

**MOZART WISE**

# MOZART WISE

## MOZART WISE ?

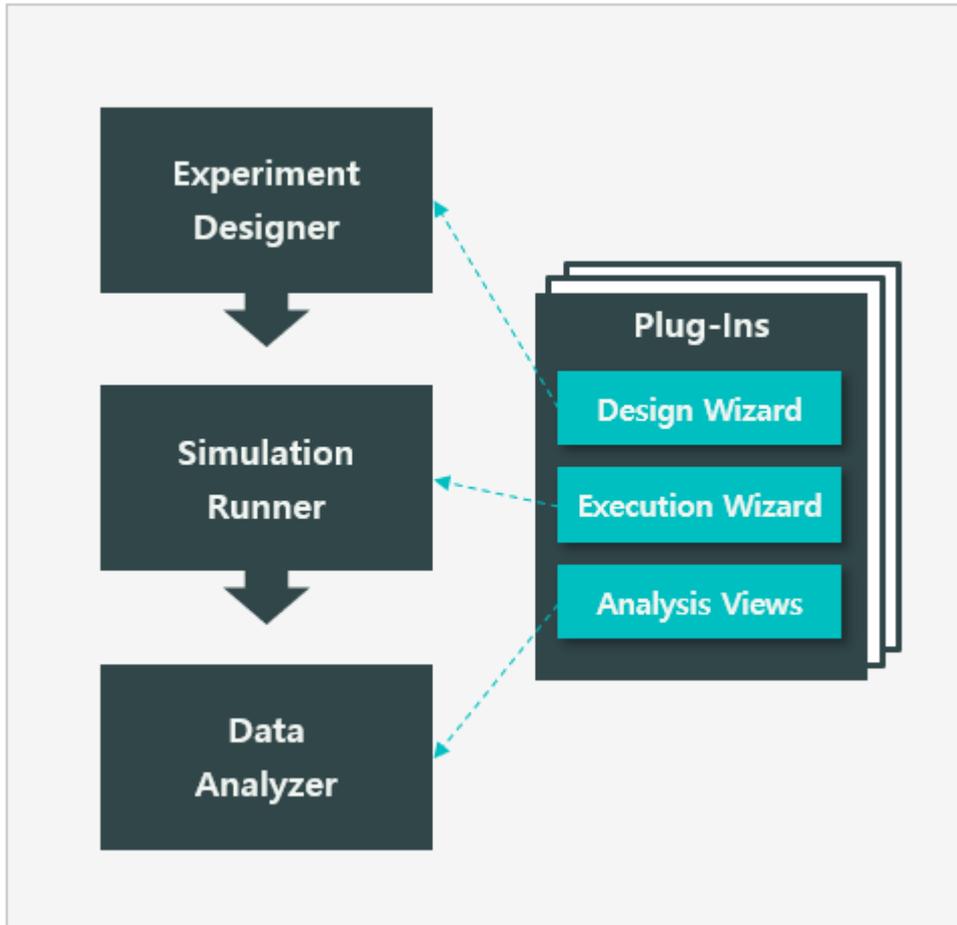
MOZART WISE (**What-If Simulation Environment**)는 MOZART 로 개발된 실행모델을 기반으로 What If 시뮬레이션을 계획, 실행, 분석할 수 있도록 지원하는 통합 도구입니다. WISE 를 사용하여 다음의 작업들을 쉽게 할 수 있습니다.

- MOZART 시뮬레이션 모델의 Input 을 변경하여 쉽게 What-If 시나리오를 만들고 실행할 수 있습니다. (What-If 계획 생성 및 실행)
- 장비고장/PM계획 시의 공장 영향평가, Demand 변경에 따른 납기, Bottle Neck 지점의 변화 예측 등 다양한 What-If 분석이 가능합니다. (What-If 분석)
- 머신러닝 및 다양한 휴리스틱 방식들을 적용하여 최적의 계획 및 스케줄을 생성할 수 있는 시뮬레이션 입력인자의 값을 찾을 수 있습니다. (시뮬레이션 최적화)

---

## WISE Client 의 구성

WISE Client 는 기본적으로 **Experiment Designer, Simulation Runner, Data Analyzer** 의 세가지 모듈로 구성됩니다. 그리고 주요 활용 시나리오들을 지원할 수 있는 여러가지 **플러그인들** 을 제공하여 기능을 확장할 수 있도록 구성되어 있습니다.

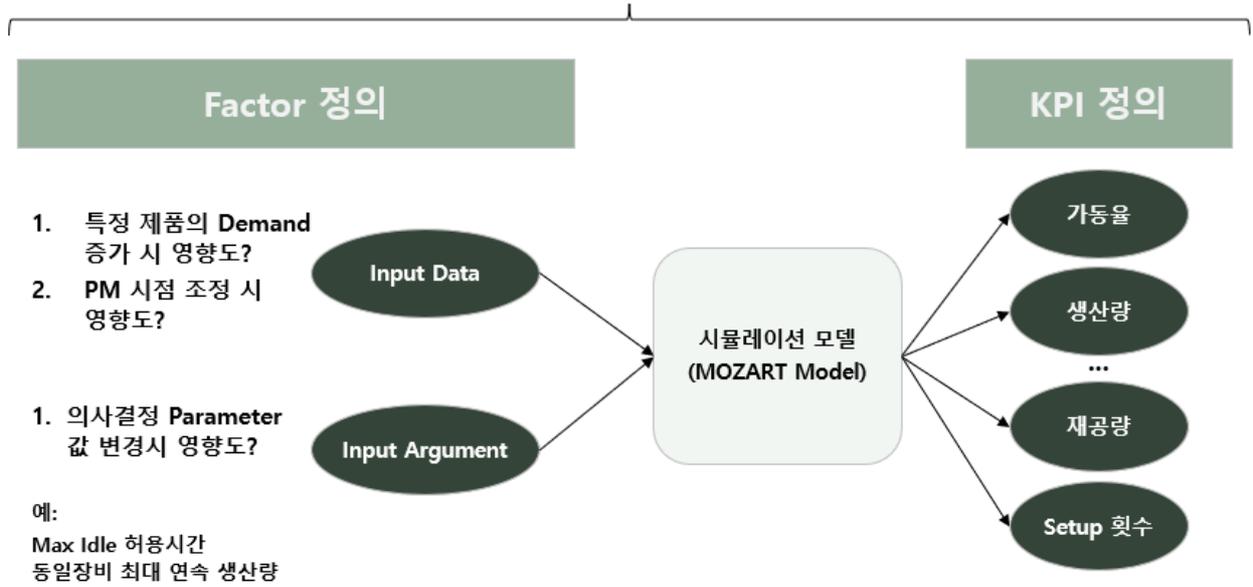


WISE 클라이언트 구성요소

## Experiment Designer

Experiment Designer에서는 Base Model 구동 시 관심있는 독립변수(Factor) 들과 수준 (Factor Value) 들을 정의하여 실험을 계획하고 결과평가를 위한 종속변수(KPI) 들을 정의하는 작업을 합니다.

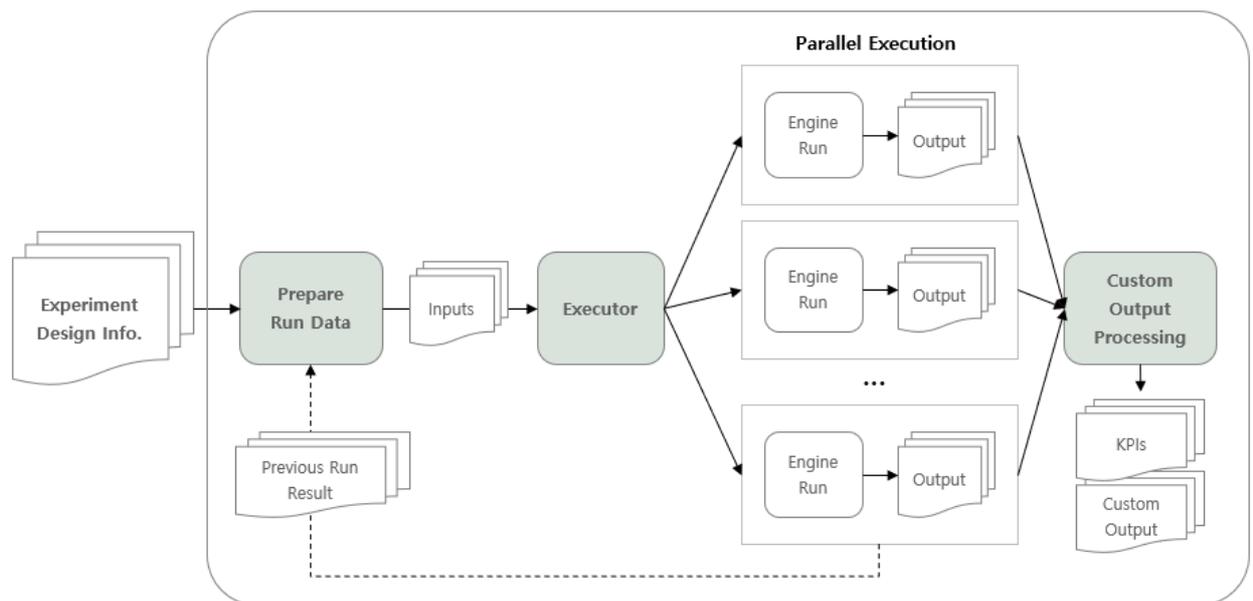
# Designer 기능



## Experiment Designer 개념

### Simulation Runner

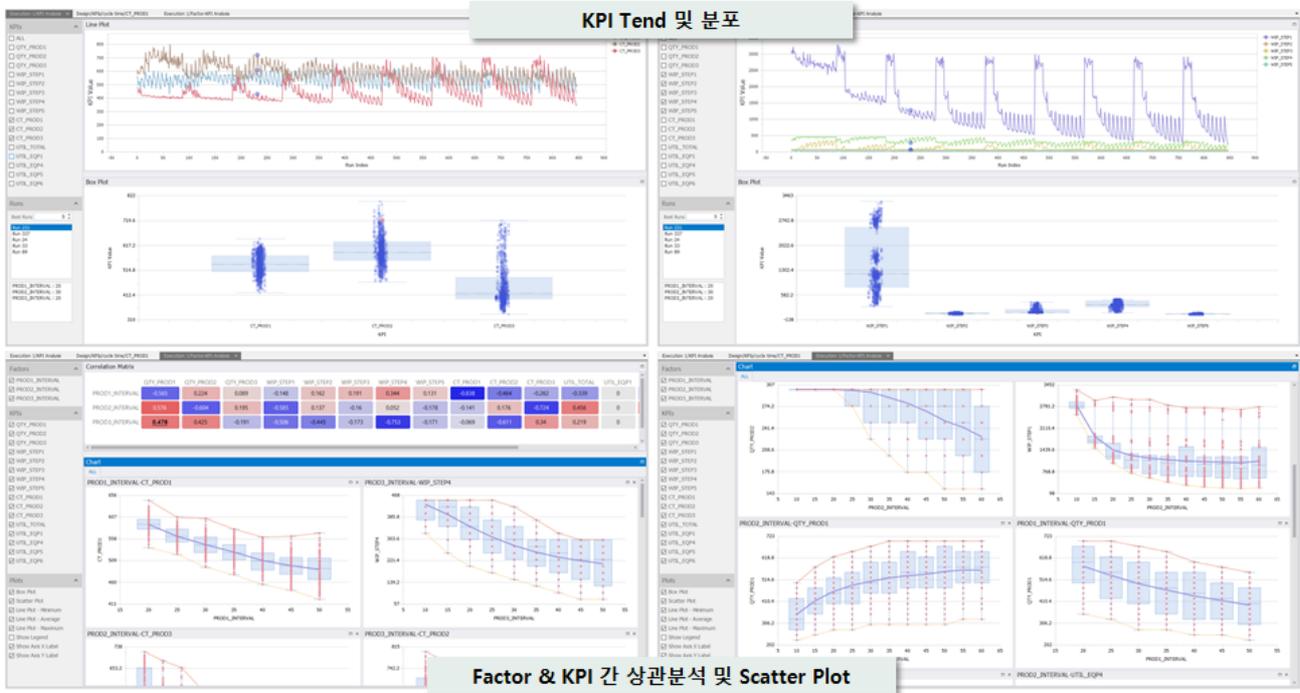
Designer 에서 정의한 Factor 들의 조합에 따라 실행할 Run 들의 Input 을 생성하고 서버자원의 활용이 가능한 범위에서 병렬로 모델을 실행합니다. 모델이 실행된 후 실험에서 정의한 KPI 들 사용자 정의 Output 을 생성하여 결과 분석을 가능하게 합니다.



## Simulation Runner 기능 구조

# Data Analyzer

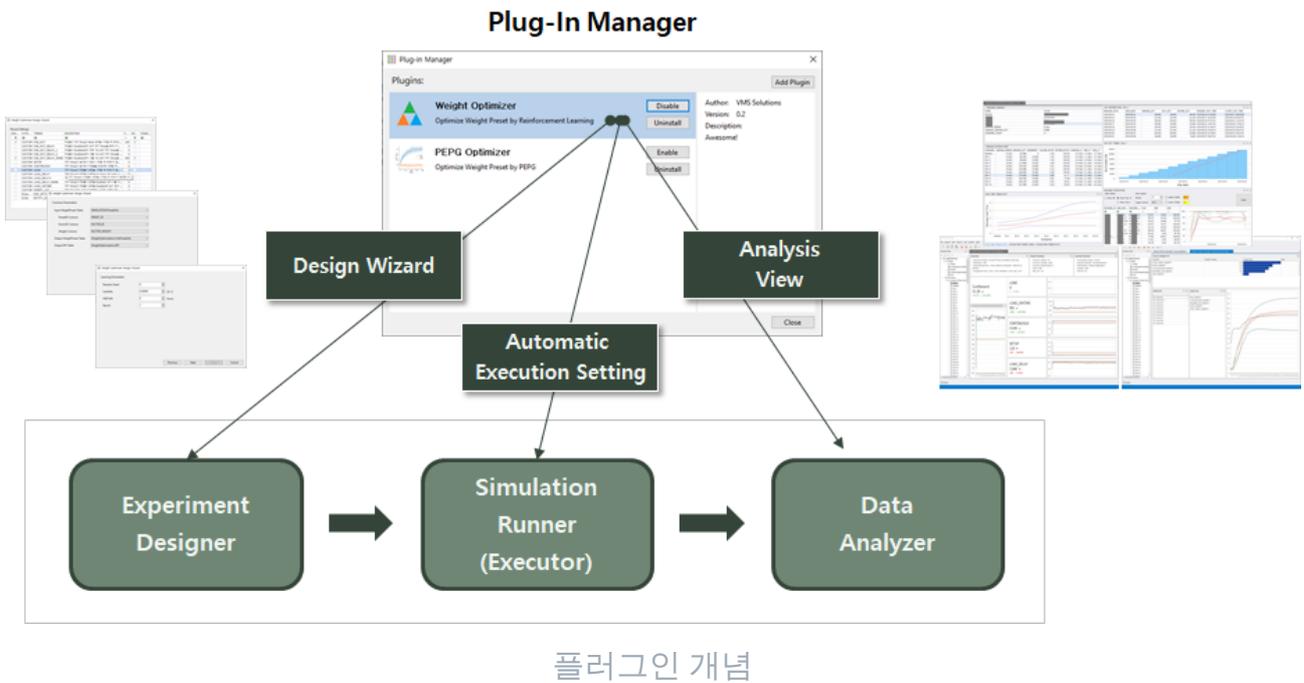
기본적인 Output 을 사용하여 Factor 와 KPI의 상관관계와 추세, Run 별 KPI 의 변화 등을 쉽게 확인 할 수 있는 분석 뷰를 제공합니다.



분석 View 예시

# Plug-Ins

WISE 의 기본적인 Designer, Runner 기능을 활용하여 특수 목적의 What-If 분석 및 Optimizer 모듈을 Plug-In 형태로 제공하여 사용자가 손쉽게 다양한 분석을 수행할 수 있도록 지원합니다.



## WISE 활용 방안

WISE 는 MOZART 로 만든 스케줄, 플래닝 엔진을 사용하여 시나리오를 확장합니다. 스케줄 모델은 생산계획, 스케줄 수립 관점에서 가상의 제조회장(Digital Twin)이 됩니다. 따라서 이러한 가상의 제조회장을 통해 생산 조건이 변경됨에 따라 생산지표변화를 확인할 수 있으며, 더 나아가 최적의 생산지표를 만들기 위해 스케줄 디스패칭 룰의 가중치를 최적화하는 시뮬레이션 최적화 기법을 생산운영에 적용할 수 있습니다.

## 생산조건 변경의 영향도 분석

- 생산조건을 정의할 수 있는 다양한 Simulation Input 데이터를 Factor 들로 정의(장비정보, 제품별 Demand, PM 계획 등)
- Factor 별 관심있는 설정 값을 데이터 테이블 혹은 Expression 형태로 입력하여 실험계획 생성
- 생산조건 변경의 영향도를 평가할 수 있는 수치를 KPI로 정의 (가동율, 재공수준, 제품별 생산량, 제품별 Lead Time 등)
- 실험계획의 인자 별 수준에 따른 KPI 결과 분석 가능



What if 시뮬레이션 활용절차

### 활용 예

- 장비 PM 에 따른 납기지연 Risk 검토
- 장비 PM 시 재공분포변화 예측
- 장비 PM 시점 별 영향도 분석을 통해 PM 시기 결정
- 특정 제품의 Demand 증가, 감소 시 납기 충족 여부, 장비 가동 예측
- 디스패칭 Rule 의 추가/변경 시 효과 평가

### Demo 동영상 보기

## 디스패칭 룰 가중치 최적화(Simulation Optimization)



Weight Optimizer 적용절차 및 방안

- 다중 디스패칭 Factor 의 가중합(Weighted Sum) 방식을 사용하는 스케줄러에 적용가능
- Wizard의 최소 설정을 통해 기존 스케줄러의 디스패칭 룰 가중치의 최적 설정값 도출
- 학습 결과검토 후 주기적인 학습을 통해 변동되는 생산현황에 따른 최적의 가중치를 찾고 현장에 적용할 수 있도록 자동화 가능

## 활용 예

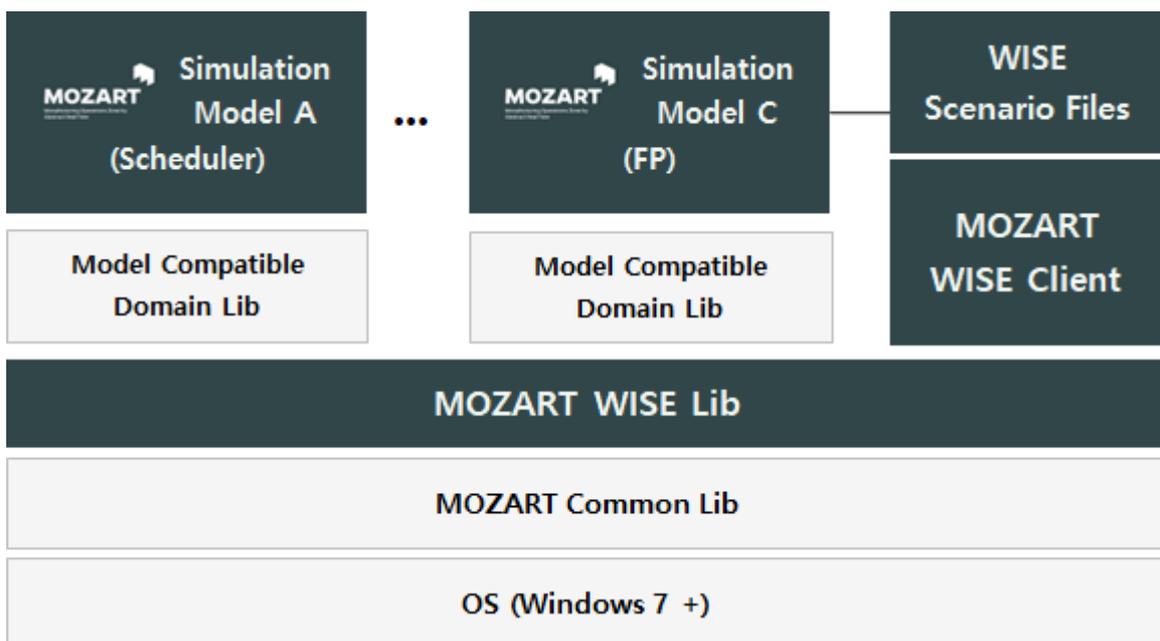
- 스케줄러 초기 셋업 시 디스패칭 룰의 효과 분석 및 초기 가중치 설정
- 스케줄러에 적용된 디스패칭 룰 가중치 자동 설정
- 신규 디스패칭 Factor 개발 시 가중치 설정 및 효과 평가

## Demo 동영상 보기

# 시스템 구성

## WISE Client

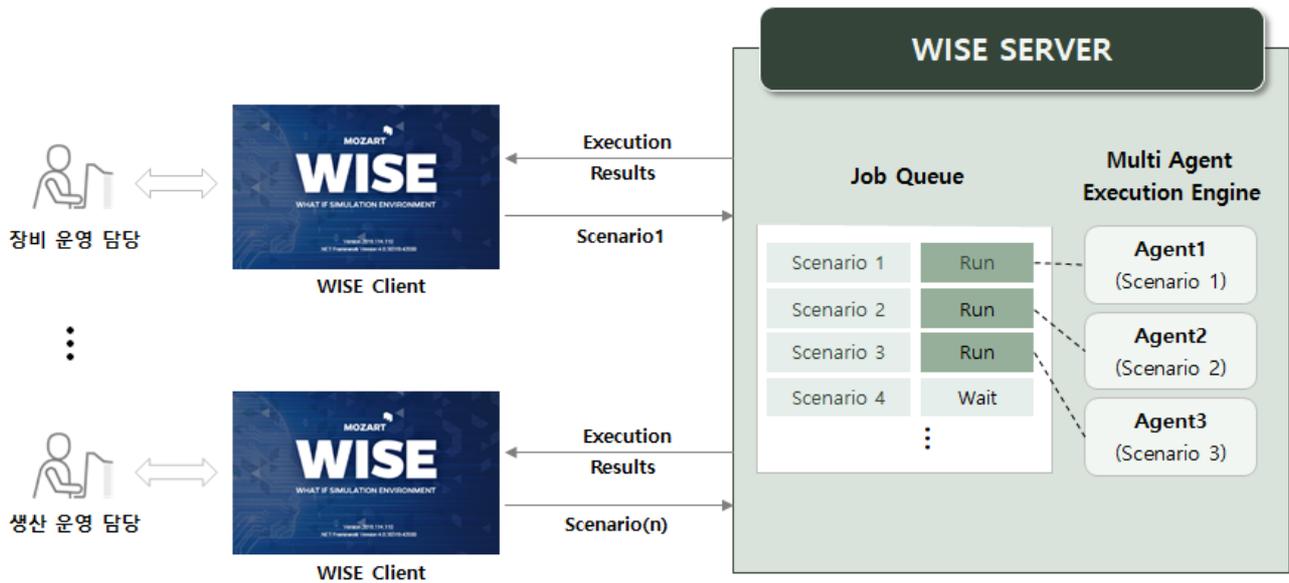
MOZART WISE 는 클라이언트 프로그램으로 제공되며, 독립적으로 구동 가능합니다. 또한 MOZART 서버 환경에서도 구동이 가능합니다. WISE 는 다양한 버전의 MOZART 모델을 독립적으로 실행할 수 있습니다.



