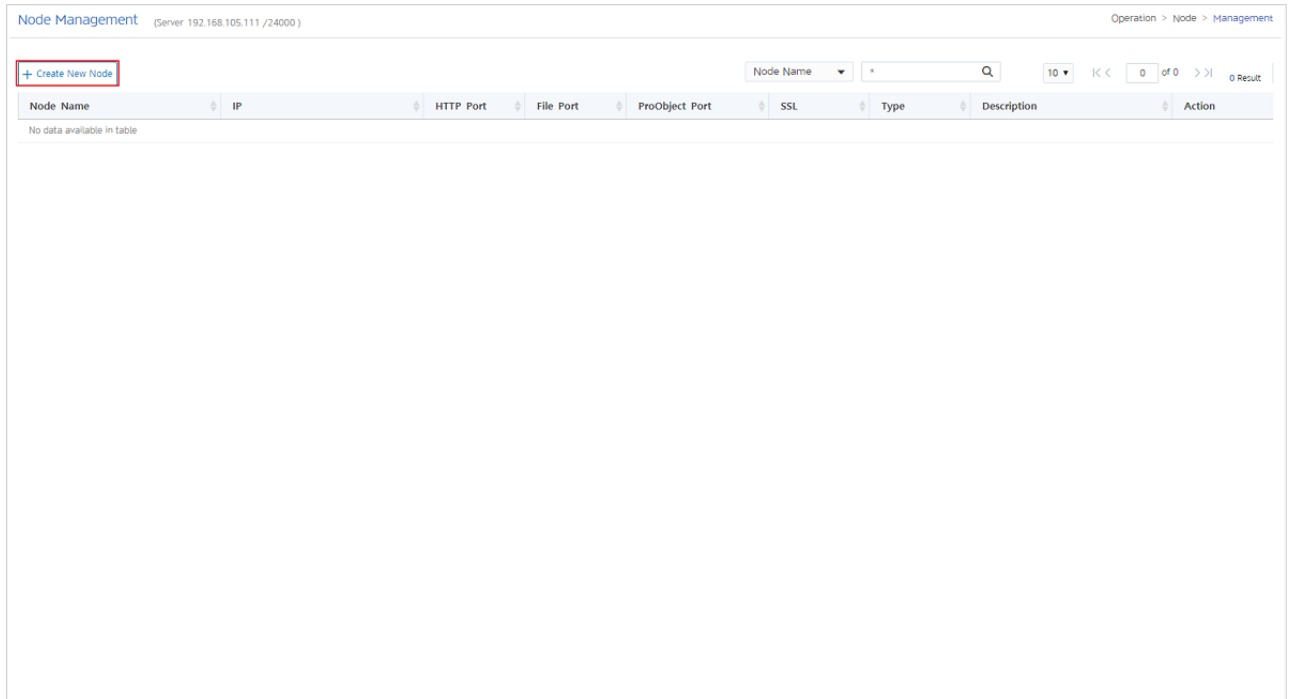
1. **ProManager**의 **네비게이션 영역**에서 **[Node] > [Node Management]**를 선택해서 사용자가 구성한 환경을 기준으로 노드 설정을 추가한다.

Node Name	Node ID	IP	File Port	Http Port	ProObject Port	SSL	Admin	Node Type	Action
Test	ab53160f6bbdf2865628d022257...	192.168.105.111	4445	14000	6777	FALSE		TEST	Action
Master	ab535a2b43de29c64d1d83607...	192.168.105.111	5445	24000	7777	FALSE		MASTER	Action
Runtime	ab53a1e0e08933d85a14320d4...	192.168.105.111	6445	34000	8777	FALSE		RUNTIME	Action

[ProManager] - Node Management 화면

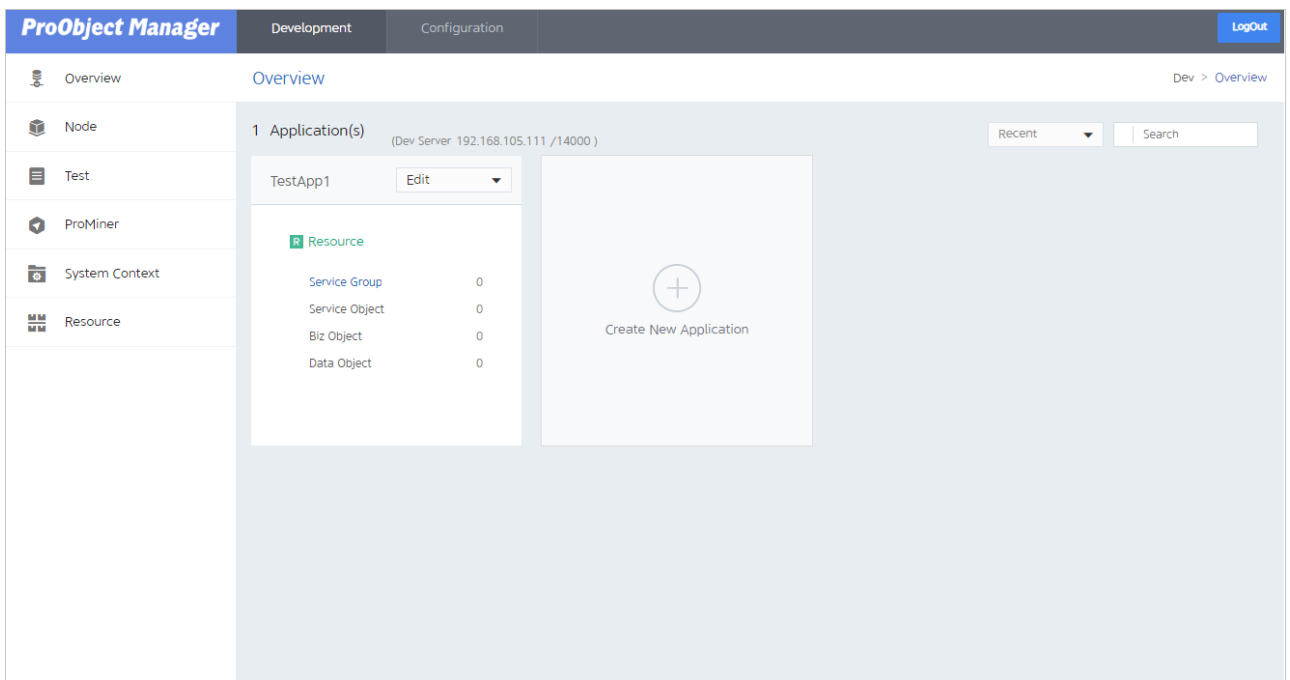
항목	설명
Node Type	<p>다음은 Node Type에 대한 설명이다.</p> <ul style="list-style-type: none"> ◦ Master : Ops 서버 환경 ◦ Runtime : Runtime 서버 환경 ◦ Test : Dev 서버 환경

2. **ProManagerOps**의 네비게이션 영역에서 **[Node] > [Node Management]**를 선택해서 사용자가 구성한 런타임 환경을 기준으로 노드 설정을 추가한다.



[ProManagerOps] - Node Management 화면

3. ProManager의 네비게이션 영역에서 [Overview] > [APP&SG]를 선택한 후 [Create New Application] 버튼을 클릭해서 애플리케이션과 서버 그룹을 생성한다.



[ProManager] - Overview 화면

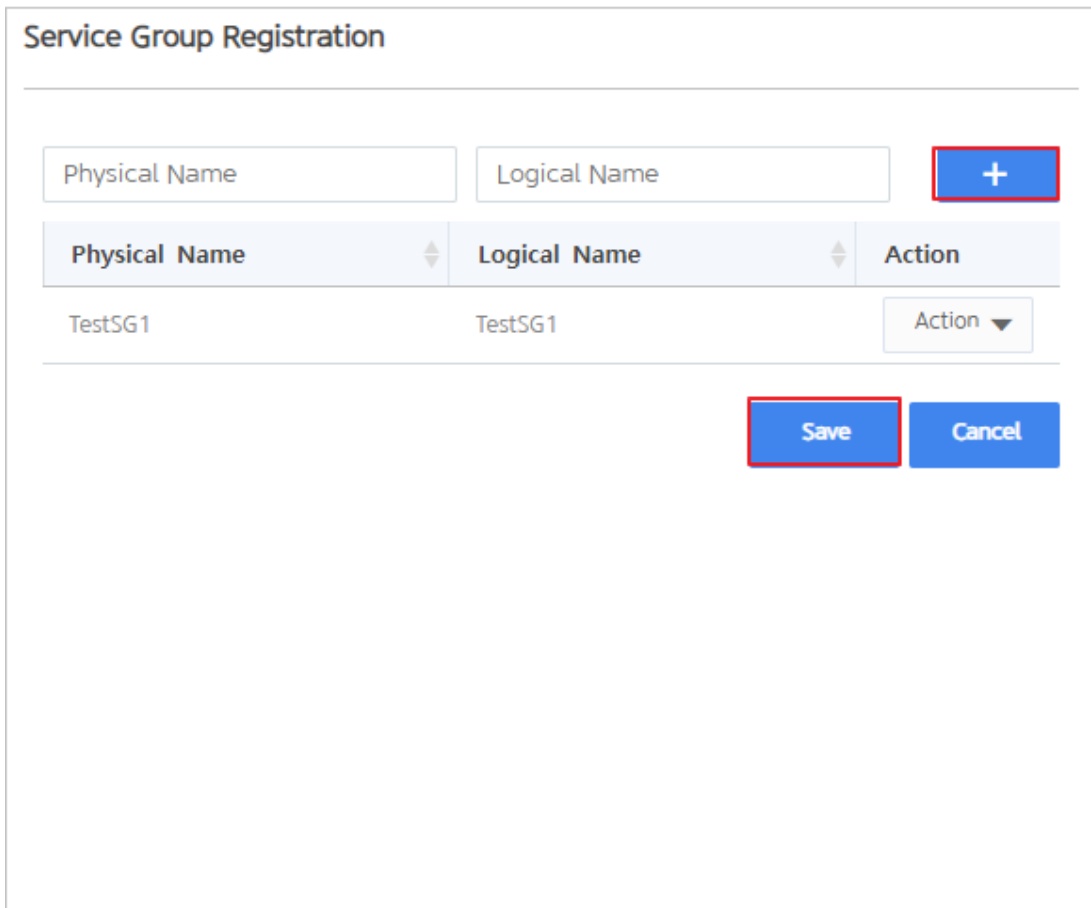
Create Application 화면에 각 항목을 입력하고 [OK] 버튼을 클릭한다.

Create Application

Physical Name	<input type="text" value="TestApp1"/>
Logical Name	<input type="text" value="TestApp1"/>
Package	<input type="text" value="com.tmax"/>
Description	<input type="text"/>

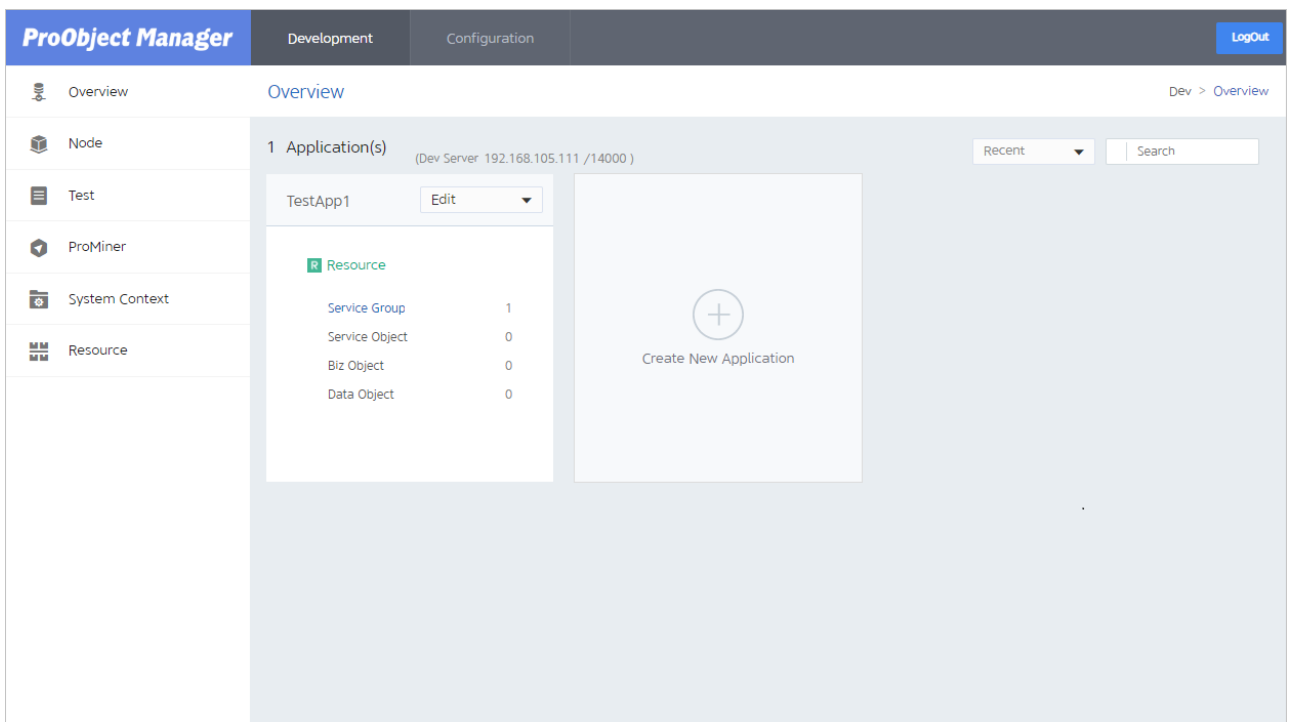
Create Application 화면

Service Group Registration 화면에 'Physical Name', 'Logical Name'에 값을 입력하고 [+] 버튼을 클릭한다. 각 항목을 입력하고 [Save] 버튼을 클릭한다.



Service Group Registration 화면

서버 그룹을 등록하면 **Overview 화면**에 서버 그룹의 Count가 증가한다.



[ProManager] - Overview 화면 - 서버 그룹 추가

4. ProManager의 네비게이션 영역에서 [Overview] > [Meta] 메뉴를 선택하면 Meta Dictionary 화면에서 등록된 메타정보를 조회할 수 있다.

Physical Name * Logical Name * Resource Group * Search 7 Results 1 of 1 10

<input type="checkbox"/>	Meta Type	Physical Name	Logical Name	Resource Group	Field Type	Length	Update Time	Comments
<input type="checkbox"/>	non-persistent	sal	sal	sal	int	7	2019-12-03 03:56:20	sal
<input type="checkbox"/>	non-persistent	mgr	mgr	mgr	int	4	2019-12-03 03:56:46	mgr
<input type="checkbox"/>	non-persistent	job	job	job	String	9	2019-12-03 03:57:08	job
<input type="checkbox"/>	non-persistent	ename	ename	ename	String	20	2019-12-03 03:55:29	ename
<input type="checkbox"/>	non-persistent	empno	empno	empno	int	4	2019-12-03 03:57:37	empno
<input type="checkbox"/>	non-persistent	deptno	deptno	deptno	int	2	2019-12-03 03:57:59	deptno
<input type="checkbox"/>	non-persistent	comm	comm	comm	int	7	2019-12-03 03:58:14	comm

Add Add Excel Change Delete

[ProManager] - Meta Dictionary 화면

Meta Dictionary 화면([\[ProManager\] - Meta Dictionary 화면](#))에서 **[Add]** 버튼을 클릭해서 아래와 같은 방법으로 몇 가지 샘플을 만든다.

Create Meta Field

Physical Name

Logical Name

Resource Group

Comments

Field Type

Length

Type non-persistent persistent

Primary Key yes no

DataSource

Table

Column

Masking Use Unuse

From To

Encrypt Use Unuse

Default Value

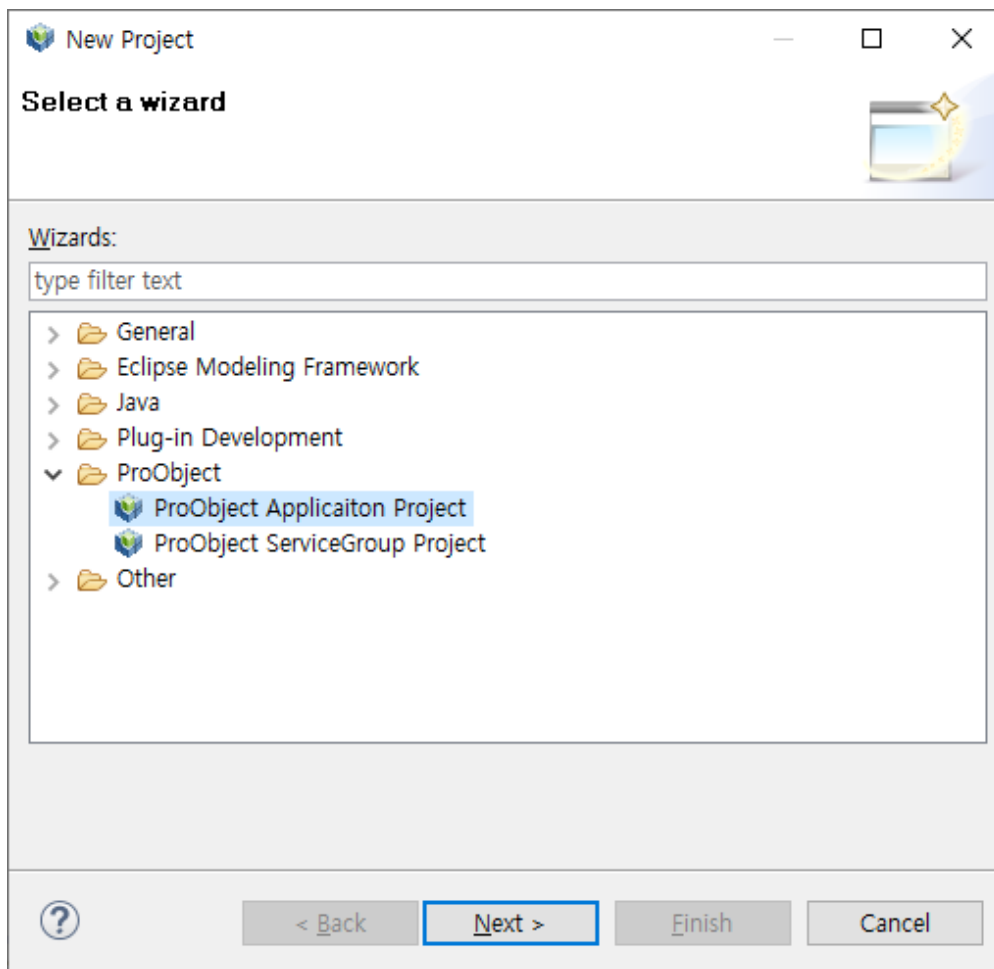
[ProManager] - Create Meta Field 화면

3.4.2. 프로젝트 생성

ProStudio를 실행한 후 다음의 과정으로 프로젝트를 생성한다.

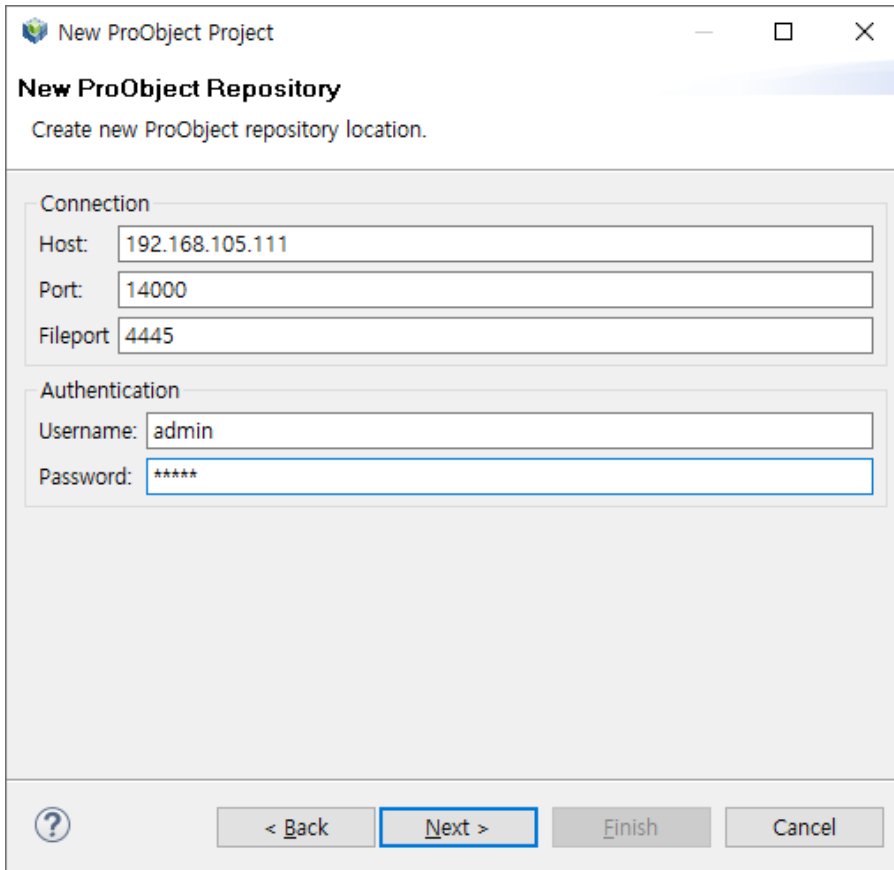
1. ProStudio의 메뉴에서 [New] > [Project]를 선택한 후 **New Project 화면에서 [ProObject] > [ProObject Application Project]**를 선택한다. Application Project 정보를 입력하고 [Next] 버튼을

클릭한다.



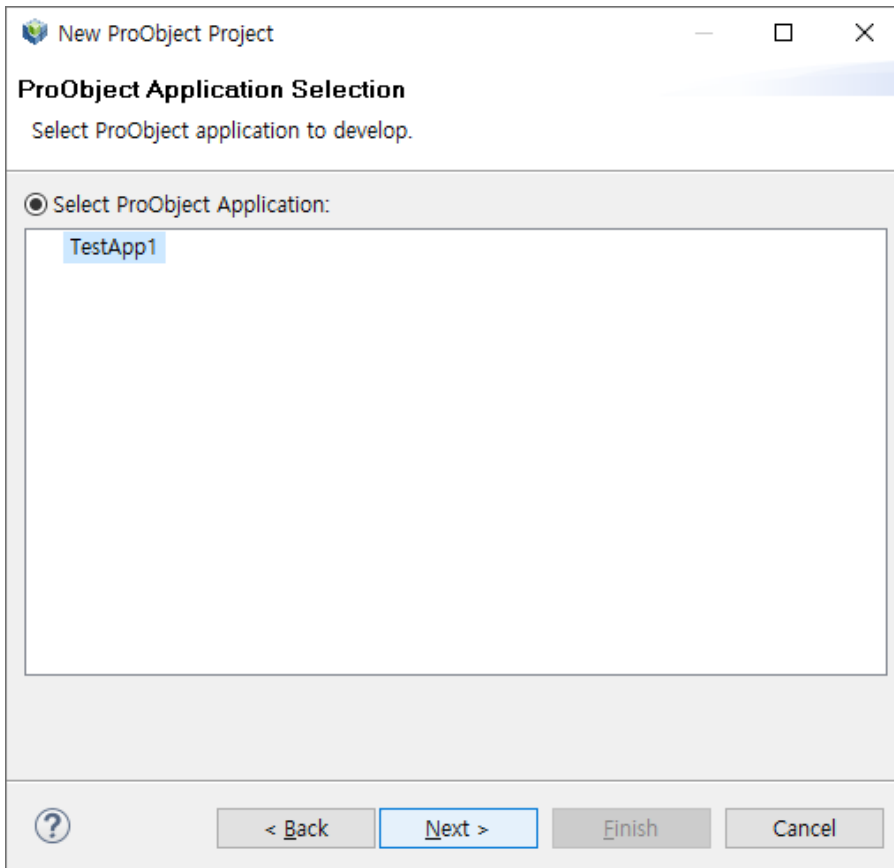
ProStudio - 프로젝트 생성 (1)

사용자가 설정한 Dev 서버(ProManager) 환경의 정보와 계정명을 입력한다.



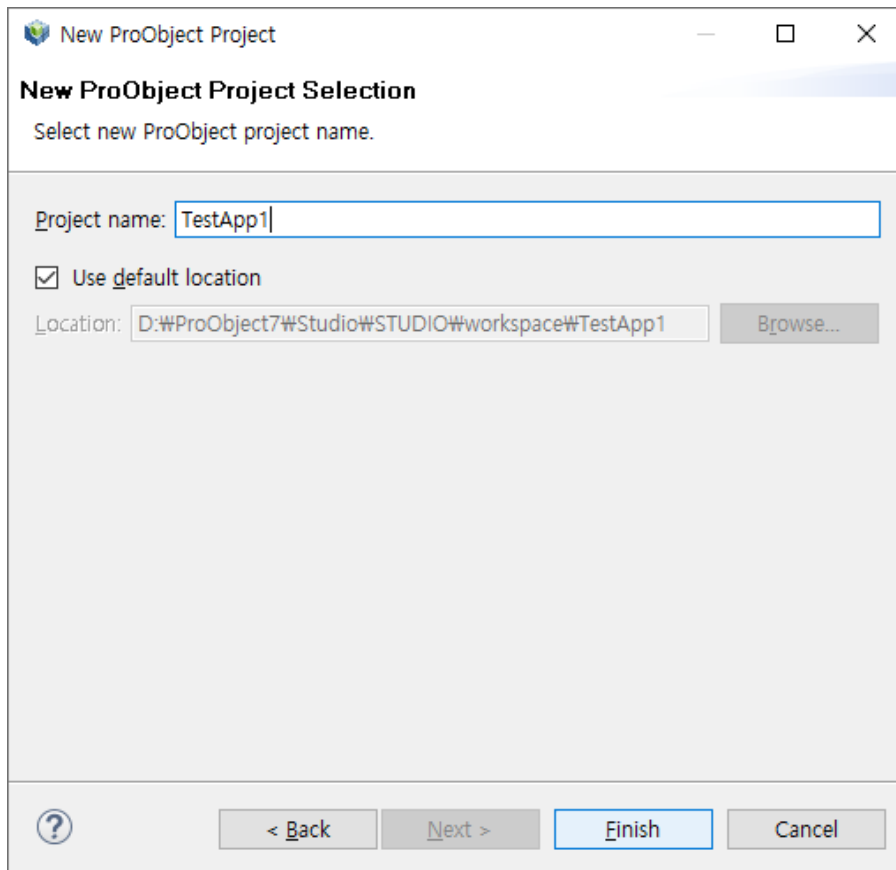
ProStudio - 프로젝트 생성 (2)

ProManager에서 추가한 애플리케이션을 선택한 후 **[Next]** 버튼을 클릭한다.



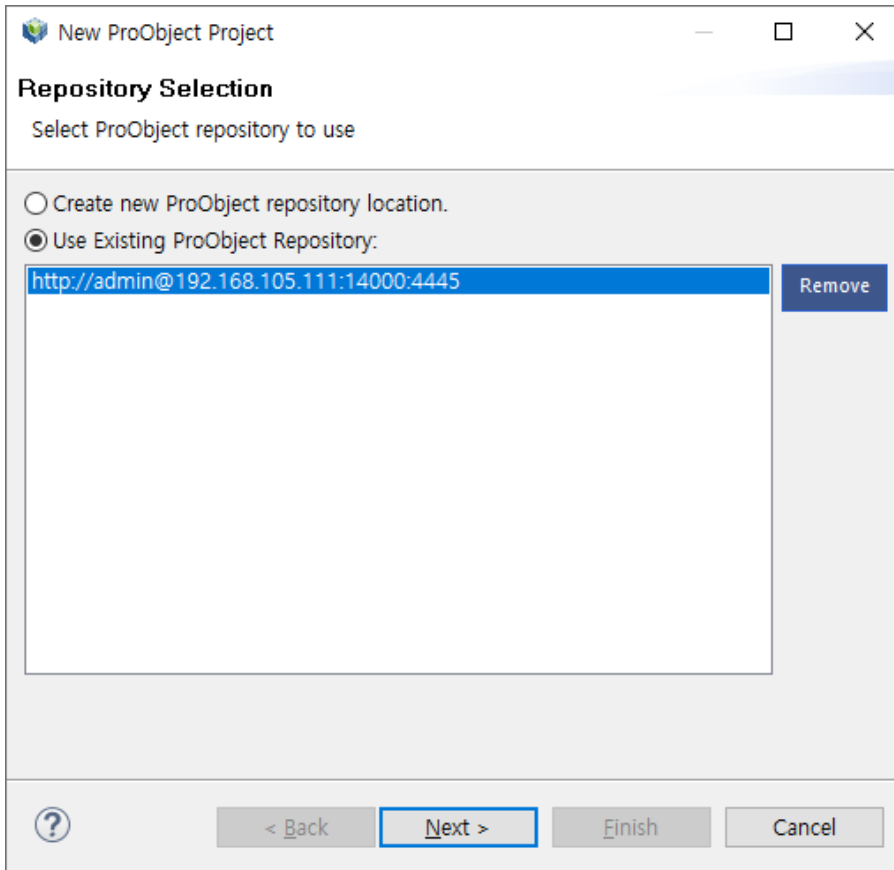
ProStudio - 프로젝트 생성 (3)

프로젝트명을 입력하고 **[Finish]** 버튼을 클릭한다.



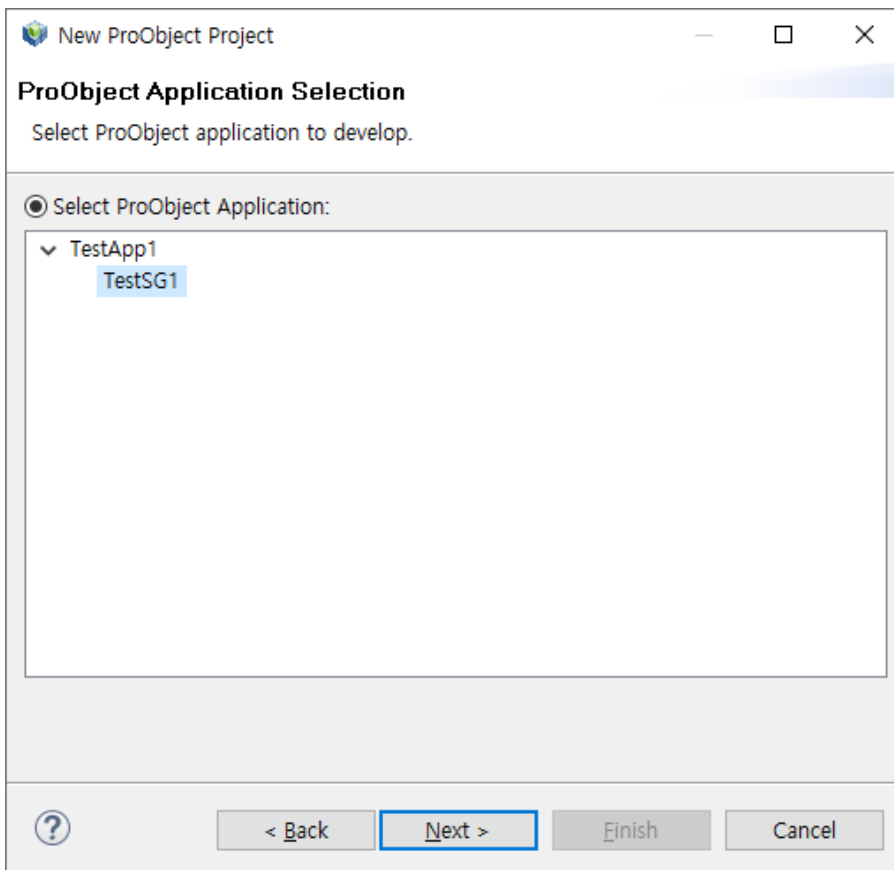
ProStudio - 프로젝트 생성 (4)

2. **ProStudio**의 메뉴에서 **[New] > [Project]** 선택한 후 **New Project 화면**에서 **[ProObject] > [ProObject Service Group Project]**를 선택한다(Application Project를 생성할 때 사용자 정보를 입력하였기 때문에 넘어가면 된다).



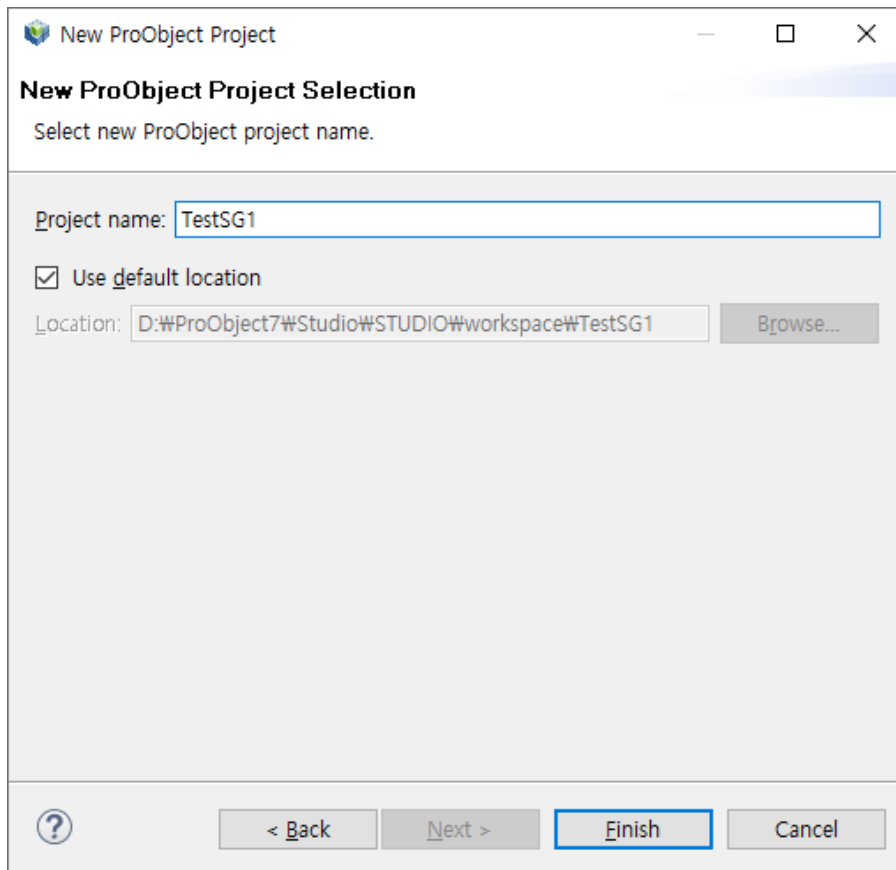
ProStudio - 서버 그룹 생성 (1)

ProManager에서 생성한 Service Group을 선택한 후 **[Next]** 버튼을 클릭한다.



ProStudio - 서버 그룹 생성 (2)

프로젝트명을 입력하고 **[Finish]** 버튼을 클릭한다.



ProStudio - 서버 그룹 생성 (3)

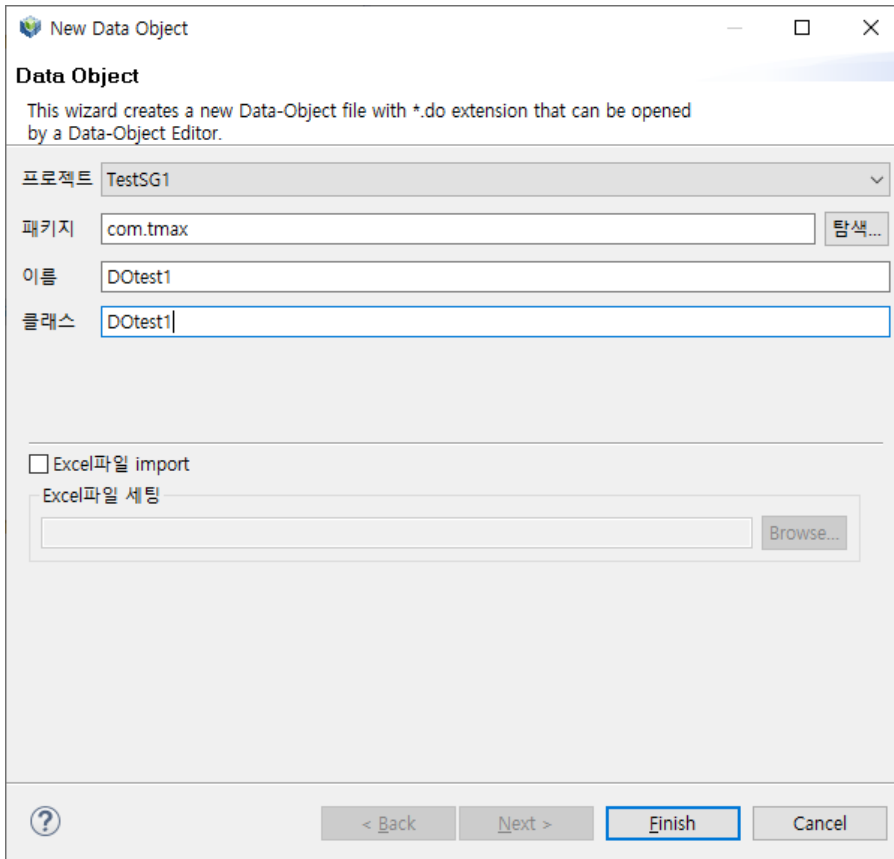
3.4.3. DO, SO 생성

ProStudio에서 테스트할 DO, SO 리소스를 생성하고 커밋한다.

DO 리소스 생성

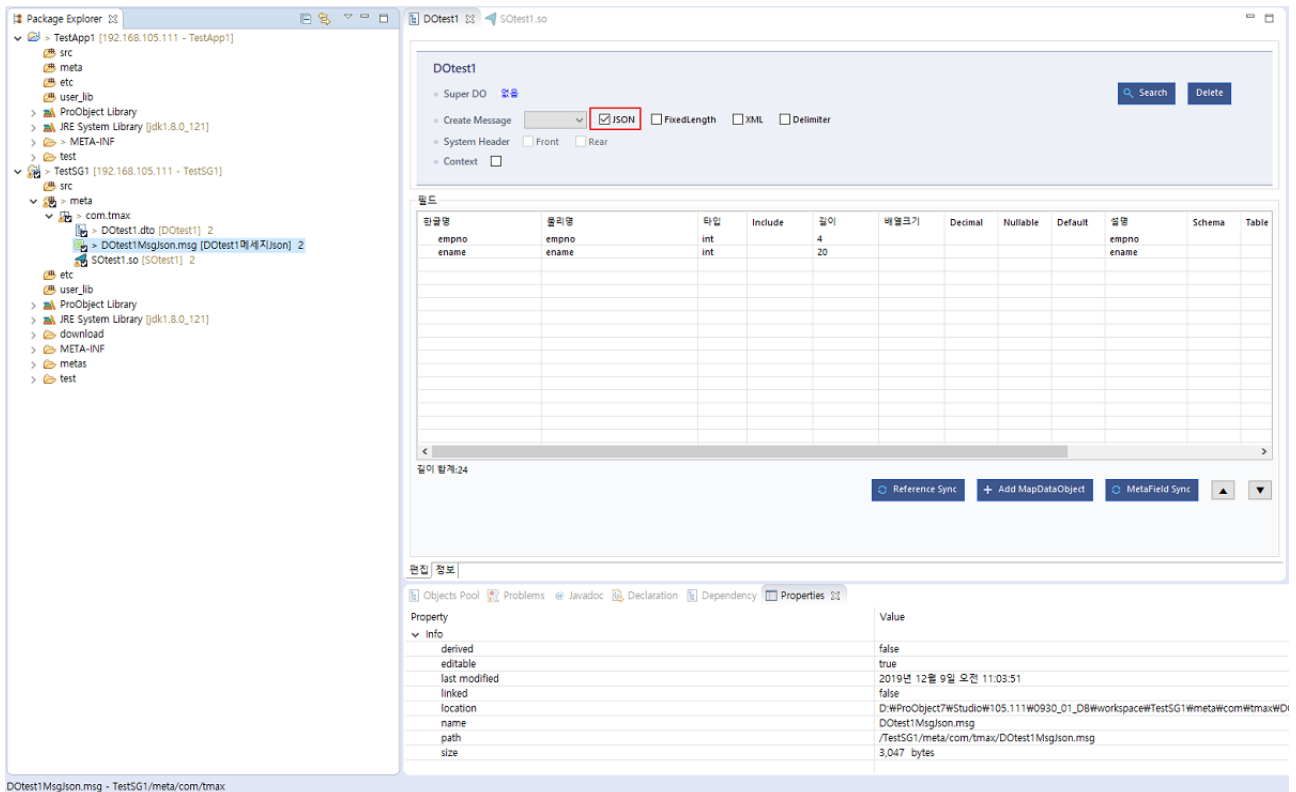
다음의 과정으로 DO 리소스를 생성한다.

1. ProStudio의 네비게이터 영역에서 'TestSG1'을 선택한 후 컨텍스트 메뉴에서 **[New] > [DataObject]**를 선택한다.
2. **New Data Object 화면**에서 사용자가 원하는 이름으로 설정한 후 **[Finish]** 버튼을 클릭한다.



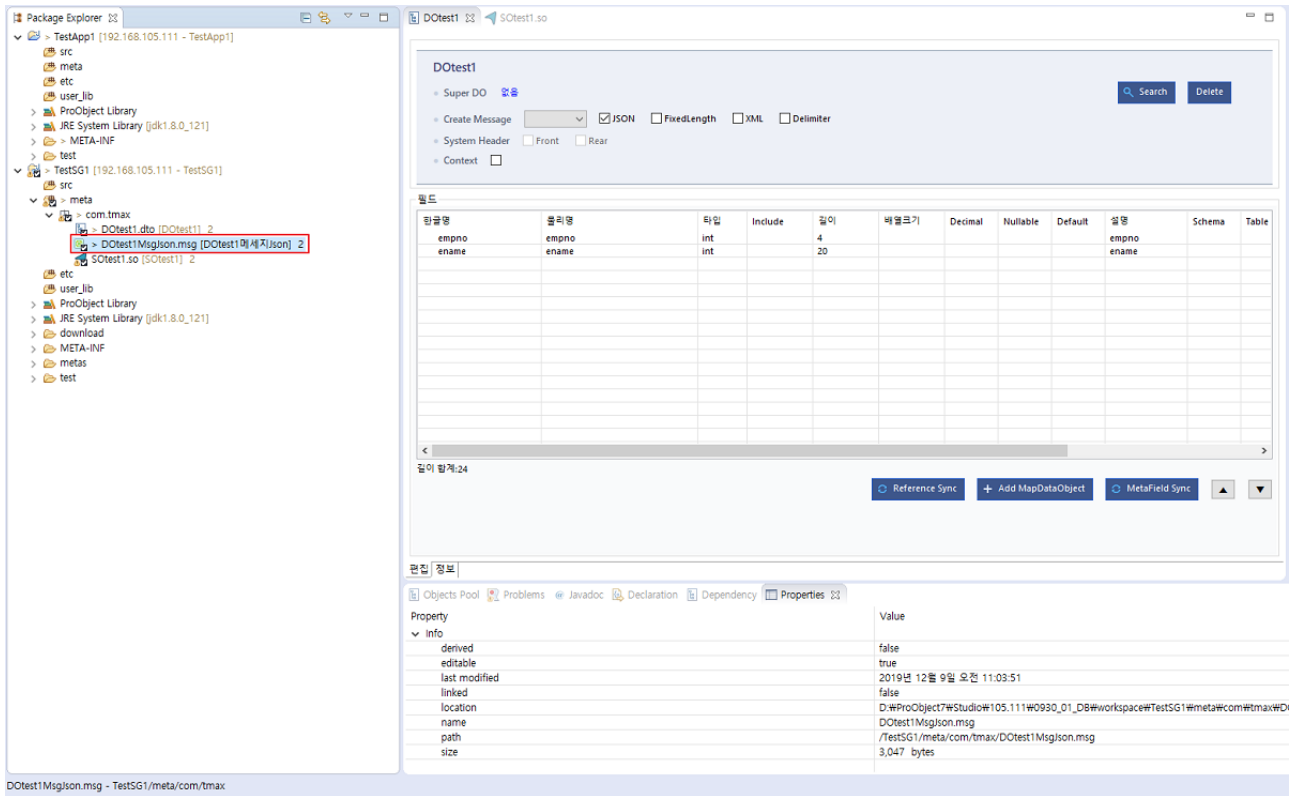
DO 리소스 생성 (1)

3. 생성한 DO를 열어 Create Message에 JSON을 체크한다.



DO 리소스 생성 (2)

4. 네비게이터의 서비스 그룹에 아래와 같이 .msg 파일이 생기는지 확인한다.

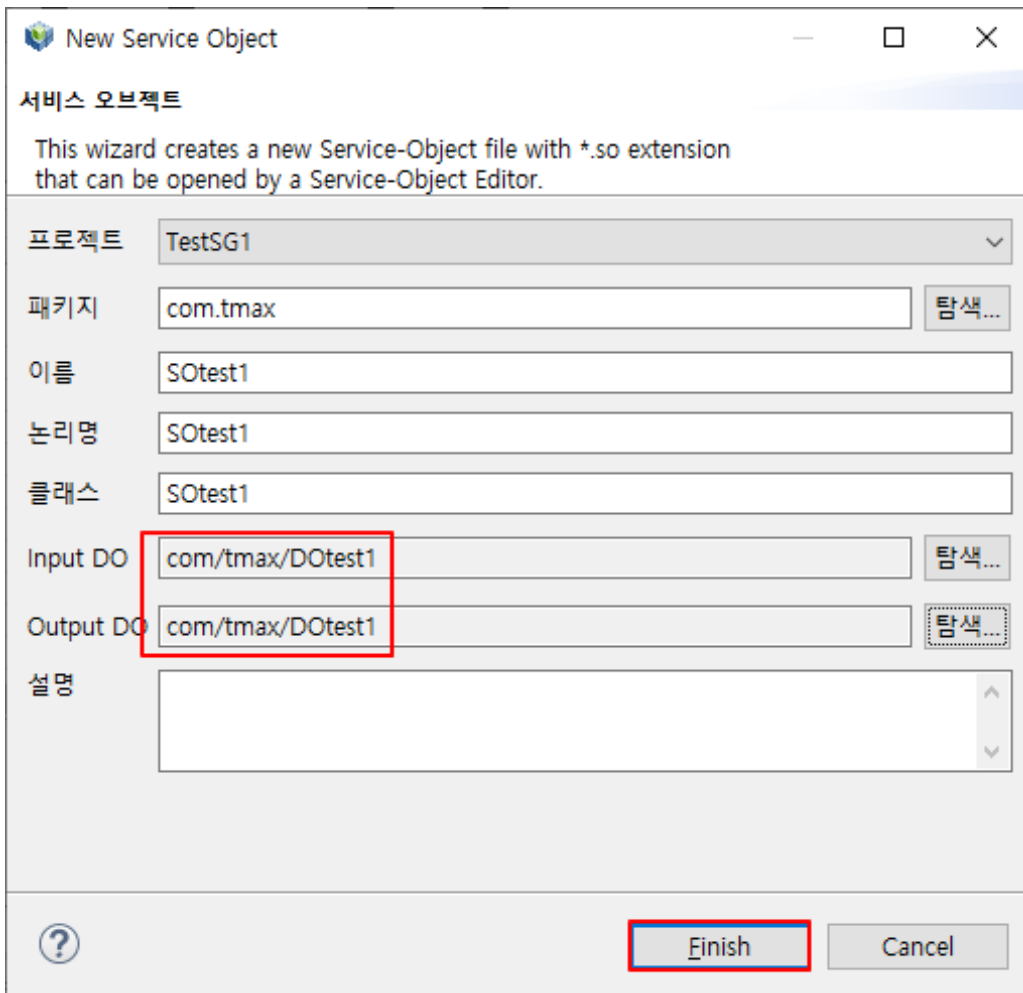


DO 리소스 생성 (3)

SO 리소스 생성

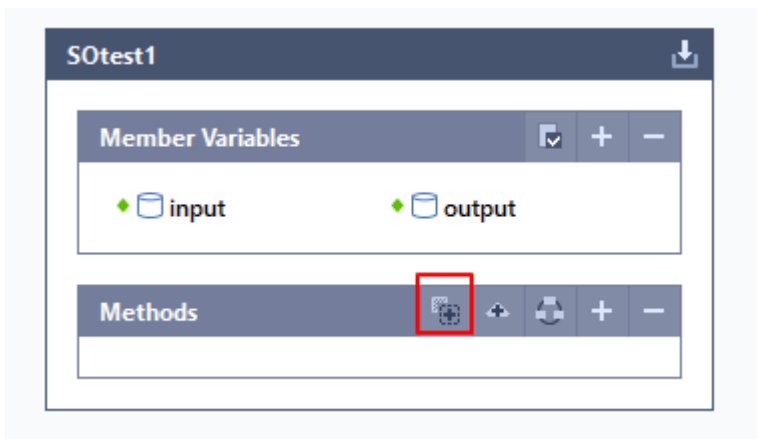
다음의 과정으로 SO 리소스를 생성한다.

1. 생성한 서비스 그룹 TestSG1를 선택한 후 컨텍스트 메뉴에서 **[New] > [Service Object]**를 선택한다.
2. **New Service Object** 화면에서 사용자가 원하는 이름을 설정하고 Input DO, Output DO에는 위에서 생성한 DO를 선택한 후 **[Finish]** 버튼을 클릭한다.