WISE 클라이언트 시스템 구성

## WISE Server

분석가는 WISE 클라이언트에서 시나리오를 만들고 서버에 실행 요청을 하면, 서버에서 실행한 결과를 클라이언트로 다운받을 수 있습니다. 로컬환경에서 하기 힘들거나 다양한 실험을 계획하고 실행해야 할 때 서버를 통해 배치 실행, 결과확인이 가능합니다.



WISE 서버 사용 이미지

# WISE 설치

## 설치파일 다운로드

홈페이지의 제품 다운로드 경로를 통해 제품 별 설치파일(.MDZ) 과 필요한 플러그인 설치파일(.vplugin 파일)을 모두 다운로드 받습니다. MOZART 116.3 버전부터 WISE 설치파일을 제공합니다. [여기](#)에서 다운로드 가능합니다.

## MOZART WISE 설치 및 실행

MOZART CLIENT INSTALLER 를 실행시키고 Install 기능으로 WISE.202x.xxx.x.x.MDZ 파일을 설치합니다. 파일을 설치하는 경우 LinqPad 드라이버도 함께 설치됩니다.

 MOZART WISE는 .NET Framework 4.6.1 또는 이후 버전이 설치된 PC에서 동작합니다.

설치된 파일은 아래 경로에서 확인 할 수 있습니다.

### MOZART WISE 설치경로

```
%ProgramFiles%\VMS\Mozart\Client\WISE
```

### LinqPad Driver 설치경로

```
%LocalAppData%\LINQPad\Drivers\DataContext\4.6\Mozart.LinqPad.Driver (f77657
```

## WISE 플러그인 설치

WISE 플러그인은 WISE의 기능을 확장하거나 특정 목적의 시나리오를 쉽게 작성하고 실행할 수 있도록 WISE에 추가되는 기능 집합입니다. 아래의 총 4가지 플러그인들을 사용할 수 있으며, Plug-Ins 기능을 통해 추가/삭제하거나 활성화할 수 있습니다.

### Multi Model Executor 플러그인

WISE 시나리오를 복수의 모델, 복수의 실행엔진을 사용할 수 있도록 확장시켜주는 플러그인입니다.

### Weight Optimizer 플러그인

MOZART Simulation Module의 Weight Sum 방식 Preset을 사용하는 모델에 대해 Factor 가중치를 최적화하기 위해 Reinforcement Learning을 사용할 수 있도록 지원하는 Optimizer 플러그인입니다.

### AOWSA 플러그인

Advanced Optimal Weight Search Algorithm을 사용하여 Scheduler에 적용된 Preset Factor에 대한 최적 가중치를 찾아주는 Optimizer 플러그인입니다.

### PEPG 플러그인

MOZART 모델에 대해 Parameter Exploring Policy Gradient 방식의 Reinforcement Learning을 통해 최적의 파라미터를 찾는 Optimizer 플러그인입니다.

모든 플러그인은 아래와 같은 절차로 설치할 수 있습니다. 플러그인이 정상적으로 설치되면 [Plug-Ins] 메뉴하단에 플러그인별 메뉴가 활성화됩니다.

1. WISE를 실행합니다.
2. [Plug-Ins]>[Plug-In Manager]를 실행합니다.
3. 우측 상단의 "Add Plugin" 버튼을 통해 설치하고자 하는 .vplugin 파일을 선택합니다.
4. 설치된 플러그인 리스트에 대상 플러그인이 표시된 것을 확인합니다.
5. 플러그인의 "Enable" 버튼을 통해 사용하고자 하는 플러그인을 활성화합니다.

[Plug-Ins]>[Plug-In Manager]를 사용하여 설치된 플러그인 리스트를 조회하고, 비활성화, 삭제  
제가 가능합니다. 플러그인의 설치 경로는 아래와 같습니다.

```
%AppData%\Mozart\MOZART WISE\Plugins
```

# WISE 기능

→ WISE Scenario

/wise-1/wise-1

→ Experiment Designer

/wise-1/experiment-designer

→ Simulation Runner

/wise-1/simulation-runner

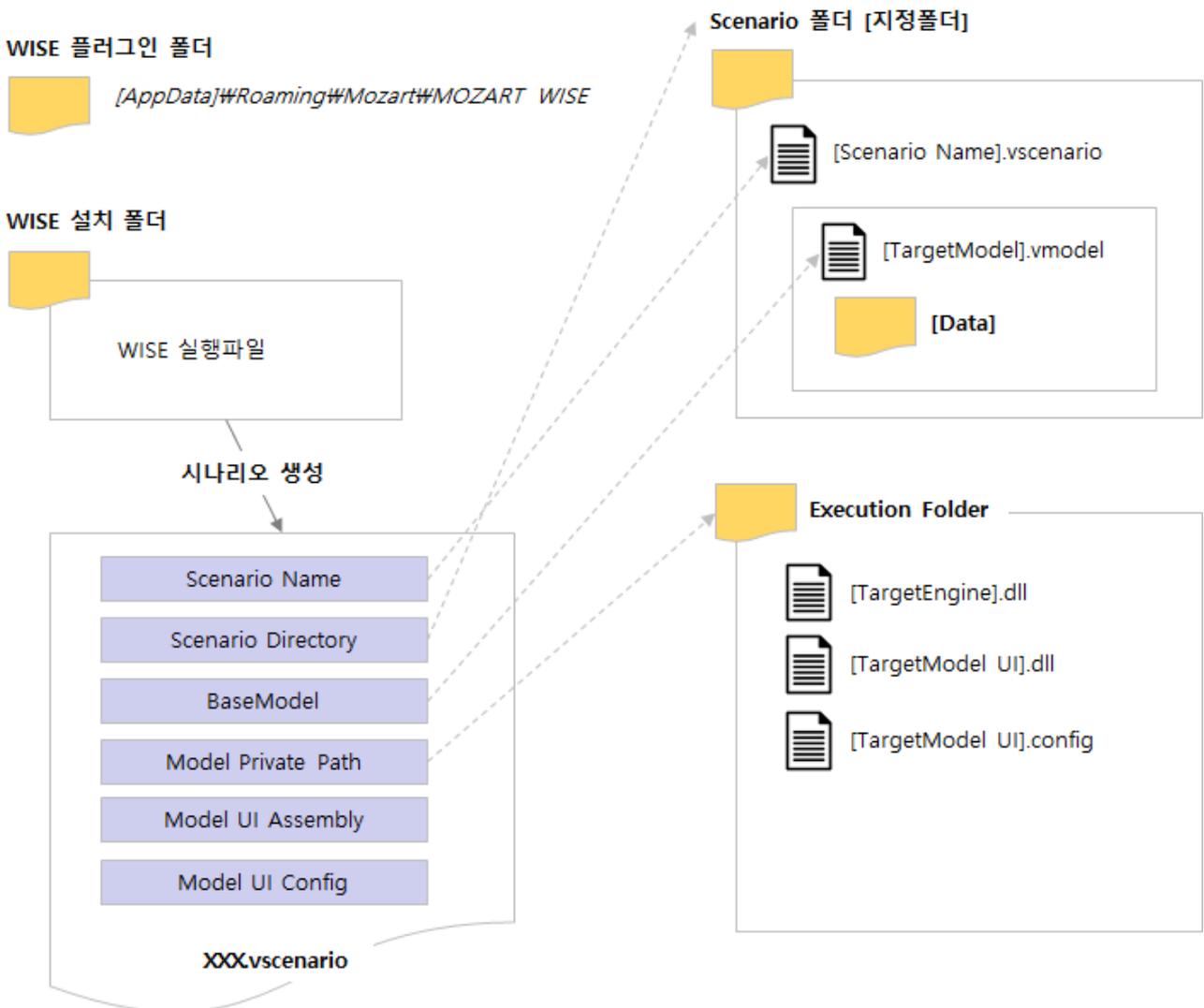
→ Data Analyzer

/wise-1/data-analyzer

# WISE Scenario

## 개요

시나리오(Scenario)는 WISE에서 프로젝트를 구성하는 단위입니다. 시나리오는 Scenario Directory, Base Model, Model Private Path, Model UI Assembly, Model UI Config 등의 경로 및 파일 정보를 저장합니다. 아래 그림은 WISE 설치경로 및 시나리오, 실행폴더의 관계를 표시합니다.



WISE 관련 폴더 구조

## 시나리오 생성 방법

WISE 의 [File]>[New Scenario] 메뉴를 사용하여 시나리오를 생성할 수 있습니다. 시나리오 설정화면에 아래와 같은 내용을 설정합니다.

### Scenario Name

시나리오 명칭입니다.

### Scenario Directory

시나리오 관련 파일이 저장되는 프로젝트 폴더입니다. 사용자가 임의로 지정하여 설정합니다. 시나리오 생성시 시나리오 파일과 Base Model, Base Model 의 기본 실행 데이터가 저됩니다.

### Base Model

시뮬레이션을 수행할 대상 모델입니다. 시나리오를 통해 실험계획을 할 때 Base Model 의 Input 데이터와 Argument 를 대상으로 Factor 를 정의할 수 있습니다. 시나리오 생성시점에 지정된 Base Model 과 Base Model 위치에 포함된 Data 폴더를 시나리오 폴더로 복제합니다. 따라서 시나리오 생성 이후 Base Model 이 변경된 경우에는 시나리오 폴더의 Base Model 을 업데이트 해주면 변경내용이 시나리오에 자동 반영됩니다.

### Model Private Path

Base Model 의 구동을 위한 DLL 파일이 위치한 경로입니다. 폴더내 Base Model 실행 DLL 이 구동되기 위해 필요한 참조 DLL 전체가 있어야만 시나리오가 정상적으로 구동됩니다. 실행 옵션에 따라 본 폴더에 모델 실행을 위한 파일들이 추가적으로 필요합니다.

 실행옵션 중 [Execute Model in Independent Process] 와 [Execute Model Using Libraries in Private Path] 옵션을 적용하기 위해서는 아래의 세가지 파일들을 MOZART Studio 실행폴더로 부터 Model Private Path 에 복제해 두어야만 시나리오가 실행됩니다.

- MozartAgent.exe
- MozartAgent.exe.config
- Mozart.Task.Model.dll

## **Model UI Assembly**

Base Model 의 검증 UI DLL 파일명입니다. 지정된 명칭의 파일이 항상 Model Private Path 위치에 있어야 정상 구동됩니다.

## **Model UI Config**

Model UI 의 메뉴 파일명입니다. 지정된 명칭의 파일이 항상 Model Private Path 위치에 있어야 정상 구동됩니다.

# Experiment Designer

## Factor

Factor 는 시뮬레이션의 결과에 영향을 주는 시뮬레이션 입력항목 중 What-If 시뮬레이션에서 관심있는 항목을 정의하기 위해 사용하는 요소입니다. WISE 의 Scenario View 의 트리 메뉴에서 [Design>Factors] 노드 메뉴를 사용하여 Factor 와 실험할 각 수준(Factor Value)을 정의할 수 있습니다.

## Factor 등록 절차

Factor 는 반드시 Factor Group 폴더 하위에 추가할 수 있기 때문에, 먼저 Factor Group 을 추가한 후 Factor 를 추가합니다.

1. [Design]>[Factors] 노드에서 [Add] 버튼을 눌러 Factor 그룹을 추가합니다.
2. 추가된 Factor 그룹 노드에서 [Add] 버튼을 눌러 Factor 를 추가합니다.
3. Factor 의 평가 값을 지정할 수 있는 Expression 을 편집합니다. Expression 의 결과는 Scalar Value 를 반환하는 수식을 입력하거나 지정된 값을 입력해야 합니다. 이 수식의 값은 이후 Run 수행시점에 해당 Run 수행시 지정된 Factor 의 값으로 Result 에 기록됩니다. 이를 통해 Run 별 Factor 값과 KPI 의 값의 관계를 분석할 수 있습니다.

## Factor Value 정의

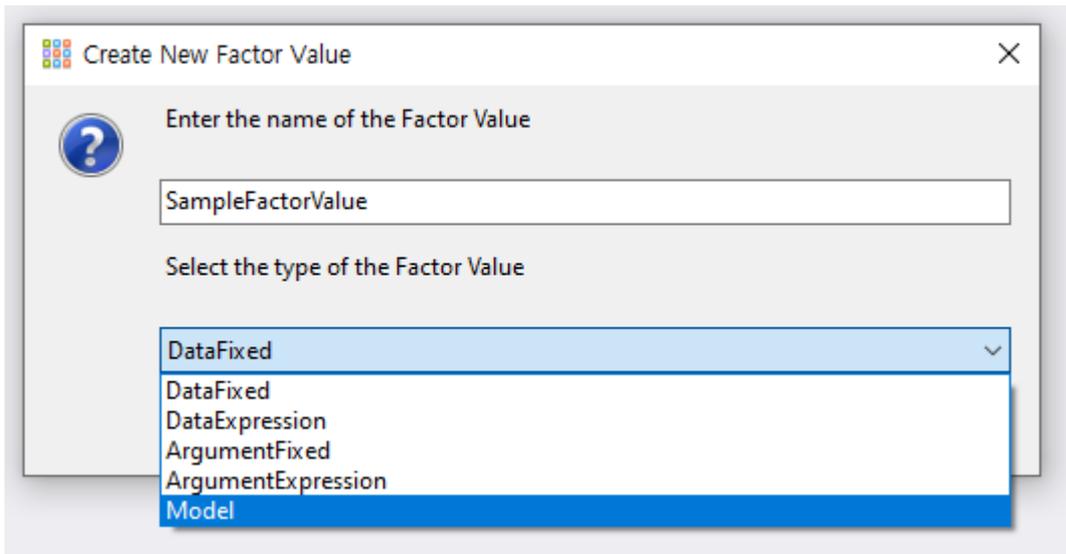
Factor Value 는 해당 Factor 에 대해 실험할 수준을 정의합니다. 한 개의 Factor 에 대해 여러 개의 Factor Value 를 등록할 수 있습니다. Factor Value 를 정의할 수 있는 대상은 Base Model 의 모든 Input 과 Input Argument 항목입니다. 즉, Factor Value 는 Base Model 에서 수정할 수 있는 정보 중 Base Model 과 다른 하나의 실행 케이스를 정의합니다.

예를 들어 Input Argument 에 `DefaultSetupTimeMinute` 이라는 변수가 정의되어 있다고 할 때 Factor 를 Setup Time 으로 정의한 후, Factor Value 에 `DefaultSetupTimeMinute` 의 값을 10, 15, 20, 25, 30 로 설정한 5개의 Factor Value 를 등록할 수 있습니다. 이 경우 등록된 Factor Value 가 하나의 실행케이스가 됩니다. Base Model 의 `DefaultSetupTimeMinute` 값이 5로 설정되어 있었다면 상기와 같은 실험계획은 Default Setup Time 의 변화함에 따른 관심 KPI 의 변화정도를 알아보기 위한 실험이 됩니다.

## Factor Value 등록 절차

Factor Value 를 등록할 수 있는 대상은 Key 값이 설정된 모든 Input Table 과 Input Argument 이며, 등록하는 방법은 Expression 을 사용하는 방법과 직접 값을 입력하는 두 가지 방법이 있습니다.

1) Factor 가 등록된 상태에서 Factor 를 선택한 후 [Add] 버튼으로 Factor Value 를 추가합니다.



Factor Value 등록 창

2) Factor Value 의 명칭을 입력하고, 아래의 5가지 유형 중 한가지를 선택합니다.

- **DataFixed** : Input Table 을 직접 사용자가 편집하여 정의합니다.
- **DataExpression** : 변경대상 Input Table 과 해당 Table 의 데이터를 변경하는 Expression 을 정의합니다.
- **ArgumentFixed** : Input Argument 들 중 특정 Argument 의 값을 편집합니다.
- **ArgumentExpression** : Input Argument 들 중 특정 Argument 의 값을 업데이트하는 Expression 을 정의하여 값을 정의합니다.
- **Model** : 지정된 모델의 Input Argument와 Input Data를 그대로 사용합니다.

하나의 Factor 에 서로 다른 유형의 Factor Value 를 추가할 수 있으며, 등록된 Factor Value 별로 대상 Factor 에 대한 하나의 테스트 케이스를 정의합니다. Factor Value 는 복수의 데이터를 변경하는 방식으로 정의할 수 있습니다. 예를 들어 ArgumentFixed 를 사용하는 경우 "A",

"B" 옵션을 한번에 변경하여 하나의 Factor Value 를 정의할 수 있습니다. 뿐만 아니라 하나의 Factor Value에 여러 입력 데이터 테이블을 수정하여 등록할 수 있습니다.

## 유형별 Factor Value 등록 절차 및 예

### DataFixed 기능

1. Factor Value 편집화면의 좌측 트리상단의 [Add] 명령을 실행합니다.
2. Input Table 선택창에서 변경대상 테이블을 선택합니다. Input Table 중 Primary Key 가 설정된 테이블 표시됩니다. 만일 선택을 위한 트리에 테이블이 표시되지 않는다면 Base Model 에 변경이 필요한 Input Table 에 Key를 설정해야 합니다.
3. 테이블이 선택되면 오른쪽 그리드에 Base Model 의 입력데이터가 로드됩니다.
4. 데이터 중 변경이 필요한 항목의 데이터를 수정합니다.
5. 수정을 한 항목은 행이 노란색으로 변경되어 수정된 데이터가 표시됩니다.
6. 수정된 값을 저장합니다.

The screenshot shows a software interface with a tree view on the left containing 'Inputs', 'BOP', and 'Product'. The main area displays a table with the following data:

LINE_ID	PRODUCT_ID	PROCESS_ID	INPUT_INTERVAL	INPUT_STDDEV
LINE01	PROD01	PROC01	100	20
LINE01	PROD02	PROC02	100	60
LINE01	PROD03	PROC03	100	10

Data Fixed 유형의 Factor Value 입력 예시

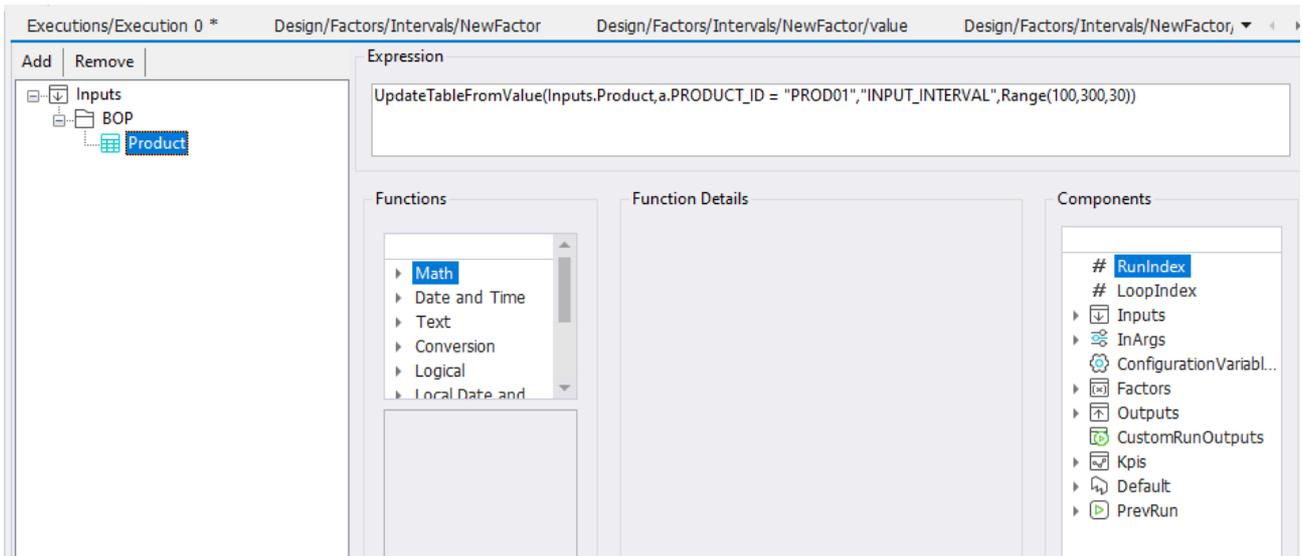
상기 예시에서는 모델의 제품(Product) 데이터 테이블에서 "PROD02" 번 제품의 INPUT\_STDDEV 컬럼 값을 60으로 변경한 테이블을 Factor Value 로 정의한 예시입니다.

동일한 방식으로 왼쪽 트리에 변경이 필요한 추가적인 Input Table 을 등록하고 값을 변경할 수 있습니다. 이 경우 해당 Factor Value 는 변경된 모든 정보로 정의되는 하나의 테스트 케이스가 됩니다. Fixed 유형의 경우 Factor Value 1개로 실행 케이스 1개만 생성 가능합니다.

### DataExpression 기능

1. Factor Value 편집화면의 좌측 트리상단의 [Add] 명령을 실행합니다.

2. Input table 선택창에서 변경대상 테이블을 선택합니다. Input table 중 primary key 가 설정된 table 표시됩니다. 만일 선택을 위한 트리에 값이 노드가 표시되지 않는다면 Base Model 에 변경이 필요한 Input Table 에 Key를 설정해야 합니다.
3. 테이블이 선택되면 오른쪽 화면에 Expression 을 편집할 수 있는 창이 열립니다.
4. Expression 편집을 통해 대상 Table 을 업데이트하는 수식을 정의합니다. Expression 편집기에 대한 자세한 사용법은 [여기](#)를 참조합니다.