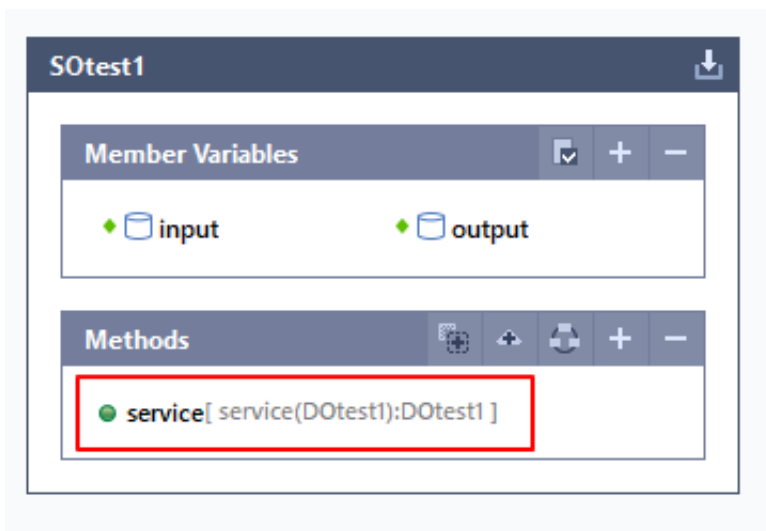
SO 리소스 생성 (1)

3. SO를 열어 비구현 메소드 추가한다.



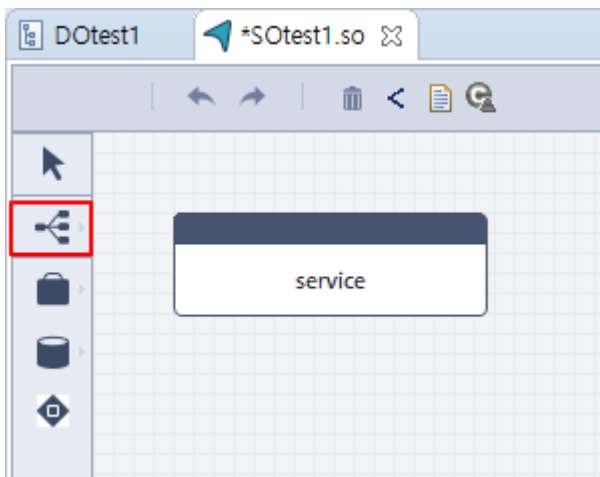
SO 리소스 생성 (2)

4. 메시지 생성 여부를 묻는 대화상자에서 [OK] 버튼을 클릭하면 메소드가 생성된다.

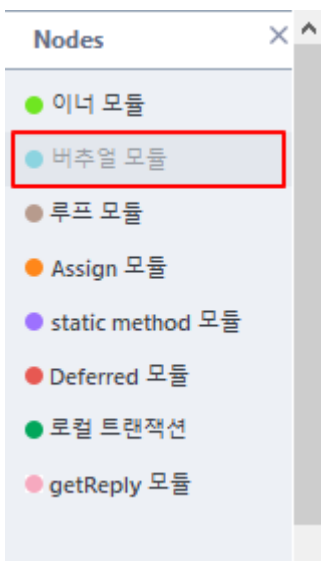


SO 리소스 생성 (3)

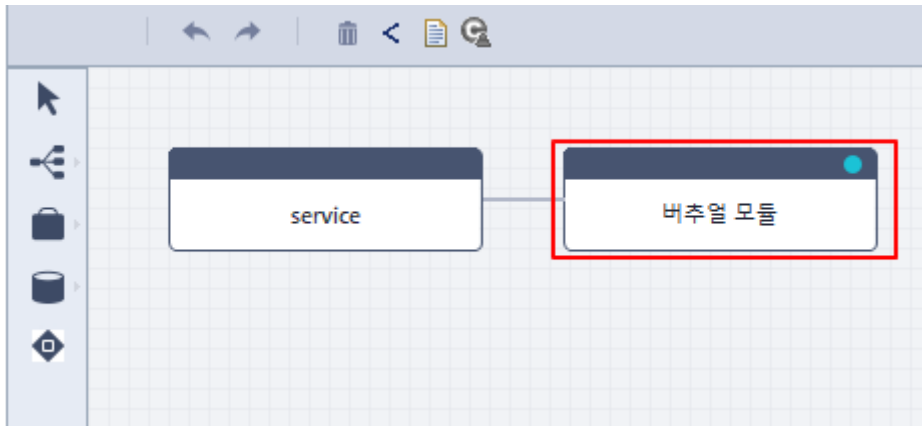
5. 생성된 메서드를 더블클릭하면 버추얼 모듈을 추가할 수 있다.



SO 리소스 생성 (4)



SO 리소스 생성 (5)



SO 리소스 생성 (6)

6. 버추얼 모듈을 더블클릭해서 나오는 SOtest1.so 파일에 다음과 같이 return output을 넣는다. 이후 저장을 하게되면 에러가 발생하지 않는다(return 값이 없으면 에러가 발생한다).

```

DCTest1 | SOtest1.so | DD 에디터

private ProObjectLogger logger = ServiceLogger.
public com.tmax.DCTest1 input = new com.tmax.DO
public com.tmax.DCTest1 output = new com.tmax.D

@Override
public com.tmax.DCTest1 service (com.tmax.DCTest
{
    //[BEGIN_NODE_BLOCK, , service (DCTest1)]
    {

        //[BEGIN_NODE_BLOCK, 0, service (DCTest1)]
        {
            //버추얼 모듈
            //[BEGIN_VIRTUAL_CODE_BLOCK, 0, service (DCTest
            {
                logger.info ("##### Test #####");
                return output;
            }
        }
    }
}

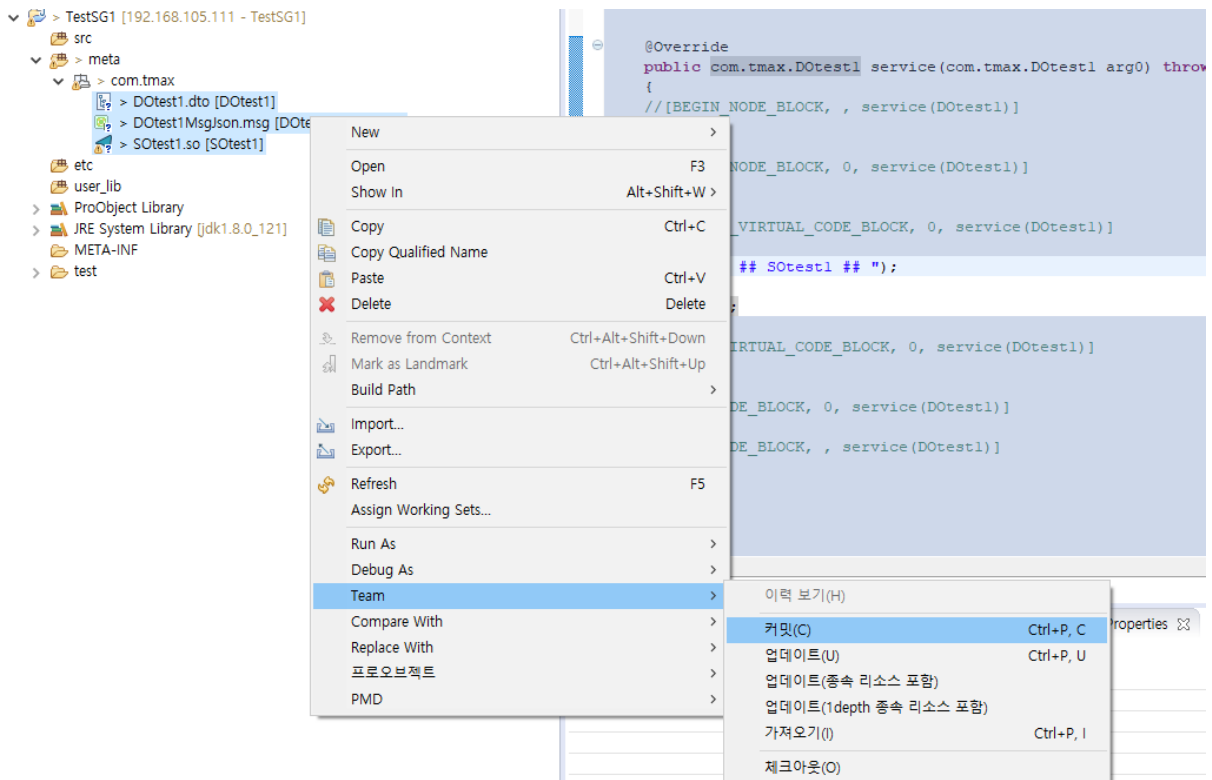
```

SO 리소스 생성 (7)

리소스 커밋

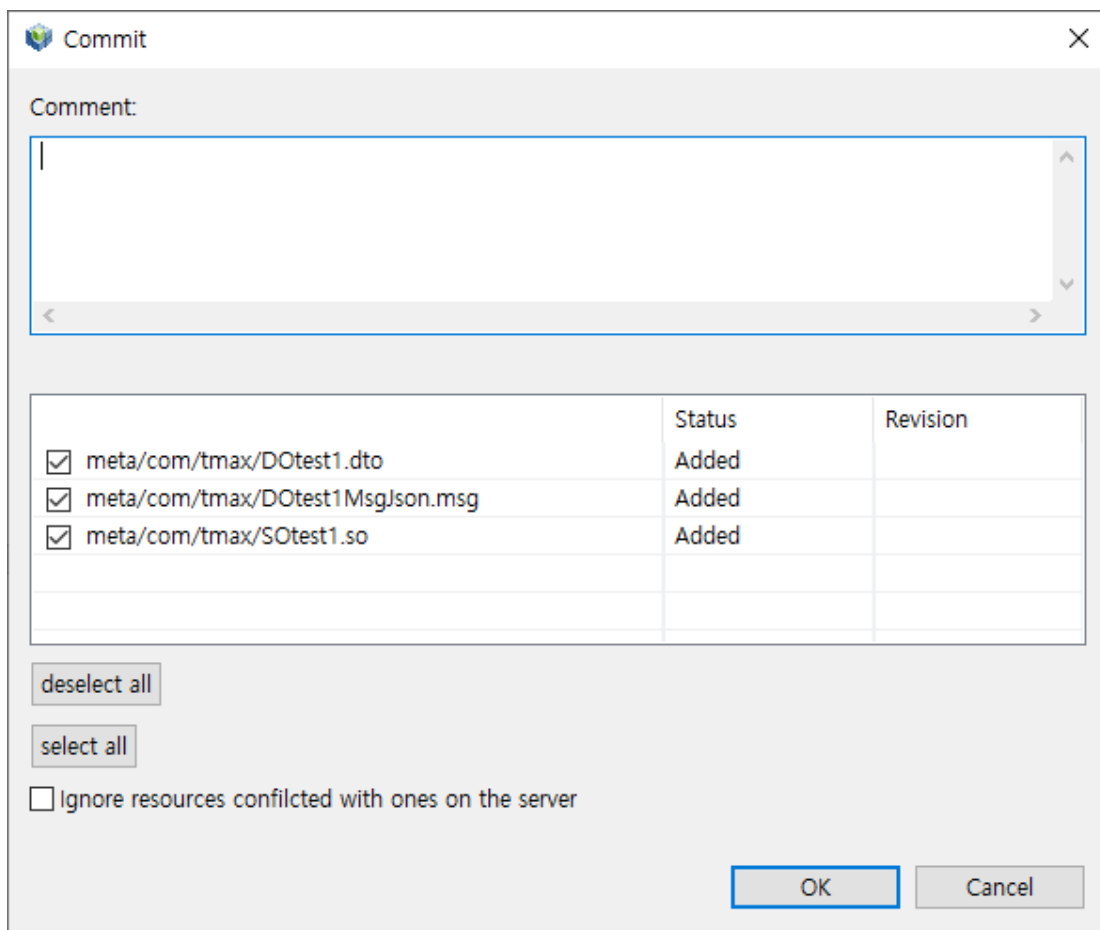
다음의 과정으로 커밋한다.

1. 생성된 DO와 SO를 선택한 후 컨텍스트 메뉴에서 [Team] > [커밋]을 선택한다.



리소스 커밋 (1)

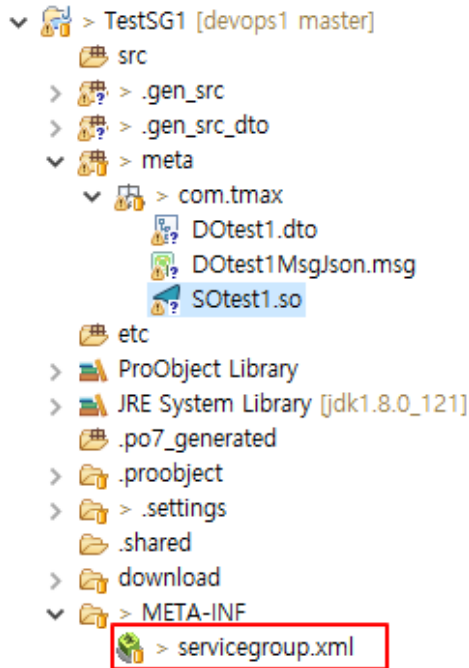
2. 커밋 정보를 확인한 후 **[OK]** 버튼을 클릭한다.



리소스 커밋 (2)

3. servicegroup.xml 파일을 생성한다. 생성한 SO 파일은 선택한 후 컨텍스트 메뉴에서 **[프로오브젝트] >**

[Generate Application DD]를 선택한다.



리소스 커밋 (3)



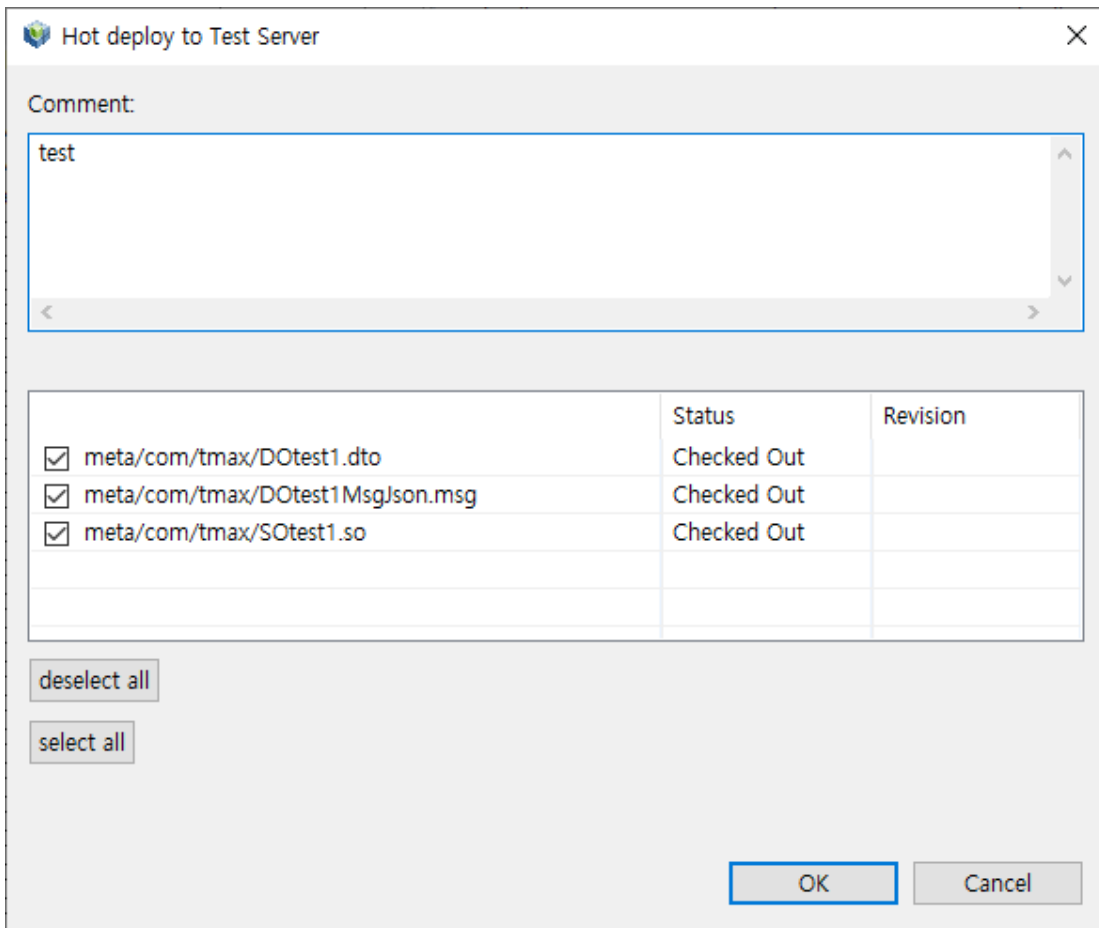
DB 버전에서는 생성한 servicegroup.xml 파일은 HotDeploy할 필요 없다. SO를 HotDeploy할 때 자동으로 반영된다.

3.4.4. HotDeploy 및 Servcie Test

다음의 과정으로 **ProManager**에서 리소스를 HotDeploy하고 Servcie Test를 진행한다.

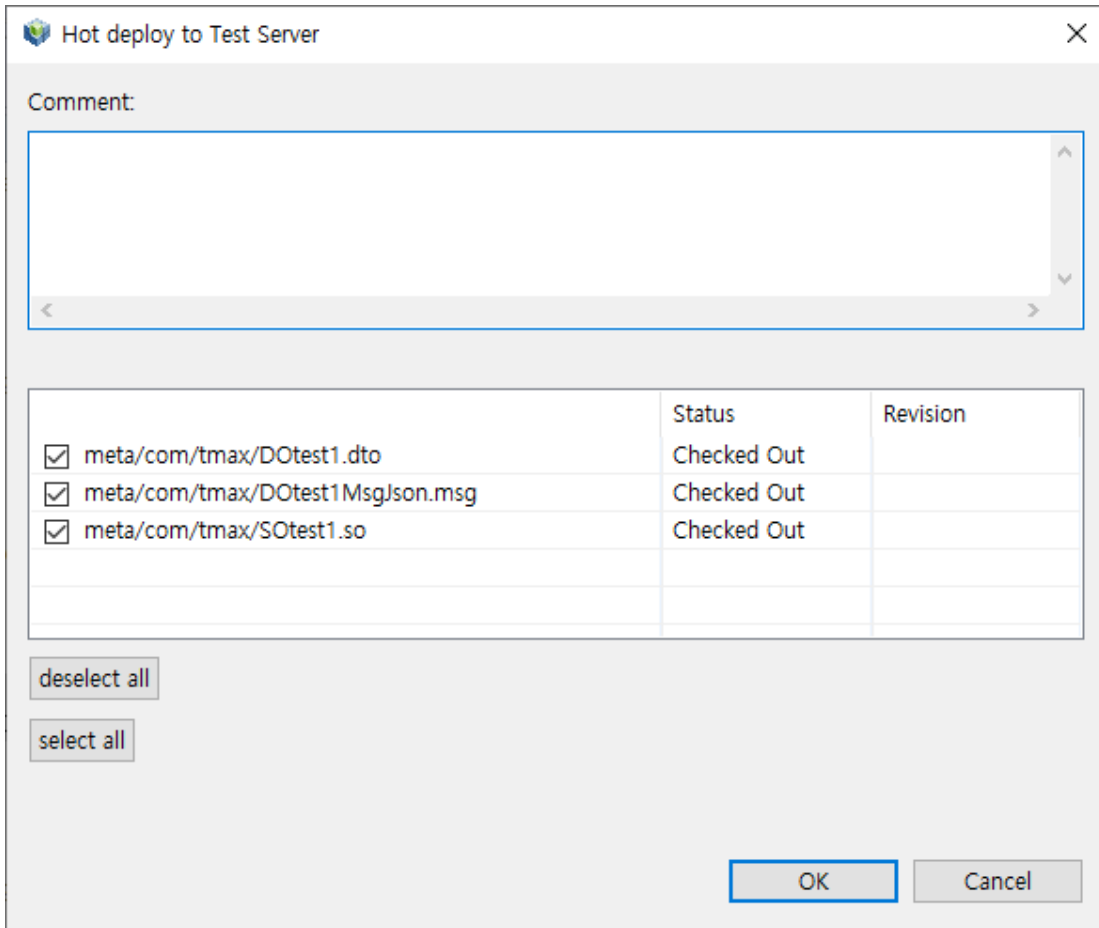
1. **ProStudio**에서 리소스들을 HotDeploy한다.

생성한 리소스를 선택한 후 컨텍스트 메뉴에서 [**프로오브젝트**] > [**Hot deploy to Test Server**]를 선택한다.
HotDeploy하는 과정에서 DEV 서버 로그에 에러가 발생하지 않는지 확인한다.
(`${PROOBJECT_HOME}/logs/ProObject.log`)



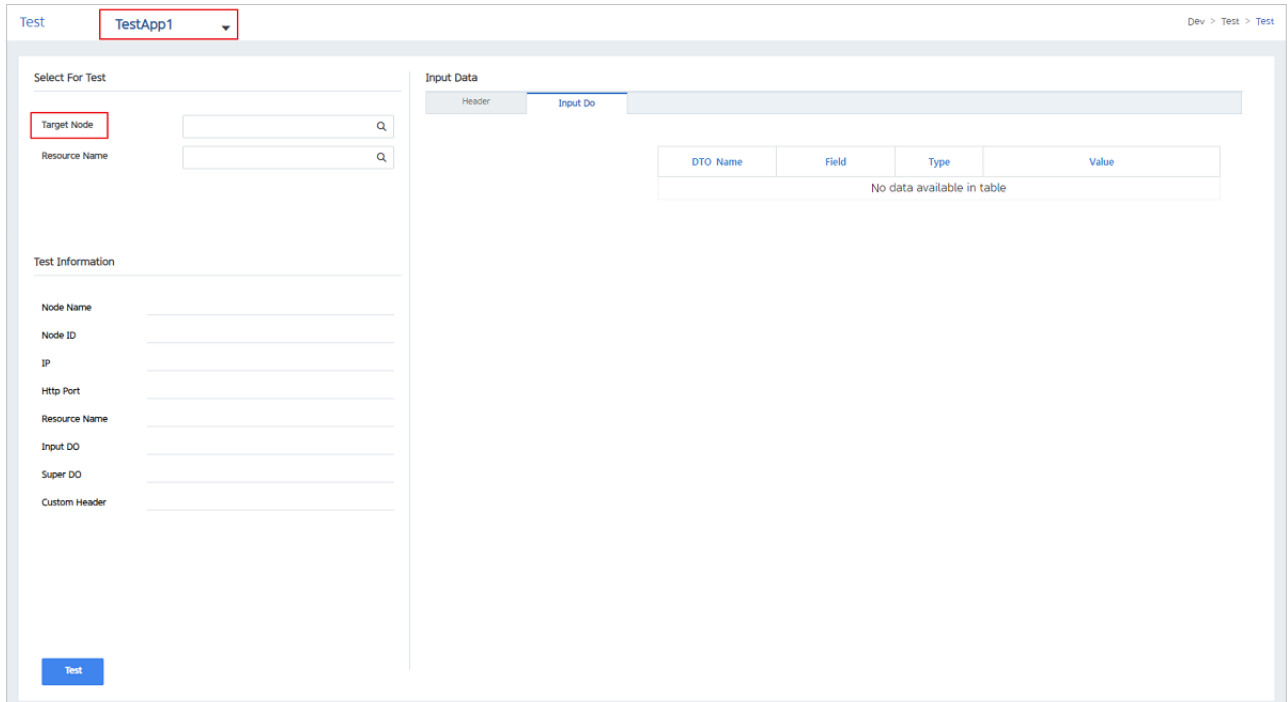
ProStudio - HotDeploy (1)

2. `${PROOJECT_HOME}/system/config`에서 설정한 `dbio_config.xml` 파일을 `${PROOJECT_HOME}/application/AppName/config`에 넣어준다.



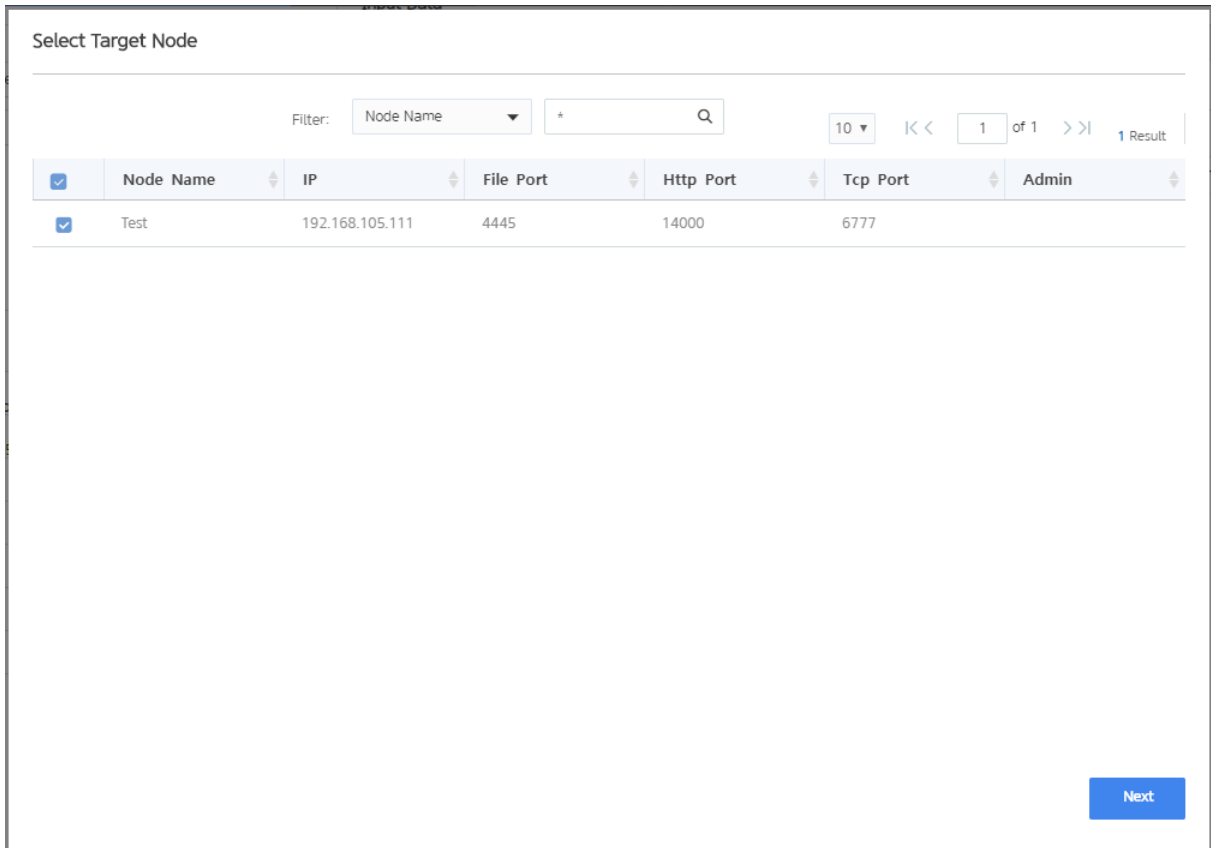
ProStudio - HotDeploy (2)

3. ProManager의 [Development] 메뉴에서 [Test]를 선택한 후 목록에서 'AppName'을 선택한다. 'Target Node' 항목에서 [검색] 버튼을 클릭한다.



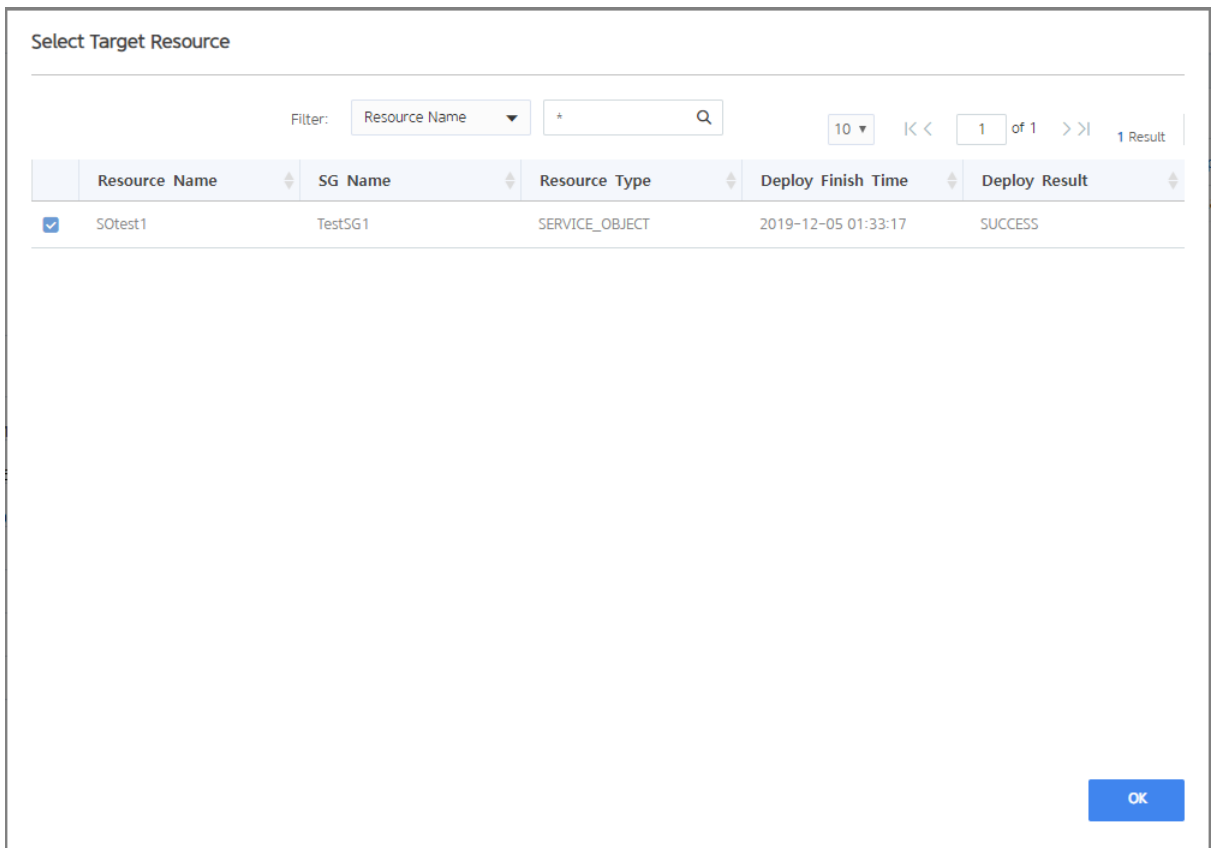
ProManager 테스트 (1)

Select Target Node 화면에서 노드를 선택한 후 [Next] 버튼을 클릭한다.



ProManager 테스트 (2)

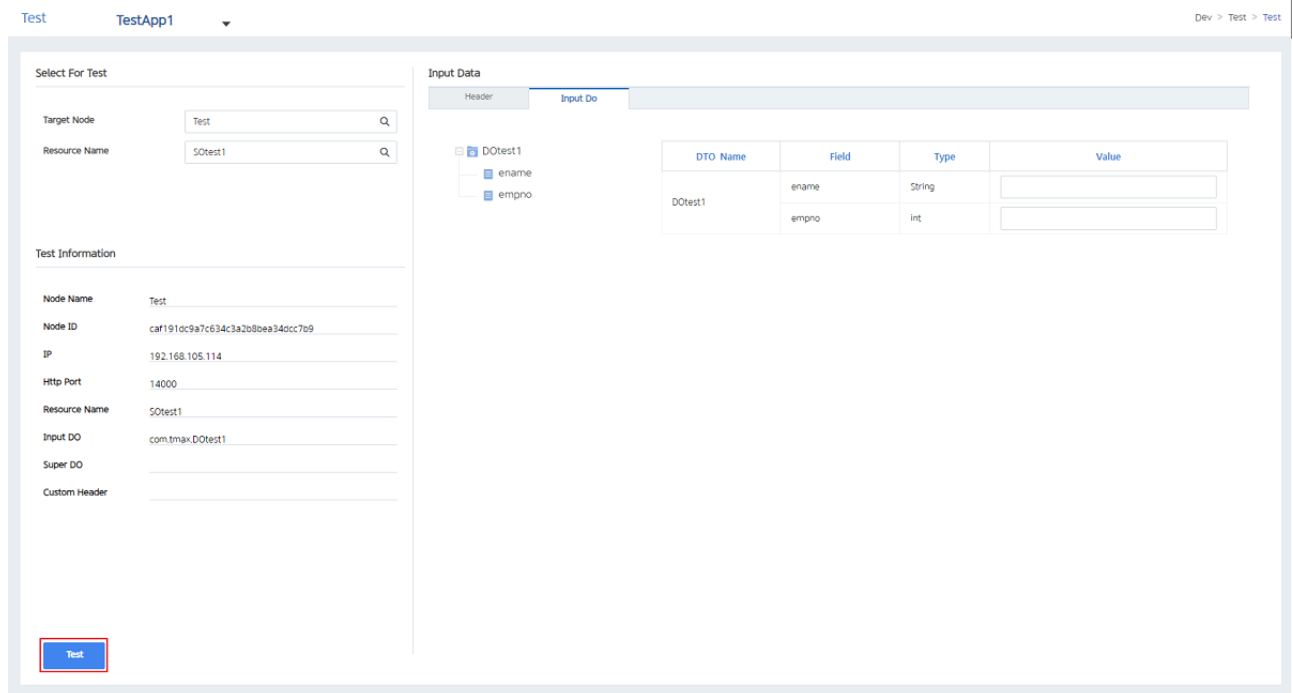
- 'Resource Name' 항목에서 생성한 SO를 체크하여 [OK] 버튼을 클릭한다.



ProManager 테스트 (3)

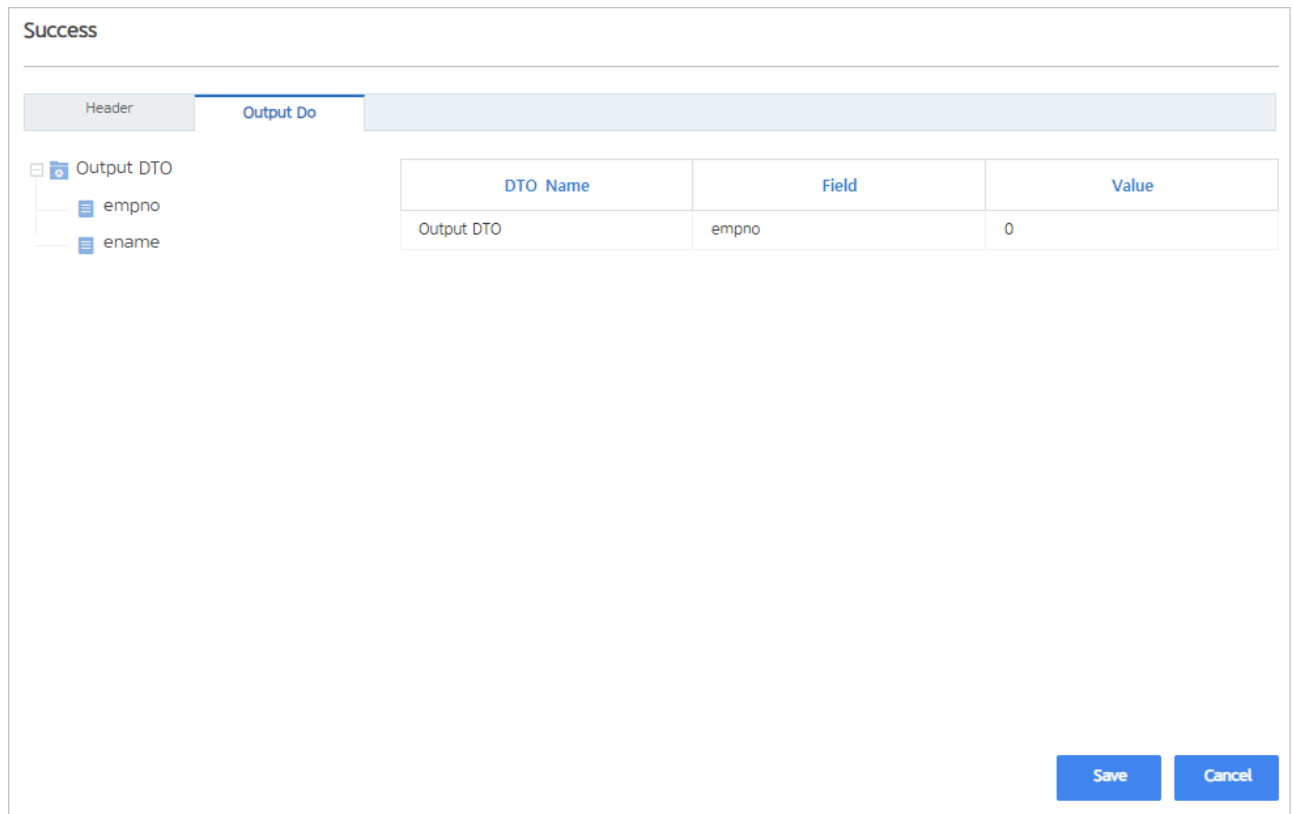
- `${PROOBJECT_HOME}/logs/ProObject.log`를 tail을 설정해 놓은 상태에서 아래와 같이 [Test] 버튼을

클릭한다.



ProManager 테스트 (4)

6. 아래와 같은 화면이 나타나고 로그에 .so 파일에 작성한 로그가 찍히는 것을 확인하면 성공이다.



ProManager 테스트 (5)

```
[2019.12.09 10:57:33] [INFO] [GUID-12161722231994139331697520112325099647]
<<TestApp1.TestSG1.S0test1>> is ready for execution : Method - service, WaitObject - null
[2019.12.09 10:57:33] [INFO] [GUID-12161722231994139331697520112325099647] ##### Test #####
[2019.12.09 10:57:33] [INFO] [GUID-12161722231994139331697520112325099647]
```

```
<<TestApp1.TestSG1.S0test1>> done for executing : Method - service
[2019.12.09 10:57:33] [INFO] [GUID-121617222231994139331697520112325099647]
<<TestApp1.TestSG1.S0test1>> is done!
```

다음과 같은 에러가 발생한다면 `${PROOJECT_HOME}/config/system.properties` 파일의 `SYSTEM_APPLICATION` 설정이 등록된 Application명과 같은지 확인한다.

```
[RTE-0110] : ApplicationNotFound
```

3.4.5. 빌드 및 런타임 서버 배포

다음의 과정으로 **ProManagerOps**를 실행해서 런타임 서버에 리소스를 빌드하고 배포한다.

1. 기존 Git 버전에서 Jenkins 빌드를 통해 OPS로 요청을 보내고 배포가 가능했다면 DB 버전에서는 Jenkins 대신 특정 셸 파일을 실행하여 OPS로 요청을 보낸다.

`${PROOJECT_HOME}/system/devops/devbuild` 경로에 있는 파일에 755 권한을 부여한다.

```
-rwxr-xr-x. 1 po7dev po7dev 15749 Dec  3 00:46 build.gradle
-rwxr-xr-x. 1 po7dev po7dev   668 Dec  4 00:11 devBuildTrigger.sh
```

2. `devBuildTrigger.sh` 파일을 사용자 환경에 맞게 수정한다.

```
#!/bin/bash

DEVOPS_HOME=${PROOJECT_HOME}/system/devops
export DEVOPS_HOME

CLASSPATH=${CLASSPATH}:${DEVOPS_HOME}/jenkins/devclient/proobject-devclient-7.0.0.0.jar
CLASSPATH=${CLASSPATH}:${DEVOPS_HOME}/binary/proobject-devserver-7.0.0.0.jar
CLASSPATH=${CLASSPATH}:${DEVOPS_HOME}/binary/proobject-devserver-dto-7.0.0.0.jar
CLASSPATH=${CLASSPATH}:${DEVOPS_HOME}/jenkins/script/3rdPartyLib/proobject-srcgen-7.0.0.0-jar-with-dependencies.jar
CLASSPATH=${CLASSPATH}:${DEVOPS_HOME}/jenkins/script/3rdPartyLib/proobject-client-7.0.0.0-jar-with-dependencies.jar

java -classpath ${CLASSPATH} com.tmax.proobject.devclient.build.DevPartialBuildCall
"192.168.105.111" "14000" $1 $2 $3 $4
```

3. 리소스 경로에 있는 xml 파일을 `devbuild` 경로에 넣는다.
 - a. 로컬에서 `AppName`, `SGName` 폴더를 만든다.
 - b. 각각 폴더에 `meta` 폴더를 만든다.
 - c. `${PROOJECT_HOME}/resources/metas/AppName/SGName` 경로에서 모듈별 타입 폴더를 위에서 만든 `meta` 폴더 아래에 생성한다.

다음은 모듈별로 생성되는 폴더이다.

모듈별 타입	폴더
DO, DOMsg	PO_IO
DOF, QO	PO_DBIO
BO, SO	PO_MODULES
JO	PO_BATCH

- d. 스튜디오에서 생성한 application.xml 파일을 복사하여 AppName\meta 폴더에 넣는다.
- e. 스튜디오에서 생성한 servicegroup.xml 파일을 복사하여 AppName\meta 폴더에 넣는다.
- f. AppName, SGName 폴더를 \${PROOJECT_HOME}/system/devops/devbuild 경로에 넣는다.

해당 예제에서 트리 구조는 아래와 같다.

```

po7dev@ns:~/proobject7/system/devops/devbuild$ ls -R TestApp1
TestApp1:
meta

TestApp1/meta:
application.xml
po7dev@ns:~/proobject7/system/devops/devbuild$ ls -R TestSG1
TestSG1:
meta

TestSG1/meta:
PO_IO PO_MODULES servicegroup.xml

TestSG1/meta/PO_IO:
D0test1.xml D0test1MsgJson.xml

TestSG1/meta/PO_MODULES:
S0test1.xml
po7dev@ns:~/proobject7/system/devops/devbuild$

```

4. 해당 경로에서 셸 파일을 실행한다.

'1'은 build number. 'TestApp1'은 Application Name, 'TestSG1'은 ServiceGroup Name이다. 'true'는 DB 버전에서 셸 파일을 사용하는 옵션이다.

```
sh devBuildTrigger.sh 1 TestApp1 TestSG1 true
```

```

po7dev@ns:~/proobject7/system/devops/devbuild$ sh devBuildTrigger.sh 1 TestApp1 TestSG1 true
12월 09, 2019 10:40:04 오전 com.tmax.proobject.devclient.build.DevPartialBuildCall main
정보: Dev Partial Build Call is Start
12월 09, 2019 10:40:04 오전 com.tmax.proobject.devclient.build.DevPartialBuildCall main
정보: property path : /home/po7dev/proobject7/system/devops/devbuild/gradle.properties
12월 09, 2019 10:40:07 오전 com.tmax.proobject.devclient.build.DevPartialBuildCall
srcgenDevResourcesToDevBuildDir
정보: Is Success copy Dev Resources To DevBuildDir : true
12월 09, 2019 10:40:07 오전 com.tmax.proobject.devclient.build.DevPartialBuildCall main

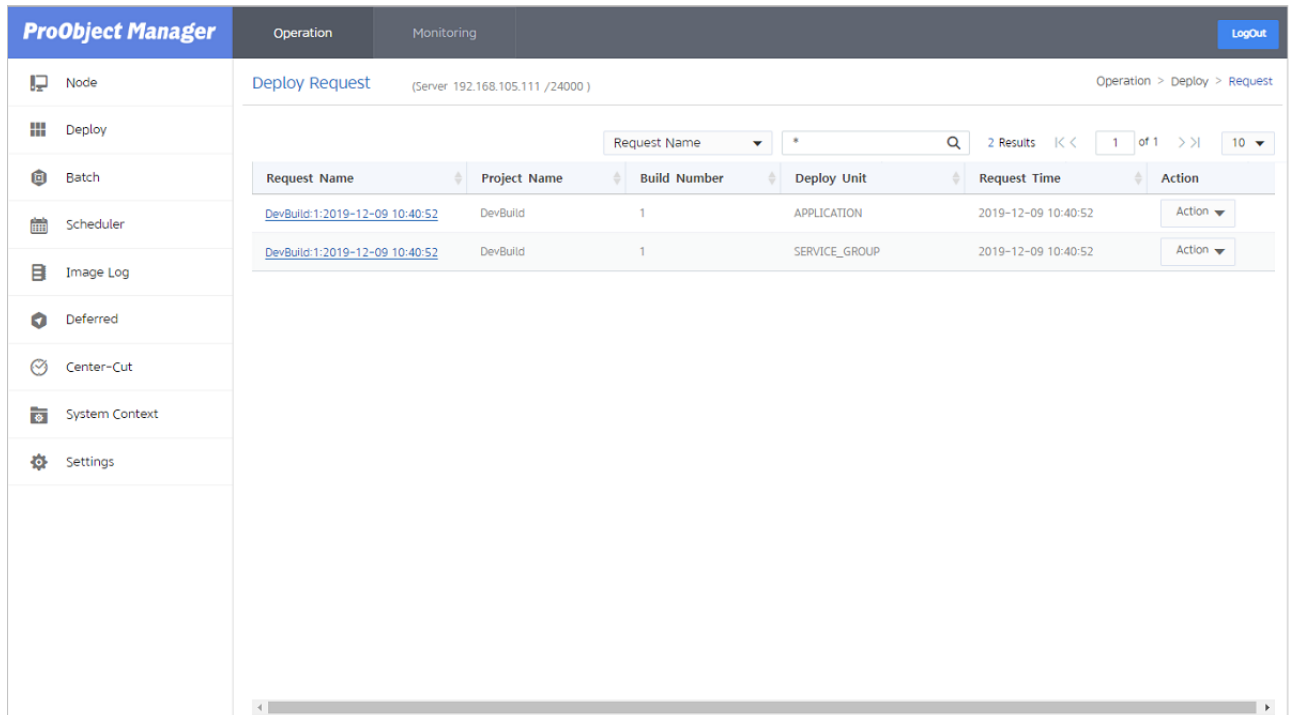
```

```

정보: requestId : 1 appName : TestApp1 sgName : TestSG1 serverIp : 192.168.105.111 serverPort : 14000
12월 09, 2019 10:40:07 오전 com.tmax.proobject.devclient.build.DevPartialBuildCall main
정보: return of get build class path is true
12월 09, 2019 10:40:56 오전 com.tmax.proobject.devclient.build.DevPartialBuildCall main
정보: DevPartialBuild Success! true
12월 09, 2019 10:40:56 오전 com.tmax.proobject.devclient.build.DevPartialBuildCall main
정보: Dev Partial Build Call is End
po7dev@ns:~/proobject7/system/devops/devbuild$

```

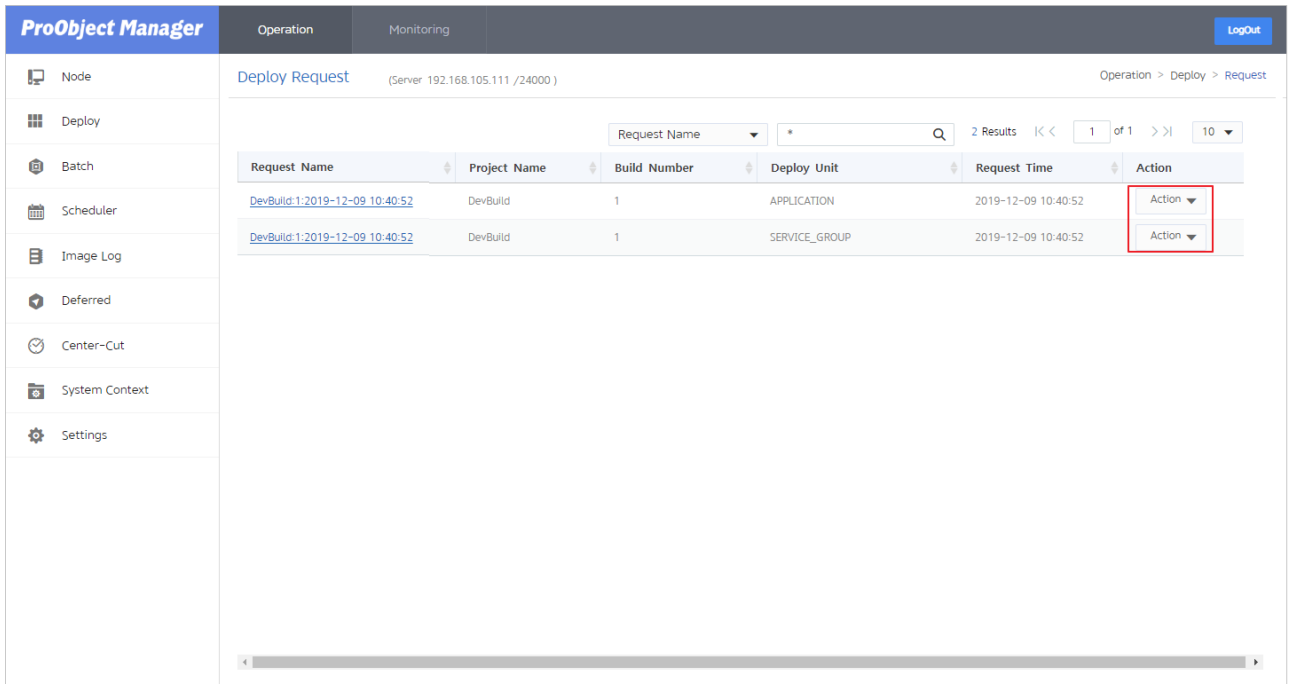
5. ProManagerOps의 네비게이션 영역에서 [Deploy] > [Request]를 선택한 후 Deploy Request 화면에서 Deploy Request가 들어왔는지 확인한다.