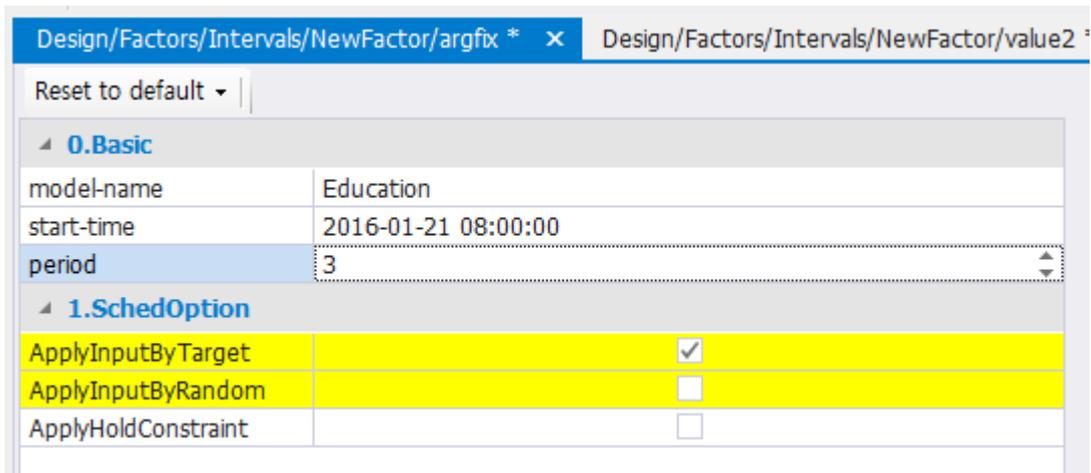
Data Expression 유형 Factor Value 정의 예시

상기 예제에서는 모델의 제품(Product) 데이터 테이블의 값을 함수를 통해 업데이트하는 예제입니다. UpdateTableFromValue 함수를 사용하여 Product 테이블의 제품 "PROD01" 의 INPUT\_INTERVAL 컬럼 값을 100부터 300까지 30 단위로 업데이트하는 수식입니다. 이러한 방식으로 정의하는 경우 하나의 Factor Value 를 사용하여 다수의 케이스를 만들 수 있습니다. 예제의 경우 총 11개의 케이스를 생성합니다. Expression 유형의 Factor Value 는 이와 같이 하나의 Factor Value 를 통해 다수의 테스트 케이스를 정의할 수 있습니다.

```
UpdateTableFromValue(Inputs.Product,a.PRODUCT_ID = "PROD01", "INPUT_INTERVAL"
```

## ArgumentFixed 기능

1. Input Argument 리스트가 표시된 입력 창에서 필요한 Argument 를 수정합니다.
2. 수정된 값은 노란색으로 변경여부가 표시됩니다.
3. 수정 후 저장합니다.

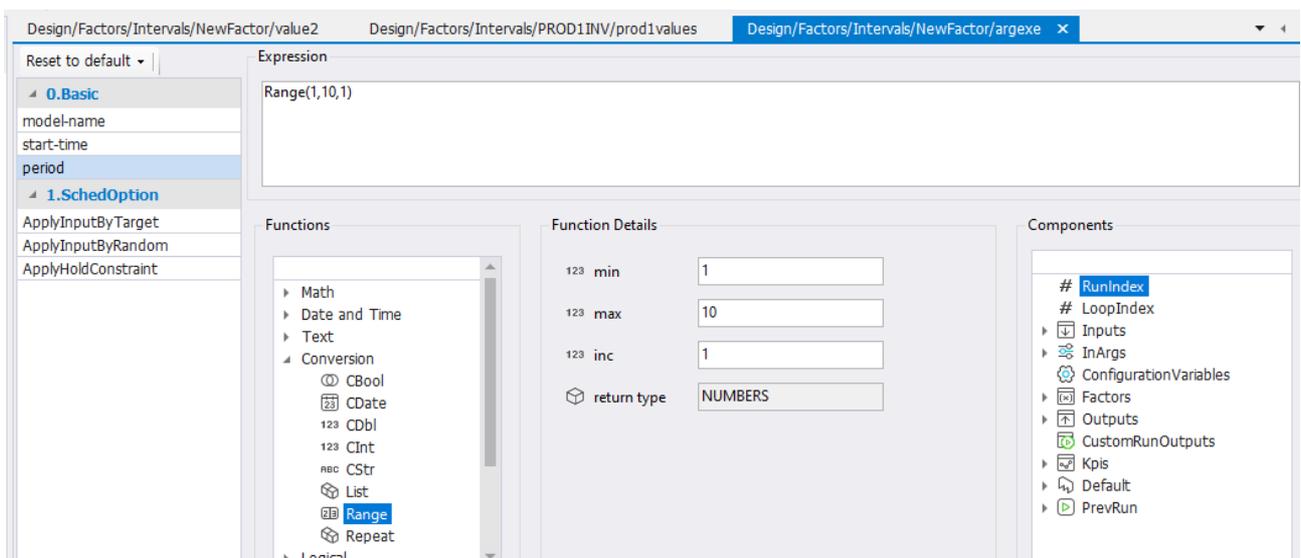


ArgumentFixed Factor Value 입력 예시

상기 예제는 Input Argument 중 두 개의 Argument 의 값을 변경한 것을 하나의 케이스로 등록한 예시입니다. 예제에서는 하나의 케이스를 정의할 때 두 개의 Argument 값을 변경하였습니다.

### ArgumentExpression 기능

1. Input Argument 리스트가 표시된 왼쪽 창에서 변경대상 Argument 를 선택합니다.
2. 오른쪽의 Expression 편집창에서 Argument 업데이트 수식을 편집합니다.
3. 수정 후 저장합니다.



ArgumentExpression Factor Value 입력 예시

위 예시는 Input Argument 중 Period (계획기간) 를 1일에서 10일까지 하루 씩 증가시키면서 모델을 실행시키도록 Factor Value 를 추가하는 예시입니다. KPI 를 수행시간이나 제품 출하

수량 등으로 설정하는 경우 계획기간 증가에 따른 수행시간의 관계 혹은 생산량의 관계 등을 테스트할 수 있습니다. 역시 Expression 형태의 경우 하나의 Factor Value 를 통해 실제 수행되는 다수의 케이스를 정의할 수 있습니다.

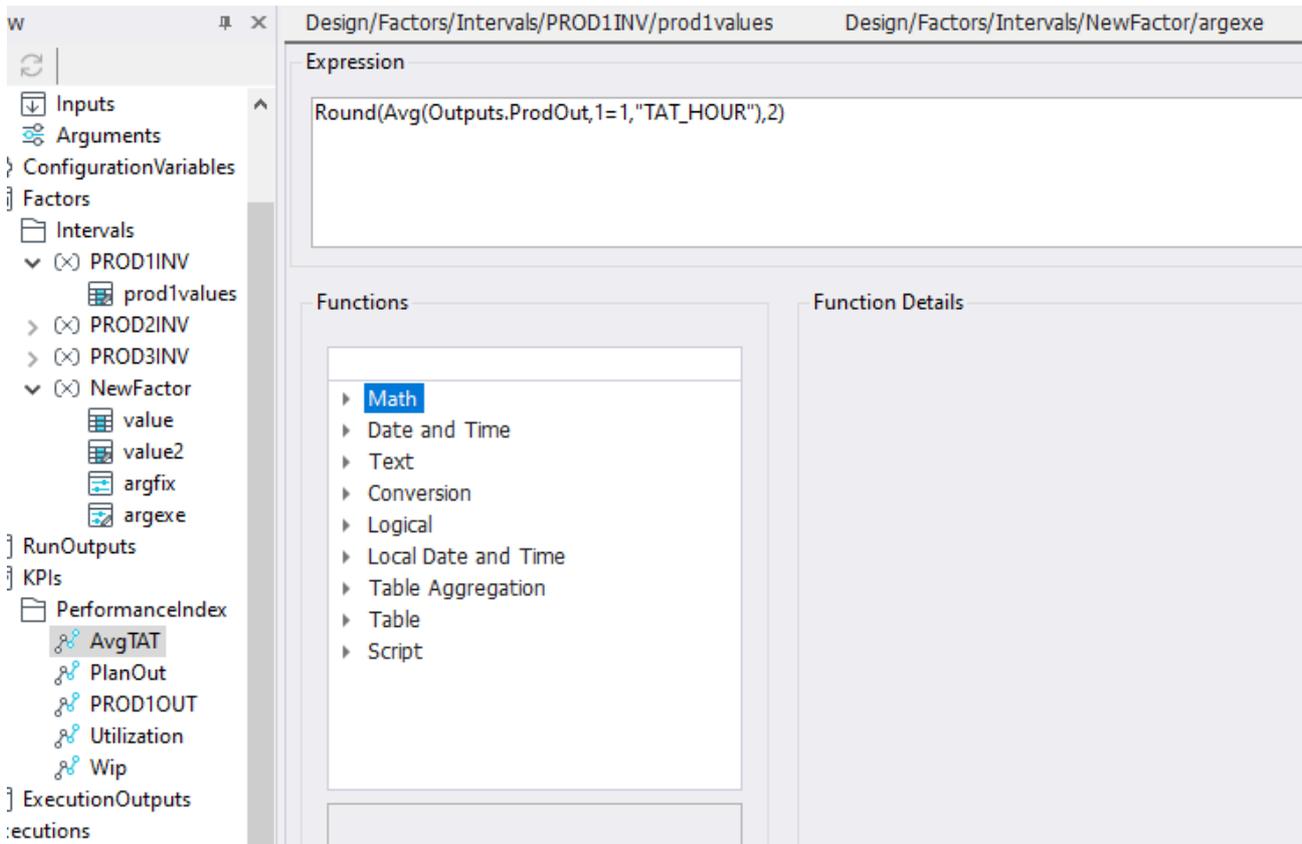
---

## KPI

KPI 는 시뮬레이션 결과 평가를 위한 Index 입니다. Factor 조합별로 모델이 실행되며, 각 Factor 조합에 따라 시뮬레이션 결과를 평가하여 목적에 따라 좋은 결과를 산출하는 Factor 조합을 찾거나 Factor 값과 KPI 의 관계를 확인하기 위해 관심있는 항목을 KPI 로 정의할 수 있습니다.

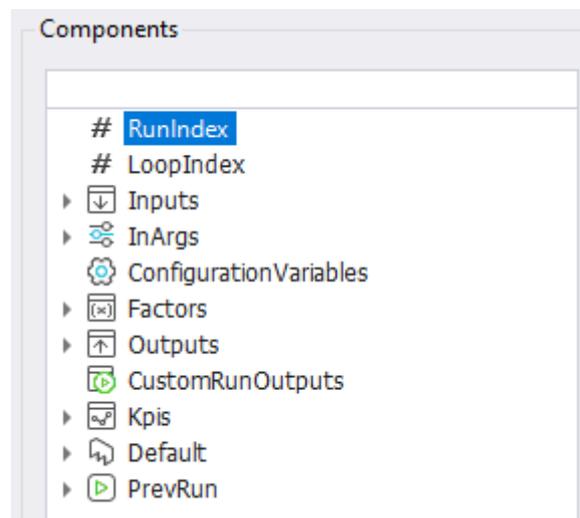
### KPI 등록 절차

1. [Design]>[KPIs] 노드에서 [Add] 버튼을 눌러 KPI 그룹을 추가합니다.
2. 추가된 KPI 그룹 노드에서 [Add] 버튼을 눌러 KPI 를 추가합니다.
3. KPI 명칭을 입력합니다.
4. KPI 값이 높을 수록 좋은 값인지, 낮을 수록 좋은 값인지를 선택합니다. KPI 관련 결과 조회시점에 KPI 소팅에 활용하기 위해 필요한 속성입니다.
5. KPI 항목이 추가된 후 KPI 값을 산출하기 위한 Expression 을 정의합니다. Expression 의 결과값은 Scalar Value 가 나오도록 작성해야 합니다.
6. Expression 작성 후 저장합니다.



KPI 작성 예시

상기 예시는 Model 의 Output 중 ProdOut 이라는 Output 테이블의 TAT\_HOUR 컬럼 값의 무작위 평균값을 산출하고 Round 처리한 값으로 KPI 를 정의한 예시입니다. KPI 를 정의하기 위해서는 위와 같이 Expression 을 사용하거나 Linq Script 를 사용할 수 있습니다. 또한 Expression 에서 사용될 수 있는 데이터는 Expression Editor 의 우측 Components 트리에 포함된 모든 데이터를 포함합니다. 각 요소에 대한 설명은 [Expression 편집기능](#)을 참조합니다.



Expression 편집시 사용할 수 있는 데이터 요소

# RunOutputs

RunOutput은 시뮬레이션 종료 후 결과 분석에 필요한 데이터를 미리 추출 및 가공해 놓는 모델 단위의 사용자 정의 아웃풋입니다. 각 모델의 실행이 완료될 때마다 RunOutput 데이터를 생성합니다. 하나의 RunOutput은 스키마와 스키마를 채우기 위한 Expression으로 구성되어 있습니다. 기록된 RunOutput은 개별 Run의 RunOutputs 노드에서 확인할 수 있습니다.

## RunOutput 등록 절차

1. [Design]>[RunOutputs] 노드에서 [Add] 버튼을 눌러 RunOutput 그룹을 추가합니다.
2. 추가된 RunOutput 그룹 노드에서 [Add] 버튼을 눌러 RunOutput을 추가합니다.
3. RunOutput 명칭을 입력합니다.
4. RunOutput 항목이 추가된 후 [Schema] 탭에서 추출 및 가공한 데이터를 저장할 스키마를 정의합니다.
5. [OnEndRun] 탭에서 스키마를 채우기 위한 Expression을 정의합니다. Expression의 결과 값은 DataTable 형식이어야 합니다.
6. Expression 작성 후 저장합니다.

Name	Data Type	Default Value	Allow Nulls	Is Primary Key
EQP_GROUP	string		<input type="checkbox"/>	<input checked="" type="checkbox"/>
AVG_BUSY	double		<input checked="" type="checkbox"/>	<input type="checkbox"/>
AVG_SETUP	double		<input checked="" type="checkbox"/>	<input type="checkbox"/>
AVG_PM	double		<input checked="" type="checkbox"/>	<input type="checkbox"/>
AVG_IDLE	double		<input checked="" type="checkbox"/>	<input type="checkbox"/>

RunOutput 스키마 작성 예시

Expression
RunScript("C:\\EQP_GROU_UTIL.linq")

RunOutput OnEndRun Expression 작성 예시

위 예시는 모델 실행이 종료된 후 장비 그룹별 평균 가동률을 계산해서 저장하는 RunOutput을 정의한 예시입니다. 현재 Expression에서는 DataTable을 생성하거나 DataTable에 Row를

추가할 수 있는 함수를 제공하고 있지 않기 때문에, 반드시 LINQPad 스크립트를 실행해서 데이터를 생성해야 합니다.

---

## ExecutionOutputs

ExecutionOutput은 Simulation 종료 후 결과 분석에 필요한 데이터를 미리 추출 및 가공해 놓는 시나리오 단위의 사용자 정의 아웃풋입니다. 각 모델의 실행이 완료될 때마다 또는 시나리오에 정의된 모든 모델의 실행이 완료된 후의 두 가지 시점에 ExecutionOutput 데이터를 생성합니다. RunOutput은 실행된 모델의 수 만큼 테이블이 생성되지만 ExecutionOutput은 하나의 테이블만 생성됩니다. 하나의 ExecutionOutput은 스키마와 스키마를 채우기 위한 OnEndRun Expression 및 OnEndExecution Expression으로 구성되어 있습니다. 기록된 ExecutionOutput은 Execution의 ExecutionOutputs 노드에서 확인할 수 있습니다. 모든 시나리오에는 Run 별 Factor와 KPI 값을 기록하는 DefaultReport ExecutionOutput이 등록되어 있습니다.

### ExecutionOutput 등록 절차

1. [Design]>[ExecutionOutputs] 노드에서 [Add] 버튼을 눌러 ExecutionOutput 그룹을 추가합니다.
2. 추가된 ExecutionOutput 그룹 노드에서 [Add] 버튼을 눌러 ExecutionOutput을 추가합니다.
3. ExecutionOutput 명칭을 입력합니다.
4. ExecutionOutput 항목이 추가된 후 [Schema] 탭에서 추출 및 가공한 데이터를 저장할 스키마를 정의합니다.
5. [OnEndRun] 탭에서 모델 실행이 완료될 때마다 스키마를 채우기 위한 Expression을 정의합니다. Expression의 결과값은 DataTable 형식이어야 합니다.
6. [OnEndExecution] 탭에서 모든 모델의 실행이 완료된 후 스키마를 채우기 위한 Expression을 정의합니다. Expression의 결과값은 마찬가지로 DataTable 형식이어야 합니다.
7. Expression 작성 후 저장합니다.

Design/ExecutionOutputs/Outputs/EQP\_GROU\_UTIL\_REPORT x

Schema Change Sequence

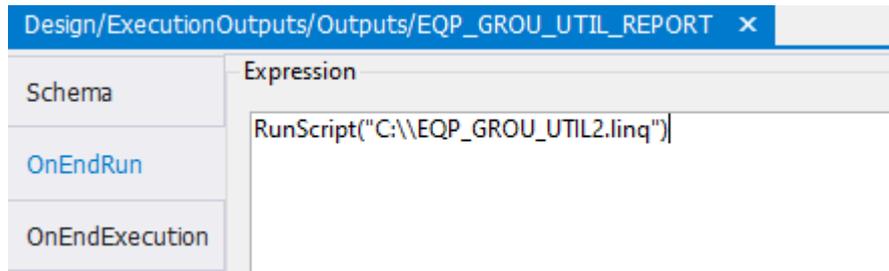
OnEndRun Local Filter (Enter filter expression for local data table...)

OnEndExecution Description

Name	Data Type	Default Value	Allow Nulls	Is Primary Key
RUN_INDEX	int		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
EQP_GROUP	string		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
AVG_BUSY	double		<input checked="" type="checkbox"/>	<input type="checkbox"/>
AVG_SETUP	double		<input checked="" type="checkbox"/>	<input type="checkbox"/>
AVG_PM	double		<input checked="" type="checkbox"/>	<input type="checkbox"/>
AVG_IDLE	double		<input checked="" type="checkbox"/>	<input type="checkbox"/>

DataAction

ExecutionOutput 스키마 작성 예시



ExecutionOutput OnEndRun Expression 작성 예시

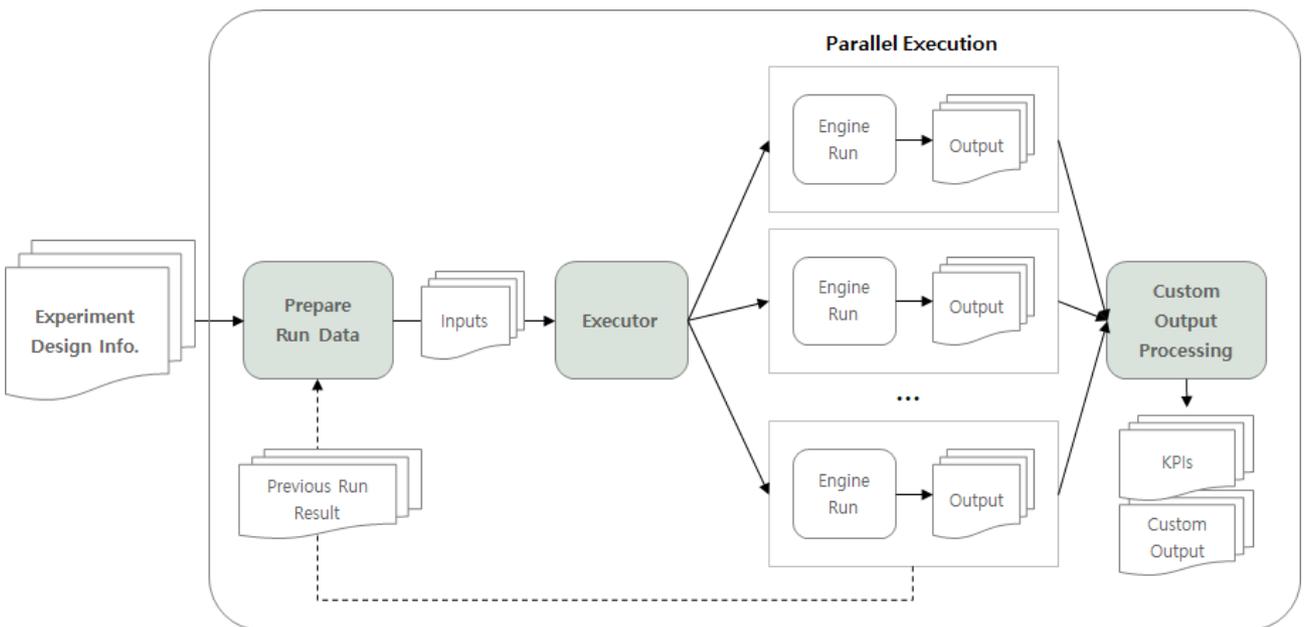
위 예시는 RunOutput과 유사하게 모델 실행이 종료되었을 때 장비 그룹별 평균 가동률을 계산해서 저장하는 ExecutionOutput을 정의한 예시입니다. 현재 Expression에서는 DataTable을 생성하거나 DataTable에 Row를 추가할 수 있는 함수를 제공하고 있지 않기 때문에, 반드시 LINQPad 스크립트를 실행해서 데이터를 생성해야 합니다.

# Simulation Runner

## Runner 의 구조

Simulation Runner 는 아래 그림과 같이 크게 3개의 기능으로 구성됩니다.

- **Prepare Run Data** : 실험계획에 정의된 Factor 들의 조합으로 Base Model 의 입력데이터를 계획한 Factor Value 로 치환하여 실행시 필요한 Input Data 를 준비합니다.
- **Executor** : 준비된 Input Data Set 별로 모델을 실행합니다. 이때, 성능을 고려하여 동시 병렬실행을 지원합니다.
- **Custom Output Processing** : Model 실행결과별로 정의된 KPI 데이터를 추출하고, KPI 이외에 사용자가 정의한 추가 Output (RunOutput, ExecutionOutput) 데이터를 추출하여 시나리오 결과로 기록합니다.



Simulation Runner 의 실행 구조

## 실행 설정

Runner 의 설정을 위해서는 시나리오 탐색기의 Executions 하위에 Execution 을 생성합니다. Execution 단위로 실행 옵션을 정의할 수 있습니다. 실행 옵션의 기본 설정 항목은 아래 3가지 탭으로 나누어 설정 가능합니다.

- General : 실험계획시 설계된 Factor 의 반영 여부 및 모델 실행 방식등을 정의합니다.
- I/O : 시뮬레이션 Input, Output 에 대한 저장 옵션입니다.
- Database : 시나리오의 Input, Output 을 데이터베이스와 연계하여 실행하기 위한 옵션을 설정합니다.