ProManagerOps 테스트 (1)

6. `${PROOBJECT_HOME}/system/config`에서 설정한 `dbio_config.xml` 파일을 런타임 서버 `${PROOBJECT_HOME}/application/AppName/config`에 넣어준다.

ProManager의 [Deploy] > [Request]에서 빌드된 요청의 'Action' 항목에서 'Deploy'를 선택한다. 초기 한번만 Deploy이고 이후 요청이 들어올 경우 HotDeploy이다.



ProManagerOps 테스트 (2)

- 배포하는 과정에서 Ops 서버 로그에 에러가 발생하지 않는지 확인한다(`${PROOBJECT_HOME}/logs/ProObject.log`). RTE 서버에 다음 경로에 .jar 파일의 형태로 배포된다.

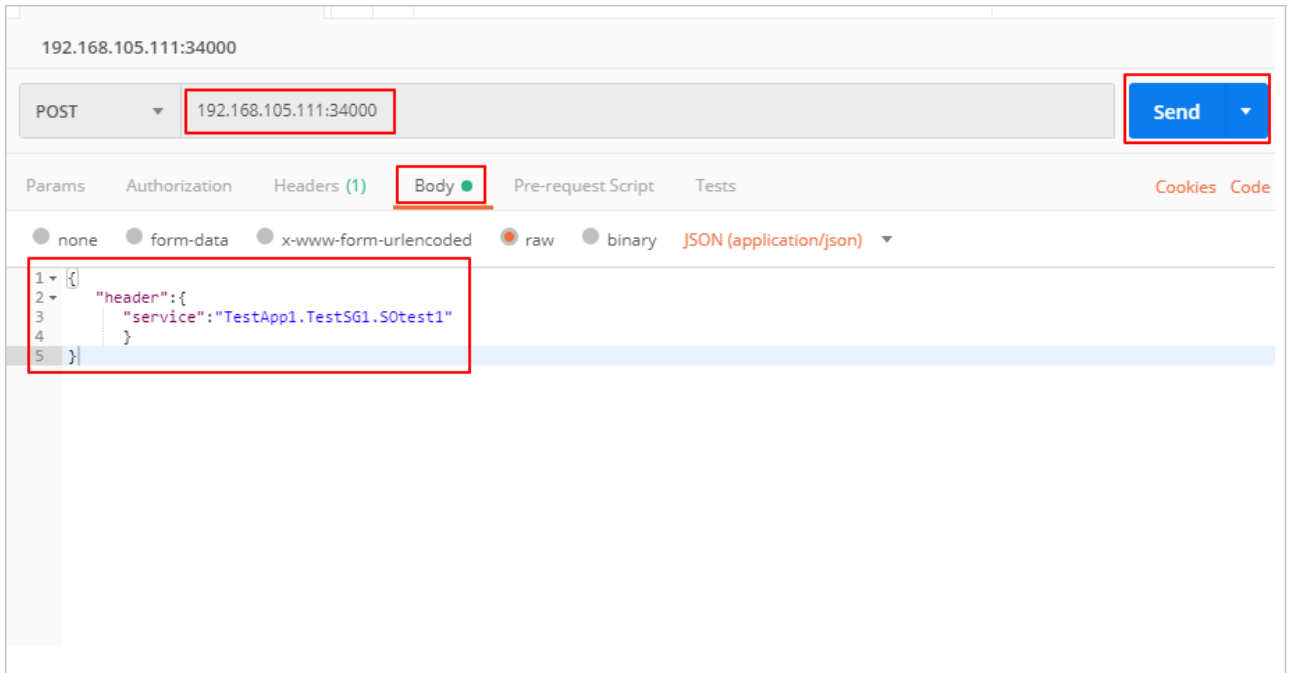
```
/home/po7rte/proobject7/application/TestApp1/servicegroup/TestSG1
```

- 다음은 vim TestSG1.jar 결과이다.

```
" zip.vim version v27
" Browsing zipfile /home/po7rte/proobject7/application/TestApp1/servicegroup/TestSG1/TestSG1.jar
" Select a file with cursor and press ENTER

META-INF/
META-INF/MANIFEST.MF
com/
com/tmax/
com/tmax/proobject/
com/tmax/proobject/test/
com/tmax/proobject/test/runtime/
com/tmax/proobject/test/runtime/servicegroup/
com/tmax/proobject/test/runtime/servicegroup/main/
com/tmax/proobject/test/runtime/servicegroup/main/service/
com/tmax/proobject/test/runtime/servicegroup/main/service/memorycalc/
com/tmax/proobject/test/runtime/servicegroup/main/service/memorycalc/ProObjectMemoryCalculatorFor
LengthCalc.class
com/tmax/S0test1Executor.class
com/tmax/S0test1.class
```

- `${PROOBJECT_HOME}/logs/ProObject.log`를 tail 걸어 놓은 상태에서 Postman에서 아래와 같이 서비스 호출을 실행한다(Application명, Service Group명, SO파일명)..



ProManagerOps 테스트 (3) - 서비스 호출

10. RTE 서버의 `${PROOBJECT_HOME}/logs/ProObject.log`에 정상적으로 로그가 저장되었는지 확인한다.

다음과 같은 에러가 발생한다면 `${PROOBJECT_HOME}/config/system.properties` 파일의 `SYSTEM_APPLICATION` 설정이 등록된 Application명과 같은지 확인한다.

```
[RTE-0110] : ApplicationNotFound
```

3.5. 제거

설치된 개발 서버(DevOps)를 제거하기 위해서는 실행중인 서버를 모두 종료 후 설치한 폴더의 파일을 제거해주어야 한다. 설치 파일 삭제에 실패한 경우 해당 폴더로 이동한 후 명령어를 실행해서 삭제를 진행한다.

4. ProStudio 설치 및 제거

본 절에서는 ProObject Studio(이하 ProStudio)를 설치하고 제거하는 방법에 대해서 설명한다.

4.1. 설치

ProStudio는 별도의 인스톨러 없이 제공된 압축파일을 해제하는 과정으로 설치를 진행한다. 제공된 ProStudio.zip 파일을 임의의 디렉터리에서 압축을 해제한다.

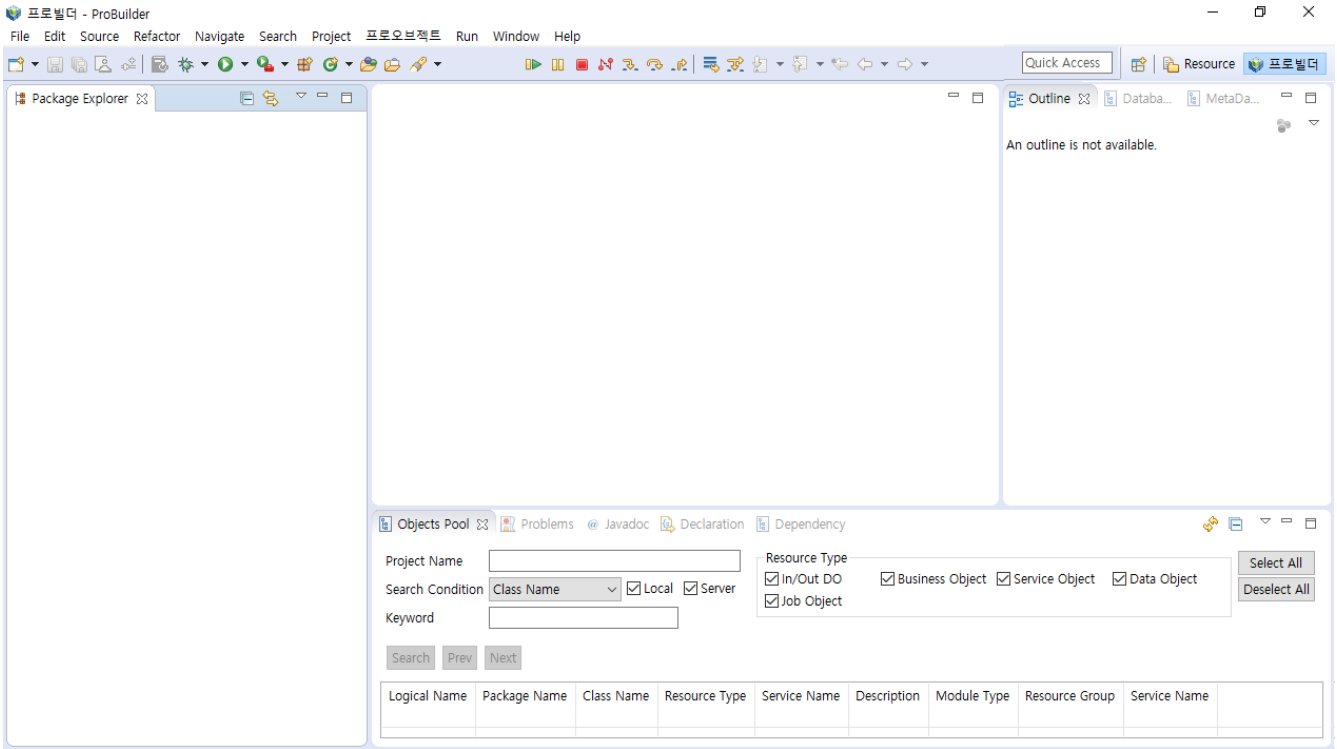
압축을 해제하면 다음과 같이 파일이 생성되어 있어야 한다.

이름	수정한 날짜	유형	크기
configuration	2020-01-20 오후 1:55	파일 폴더	
features	2019-05-20 오후 8:38	파일 폴더	
p2	2019-05-20 오후 8:38	파일 폴더	
plugins	2019-05-20 오후 8:38	파일 폴더	
readme	2020-01-20 오후 1:54	파일 폴더	
.eclipseproduct	2016-02-03 오전 10:08	ECLIPSEPRODUCT 파일	1KB
artifacts.xml	2019-02-22 오후 5:41	XML	146KB
eclipsesec.exe	2019-02-20 오후 5:31	응용 프로그램	18KB
ProObject.exe	2019-02-20 오후 5:31	응용 프로그램	306KB
ProObject.ini	2019-02-22 오후 5:39	구성 설정	1KB

[ProStudio] - 설치 후 디렉터리

4.2. 설치 확인

ProObject.exe 파일을 클릭해서 ProStudio를 실행한다. **Open Perspective**에서 "프로빌더"를 선택하고 다음과 같이 프로그램이 실행되는 것을 확인한다.



[ProStudio] - workspace



스튜디오 기동과 관련된 자세한 내용은 "ProObject Studio 개발자 안내서"를 참조한다.

4.3. 제거

ProStudio를 설치했던 폴더와 생성한 리소스 파일의 경로에 있는 파일을 삭제한다. 파일 삭제에 실패한 경우 해당 폴더로 이동한 후 명령어를 실행해서 삭제를 진행한다.

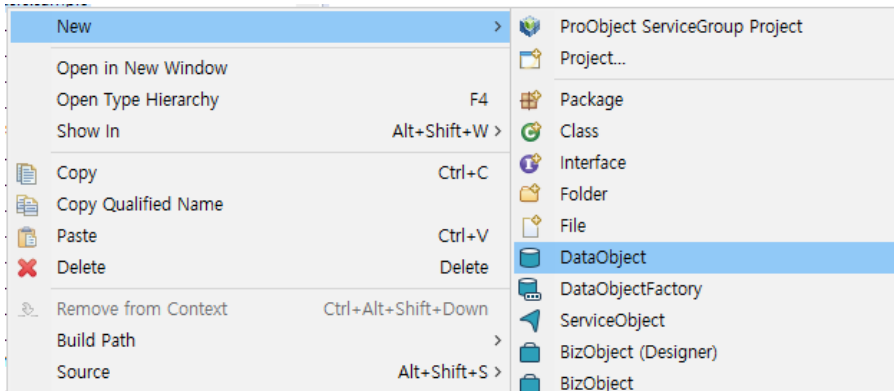
Appendix A: 시작하기

본 부록에서는 ProObject Studio를 사용해서 DO, DOF, BO, SO를 생성한 서비스를 등록하고 테스트하는 과정에 대해서 설명한다.

A.1. DataObject(DO) 생성

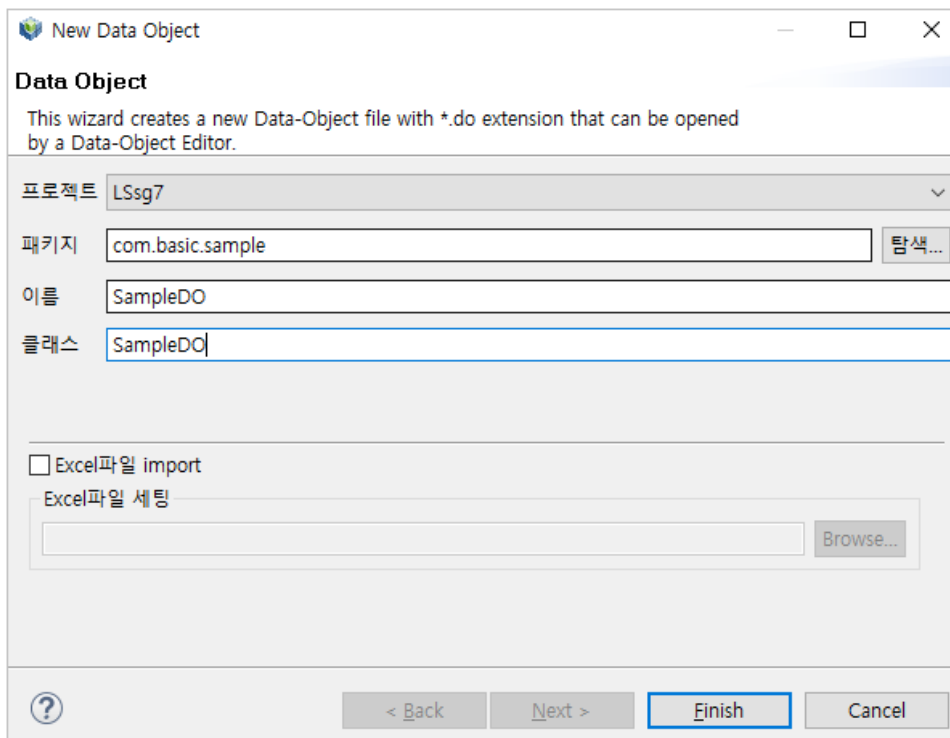
다음은 DO를 생성하는 과정에 대한 설명이다.

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



DO 생성

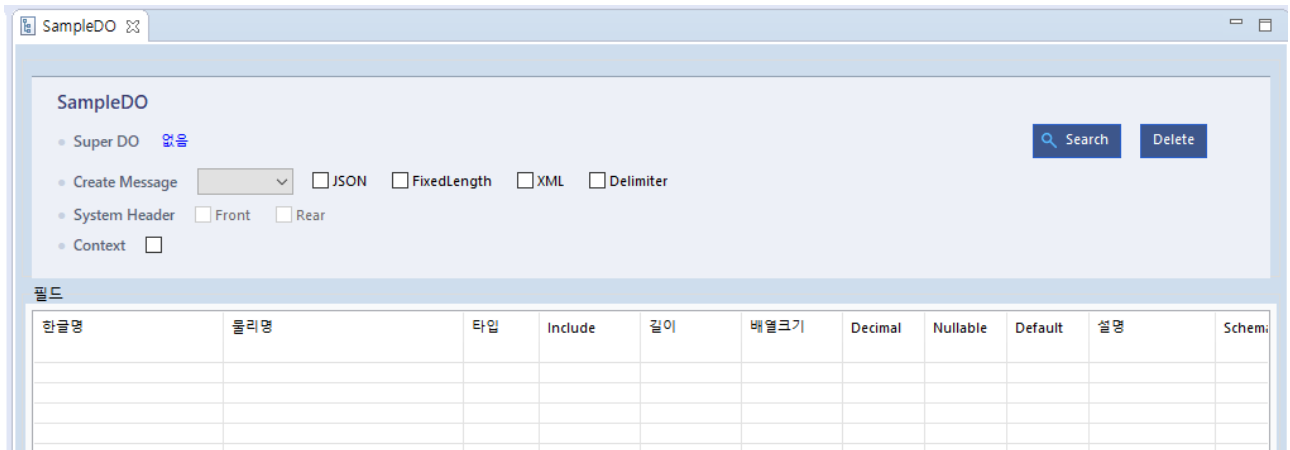
2. **New Data Object 화면**에서 프로젝트를 선택한 후 DataObject 정보를 입력한다. 각 항목을 입력한 후 **[Finish]** 버튼을 클릭한다.



New Data Object 화면

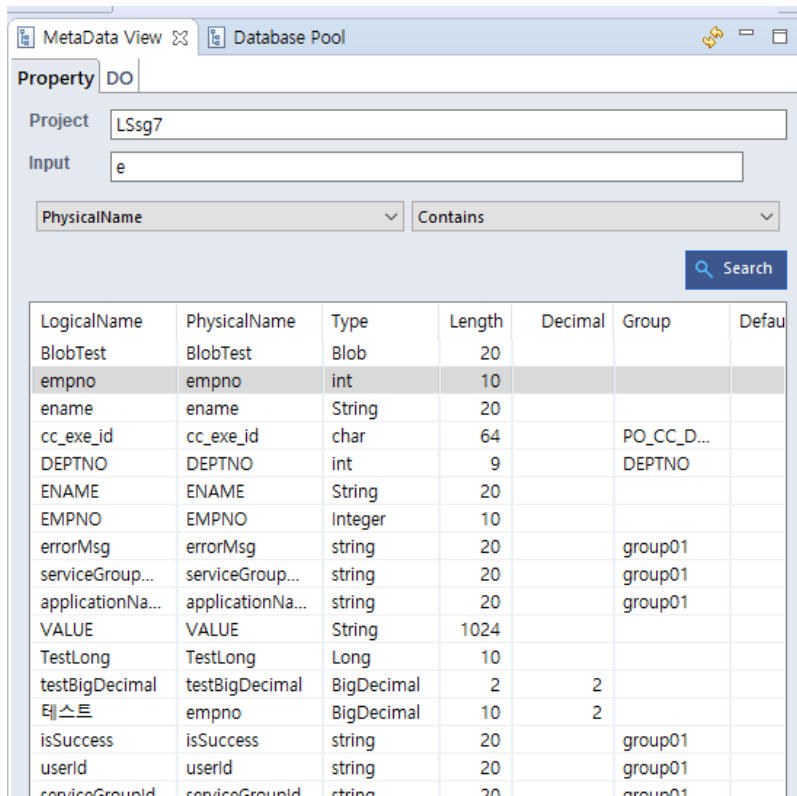
DO 디자인 화면에서 **Package Explorer**의 meta 아래 '{물리명}.dto', .gen_src_dto 아래 '{물리명}.java'가

정상적으로 추가되었는지 확인한다.



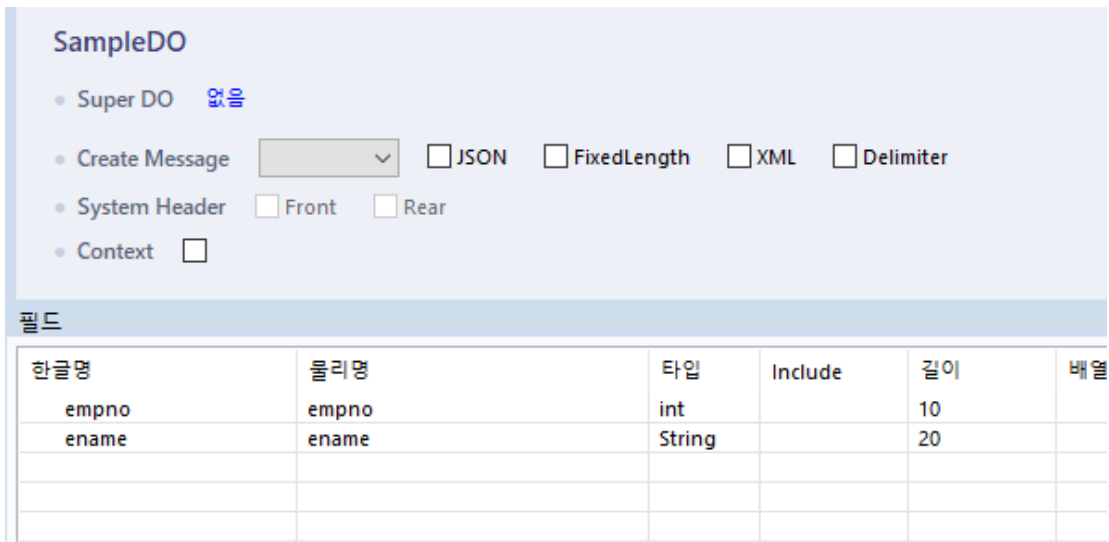
DO 디자인 화면

3. **MetaData View**에서 DO의 필드에 추가할 메타를 검색한다. 메타는 사전에 등록되어 있어야하며, **ProManager**에서 등록할 수 있다.



MetaData view 화면

검색한 메타들을 드래그 앤 드롭 방식 또는 더블클릭으로 메타들을 추가한다.

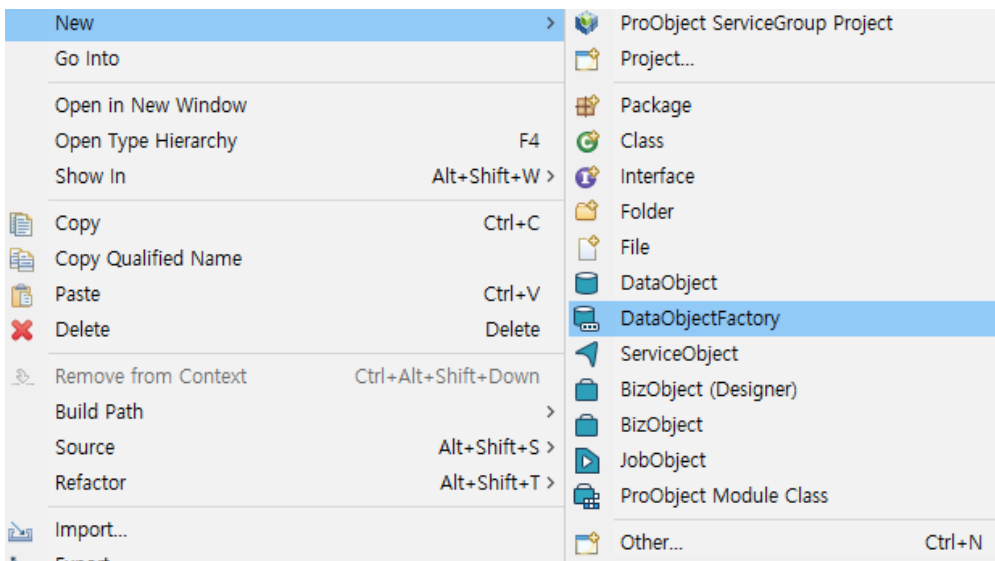


메타 추가 후 DataObject 에디터 화면

A.2. DataObject Factory(DOF) 생성

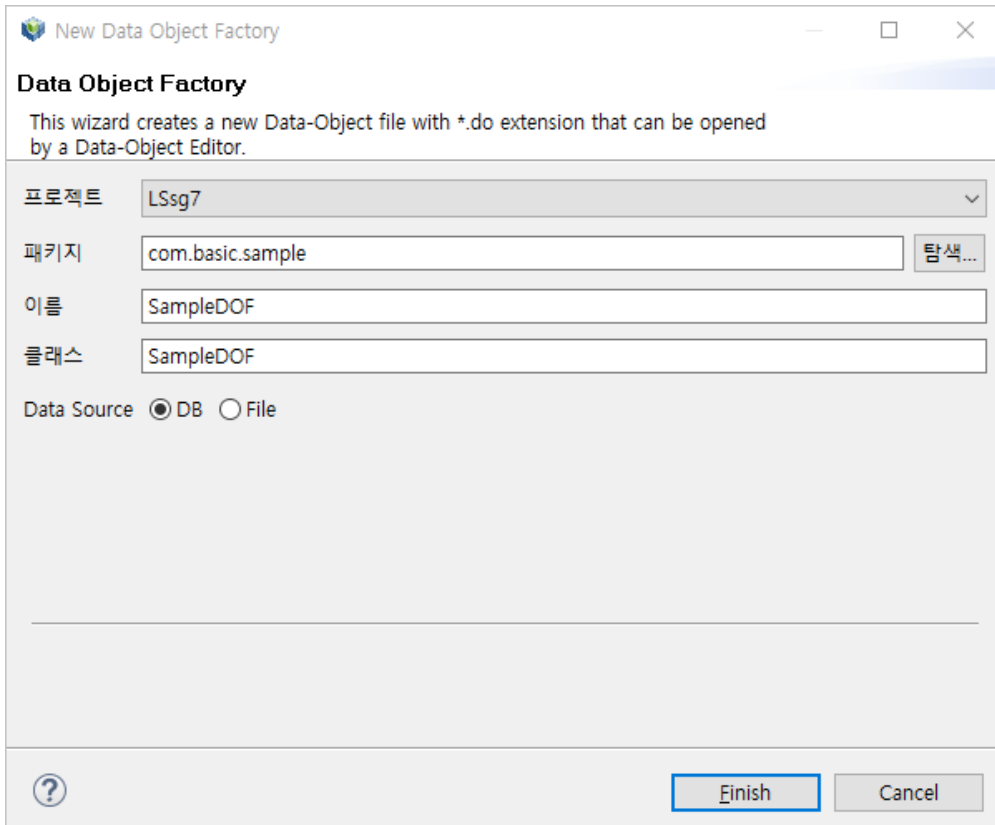
다음은 DOF를 생성하는 과정에 대한 설명이다.

1. Package Explorer의 컨텍스트 메뉴에서 **[New] > [DataObject Factory]**를 선택한다.



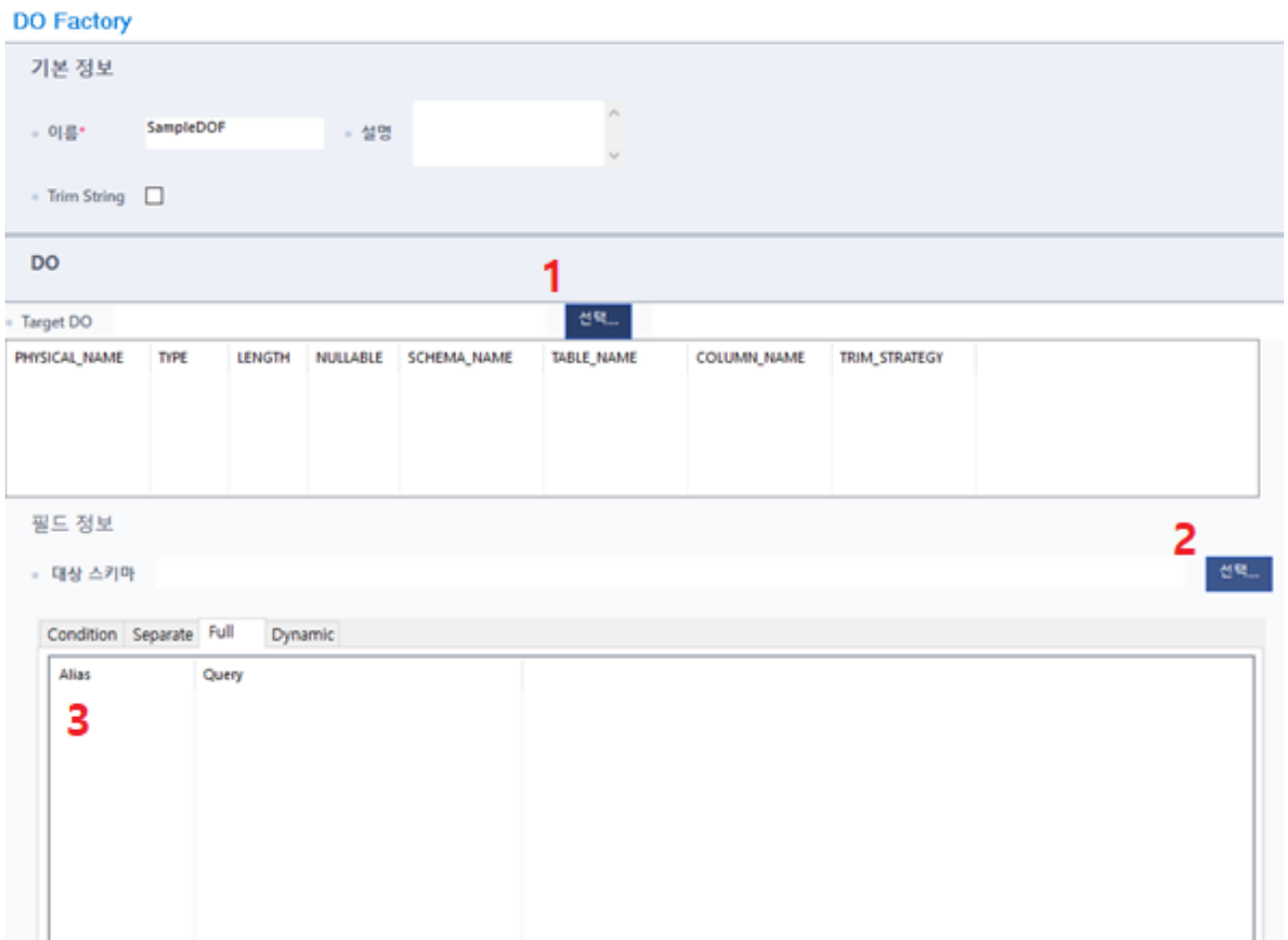
Data Object Factory 생성

2. **New DataObject Factory 화면**에서 프로젝트를 선택 후 각 항목을 입력한 후 **[Finish]** 버튼을 클릭한다.



New DataObject Factory 화면

아래의 화면과 같이 DOF가 생성된 것을 확인할 수 있다.



초기 DOF 화면

DOF를 이용하기 위해서는 Target DO 선택, 스키마 선택, 쿼리 작성 의 작업이 추가적으로 진행되어야 한다.

- a. 생성한 DOF에 Target DO를 **DOF 화면(초기 DOF 화면)**의 1번에 해당되는 부분에서 설정한다.

PHYSICAL_NAME	TYPE	LENGTH	NULLABLE	SCHEMA_NAME	TABLE_NAME	COLUMN_NAME	TRIM_STRATEGY
empno	Integer	10	false	po7_51_24	EMP	EMPNO	
ename	String	20	false	po7_51_24	EMP	ENAME	STRING

Target DO 추가된 화면

- b. 대상 스키마를 선택하는 경우 연결 정보 및 스키마를 선택해 줄 수 있다.

연결 정보: tiberob6 스키마: PO7_51_24

스키마 선택

연결 정보 및 스키마를 선택할 경우 아래의 화면에서 테이블 및 컬럼을 선택할 경우 쿼리를 자동 생성할 수 있다.

데이터를 가져올 대상 테이블과 컬럼들을 선택하세요.

테이블 목록: 필터 emp, EMP

선택	PK	컬럼명	타입	설명
<input checked="" type="checkbox"/>		EMPNO	NUMBER(4)	
<input checked="" type="checkbox"/>		ENAME	VARCHAR2	
<input type="checkbox"/>		JOB	VARCHAR2	
<input type="checkbox"/>		MGR	NUMBER(4)	
<input type="checkbox"/>		HIREDATE	TIMESTAMP	
<input type="checkbox"/>		SAL	NUMBER(7)	
<input type="checkbox"/>		COMM	NUMBER(7)	
<input type="checkbox"/>		DEPTNO	NUMBER(2)	

선택된 컬럼(들): EMP [A], EMPNO, ENAME

테이블, 컬럼 선택

- c. 쿼리를 직접 작성하는 원하는 경우 **DOF 화면(초기 DOF 화면)**의 3번에 해당되는 부분에 직접 작성할 수 있다.

Condition Separate Full Dynamic

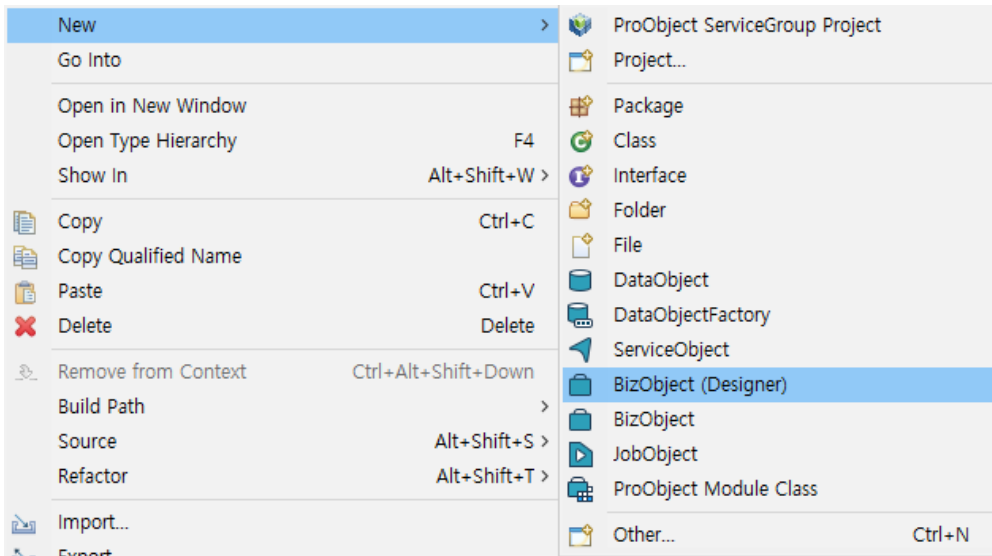
Alias	Query
select_default_0	/* com.basic.sample.SampleDOF */SELECT A.EMPNO, A.ENAMEFROM EMP AWHERE A.EMPNO = :empno AND A.ENAME = :en...
update_default_0	/* com.basic.sample.SampleDOF */UPDATE EMPSET ENAME = :enameWHERE A.EMPNO = :empno AND A.ENAME = :ename
insert_default_0	/* com.basic.sample.SampleDOF */INSERT INTO EMP (EMPNO, ENAME) VALUES (:empno, :ename)
delete_default_0	/* com.basic.sample.SampleDOF */DELETE FROM EMPWHERE A.EMPNO = :empno AND A.ENAME = :ename

쿼리 생성 결과

A.3. BizObject(BO) 생성

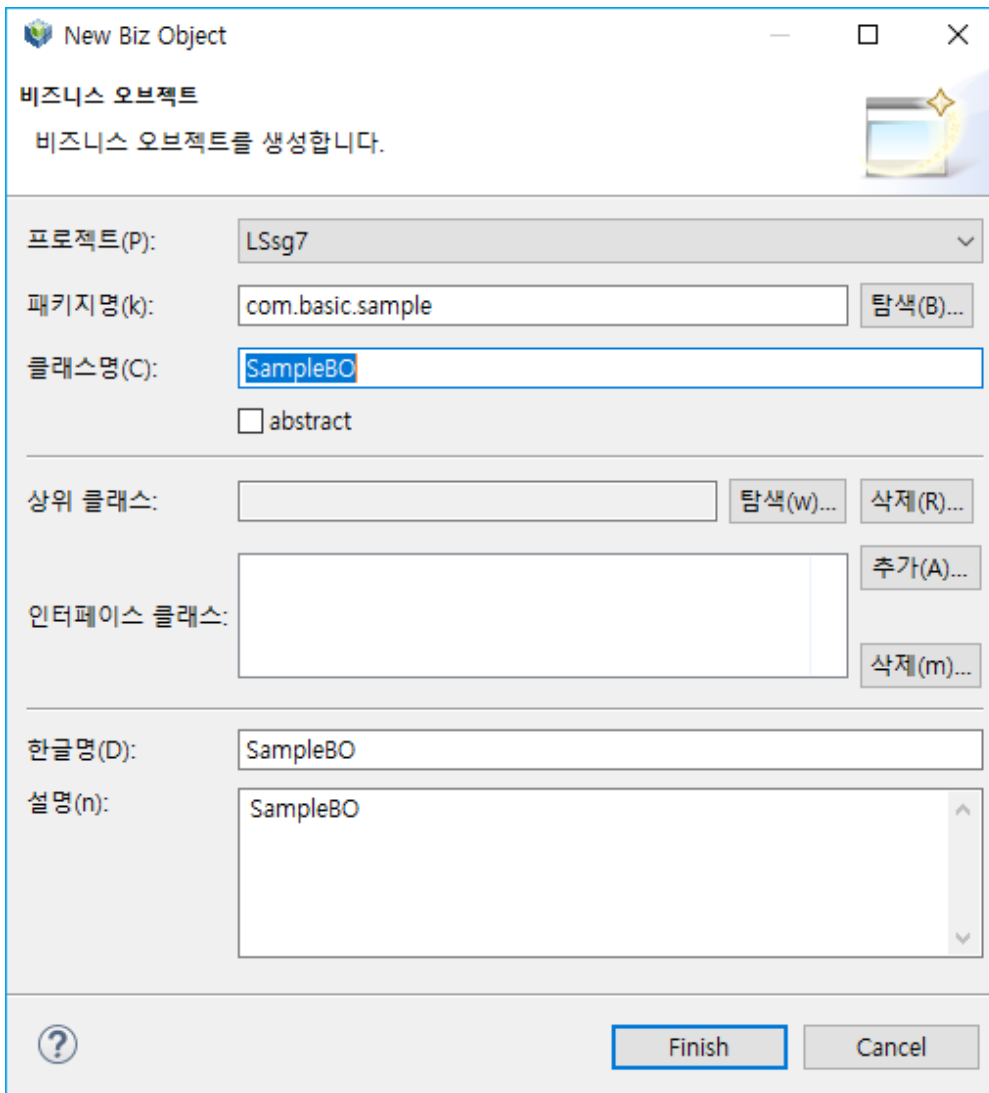
다음은 BO를 생성하는 과정에 대한 설명이다.

1. Package Explorer의 컨텍스트 메뉴에서 **[New] > [BizObject]**를 선택한다.



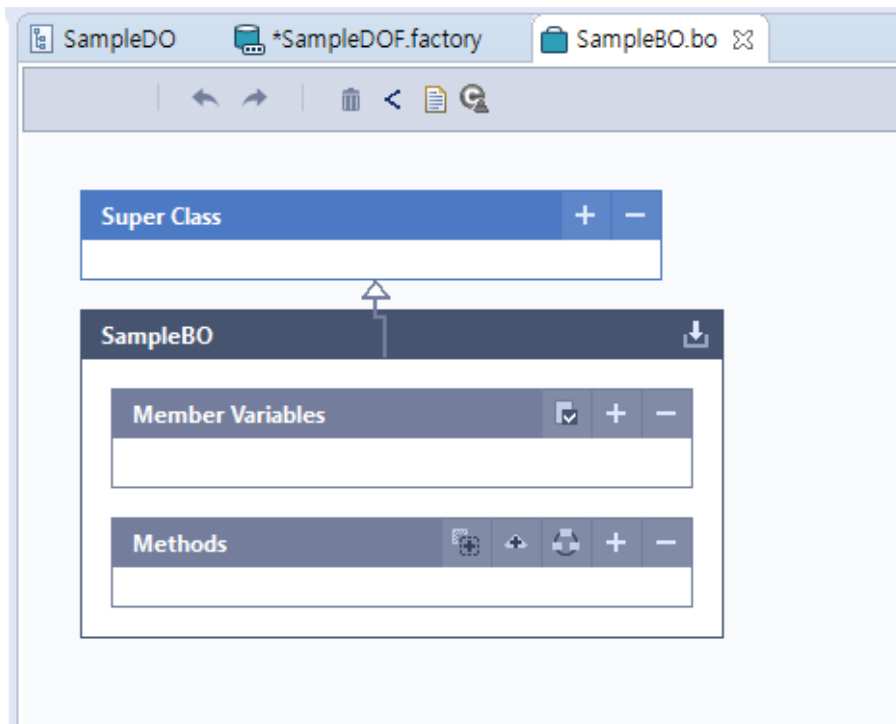
BizObject 생성

2. **New BizObject** 화면의 각 항목을 입력한 후 **[Finish]** 버튼을 클릭한다.



New BizObject 화면

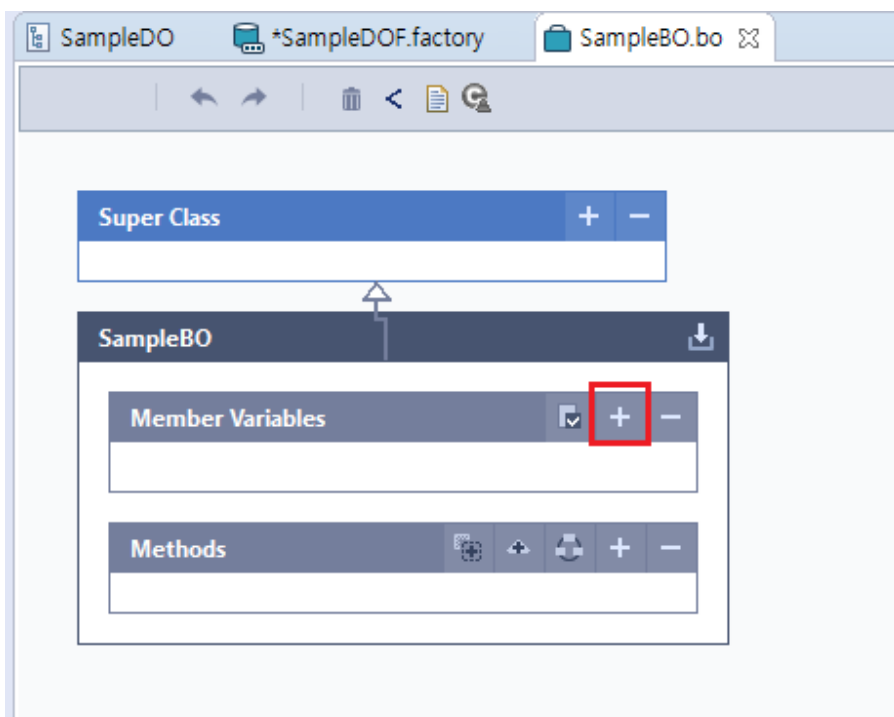
아래의 화면과 같이 BO가 생성된 것을 확인할 수 있다.



초기 BizObject 화면

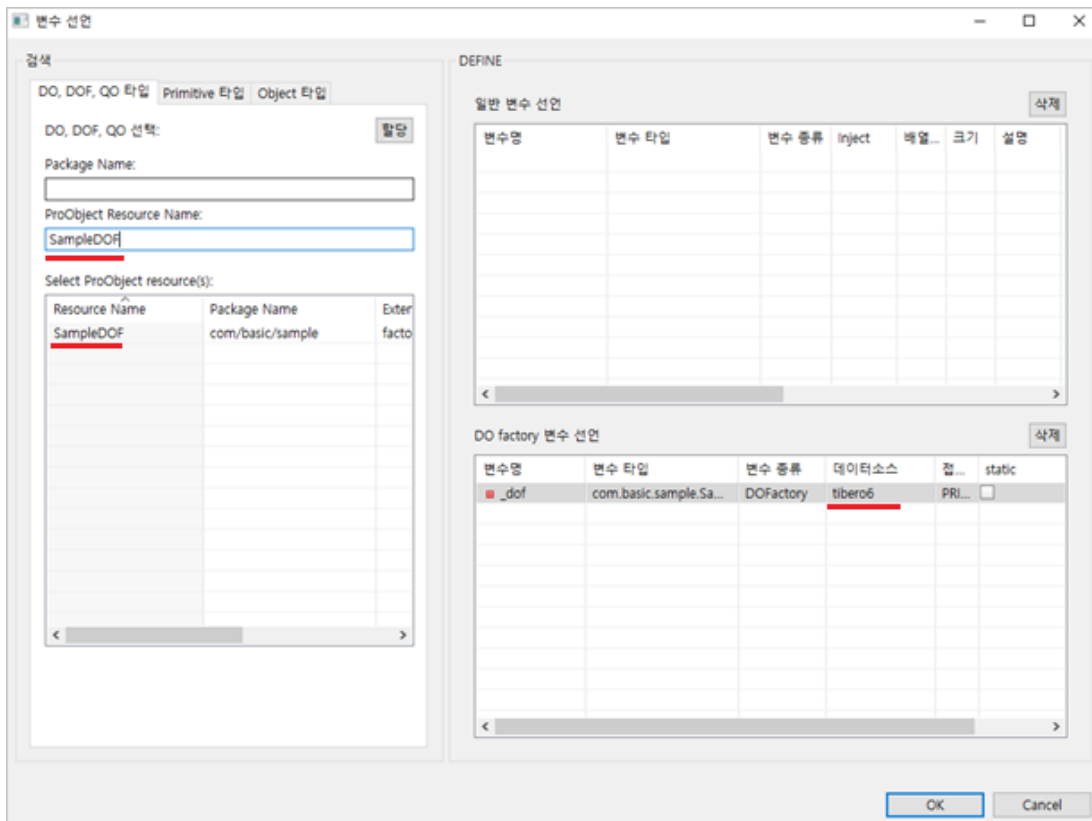
3. 생성한 BO에 이전에 생성한 DOF 모듈을 추가해야 한다.

- a. DOF를 모듈을 추가하기 위해서는 멤버 변수를 추가해야한다. 멤버 변수를 추가하기 위해서는 Member Variables의 [+]를 클릭한 후 이전에 생성한 DOF를 검색한 후 추가한다.



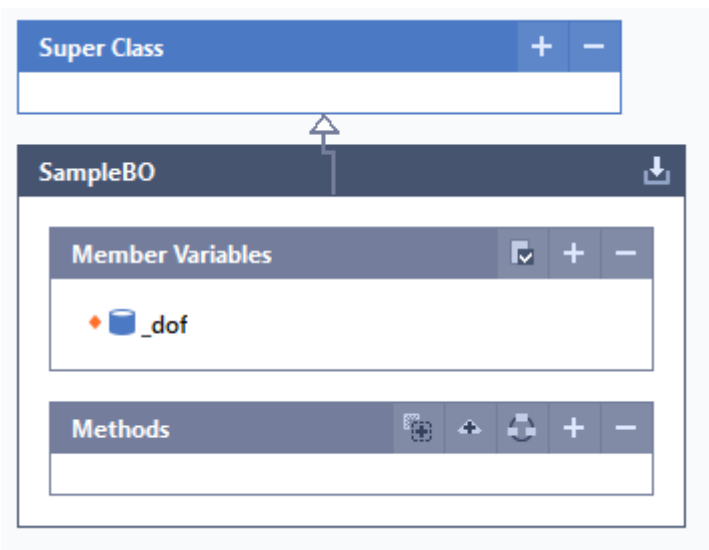
Member Variables 추가 화면 (1)

DOF를 추가하면 DO factory 'DO factory 변수 선언'에서 확인이 가능하며 사용할 데이터소스를 선택한다.



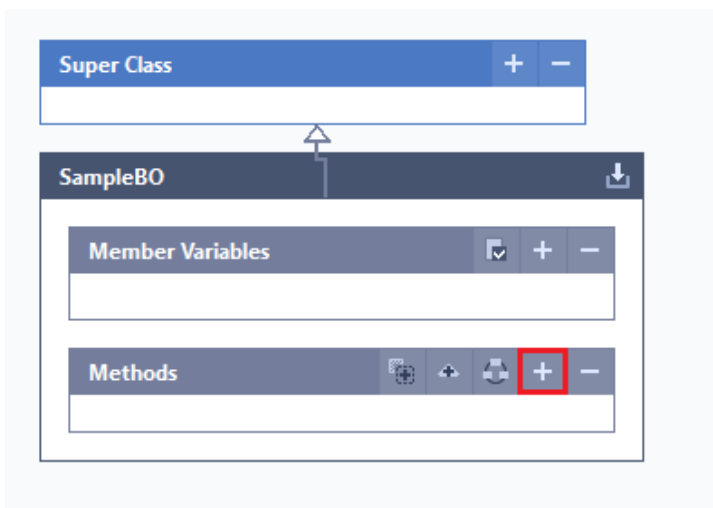
Member Variables 추가 화면 (2)

다음은 멤버 변수를 추가했을 때의 화면이다.



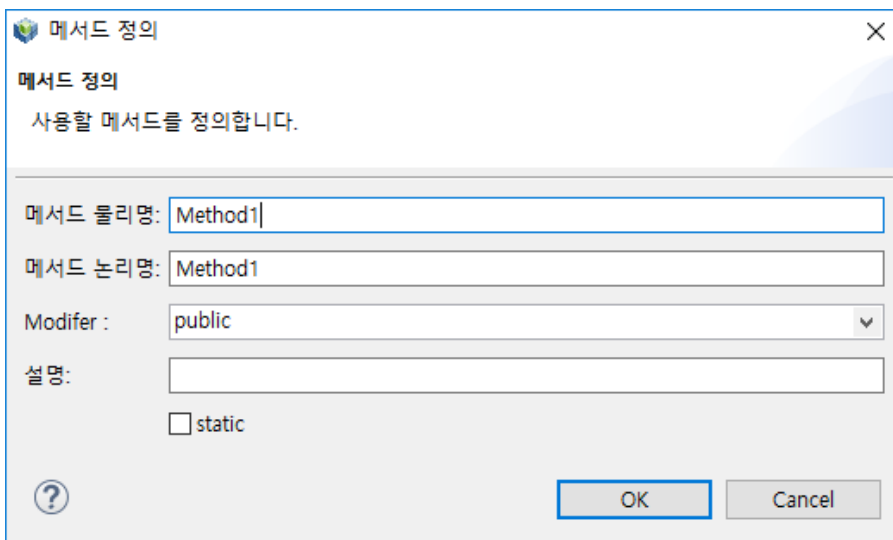
Member Variables 추가 후 화면

- b. 메소드를 생성하고, ServiceObject에서 입력 받은 값을 받아와 처리해 그 값을 다시 반환하도록 설정한다. **Methods**의 **[+]**를 클릭한 후 기본 정보를 설정한다.