## General

### Factor Options

시나리오에 정의된 Factor 중 실행시 반영 대상이 되는 Factor 를 선택합니다. 아래와 같이 실제 시나리오상에 4개의 Factor 와 Value 가 정의되어 있어도 실행옵션에서 첫번째 Factor (PROD1INV)만 선택을 하는 경우 해당 Factor 의 Factor Value 로 정의한 케이스에 대해서만 실행됩니다.

The screenshot shows the 'General' configuration window with three tabs: 'General', 'I/O', and 'Database'. The 'General' tab is active and contains the following sections:

- Factor Option:** A list of factors with checkboxes. 'PROD1INV' is checked, while 'PROD2INV', 'PROD3INV', and 'NewFactor' are unchecked.
- Execution Option:**
  - Base Model Run: Skip (dropdown menu)
  - End Condition: (empty text field with a menu icon)
  - Thread Count: 2 (spin box)
  - Executor: Simple Executor (text field)
  - Execute Model in Independent Process
  - Execute Model Using Libraries in Private Path
- Log Option:**
  - Write Model Log

Experiment 설정 창 (기본탭)

### Execution Options/Base Model Run

Base Model 에 대한 실행 여부를 설정합니다. 기본 설정값은 Skip 입니다. 옵션별 실행방식은 아래와 같습니다.

- Skip : Base Model 을 실행하지 않습니다.
- Run Directly : 시나리오 폴더의 Base Model 이 저장된 위치에서 바로 실행됩니다. 이경우 실행결과가 시나리오 폴더 Root 에 생성됩니다.
- Run In Execution Folder : Base Model 을 Execution 실행폴더 하위로 복제한 후 실행합니다

### **Execution Options/End Condition**

계획된 시나리오의 실험간 설정된 조건을 충족하는 경우 이후 계획된 실험을 중단할 수 있습니다. End Condition 에 Expression 으로 이러한 조건을 지정할 수 있습니다.

### **Execution Options/Thread Count**

계획된 시나리오의 Run 들을 병렬실행 시키기 위한 옵션입니다. 1이상의 값을 설정하는 경우에 동시에 Thread 설정 횟수만큼의 Run 을 실행시켜 전체 시나리오에 대한 실행시간을 단축할 수 있습니다.

### **Execution Options/Executor**

계획된 실험을 실행시켜주는 Executor 로 WISE 기본은 Simple Executor 입니다. 플러그인 설치에 따라 Executor 가 추가되며, 본 항목에서 실행대상을 선택할 수 있습니다.

### **Execution Options/Execute Model in Independent Process**

Run 을 실행시키는 Host 를 독립적으로 실행할 것인지 여부를 결정합니다. true 인 경우에는 각 모델 실행 Run 별로 실행을 위한 Agent 프로그램이 독립적으로 실행됩니다. 만일 Thread Count 를 3으로 했다면 Agent 3개가 동시에 실행되게 됩니다. 개별 실행시 오류가 발생하는 경우에도 전체 시나리오 실행에 영향을 주지 않습니다. false 인 경우에는 Host 한개가 Run 을 순차 혹은 병렬로 설정에 따라 실행합니다. 하나의 Process 에서 실행을 관장하게 됨으로 한개의 Run 에서 심각한 오류가 발생하는 경우에 전체 시나리오가 실행되지 못하고 종료하게 됩니다.

### **Execution Options/Execute Model Using Libraries in Private Path**

시나리오의 private path 의 파일만을 가지고 시나리오를 실행시킵니다. WISE 설치버전과 Model 의 빌드버전이 다른 경우에 이 옵션을 사용할 수 있습니다.

**i** 이 옵션을 사용하기 위해서는 Base Model 의 실행을 위한 Studio 설치 폴더에서 아래 두가지 파일이 Private Path에 반드시 포함되어 있어야 합니다.

- MozartAgent.dll
- Mozart.Task.Model.dll

### Log Option /Write Model Log

Base Model 의 실행로그를 시나리오 실행 로그에 포함하여 출력할지 여부입니다. true 인 경우 시나리오의 Run 실행 로그와 각 Run 별 모델에 포함된 실행로그가 함께 출력됩니다.

### I/O

시나리오 실행시에 수행되는 모델실행 횟수는 적게는 2~3회 부터 많게는 수백회에 이르기 까지 다양하게 설정할 수 있습니다. 이에 따라 각 모델의 실행 Run 별 생성되는 데이터가 많아질 수 있습니다. I/O 설정을 통해 필요한 Input, Output 만 선별적으로 저장할 수 있습니다.

### Input Options

Execution 에서 수행되는 Run 별 Input 저장 옵션입니다. 시나리오 실행시 실행된 데이터는 시나리오 Root 폴더 하위의 Executions/[Execution Name] 폴더에 Run 별로 생성되고, Run 별 폴더 하위의 Data 폴더에 Input 정보가 저장됩니다. Data Save Option 은 아래 4가지 유형이 있으며 기본 값은 All 입니다.

- All : 각 Run 별로 Factor Value 에 의해 변경된 Input 과 Base Model 의 Input 전체를 파일로 저장합니다.
- MODIFIED : Factor Value 에서 정의하여 Base Model 과 달라진 Input 만 저장합니다.
- CUSTOM : 하기의 Items 에 지정된 Input data 만 저장합니다. CUSTOM 상태에서만 하기 Items 의 항목을 선택할 수 있습니다.
- NONE : Run 의 Input 전체를 저장하지 않습니다.

The screenshot shows the 'I/O' configuration panel. On the left, there are three tabs: 'General', 'I/O' (selected), and 'Database'. The main area is divided into two sections: 'Input Option' and 'Output Option'. Each section contains a 'Data Save Type' dropdown menu set to 'ALL' and a list of items with checkboxes. In the 'Input Option' list, the items are 'Product', 'ProcStep', and 'Equipment'. In the 'Output Option' list, the items are 'ErrorHistory', 'EqpPlan', and 'StepTarget'. All checkboxes are currently unchecked.

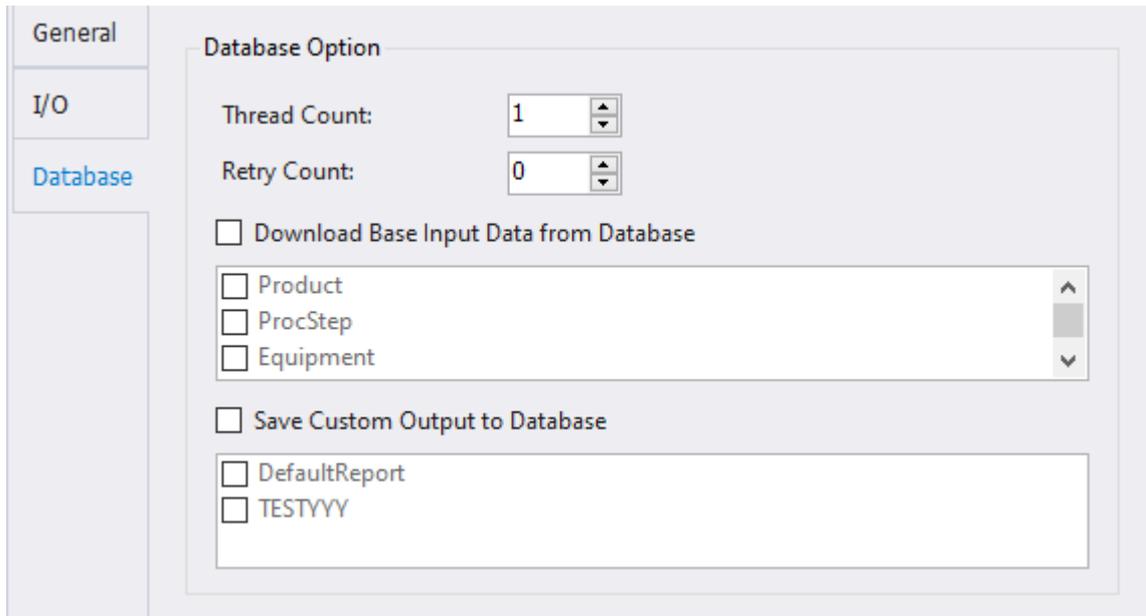
Experiment 설정창 (I/O 탭)

## Output Options

Execution 의 Run 별 Output 저장 옵션입니다. Run 별 Output 역시 Input 과 동일하게 Execution 의 Run 별 실행폴더에 생성됩니다. 실행옵션은 Input 옵션과 같습니다.

## Database

Base Model 의 실행시 Model 에 연결된 DataAction 을 기준으로 실행시점의 데이터를 사용하거나 시나리오 실행 결과를 외부에 저장하기 위한 설정을 할 수 있습니다.



Experiment 설정창(Database 탭)

### Tread Count

데이터베이스에서 Input data 를 다운로드 하거나 최종 결과를 저장하는 작업을 수행할 때 필요한 스레드 수를 지정합니다.

### Retry Count

데이터베이스 연결 실패시 재시도 횟수를 지정합니다.

### Download Base Input Data From Database

시나리오 실행 초기시점에 Base Model의 Input 데이터 중 현재 시점의 데이터로 데이터베이스에서 다운로드 할 지 여부를 설정합니다. 값이 체크되면 하위의 리스트중 다운로드 대상 Input table 을 선택할 수 있도록 활성화됩니다. 선택한 Input 만 다운로드 합니다.

### Save Custom Output to Database

시나리오에 설정된 Run Output, Execution Output 을 저장할지 여부를 결정합니다. `true` 로 설정된 경우 하위 리스트에 표시된 사용자 정의 Output 리스트가 활성화되며, 저장대상 Output 을 선택할 수 있습니다. 리스트에 표시되는 항목은 Design 의 RunOutput, Execution Output 에 등록된 결과항목들입니다.

# Data Analyzer

시나리오의 실행 기본 실행 결과는 시나리오에 계획된 실행(Run)에 대해 해당 Run 의 실행 시 Factor 값과 KPI 값으로 제공됩니다. 실행한 Execution 하위에 Report 노드를 더블클릭하여 결과를 조회할 수 있습니다. 아래 예시는 2개의 Factor 를 사용하고 각 Factor 별 7개의 수준을 정의하여 실행한 결과입니다.

Execution Analysis Report																	
Execution		Performance			Model Info		Factors				KPIs				Model		
Run Index	Run Name	Start Time	End Time	Elapsed Time	Model Name	Dll Name	New Factor	PROD1INV	PROD2INV	PROD3INV	Avg TAT	Plan Out	PROD1OUT	Utilization	Wip	Model	
0	Run 0	2020-02-13 13:04:01	2020-02-13 13:04:03	0:00:01	-	-	0	100	100	100	37.25	650	250	87.58	427.78	Open in Studio	
1	Run 1	2020-02-13 13:04:03	2020-02-13 13:04:05	0:00:01	-	-	0	130	100	100	36.79	650	225	87.58	391.67	Open in Studio	
2	Run 2	2020-02-13 13:04:03	2020-02-13 13:04:05	0:00:01	-	-	0	160	100	100	36.18	650	200	87.58	347.22	Open in Studio	
3	Run 3	2020-02-13 13:04:05	2020-02-13 13:04:06	0:00:01	-	-	0	190	100	100	36.25	650	175	87.58	311.11	Open in Studio	
4	Run 4	2020-02-13 13:04:05	2020-02-13 13:04:06	0:00:01	-	-	0	220	100	100	35.97	650	175	87.58	288.89	Open in Studio	
5	Run 5	2020-02-13 13:04:06	2020-02-13 13:04:07	0:00:01	-	-	0	250	100	100	35.45	650	150	87.58	280.56	Open in Studio	
6	Run 6	2020-02-13 13:04:06	2020-02-13 13:04:07	0:00:01	-	-	0	280	100	100	35.56	650	150	87.58	269.44	Open in Studio	
7	Run 7	2020-02-13 13:04:07	2020-02-13 13:04:09	0:00:01	-	-	0	100	130	100	36.76	650	250	87.58	380.56	Open in Studio	

Report 결과 예시

Run 별 수행시 사용된 Factor 별 값과 KPI 결과를 한눈에 볼 수 있습니다. 또한 이 리포트는 시나리오의 실행 도중에도 확인 할 수 있습니다. 상단의 Refresh 아이콘을 사용하여 실행 중 결과를 갱신할 수 있습니다. 각 Run 별 오른쪽 끝의 "Open in Studio" 메뉴를 사용하면 모델과 연결된 MOZART Studio 에서 모델을 열어 개별 Run 의 결과를 확인 할 수 있습니다.

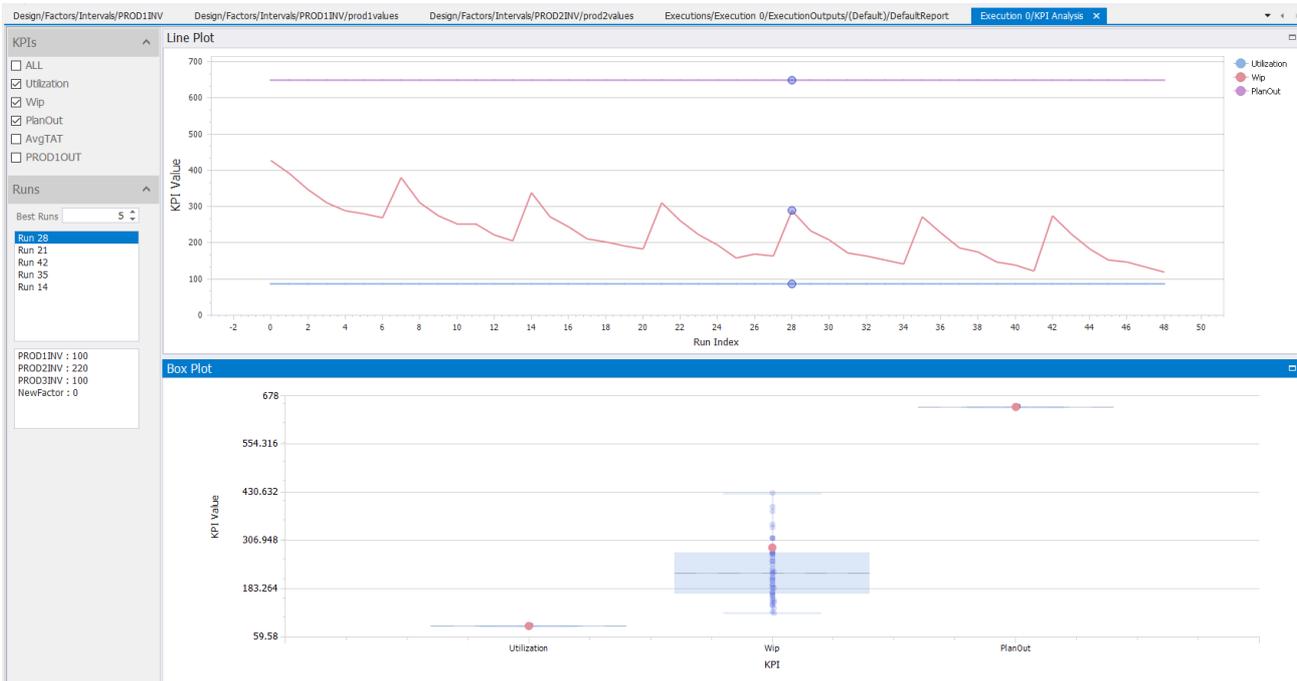
WISE 는 기본 결과 외에 기본결과에 대한 시각적인 두가지 리포트를 제공합니다.

- **KPI Analysis** : Run 에 따른 KPI 의 추세 및 KPI 결과값의 분포 등을 시각적으로 표시합니다.
- **Factor-KPI Analysis** : Factor 와 KPI 의 상관성을 수치와 차트로 표시합니다.

## KPI Analysis

KPI Analysis 화면은 좌측의 패널을 사용하여 관심있는 KPI 를 선택하여 관심있는 KPI 에 대한 차트를 볼 수 있도록 구성되어 있습니다.

KPI 값은 이상단의 Run 별 KPI 값을 조회 할 수 있는 차트와 그 하단의 KPI 별 Value 의 분포를 알 수 있는 박스플롯 차트를 제공 표시합니다.



KPI Analysis 화면예시

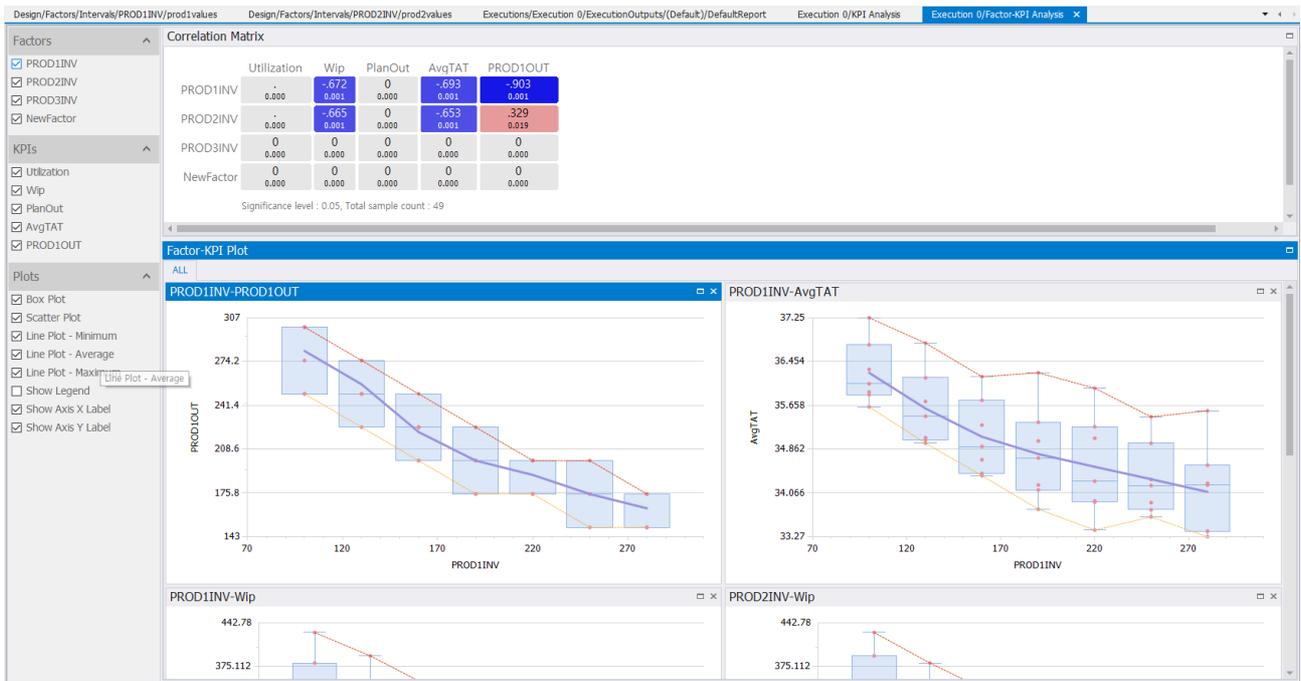
좌측 Runs 항목에서 Best Run Count 를 조정하면 각 KPI 의 소팅 기준에 따라 가장 각 점수 합계가 가장 높은 Run 을 지정된 수만큼 목록으로 표시합니다. 목록에서 Run 을 선택하면 해당 Run 의 Factor 값이 아래의 리스트에 표시되고, KPI 값이 추이와 박스플롯상에 점으로 위치가 표시됩니다.

## Factor-KPI Analysis

본 화면에서는 Factor 와 KPI 간의 상관계수를 종합적으로 확인 할 수 있으면 개별 FACTOR-KPI 간의 관계를 확인 할 수 있는 차트를 표시합니다.

### Correlation Matrix

행은 Factor, 열은 KPI 를 의미하며, 직교점에 위치한 값은 Factor - KPI 간의 상관계수입니다. 상관계수의 유의성을 확인하여 유의하지 않은 경우 회색으로 표시되고, 유의한 경우는 음양의 정도에 따라 양은 붉은색, 음은 파란색으로 표시됩니다. 채도가 높을 수록 강한 상관관계를 갖는 데이터를 표시합니다.

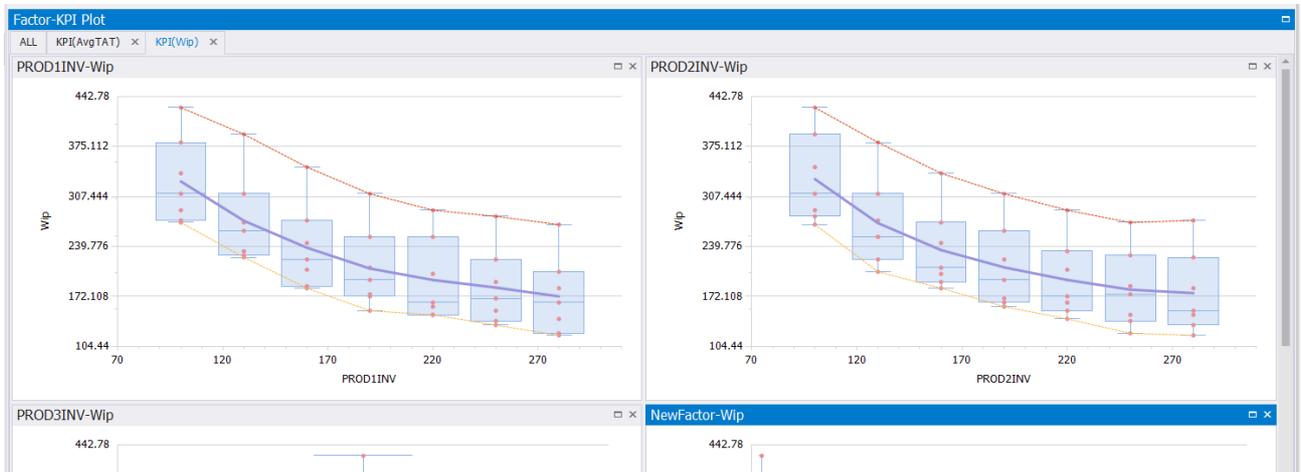


Factor-KPI Analysis 화면 예시

## Factor-KPI Plot

Correlation Matrix 에는 기본적으로 상관관계가 높게 나온 Factor-KPI Plot 이 표시됩니다. 상단 Correlation Matrix 에서 특정 Cell 을 더블클릭하면 해당 차트로 하단의 그래프 포커스가 이동합니다.

행의 제목에 있는 Factor 혹은 열제목인 KPI 를 더블 클릭하는 경우 해당 항목의 이름으로 하단에 별도의 Tab 이 추가되고 관련된 그래프가 하단에 모두 표시됩니다. 아래 이미지는 Correlation Matrix 에서 KPI 인 WIP을 더블클릭했을때 표시되는 화면을 예시적으로 표시합니다.