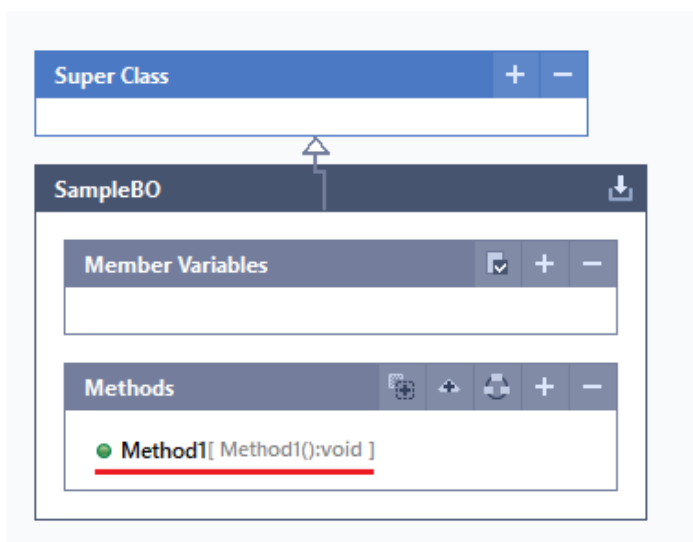
Methods 추가 화면 (1)

메서드 정의 화면에서 각 항목을 입력한 후 [OK] 버튼을 클릭한다.



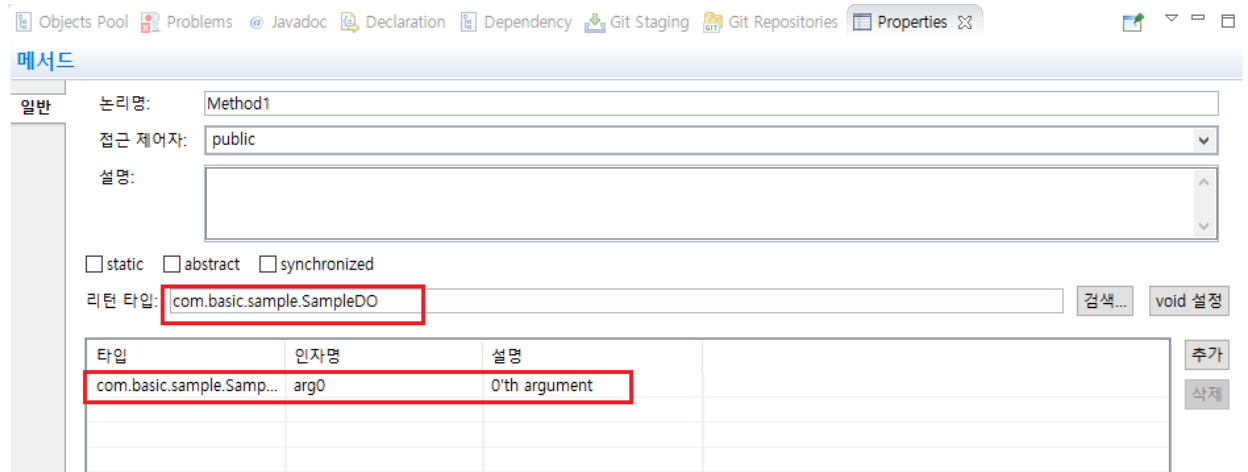
Methods 추가 화면 (2)

다음은 메소드를 추가한 후의 화면이다.





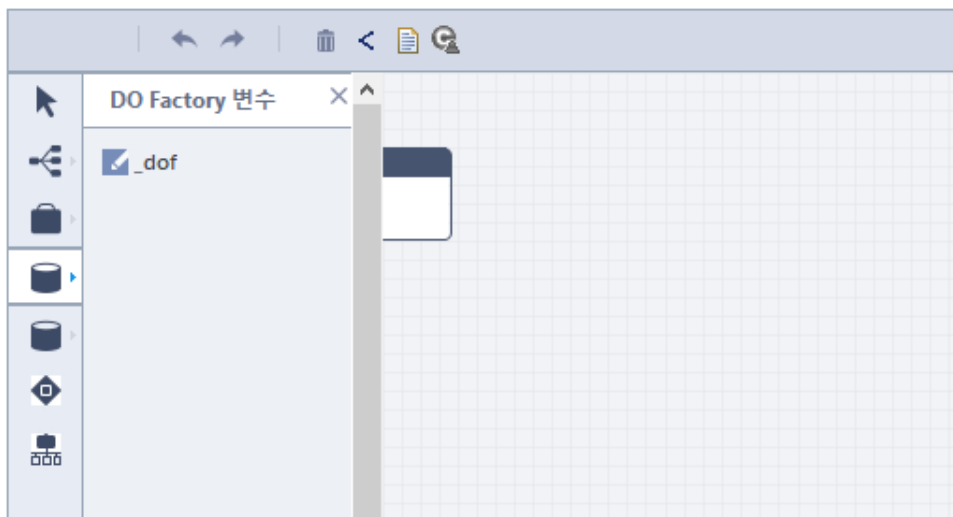
Methods 추가 후 결과 화면

추가된 메소드의 Properties에서 리턴 타입의 변경이 필요하며, 추가적으로 값을 받아오기 위한 매개변수를 추가하는 작업이 필요하다.

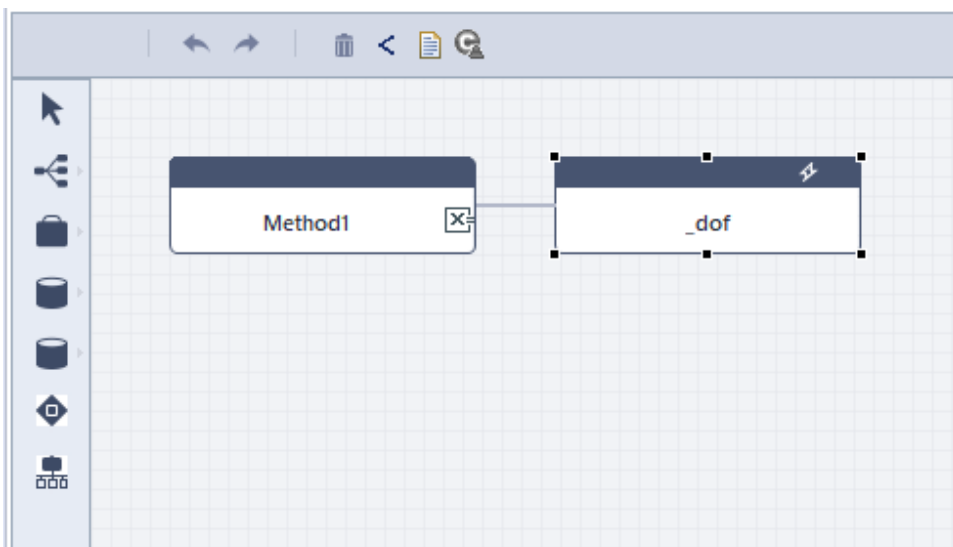


Method Property 화면

- c. **EMB Designer**의 좌측 Palette에서 모듈을 추가한다. DOF를 추가하기 위해서는 Palette에서 를 클릭하고, 를 드래그 앤드 드롭해서 디자인 화면에 넣는다.

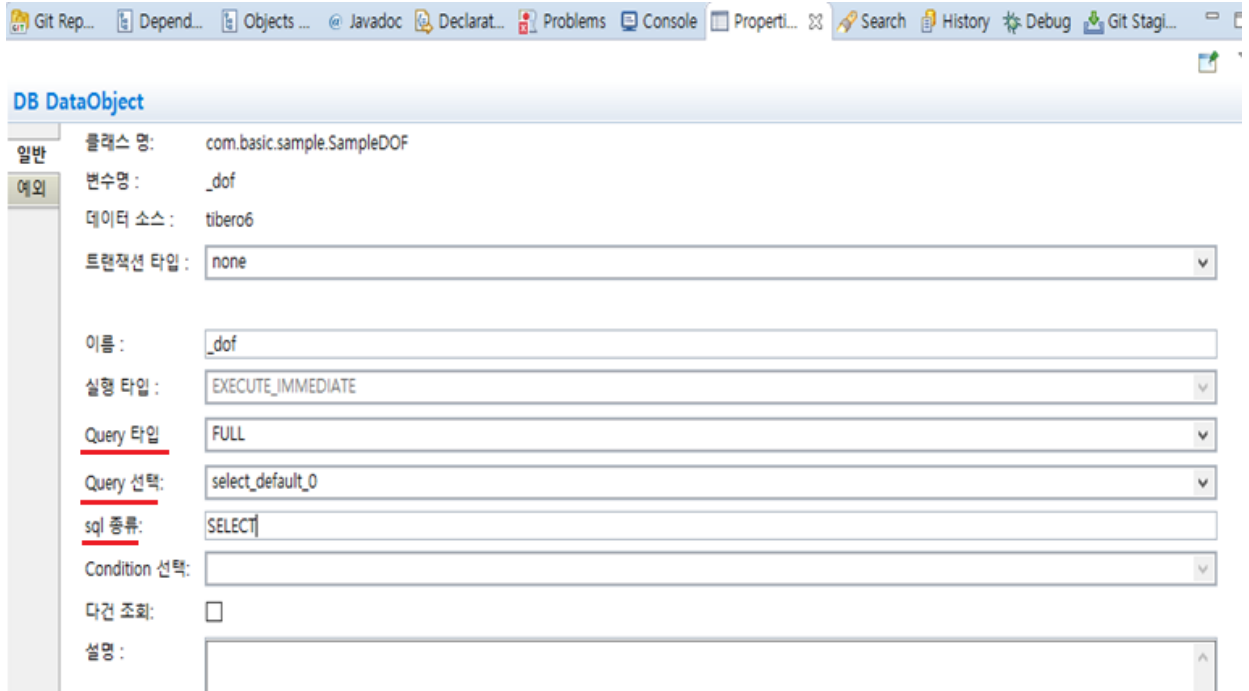


디자인 에디터 DOF 모듈 추가 화면 (1)




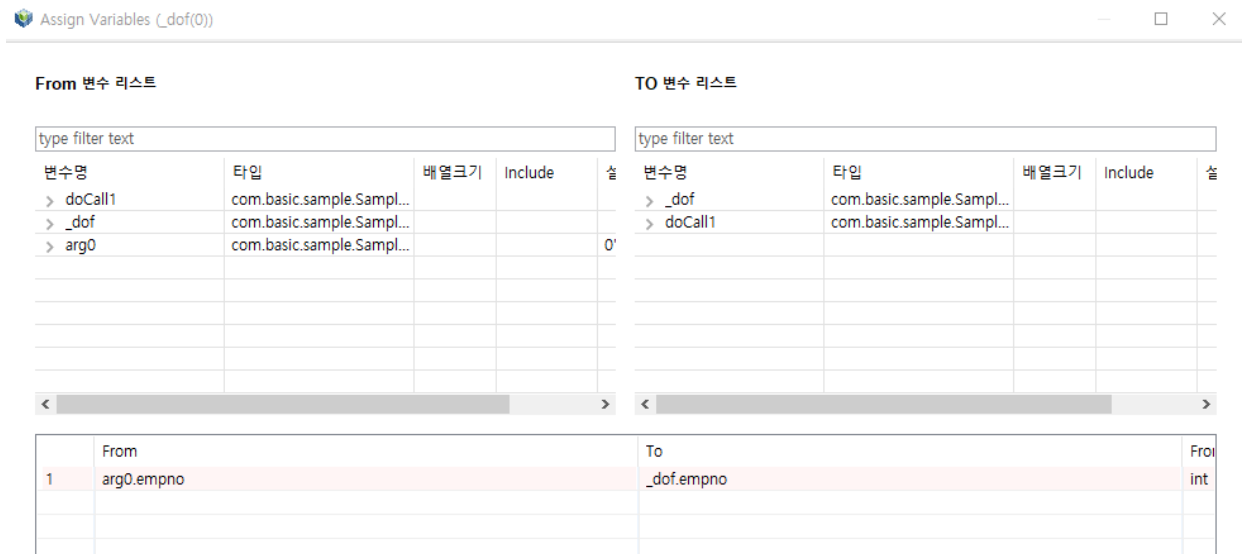
디자인 에디터 DOF 모듈 추가 화면 (2)

추가한 DOF Properties에는 'Query 타입'을 'FULL'로 선택한 후 'Query 선택' 항목을 택하면 'SQL 종류'가 자동으로 설정된다.



DOF 모듈 Properties 화면

추가한 모듈에 조회 파라미터를 매핑하기 위해 모듈에 마우스를 이동하고  버튼을 클릭한다.



파라미터 매핑 화면

조회 파라미터를 매핑할 경우 이에 대한 결과는 소스 영역에서 확인할 수 있다. 필요에 따라 결과값도 동일하게 사용한다.

- [SELECT]

```

// [BEGIN_PRE_ASSIGN_BLOCK, 0, Method1(SampleDO)]
{
    _dof.setEmpno(arg0.getEmpno());
}
// [END_PRE_ASSIGN_BLOCK, 0, Method1(SampleDO)]

```

마지막으로 메소드의 반환값을 작성한다.

```
    _dof.setFullQuery(com.basic.sample.SampleDOF.FULLQUERY.SELECT_DEFAULT_0);
    doCall1 = _dof.get();

    //[BEGIN_POST_ASSIGN_BLOCK, 0, Method1(SampleDO)]
    //[END_POST_ASSIGN_BLOCK, 0, Method1(SampleDO)]

    //[BEGIN_AFTER_CODE, 0, Method1(SampleDO)]
    return doCall1;
    //[END_AFTER_CODE, 0, Method1(SampleDO)]
}
//[END_NODE_BLOCK, 0, Method1(SampleDO)]
}
```

파라미터를 매핑하는 방법은 쿼리 종류마다 약간의 차이가 있으며, 상황에 따라 변경해서 사용한다.

▪ [INSERT]

```
//[END_BEFORE_CODE, 0, INSERT(ims204000DO)]

//[BEGIN_PRE_ASSIGN_BLOCK, 0, INSERT(ims204000DO)]
{
    doCall1.setEmpno(arg0.getEmpno());
}
{
    doCall1.setEname(arg0.getEname());
}

//[END_PRE_ASSIGN_BLOCK, 0, INSERT(ims204000DO)]

_dof.setFullQuery(com.tmax.ims204000.dof.ims204000DOF.FULLQUERY.INSERT_DEFAULT_0);
resultCnt1 = _dof.add(doCall1, true);

//[BEGIN_POST_ASSIGN_BLOCK, 0, INSERT(ims204000DO)]

//[END_POST_ASSIGN_BLOCK, 0, INSERT(ims204000DO)]

//[BEGIN_AFTER_CODE, 0, INSERT(ims204000DO)]
return doCall1;
```

▪ [UPDATE]

```
//[BEGIN_PRE_ASSIGN_BLOCK, 0, UPDATE(ims204000DO)]
{
    doCall1.setEname(arg0.getEname());
}
{
    _dof.setEmpno(arg0.getEmpno());
}

//[END_PRE_ASSIGN_BLOCK, 0, UPDATE(ims204000DO)]

_dof.setFullQuery(com.tmax.ims204000.dof.ims204000DOF.FULLQUERY.UPDATE_DEFAULT_0);
resultCnt1 = _dof.update(doCall1, true);

//[BEGIN_POST_ASSIGN_BLOCK, 0, UPDATE(ims204000DO)]
{
    doCall1.setEmpno((Integer) (_dof.getEmpno()));
}

//[END_POST_ASSIGN_BLOCK, 0, UPDATE(ims204000DO)]

//[BEGIN_AFTER_CODE, 0, UPDATE(ims204000DO)]
return doCall1;
//[END_AFTER_CODE, 0, UPDATE(ims204000DO)]
```


- [DELETE]

```

//[BEGIN_PRE_ASSIGN_BLOCK, 0, INSERT(ims204000DO)]
{
    _dof.setEmpno(arg0.getEmpno());
}
{
    _dof.setName(arg0.getName());
}

//[END_PRE_ASSIGN_BLOCK, 0, INSERT(ims204000DO)]

_dof.setFullQuery(com.tmax.ims204000.dof.ims204000DOF.FULLQUERY.INSERT_DEFAULT_0);
resultCnt1 = _dof.add(doCall1, true);

//[BEGIN_POST_ASSIGN_BLOCK, 0, INSERT(ims204000DO)]

//[END_POST_ASSIGN_BLOCK, 0, INSERT(ims204000DO)]

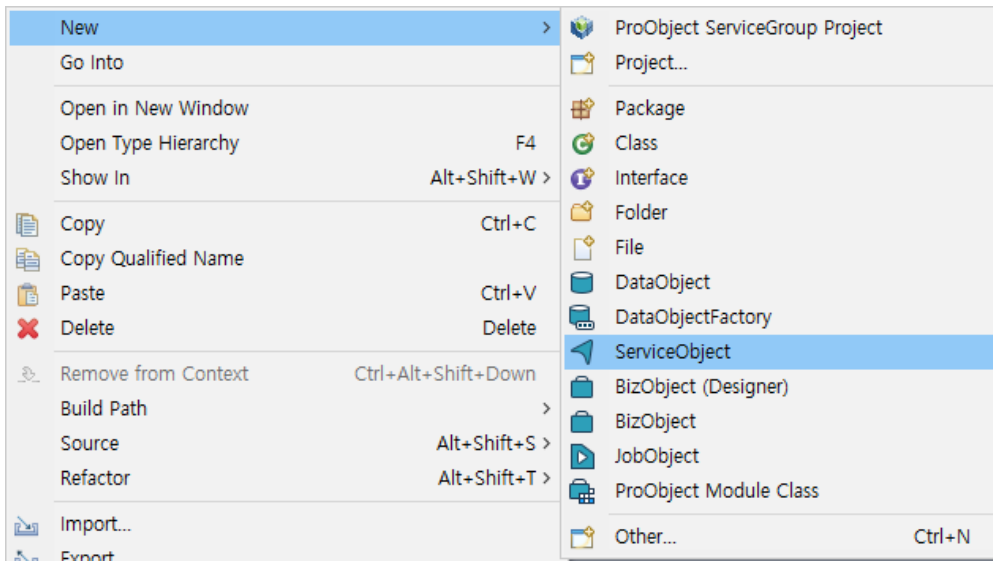
//[BEGIN_AFTER_CODE, 0, INSERT(ims204000DO)]
return doCall1;
//[END_AFTER_CODE, 0, INSERT(ims204000DO)]

```

A.4. ServiceObject(SO) 생성

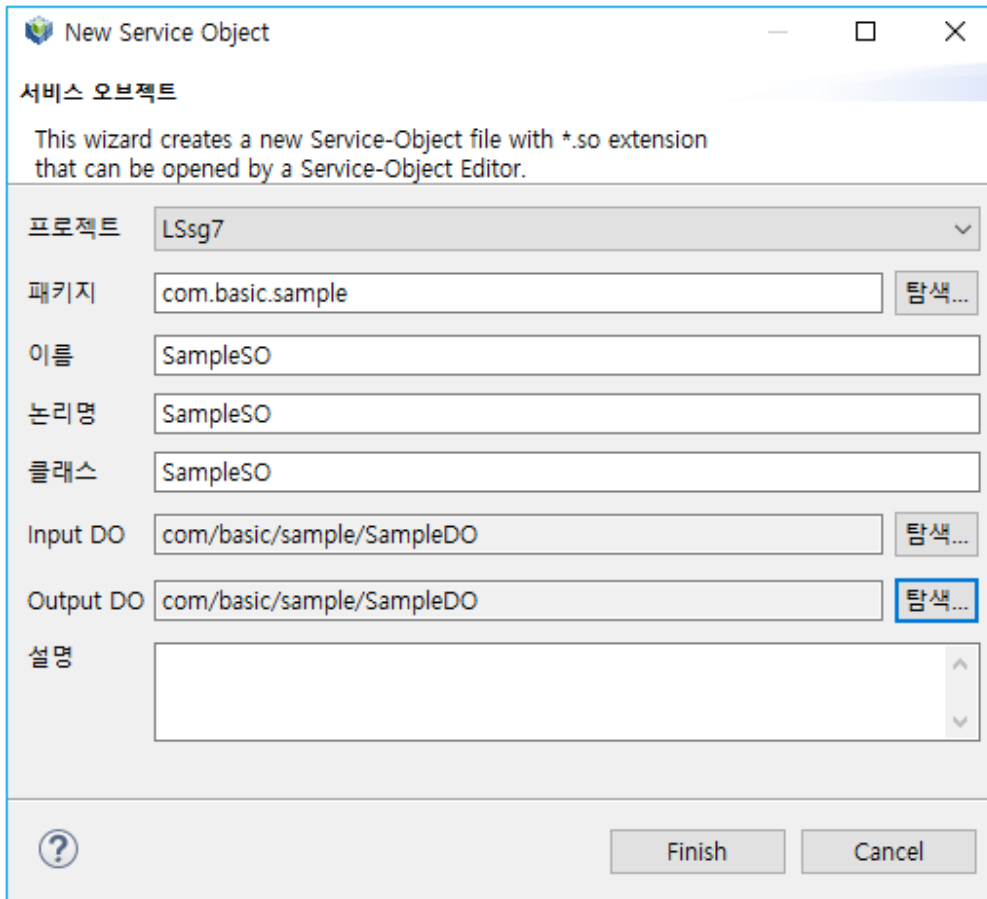
다음은 ServiceObject(SO)를 생성하는 과정에 대한 설명이다.

1. Package Explorer의 컨텍스트 메뉴에서 [New] > [ServiceObject]를 선택한다.




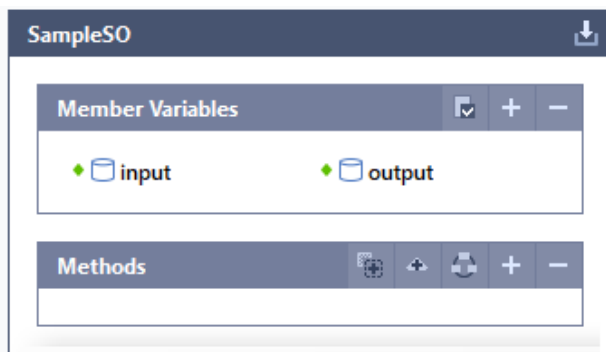
ServiceObject 생성

2. SO의 경우 다른 메타와는 달리 in/output DO를 등록해야 한다.



New ServiceObject 화면

3. SO를 생성한 후  을 클릭해서 메소드를 추가한다.



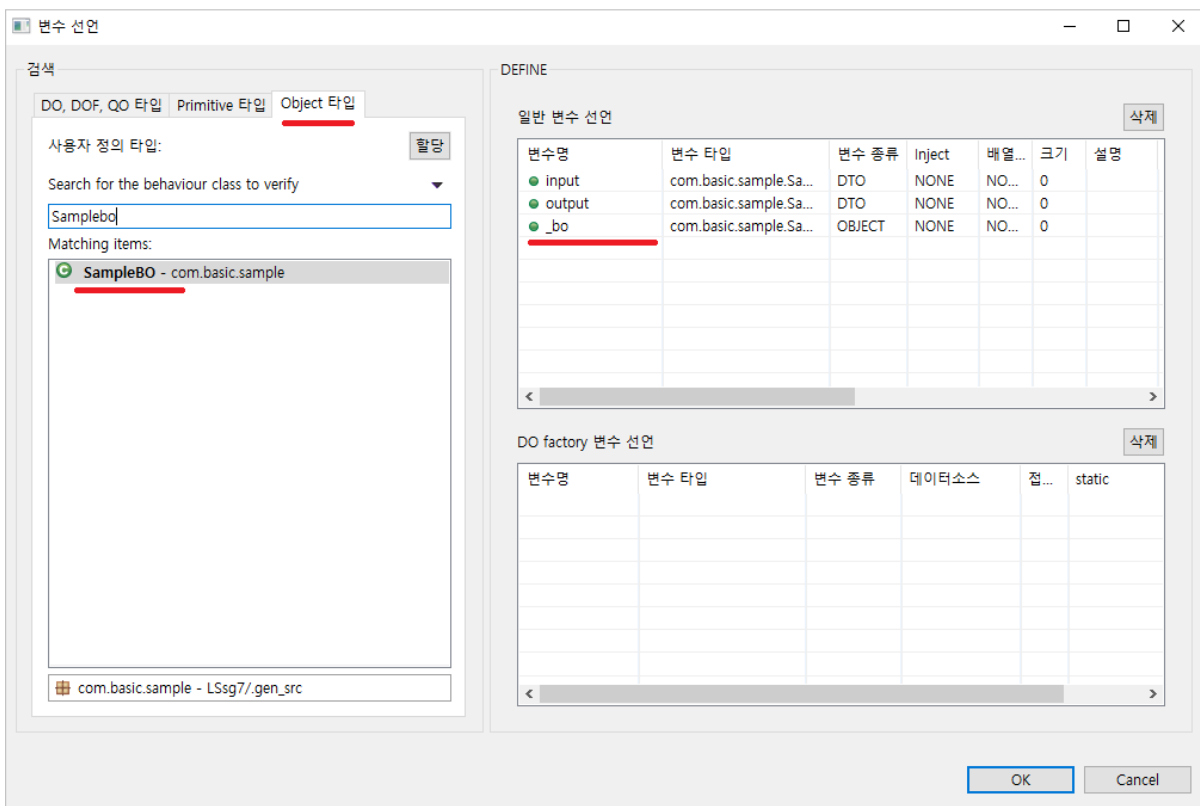
비구현 메소드 추가 화면 (1)

다음은 메소드를 추가한 결과 화면이다.



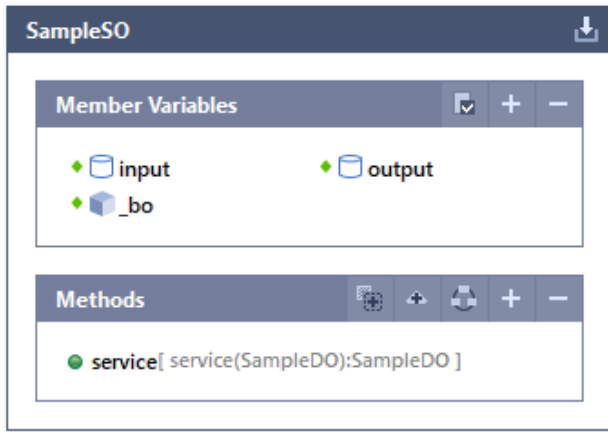
비구현 메소드 추가 화면 (2)

4. BO Member 변수를 추가하려면 **[+]**를 클릭한다. 다음의 화면에서 **[Object 타입]** 탭을 선택한 후 이전에 생성했던 BO를 검색한 후 추가한다.




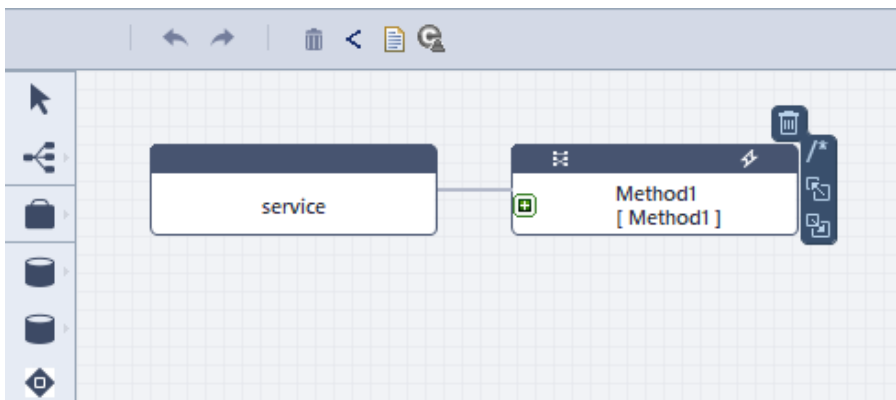
멤버 변수 추가 (1)

다음과 같이 변수가 추가된 것을 확인할 수 있다.




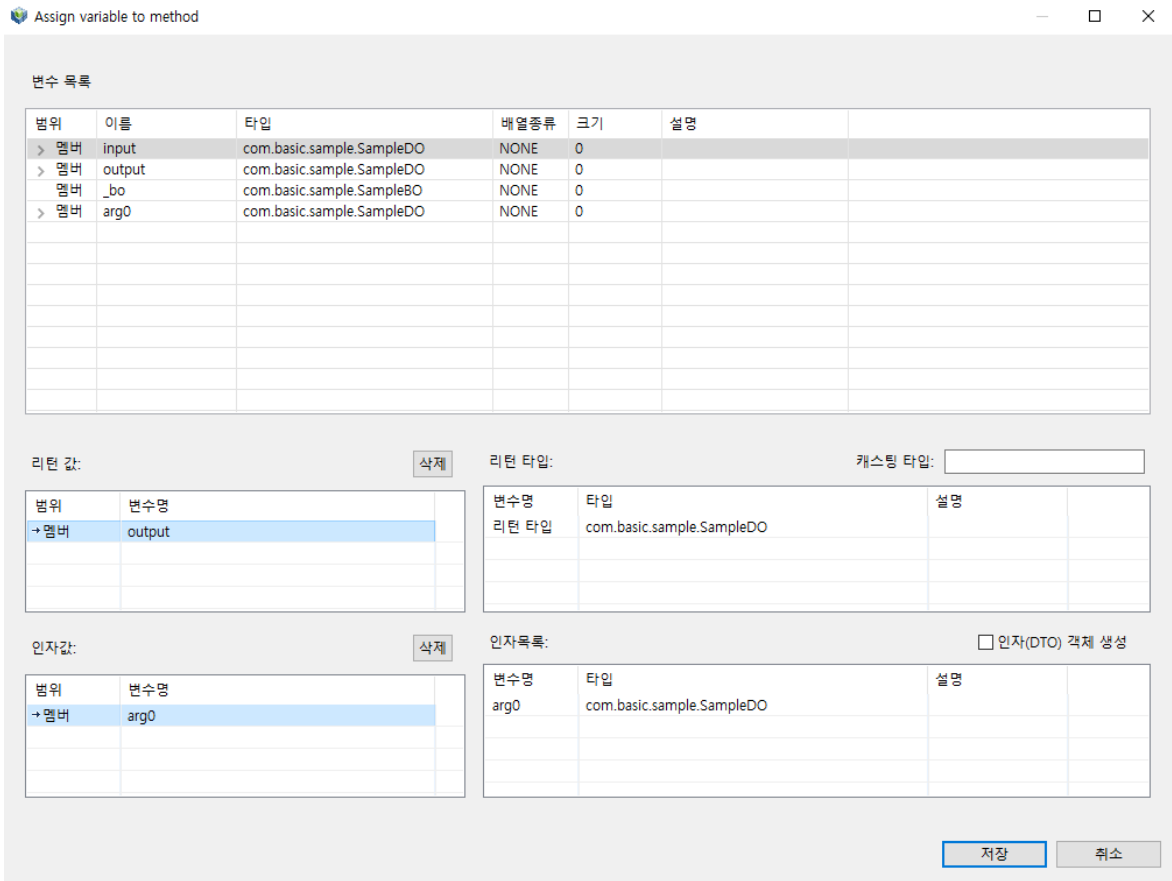
멤버 변수 추가 (2)

5. service 메소드를 더블클릭하여 디자인 화면으로 이동한다. 디자인 화면에서  을 클릭한 후 BO 모듈을 추가한다.



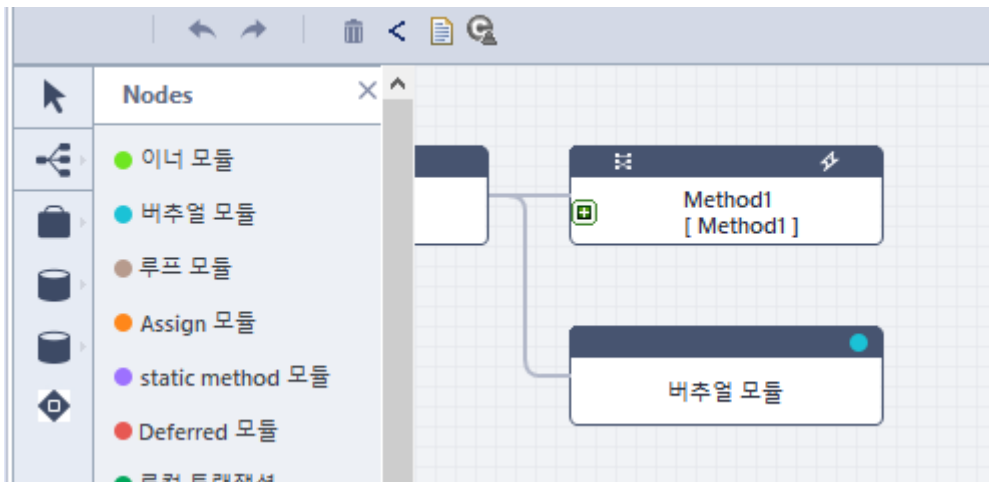
디자인 - BO 모듈 추가

입력 받은 값을 BO에 전달하고, BO에서 처리한 값을 다시 받아오게 해야 한다. 추가한 BO 모듈의  을 클릭하여 해당 작업을 진행할 수 있다.



리턴 값 - 인자값 추가

6. 마지막으로 버추얼 모듈을 추가하여, 반환 값을 작성한다.



버추얼 모듈 추가

```

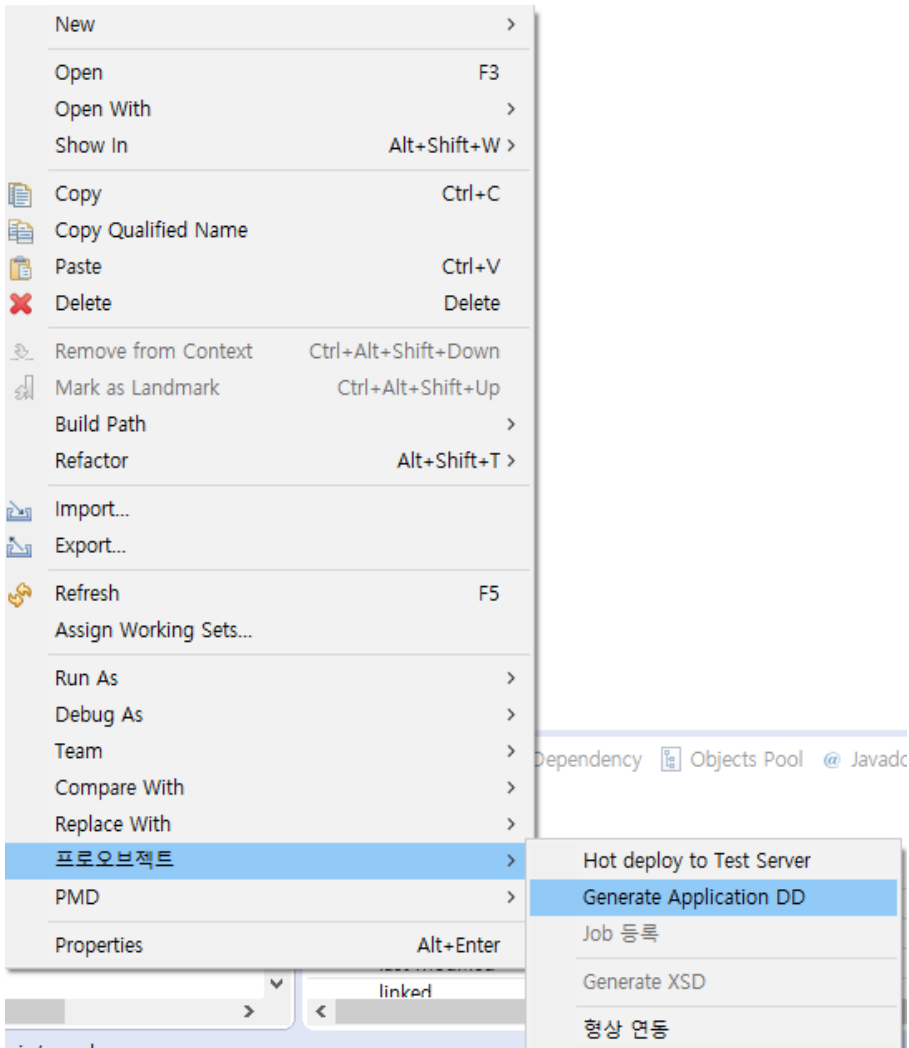
//[BEGIN_NODE_BLOCK, 1, service(SampleDO)]
{
//버추얼 모듈
//[BEGIN_VIRTUAL_CODE_BLOCK, 1, service(SampleDO)]
{
return output;
}
//[END_VIRTUAL_CODE_BLOCK, 1, service(SampleDO)]
}
//[END_NODE_BLOCK, 1, service(SampleDO)]
}
//[END_NODE_BLOCK, , service(SampleDO)]

```

버추얼 모듈 반환 값 추가

A.5. 서비스 등록(Generate Application DD)

만든 서비스를 PO 런타임에서 돌리기 위해 서비스를 등록해야 한다. 등록할 SO를 선택한 후 컨텍스트 메뉴에서 **[프로오브젝트] > [Generate Application DD]**를 선택한다.



Generate Application DD

등록한 SO에 대한 정보는 [META-INF]의 servicegroup.xml에 반영된다.

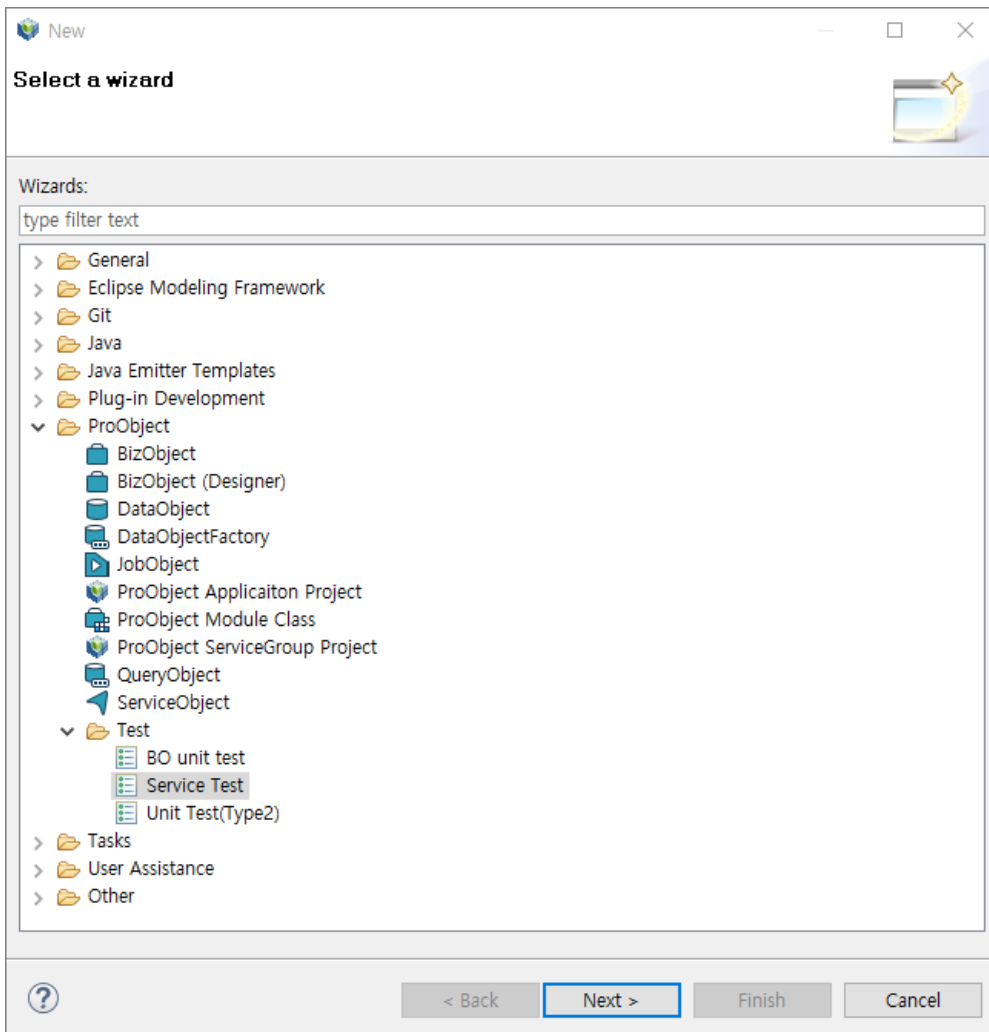
- ▼ LSSg7 [Name=LSSg7]
 - ▼ Services
 - FrismTestSO [Name=FrismTestSO, Class=com.tmax.so.FrismTestSO]
 - TestSO [Name=TestSO, Class=com.inicis.pg.dto.cm.TestSO]
 - TestSCSO [Name=TestSCSO, Class=com.tmax.sc.test.TestSCSO]
 - BigDecimalSO [Name=BigDecimalSO, Class=tmax.test.bigdecimal.BigDecimalSO]
 - SampleSO [Name=SampleSO, Class=com.basic.sample.SampleSO]
 - ▼ Batches
 - DevbuildTestIO [Name=DevbuildTestIO, Class=com.devbuild.test.DevbuildTestIO]

Generate Application DD - Servicegroup.xml 결과

A.6. 서비스 테스트

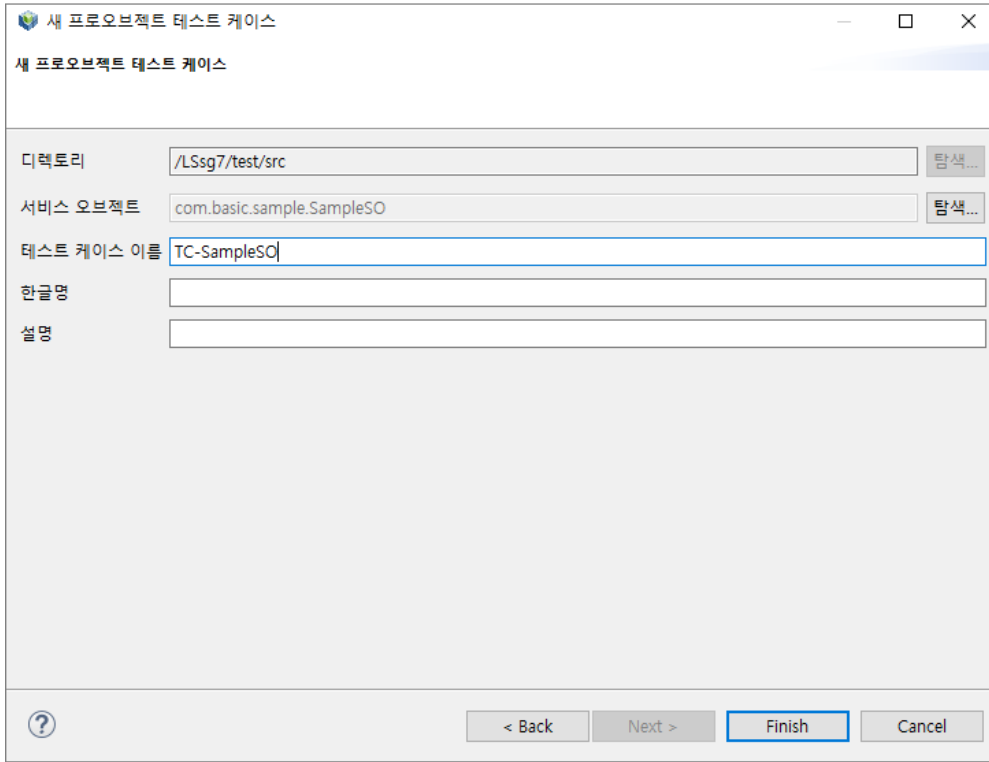
다음은 서비스 테스트 하는 방법에 대한 설명이다. 서비스 테스트는 스튜디오와 ProManager에서 진행할 수 있다(ProManager로 테스트하는 방법은 [HotDeploy](#) 및 [Service Test](#)를 참고한다).

1. 서비스 테스트를 진행해 주기 위해서는 테스트 케이스를 만든다. 업무 트리에서 **[Service Test]**를 선택한 후 **[Next]** 버튼을 클릭한다.



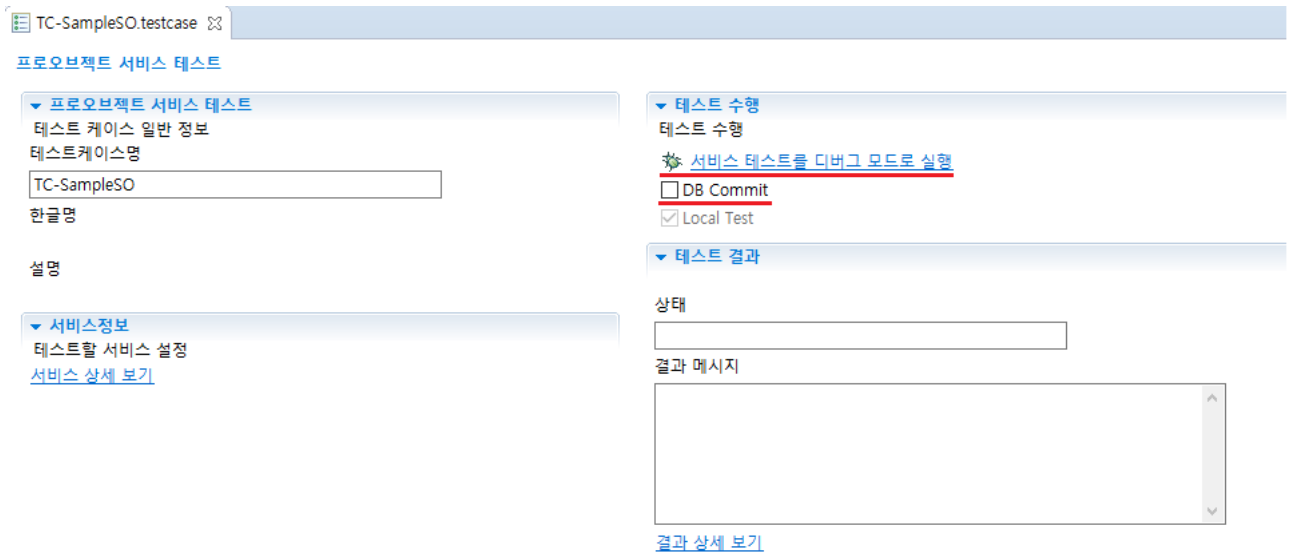
Service Test 생성 (1)

2. 새 프로오브젝트 테스트 케이스 화면에서 테스트할 서비스 오브젝트(SO)를 선택하고, 테스트 케이스 정보를 각각 작성한다.



Service Test 생성 (2)

3. 생성된 테스트 케이스는 **Pakckage Exploer**의 [test] > [src]에서 확인할 수 있다.



테스트 케이스 - 일반 화면

4. 서비스 테스트를 디버그 모드로 실행을 클릭하면 서비스 테스트가 실행되며, DB Commit을 체크하고 실행할 경우 실제로 DB에 값이 들어가는지 확인할 수 있다. 만약 입력할 데이터가 있다면 [테스트 데이터] 탭을 선택하면 다음과 같이 필드에 값을 넣고 테스트할 수 있는 화면으로 이동한다.


```
Git Repositories | Dependency | Objects Pool | Javadoc | Declaration | Problems | Console | Properties | Search | History | Debug | Git Staging
<terminated> ProObject Test Suite [Java Application] D:\OpenDK\Openjdk\java-1.8.0-openjdk-1.8.0.191-1.b12.ojdkbuild.windows.x86_64\jre\bin\javaw.exe (2019. 8. 27. 오전 11:02:14)
java.vendor.url.bug: http://bugreport.sun.com/bugreport/
sun.io.unicode.encoding: UnicodeLittle
sun.cpu.endian: little
sun.desktop: windows
sun.cpu.isalist: amd64
ProObject Local Debugging started.
Loaded Class > class com.basic.sample.SampleDO
Loaded Class > class com.basic.sample.SampleDO
Test input: empno : 7369 1
ename :

[2019.08.27 11:02:17] [Thread : [TEST-THREAD-0]] [INFO] «LSapp7.LSsg7.SampleSO» is ready for execution : Method - service, WaitObject - null
[2019.08.27 11:02:17] [Thread : [TEST-THREAD-0]] [OFF] [DATAOBJECT_FACTORY] selectStatement query :
/* com.basic.sample.SampleDOF */ 2
SELECT  A.EMPNO,
A.ENAME
FROM
EMP  A
WHERE
A.EMPNO = 7369
[2019.08.27 11:02:18] [Thread : [TEST-THREAD-0]] [INFO] «LSapp7.LSsg7.SampleSO» done for executing : Method - service
[2019.08.27 11:02:18] [Thread : [TEST-THREAD-0]] [INFO] «LSapp7.LSsg7.SampleSO» is done!
Assertion processing...
ProObject Local Debugger transaction rollback
[2019.08.27 11:02:18] [Thread : [TEST-THREAD-0]] [SEVERE] [DBSession] session closed
ProObject Local Debugging ended.
```

테스트 케이스 - 테스트 결과

1의 경우는 [테스트 데이터] 탭에서 입력값에 대한 내용이며, 2의 경우 DOF에서 설정한 쿼리가 입력 받은 값을 반영한 결과이다.