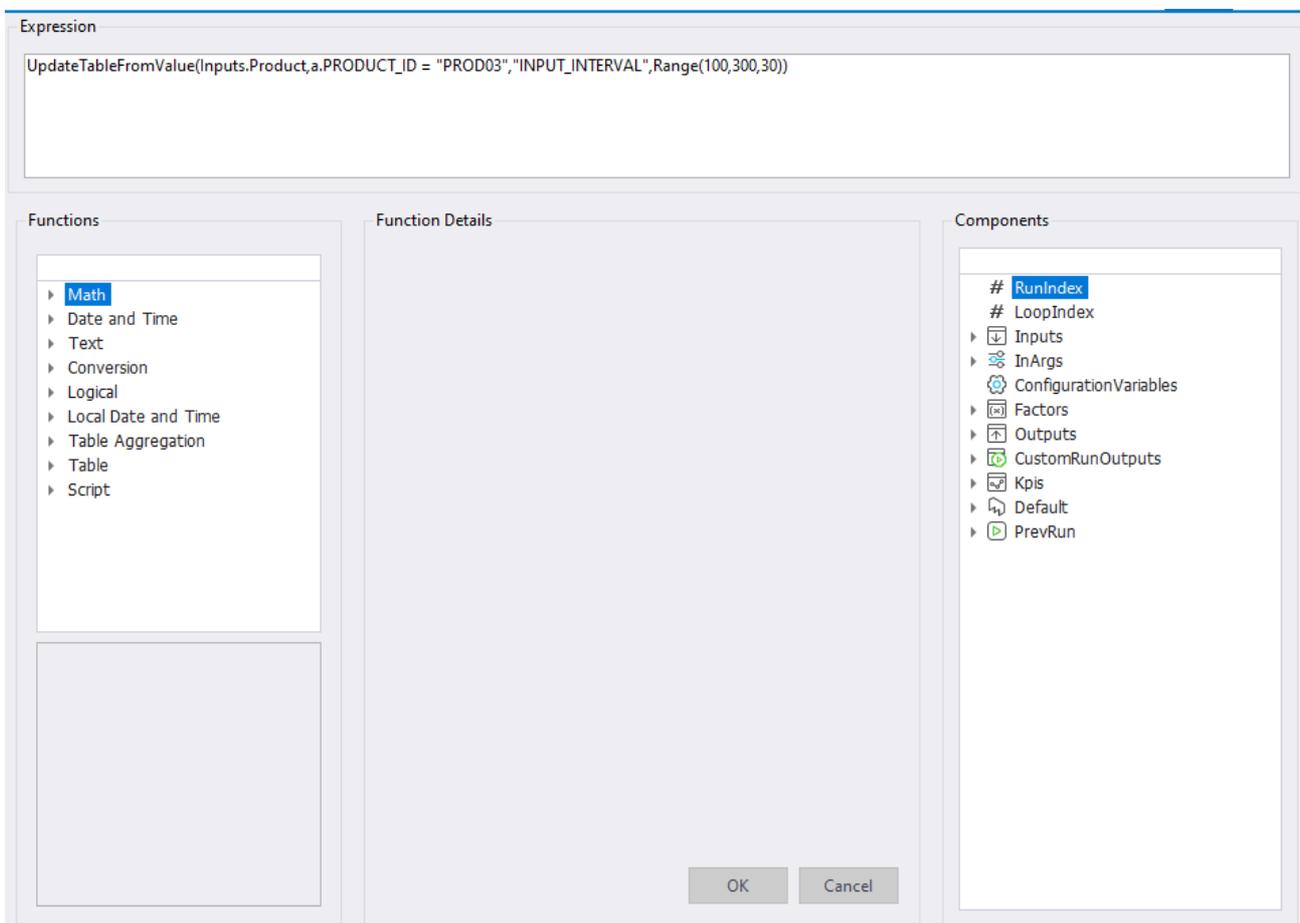


Factor-KPI Plot 의 KPI 별 탭 예시

# Expression 편집 기능

수식 편집기는 아래의 그림에서 같이 4부분으로 구성되어 있습니다.

- **Expression** : 수식이 표시되는 텍스트박스입니다. 직접 수식을 입력하거나 하단의 Function 을 사용하여 수식을 삽입할 수 있습니다.
- **Functions** : 수식에서 사용가능한 함수 리스트입니다.
- **Function Details** : 특정 함수를 선택했을 때 함수의 인수들이 표시되고 각 인수별 세부적인 수식을 입력할 수 있습니다. 모든 입력이 끝나고 OK 버튼으로 수식을 상단 Expression 창에 입력할 수 있습니다.
- **Components** : 수식에서 사용가능한 데이터 요소입니다.



Expression 편집 화면 구성

## Functions

문자열, 날짜 및 수학적, 논리연산 및 데이터 테이블의 값을 변환하는 함수들을 제공하며, LINQPad 를 사용하여 만든 .linq 파일의 스크립트를 사용할 수 있도록 기능을 제공합니다. 개별 함수에 대한 상세 설명은 추후 제공 예정입니다.

---

## Components

Base Model 에 정의된 Input, Output, Argument 의 스키마와 데이터를 모두 사용할 수 있으며, 현재 수행중인 Run 의 Index 및 이전에 수행된 Run 의 정보 등 시나리오에 정의된 대부분의 데이터요소를 수식에 사용할 수 있도록 제공합니다. 또한 시나리오에 정의된 Factor, KPI, Custom Output 까지 사용할 수 있습니다. 개별 항목에 대한 자세한 설명은 추후 제공 예정입니다.

**Untitled**

# Analysis View Designer

## 개요

시나리오에 정의된 Table 외에 추가로 다양한 소스의 Table을 추가할 수 있고, Execution 및 Run의 Output 테이블들이 포함된 Analysis View(분석화면)를 생성하여 데이터를 시각화 할 수 있습니다.

## Table 기본 등록 방법

사용할 소스 타입을 선택하고 해당 소스 파일을 등록하여 Table을 등록합니다. Table 편집화면에서 컬럼들 등록합니다.

Name	DataType	ColumnType
DEMAND	int	Static
AVG_LEADTIME	double	Static
MIN_LEADTIME	double	Static
MAX_LEADTIME	double	Static
EQP01_Util	double	Static
EQP02_Util	double	Static
TARGET_RATE	double	Static

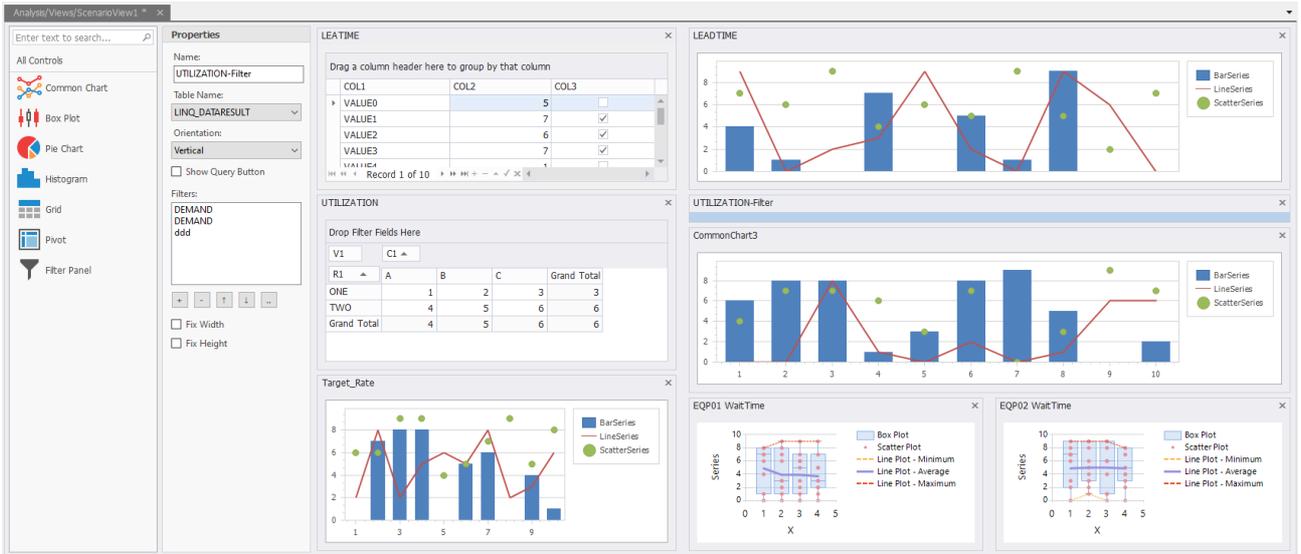
  

DEMAND	AVG_LEADTIME	MIN_LEADTIME	MAX_LEADTIME	EQP01_Util	EQP02_Util	TARGET_RATE
=	=	=	=	=	=	=
200	121.875	41.66667	183.33333	1.6369	3.30688	100
600	257.29167	41.66667	450	5.60516	9.92064	45.83333
1,000	391.04167	41.66667	716.66667	9.57341	16.03009	27.5
1,400	524.55357	41.66667	983.33333	13.39906	21.81713	19.64286
1,800	657.98611	41.66667	1,250	16.87128	27.60417	15.27778
2,200	791.38258	41.66667	1,516.66667	20.3435	33.3912	12.5
2,600	924.75962	41.66667	1,783.33333	23.81572	39.17824	10.57692
3,000	1,058.125	41.66667	2,050	27.28795	44.96528	9.16667
12,800	4,727.01521	333.33333	8,933.33333	118.53299	196.21363	240.49756

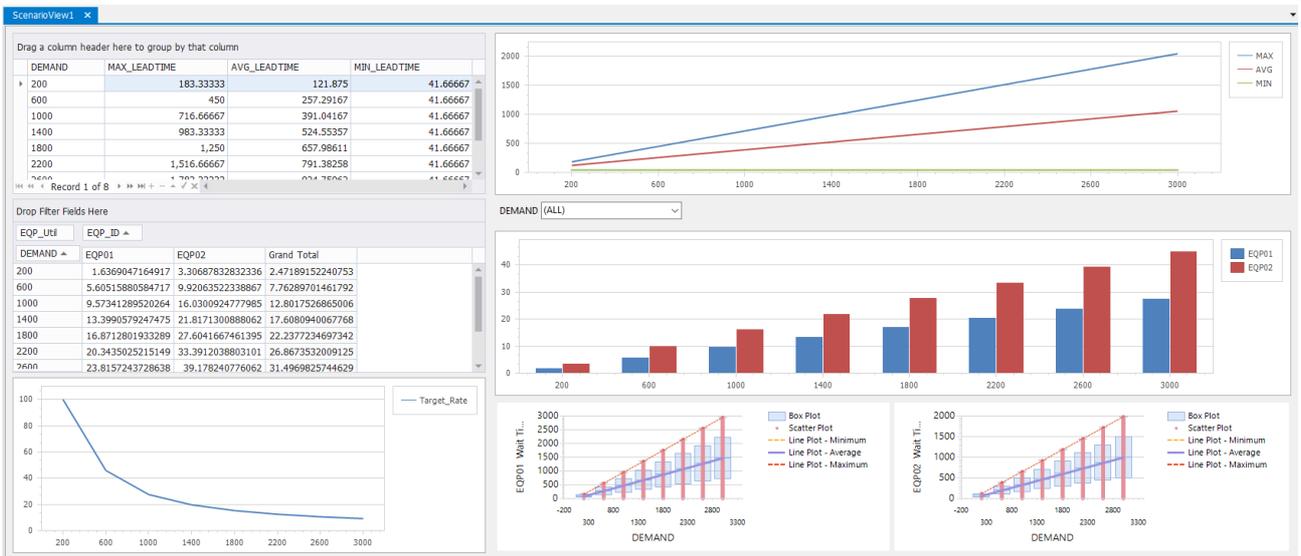
등록된 Table 화면 예시

## View 기본 등록 방법

드래그 앤 드롭으로 분석화면에 컨트롤을 추가하고, 레이아웃을 설정할 수 있습니다. 추가된 컨트롤을 선택해서 속성을 설정할 수 있습니다



사용자 분석화면 편집 예시



사용자 분석화면 결과 예시

# Analysis Table

## Analysis Table 등록

Analysis View(사용자 분석 화면)를 작성하기 위해서는 사용할 소스가 연결된 Analysis Table 을 추가해야 합니다. 등록된 Table은 소스 타입별로 설정하고, Analysis View(사용자 분석화면)에서 사용할 Table의 컬럼들을 설정합니다.

1) 시나리오 탐색기에서 Analysis의 Tables에 마우스 우클릭, **[Add]**를 선택합니다.

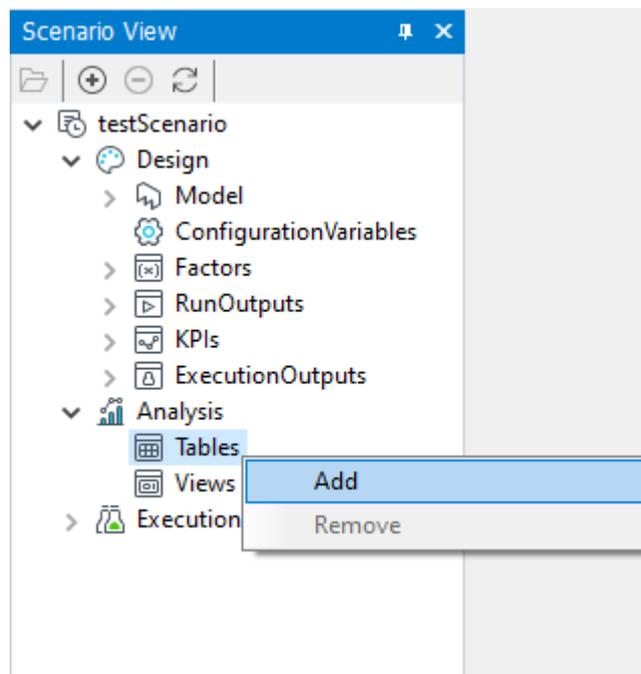
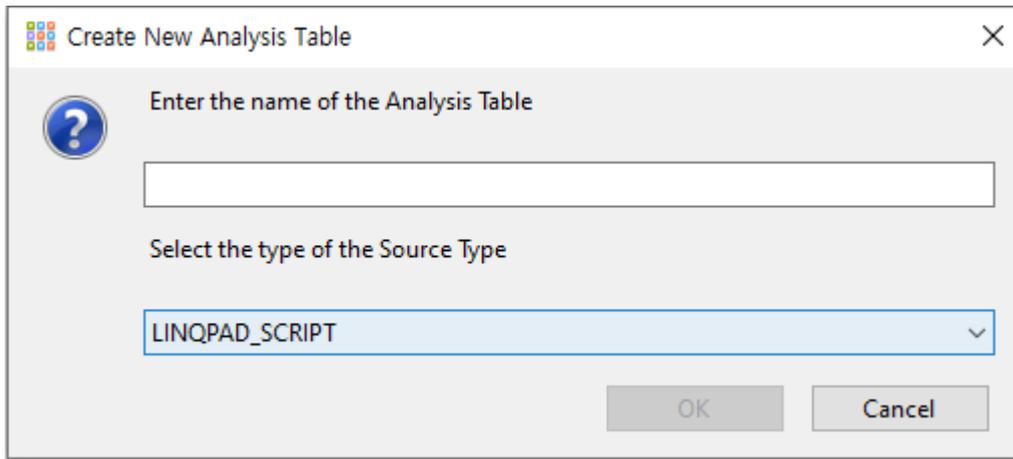


Table Add 메뉴

2) Create New Analysis Table Dialog에서 **[Enter the name of the Analysis Table]**에 Table 이름을 설정해주고, **[Select the type of the Source Type]**에서 아래의 5가지 Source Type 중 한 가지를 택하고 **[OK]** 버튼을 클릭합니다.



Create New Analysis Table 화면

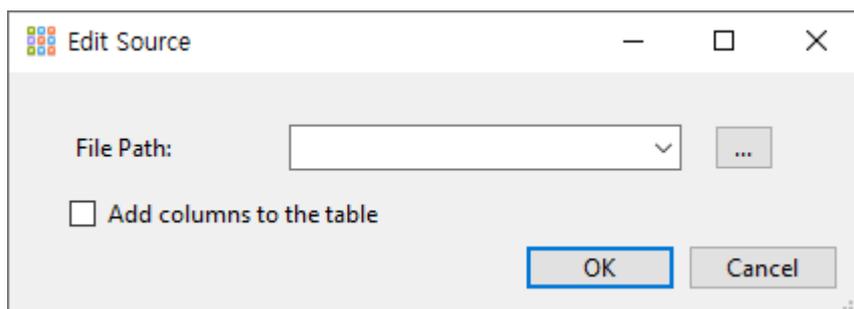
- LINQPAD\_SCRIPT : LINQPad의 스크립트 파일(.linq)
- MODEL : Mozart Studio의 모델 파일(.vmodel)
- SCENARIO : Mozart WISE의 시나리오 파일(.vscenario)
- CSV : 텍스트 데이터 파일(.csv)
- EXCEL : 엑셀 데이터 파일(.xlsx)

## Source Type 별 Table 속성

설정된 Source Type에 따라 Edit Source Dialog 표시됩니다.

### LINQPAD\_SCRIPT 파일을 통한 Table 설정

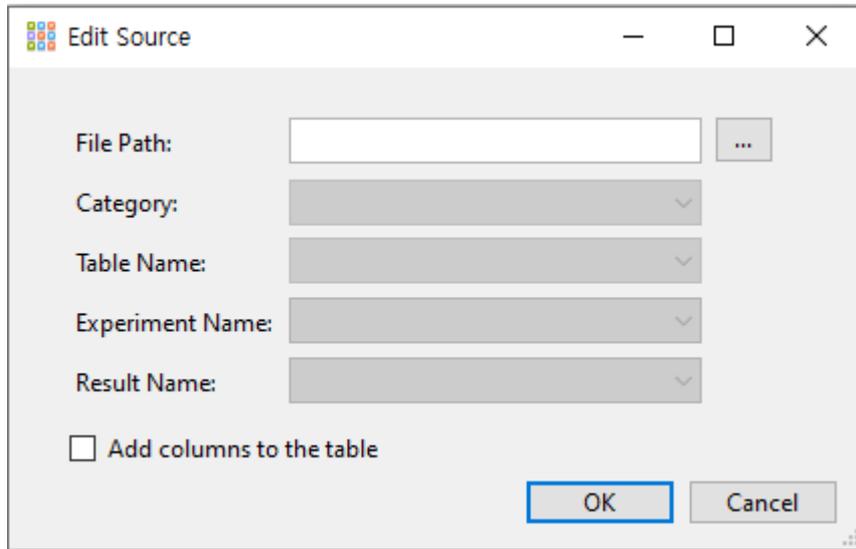
Table의 Source로서 LINQPAD\_SCRIPT를 설정할 경우, **[File Path:]**에서 Source로 사용할 LINQPad 스크립트 파일 경로를 선택합니다. **[Add columns to the table]** 설정을 통해 Source의 Column들을 Table에 추가할 지 결정하고 **[OK]** 버튼을 클릭합니다.



LINQPAD\_SCRIPT Edit Source 화면

## MODEL 파일을 통한 Table 설정

Table의 Source로서 MODEL을 설정할 경우, **[File Path:]**에서 Source로 사용할 Mozart Studio 모델 파일 경로를 선택합니다. 이후 아래의 텍스트 박스를 채웁니다.



MODEL Edit Source 화면

### Category

모델의 데이터 분류 항목

- Input : 모델의 기존 데이터
- Output : 모델의 실험 결과 데이터

### Table Name

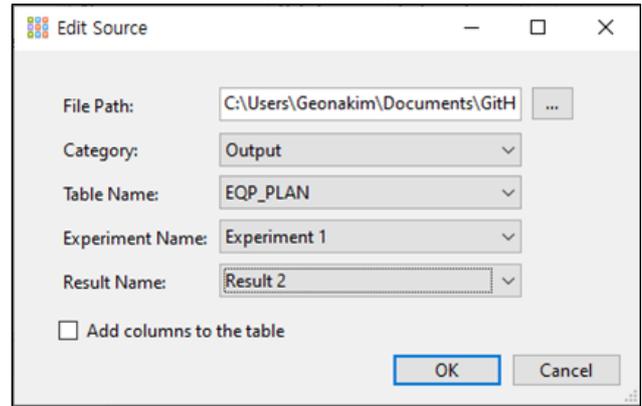
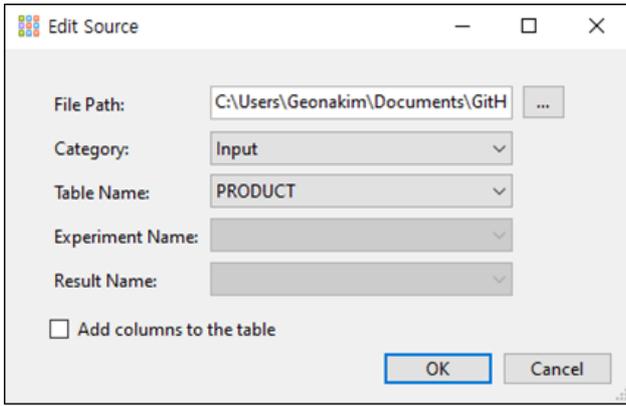
모델의 데이터 테이블 이름

### Experiment Name

실험 이름(Category의 Output 내의 Experiment가 존재하는 경우 선택 가능)

### Result Name

실험 내의 결과 이름(Experiment의 Result가 존재하는 경우 선택 가능)

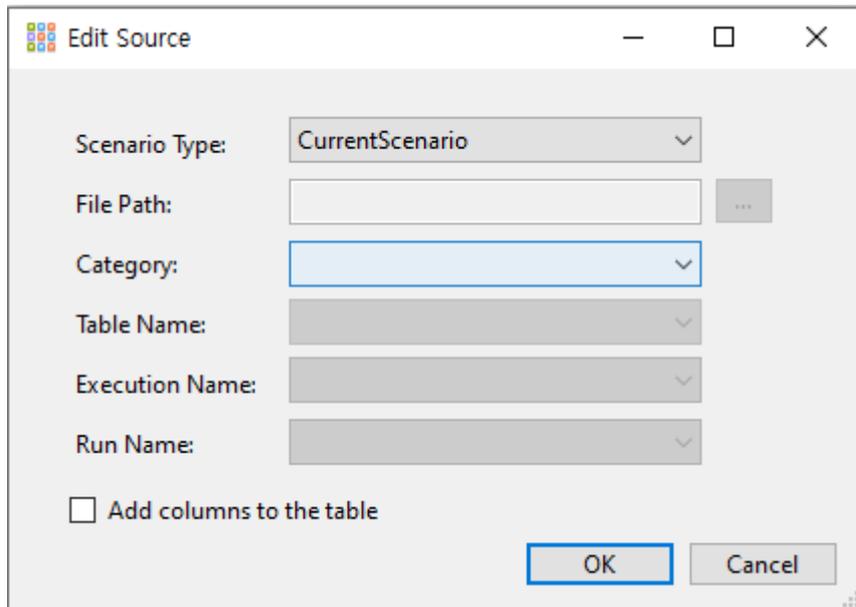


Category가 Input/Output 선택 시 Table 설정 예시

마지막으로 **[Add columns to the table]** 설정을 통해 Source의 Column들을 Table에 추가할지 결정하고 **[OK]** 버튼을 클릭합니다.

## SCENARIO 파일을 통한 Table 설정

Table의 Source로서 SCENARIO를 설정할 경우, 가장 처음 **[Scenario Type:]**에서 **[CurrentScenario]**를 선택하여 현재 Mozart WISE에서 열려있는 시나리오를 선택하거나, **[File]**을 선택하여 **[File Path:]**를 통해 다른 시나리오를 불러올지 선택할 수 있습니다. 이후 아래의 텍스트 박스를 채웁니다.



SCENARIO Edit Source 화면

### Scenario Type

## 시나리오 분류

- Current Scenario: 현재 시나리오
- File: 다른 시나리오(.vscenario) 파일

### File Path

다른 시나리오(.vscenario) 파일의 경로

### Category

시나리오 내의 데이터 분류 항목

- BaseModelInput: 모델의 기본 데이터
- Input: 각 Run별 모델의 기본 데이터
- Output: 각 Run별 모델의 결과 데이터
- RunOutput: 시뮬레이션 종료 후 결과 분석에 필요한 데이터를 미리 추출 및 가공해 놓는 모델 단위의 사용자 정의 아웃풋
- ExecutionOutput: 시뮬레이션 종료 후 결과 분석에 필요한 데이터를 미리 추출 및 가공해 놓는 시나리오 단위의 사용자 정의 아웃풋

### Table Name

시나리오 내의 테이블 이름

### Execution Name

시나리오 실행 단위 이름 (Category의 Input/Output/RunOutput 내의 Execution 존재하는 경우 선택 가능)

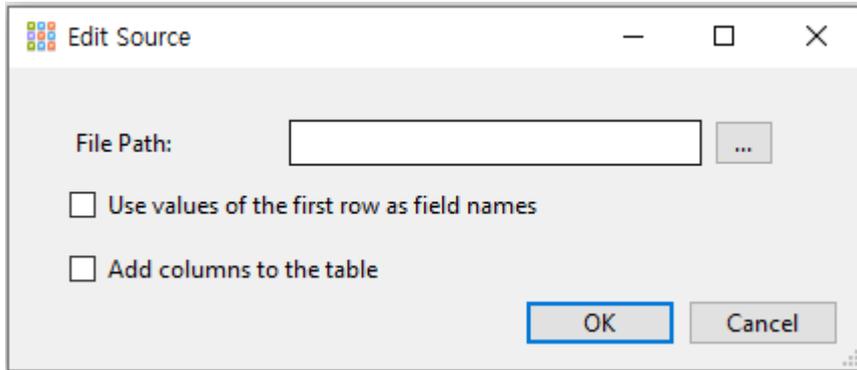
### Run Name

시나리오 실행 내의 실험 이름 (Execution의 Run 존재하는 경우 선택 가능)

마지막으로 **[Add columns to the table]** 설정을 통해 Source의 Column들을 Table에 추가할 지 결정하고 **[OK]** 버튼을 클릭합니다.

## CSV 파일을 통한 Table 설정

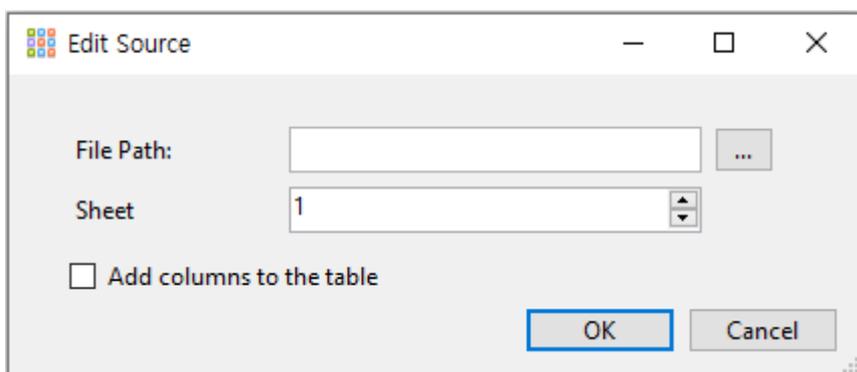
Table의 Source로서 CSV를 설정할 경우, **[File Path:]**에서 Source로 사용할 텍스트 파일 경로를 선택합니다. 이후 **[Use values of the first row as field names]**를 통해 csv파일의 첫 번째 row를 field name(column name)으로 설정할지 결정한 후 **[Add columns to the table]**를 통해 설정된 Column들을 Table에 추가할지 결정합니다. 모든 설정을 완료한 후 **[OK]** 버튼을 클릭합니다.



CSV의 Edit Source 화면

## EXCEL 파일을 통한 Table 설정

Table의 Source로서 EXCEL을 설정할 경우, **[File Path:]**에서 Source로 사용할 엑셀 파일 경로를 선택합니다. Source로 사용할 **[Sheet]**를 선택한 후 **[Add columns to the table]** 설정을 통해 Source의 Column들을 Table에 추가할지 결정합니다. 모든 설정을 완료한 후 **[OK]** 버튼을 클릭합니다.



EXCEL의 Edit Source 화면

---

## Table 편집

생성된 Table은 아래와 같이 Table 편집 화면 tab으로서 오른쪽 화면에 나타납니다.

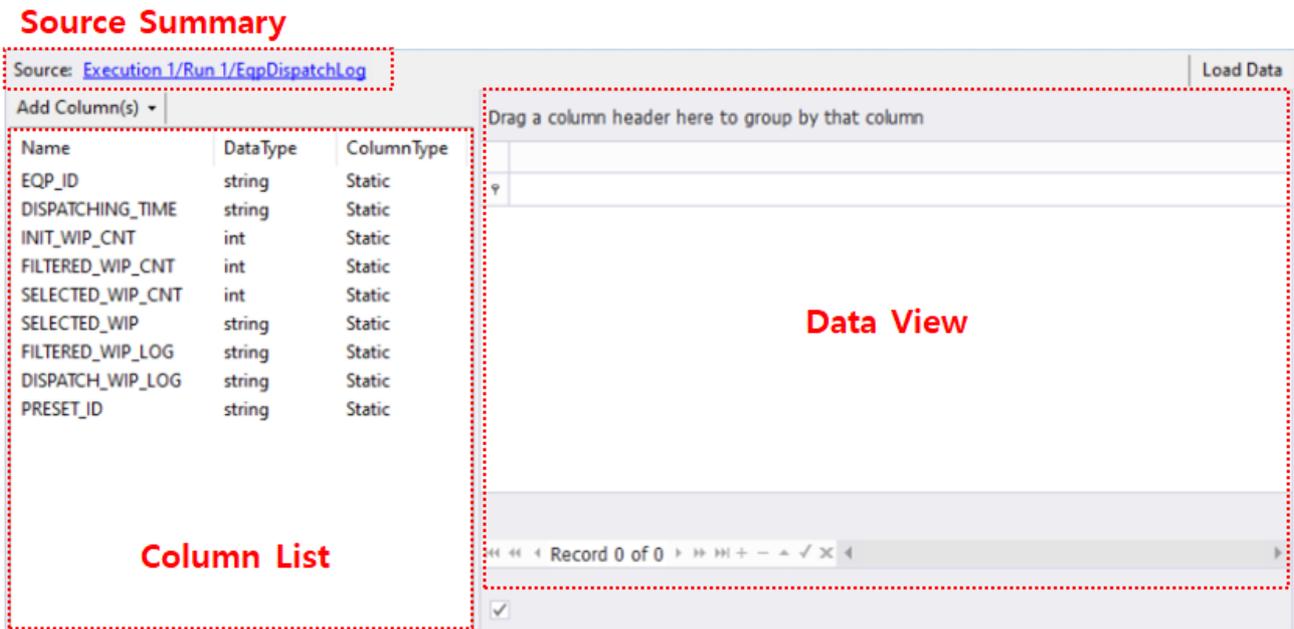


Table 편집화면

## Source Summary

Table에 연결된 소스 정보

## Column List

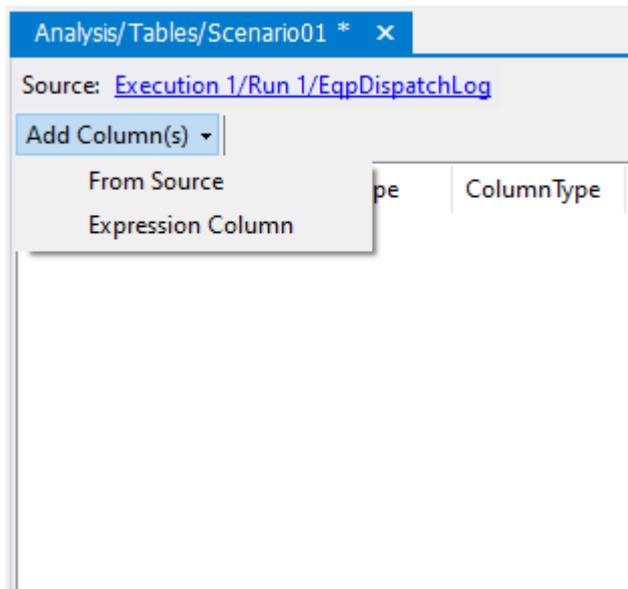
Table의 스키마 컬럼 리스트

## Data View

스키마에 따른 소스의 데이터

## Column 편집 및 추가

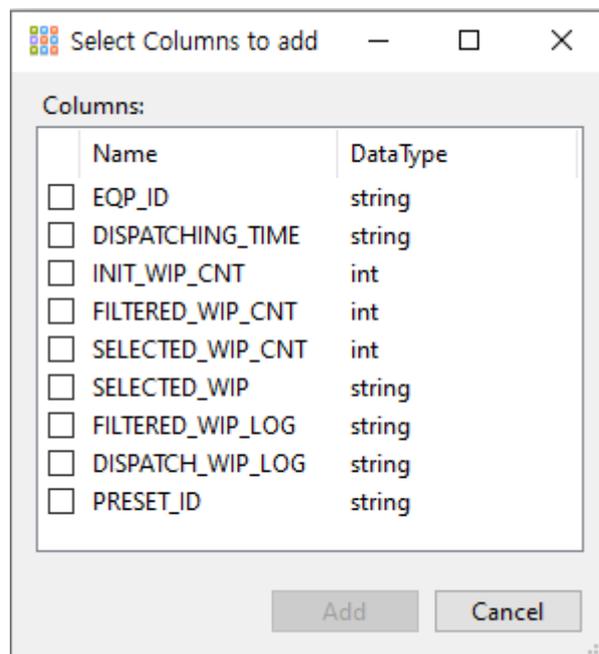
[Add Column(s)]를 클릭합니다. 연결된 Source의 Column을 추가할 경우, [From Source]를 클릭합니다. 새로운 Column을 추가할 경우 [Expression Column]을 클릭합니다.



Column 추가 메뉴

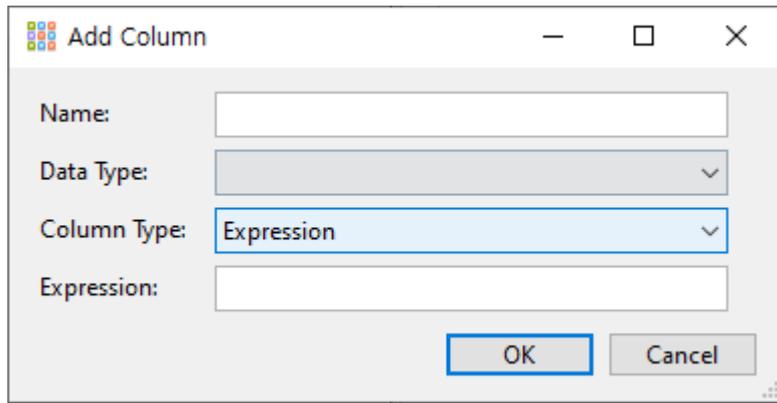
**[From Source]** : Table에 연결된 Source의 Column들을 Table에 추가합니다.

Table 편집화면에 추가되지 않은 Column들을 Select Columns to add Dialog에 표시합니다. 추가할 Column들을 선택하고 **[Add]**를 클릭합니다.



Source에 포함된 Column들 리스트

**[Expression Column]** : 기존의 Static Column들이나 새로운 Expression Column을 생성합니다.



Expression Column 추가 화면

### **Name**

컬럼 이름

### **Data Type**

컬럼 값의 이름

### **Column Type**

컬럼 타입

- Static: 기존의 컬럼 타입
- Expression: 새로운 컬럼 수식 타입

### **Expression**

수식이 표시되는 텍스트 박스

Figure 1: Expression Column Registration Example

Expression Column 등록 예시

## Load Data

Table 편집 화면 오른쪽 **[Load Data]**를 클릭한 경우, 설정된 Column을 Schema로 하여 Source의 정보를 보여줍니다.

DEMAND	AVG_LEADTIME	MIN_LEADTIME	MAX_LEADTIME	EQP01_Util	EQP02_Util	TARGET_RATE
200	121.875	41.6666666666667	183.333333333333	1.6369047164917	3.30687832832336	100
600	257.291666666667	41.6666666666667	450	5.60515880584717	9.92063522338867	45.8333333333333
1000	391.041666666667	41.6666666666667	716.666666666667	9.57341289520264	16.0300924777985	27.5
1400	524.553571428571	41.6666666666667	983.333333333333	13.3990579247475	21.8171300888062	19.6428571428571
1800	657.986111111111	41.6666666666667	1250	16.8712801933289	27.6041667461395	15.2777777777778
2200	791.382575757576	41.6666666666667	1516.66666666667	20.3435025215149	33.3912038803101	12.5
2600	924.759615384615	41.6666666666667	1783.33333333333	23.8157243728638	39.178240776062	10.5769230769231
3000	1058.125	41.6666666666667	2050	27.2879467010498	44.965277671814	9.16666666666667

Load Data 예시

# Analysis View

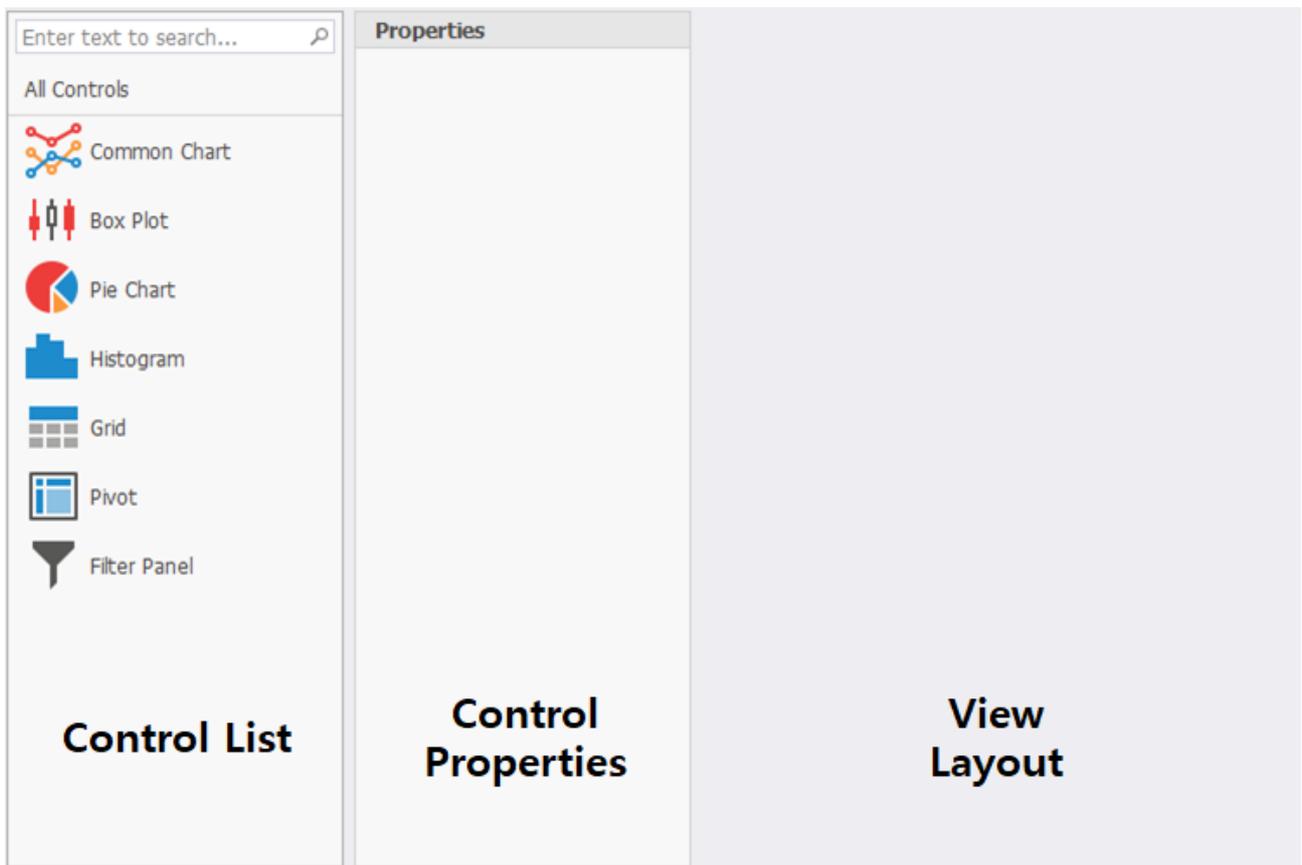
## 개요

Analysis View는 기존의 정의한 Table을 통해 사용자가 원하는 분석 화면을 만드는 기능입니다.

Analysis View를 구성하는 컨트롤들은 하나의 Table만 Source로 지정할 수 있으며, Source가 되는 Table은 미리 정의한 Analysis Table만 사용이 가능합니다.

## 화면 구성

Analysis View 화면은 크게 세 부분으로 구성되어 있습니다.



Analysis View 화면

## 컨트롤 목록(Control List)

화면에 추가 가능한 모든 컨트롤의 목록을 표시합니다.

- Common Chart
- Box Plot
- Pie Chart
- Histogram
- Grid
- Pivot
- Filter Panel

## 컨트롤 속성(Control Properties)

화면에서 선택된 컨트롤의 속성 목록을 표시합니다.

## 화면 레이아웃(View Layout)

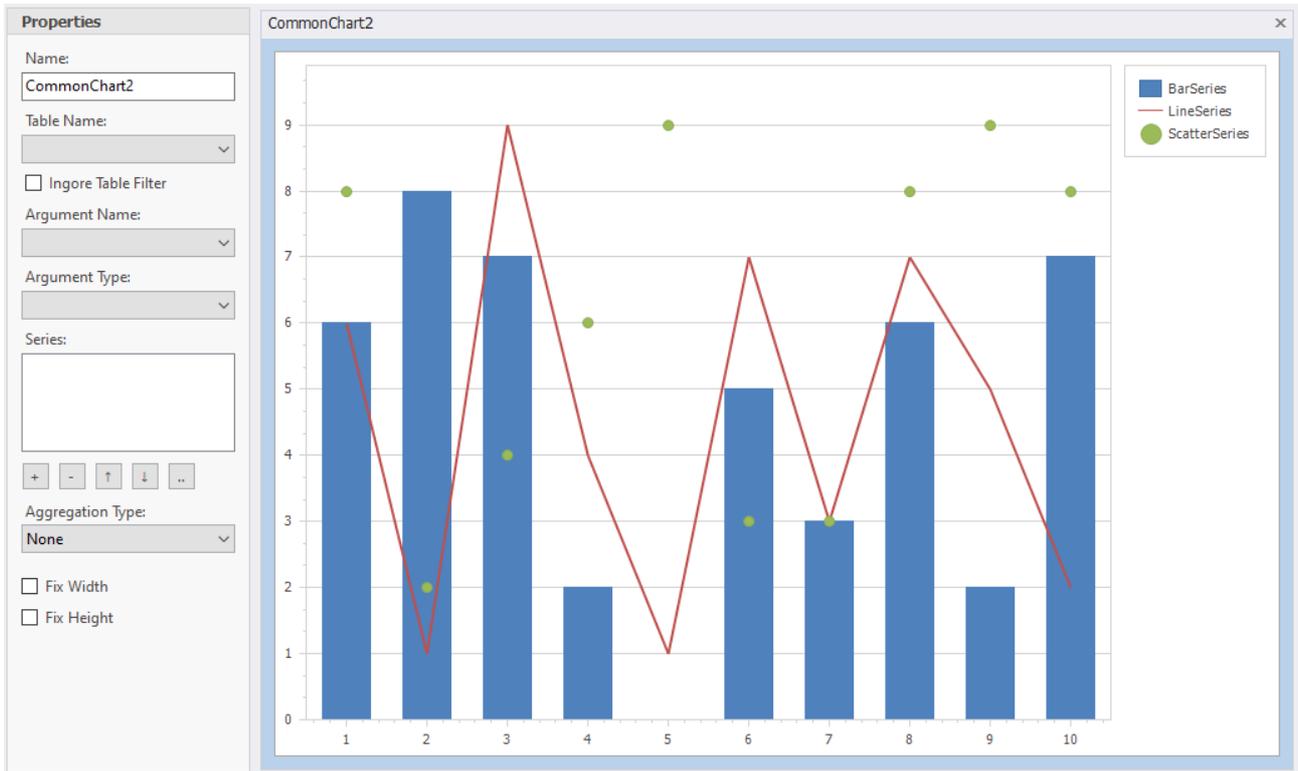
화면에 추가된 컨트롤의 레이아웃을 구성합니다.

---

## 컨트롤 속성

각 컨트롤은 소스가 되는 테이블의 데이터를 시각화하기 위해 사용자가 설정할 수 있는 여러 가지 속성을 제공하고 있습니다.

## Common Chart 컨트롤



Common Chart 컨트롤 편집화면

### Name

컨트롤의 이름

### Table Name

소스 테이블의 이름

### Ignore Table Filter

소스 테이블에 적용되는 필터 무시 여부

### Argument Name

X 축을 나타내는 컬럼 이름

### Argument Type

X축의 값 형식

- Qualitative: 각 값을 문자열 취급

- Numerical: 각 값을 숫자로 취급
- DateTime: 각 값을 날짜로 취급
- Auto: 자동으로 형식을 인식

## Series

차트에 추가될 시리즈 정보

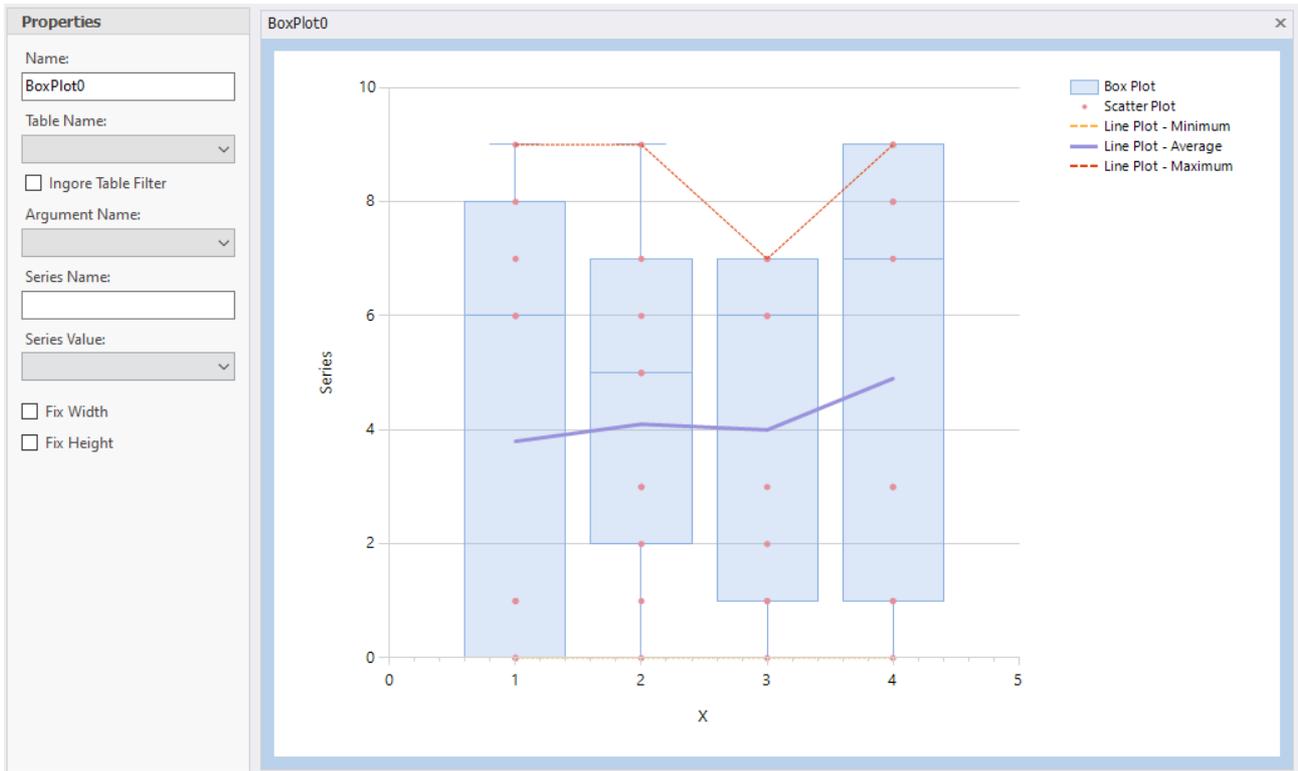
시리즈 속성 이름	설명
Name	시리즈의 이름
Type	시리즈의 유형 <ul style="list-style-type: none"> <li>• Line: 값을 선으로 표시</li> <li>• Bar: 값을 막대로 표시</li> <li>• Scatter: 값을 점으로 표시</li> </ul>
Value	Y 값을 나타내는 컬럼의 이름

## Aggregation Type

시리즈 내에 동일 X 값을 갖는 Y 값이 두 개 이상인 경우 표시 방식

- None: 모든 값을 표시함(시리즈 형식에 따라 사용 가능)
- Count: 값의 수를 표시
- Min: 값들의 최솟값을 표시
- Max: 값들의 최댓값을 표시
- Average: 값들의 평균을 표시
- Sum: 값들의 합을 표시

## Box Plot 컨트롤



Box Plot 컨트롤 편집화면

## Name

Box Plot 컨트롤의 이름

## Table Name

소스 테이블의 이름

## Ignore Table Filter

소스 테이블에 적용되는 필터 무시 여부

## Argument Name

X 축을 나타내는 컬럼 이름

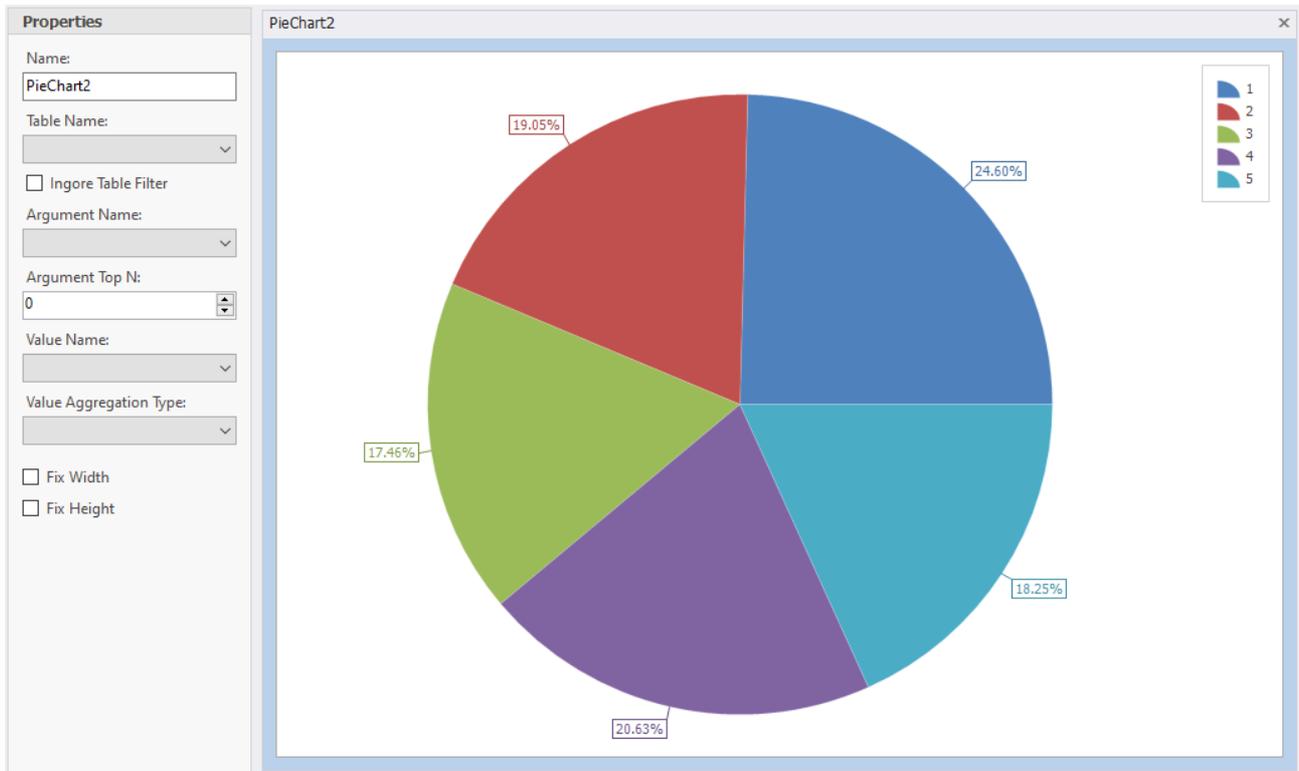
## Series Name

Y 축의 이름

## Series Value

Y 값을 나타내는 컬럼의 이름

## Pie Chart 컨트롤



Pie Chart 컨트롤 편집화면

### Name

Pie Chart 컨트롤의 이름

### Table Name

소스 테이블의 이름

### Ignore Table Filter

소스 테이블에 적용되는 필터 무시 여부

### Argument Name

영역(Slice)을 구분하는 기준이 되는 컬럼 이름

### **Argument Top N**

상위 N개의 값만 영역(Slice)을 구분해서 표시하고 나머지는 Others 영역으로 묶어서 표시

### **Value Name**

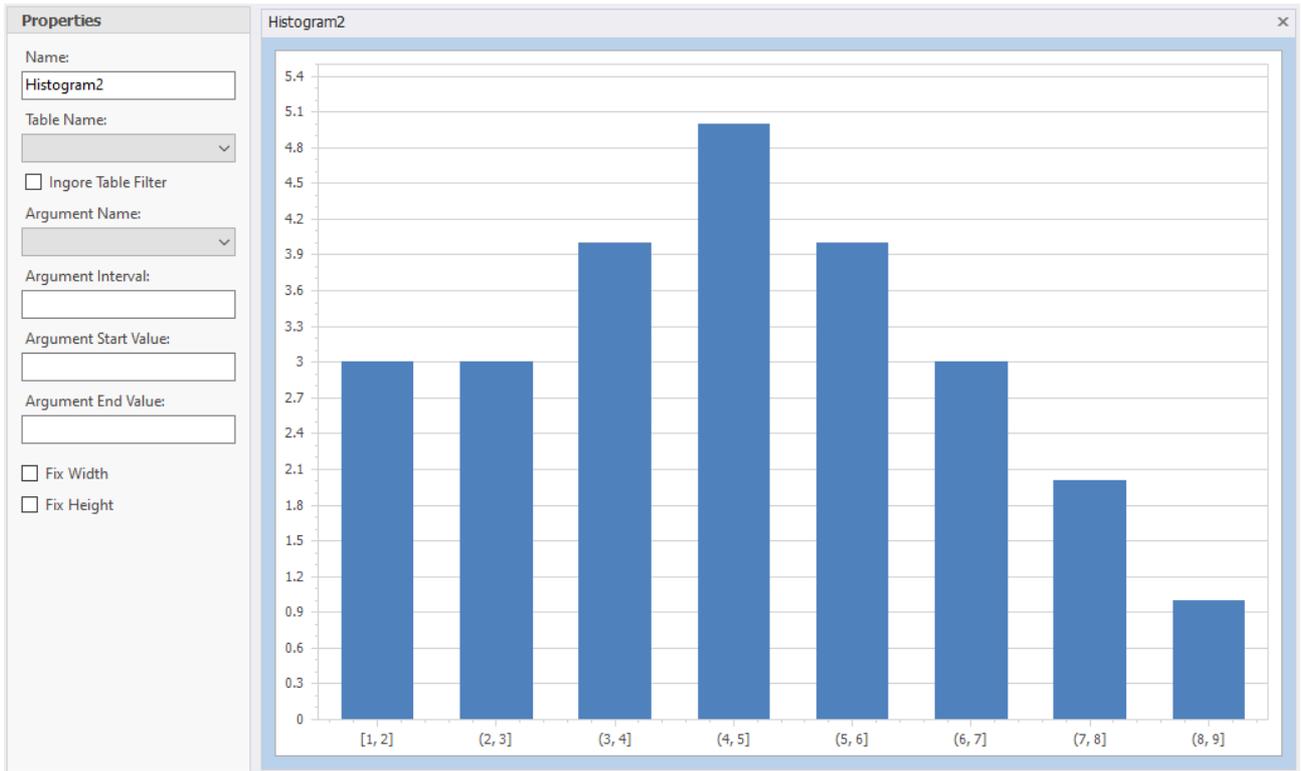
영역(Slice)들의 비율을 구성할 값의 컬럼 이름

### **Value Aggregation Type**

영역(Slice)의 비율 내에 동일한 값이 두 개 이상인 경우 표시 방식

- None: 모든 값을 표시함
- Count: 값의 수를 표시
- Min: 값들의 최솟값을 표시
- Max: 값들의 최댓값을 표시
- Average: 값들의 평균을 표시
- Sum: 값들의 합을 표시

### **Histogram 컨트롤**



Histogram 컨트롤 편집화면

## Name

Histogram 컨트롤의 이름

## Table Name

소스 테이블의 이름

## Ignore Table Filter

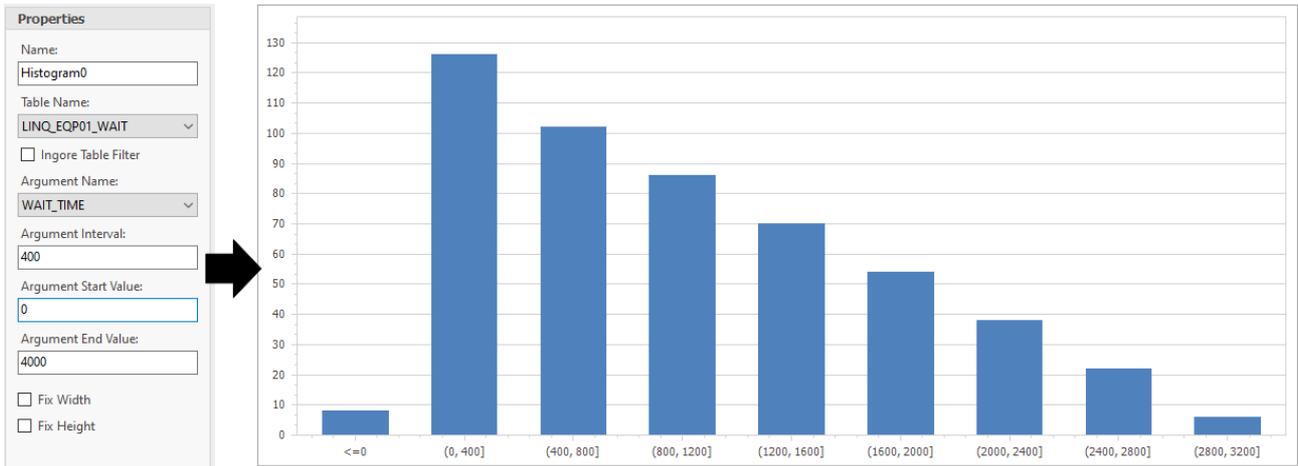
소스 테이블에 적용되는 필터 무시 여부

## Argument Name

X 축을 나타내는 컬럼 이름

## Argument Interval

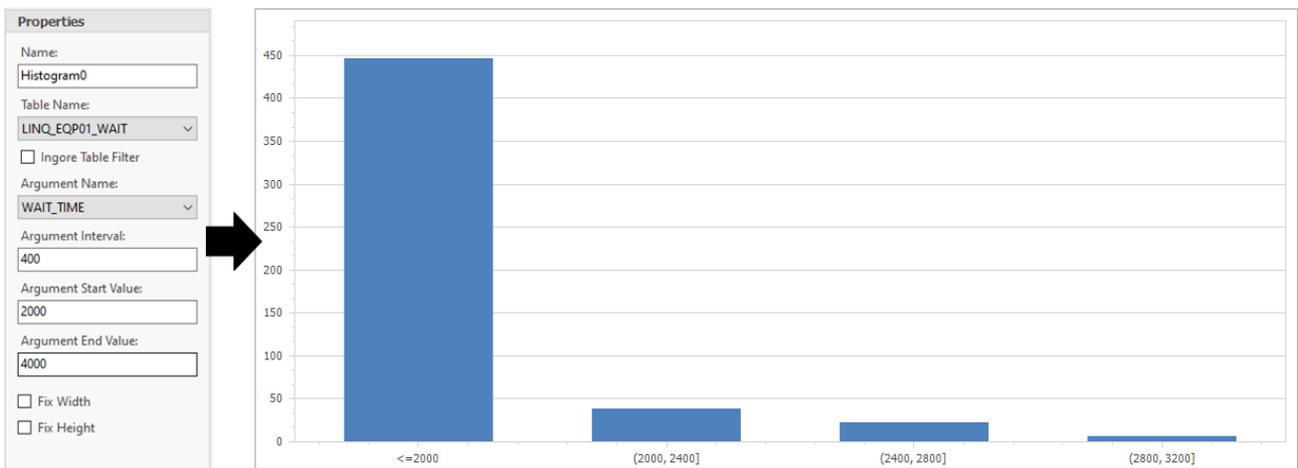
X 축 사이의 구간



0~4000 (구간 : 400) Histogram 예시

### Argument Start Value (Optional)

첫 번 구간의 시작 값

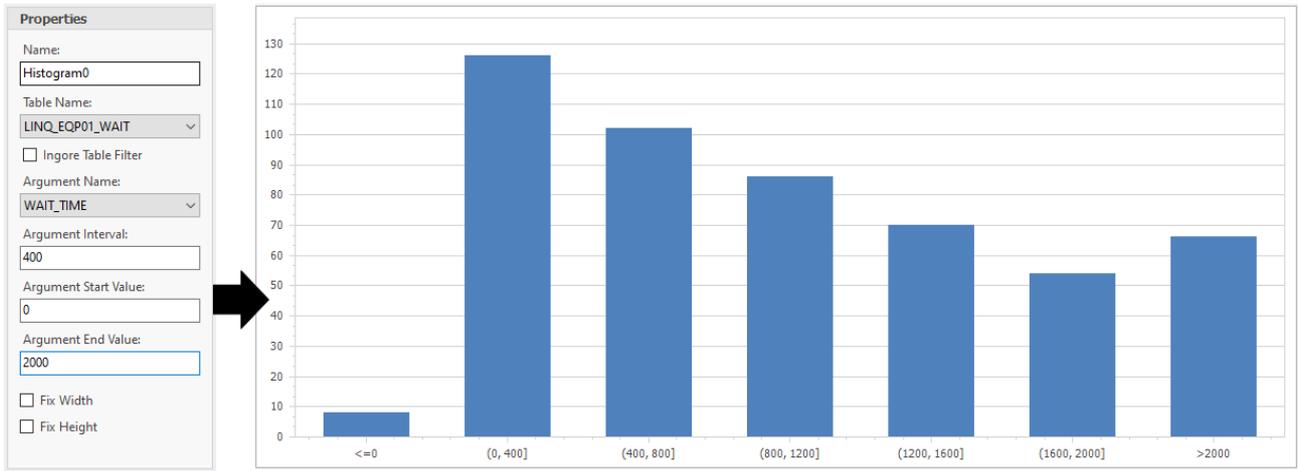


2000~4000 (구간 : 400) Histogram 예시

**i** 설정 값 이하는 한 구간(0~2000)으로 묶어서 표시가 됩니다

### Argument End Value (Optional)

마지막 구간의 끝 값



0~2000 (구간 : 400) Histogram 예시

**i** 설정 값 이상은 한 구간(2000~4000)으로 묶여서 표시가 됩니다.

## Grid 컨트롤

COL1	COL2	COL3
VALUE0	1	<input type="checkbox"/>
VALUE1	0	<input type="checkbox"/>
VALUE2	0	<input type="checkbox"/>
VALUE3	9	<input type="checkbox"/>
VALUE4	1	<input type="checkbox"/>
VALUE5	4	<input type="checkbox"/>
VALUE6	0	<input checked="" type="checkbox"/>
VALUE7	3	<input checked="" type="checkbox"/>
VALUE8	0	<input type="checkbox"/>
VALUE9	7	<input type="checkbox"/>

Grid 컨트롤 편집화면

## Name Grid

컨트롤 이름

## Table Name

소스 테이블의 이름

## Ignore Table Filter

소스 테이블에 적용되는 필터 무시 여부

## Columns

Grid의 컬럼 이름

## Pivot 컨트롤

The screenshot displays a PivotTable control interface. On the left is a Properties panel, and on the right is the PivotTable grid.

**Properties Panel:**

- Name: Pivot0
- Table Name: (dropdown menu)
- Ignore Table Filter
- Rows: (empty list)
- Columns: (empty list)
- Values: (empty list)
- Fix Width
- Fix Height

**PivotTable Grid:**

Drop Filter Fields Here

V1	C1				
R1	A	B	C	Grand Total	
ONE		1	2	3	3
TWO		4	5	6	6
Grand Total		4	5	6	6

Pivot 컨트롤 화면

## Name

Pivot 컨트롤 이름

## Table Name

소스 테이블의 이름

## Ignore Table Filter

소스 테이블에 적용되는 필터 무시 여부

## Row

Row 속성 이름	설명
Column Name	Row의 컬럼 이름
Sort Order	Row 컬럼의 정렬 방법 <ul style="list-style-type: none"><li>• None: 초기 정렬된 순서</li><li>• Ascending: 오름차순 정렬</li><li>• Descending: 내림차순 정렬</li></ul>

## Column

Columns에 위치할 컬럼 이름

Column 속성 이름	설명
Column Name	Column의 컬럼 이름



---

### Column 컬럼의 정렬 방법

Sort Order

- None: 초기 정렬된 순서
  - Ascending: 오름차순 정렬
  - Descending: 내림차순 정렬
- 

## Value

Rows와 Columns 사이에 값들을 나타낼 컬럼 이름

Value 속성 이름

설명

---

Column Name

Value의 컬럼 이름

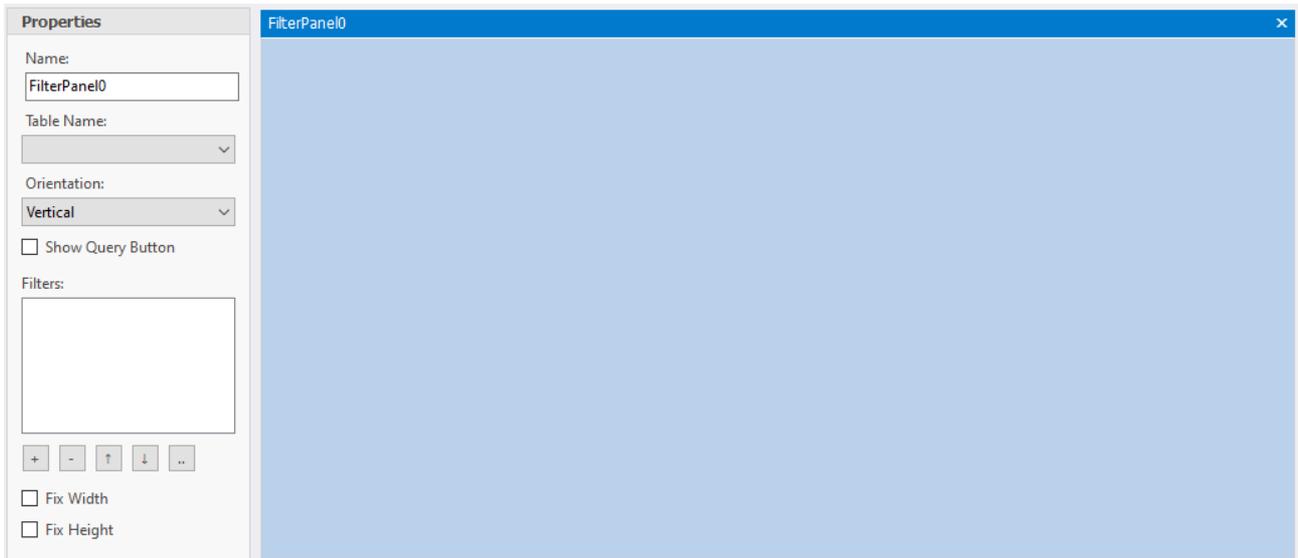
---

Aggregation Type

Value 내에 동일한 값이 두 개 이상인 경우 표시 방식

- None: 모든 값을 표시함
  - Count: 값의 수를 표시
  - Min: 값들의 최솟값을 표시
  - Max: 값들의 최댓값을 표시
  - Average: 값들의 평균을 표시
  - Sum: 값들의 합을 표시
- 

## Filter Panel



Filter Panel 편집화면

## Name

Filter Panel 이름

## Table Name

소스 테이블의 이름

## Orientation

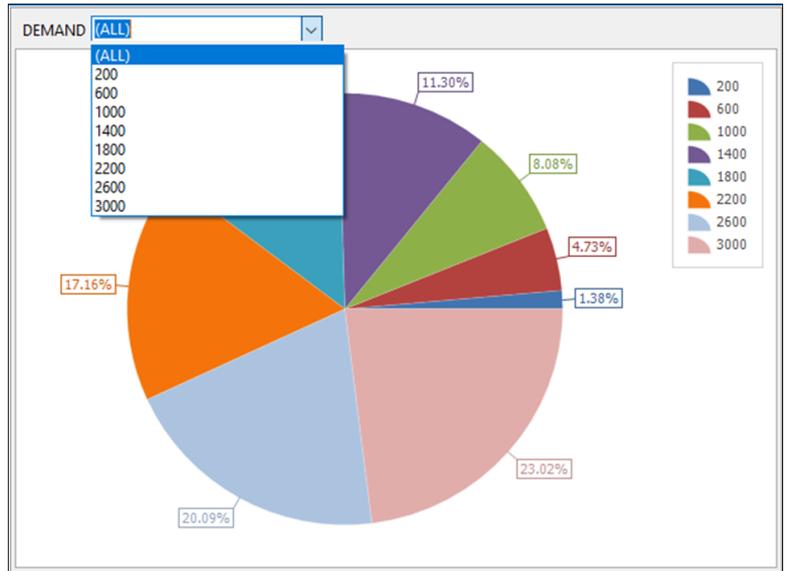
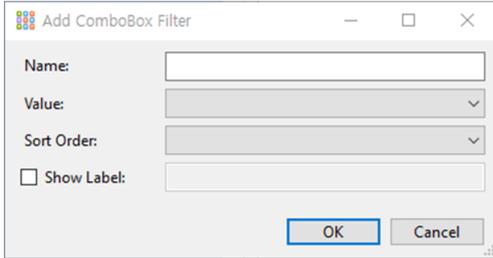
패널에 필터가 추가되는 방향

- Vertical : 세로 방향으로 필터가 추가
- Horizontal: 가로 방향으로 필터가 추가

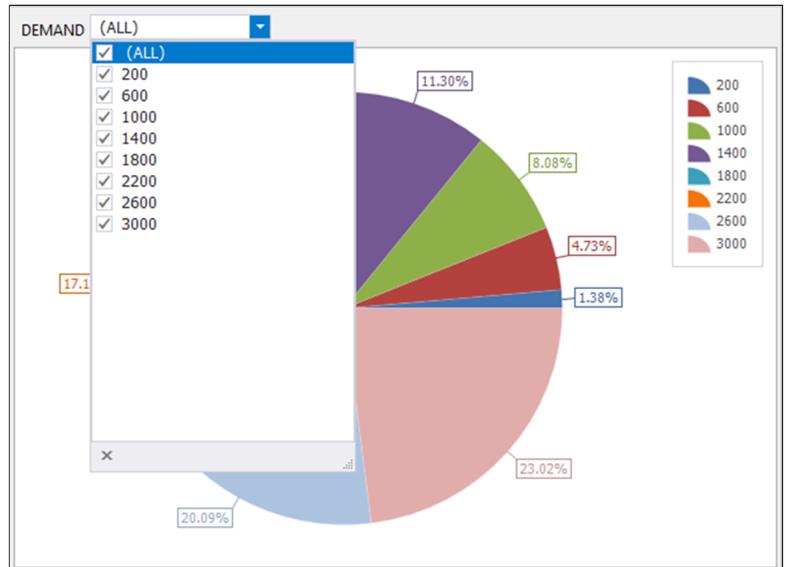
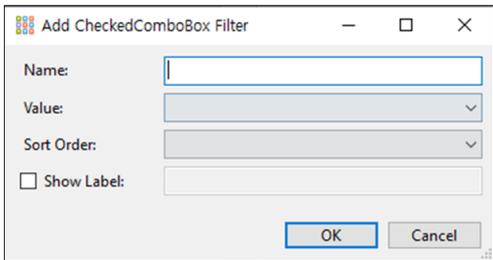
## Show Query Button Query

버튼 표시 여부를 선택

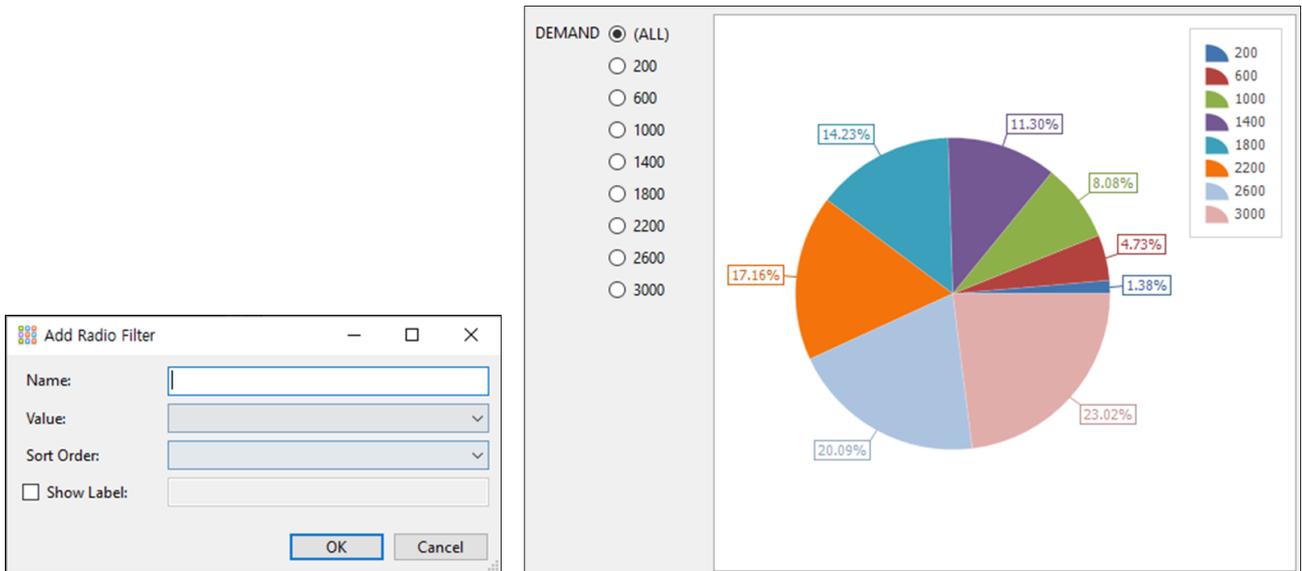
## Filters - ComboBox/CheckedComboBox/Radio Filter



ComboBox Filter 추가 예시



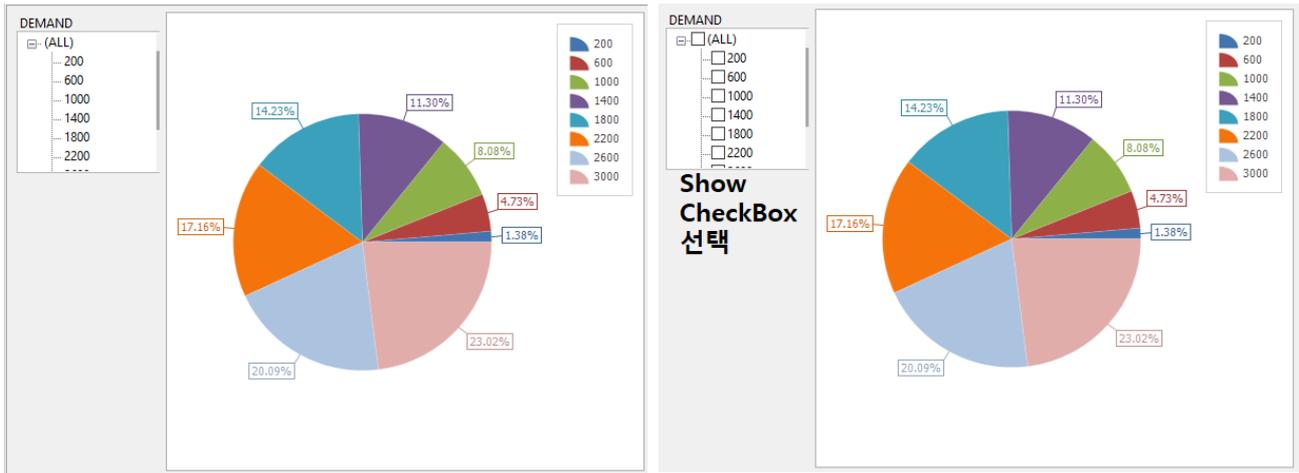
CheckedComboBox Filter 추가 예시



Radio Filter 추가 예시

속성 이름	설명
Name	ComboBox/CheckedComboBox/Radio Filter 이름
Value	필터링 대상 컬럼을 선택
Sort Order	<p>컬럼의 정렬 방법</p> <ul style="list-style-type: none"> <li>• None: 초기 정렬된 순서</li> <li>• Ascending: 오름차순 정렬</li> <li>• Descending: 내림차순 정렬</li> </ul>
Show Label	Filter Panel에 나타날 해당 Filter의 제목

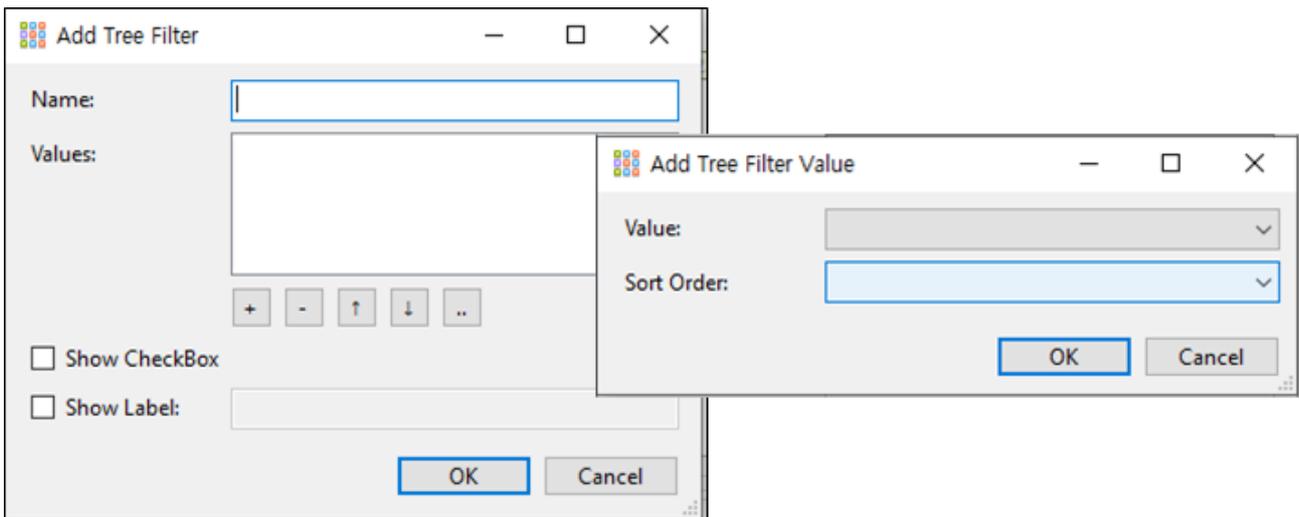
## Filters - Tree Filter



Tree Filter 추가 예시

속성 이름	설명
Name	Tree Filter 이름
Values	Filter 대상 컬럼을 선택
Show CheckBox	Tree 항목들 앞에 CheckBox 형태의 선택 옵션
Show Label	Filter Panel에 나타날 해당 Filter의 제목

### Filters - Tree Filter - Tree Filter Value



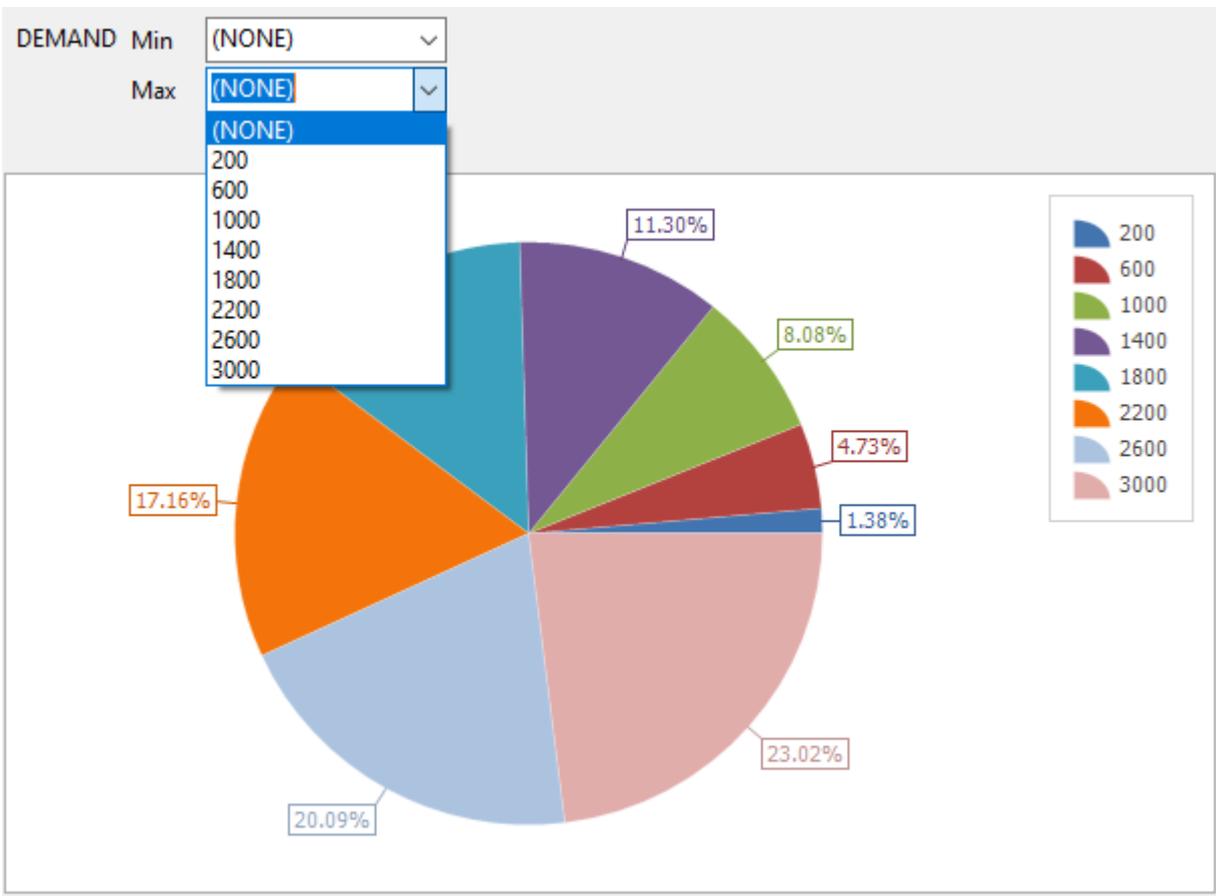
Tree Filter 화면, Tree Filter Value 화면

속성 이름	설명
-------	----

Name	Tree의 항목이 될 컬럼 이름
------	-------------------

Sort Order	컬럼의 정렬 방법
	<ul style="list-style-type: none"> <li>• None: 초기 정렬된 순서</li> <li>• Ascending: 오름차순 정렬</li> <li>• Descending: 내림차순 정렬</li> </ul>

**Filters - Range Filter**



Range Filter 추가 예시

속성 이름	설명
Name	Range Filter 이름
Value	필터링 대상 컬럼을 선택



Show Label

Filter Panel에 나타날 해당 Filter의 제목

## 레이아웃 설정

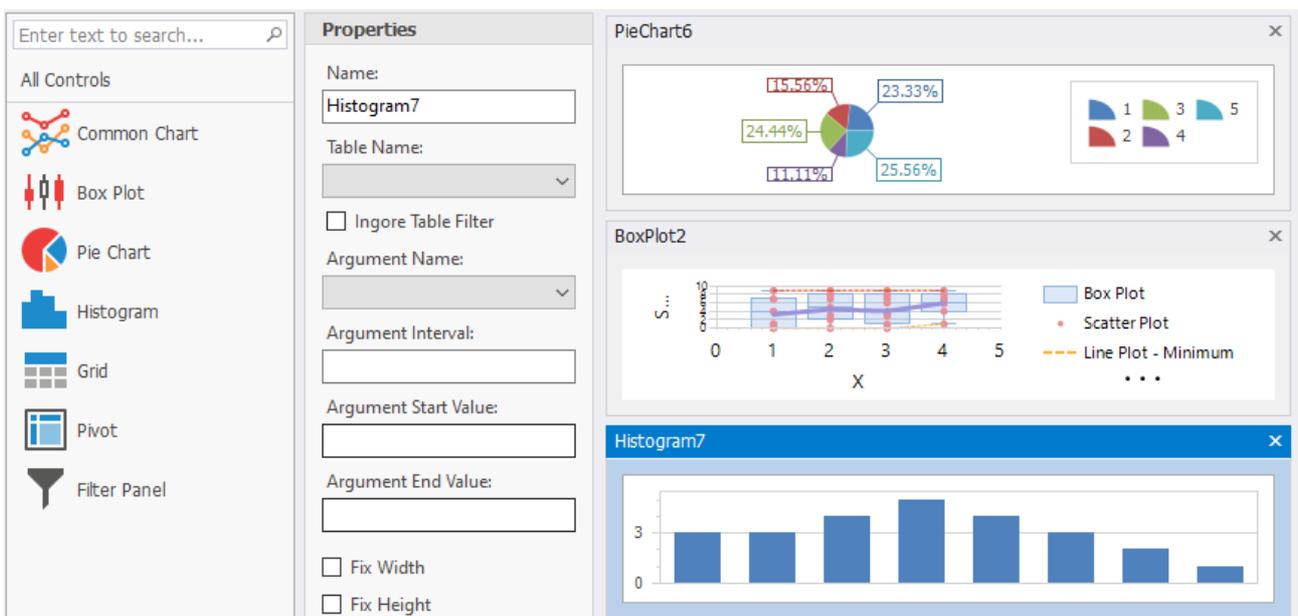
### 컨트롤 추가

분석 화면에서 추가하고자 하는 컨트롤을 마우스 왼쪽 버튼을 두 번 클릭하여 추가합니다.



컨트롤 추가 예시

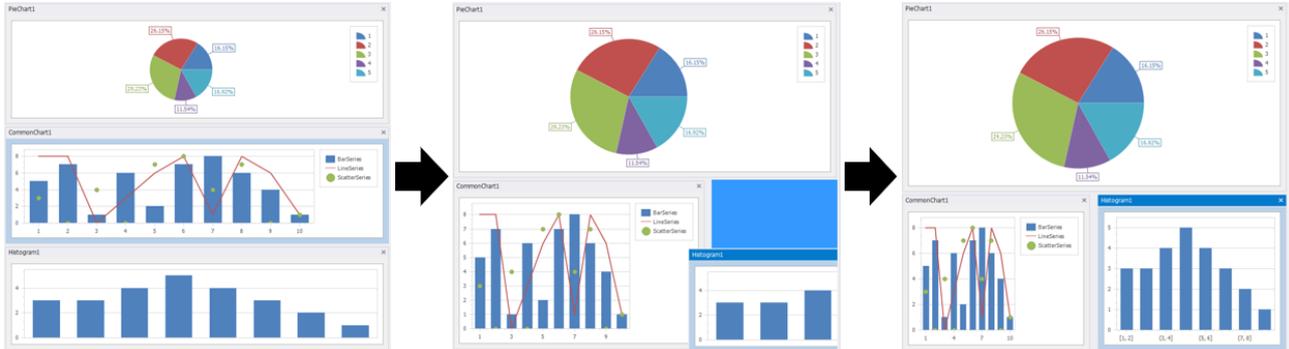
여러개의 컨트롤을 추가할 경우 아래와 같이 세로로 정렬됩니다.



여러개의 컨트롤을 추가했을 경우

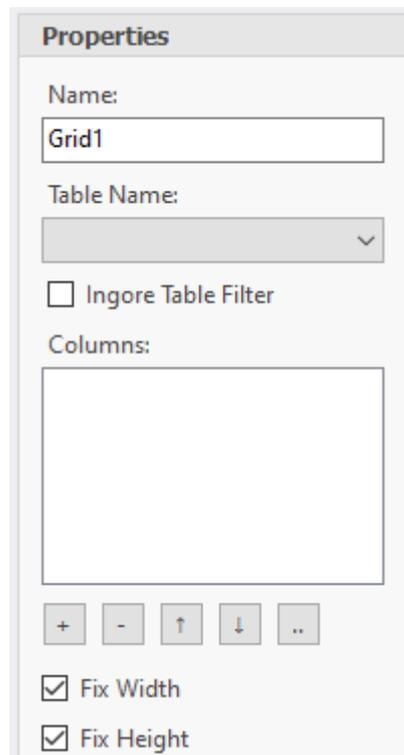
## 레이아웃 변경

분석 화면의 컨트롤은 추가 후 드래그 앤 드랍을 통해 컨트롤들의 위치를 변경하여 사용자 레이아웃을 완성할 수 있습니다.



드래그 앤 드랍을 통한 컨트롤의 레이아웃 재배치 예시

WISE 화면 사이즈가 작아져도 특정 컨트롤을 고정된 사이즈의 비율로 조정하게 하고 싶은 경우, 해당 컨트롤의 속성에서 Width와 Height를 고정하는 옵션을 설정합니다.



컨트롤의 레이아웃 옵션(Fix Width, Fix Height)

옵션 이름

설명

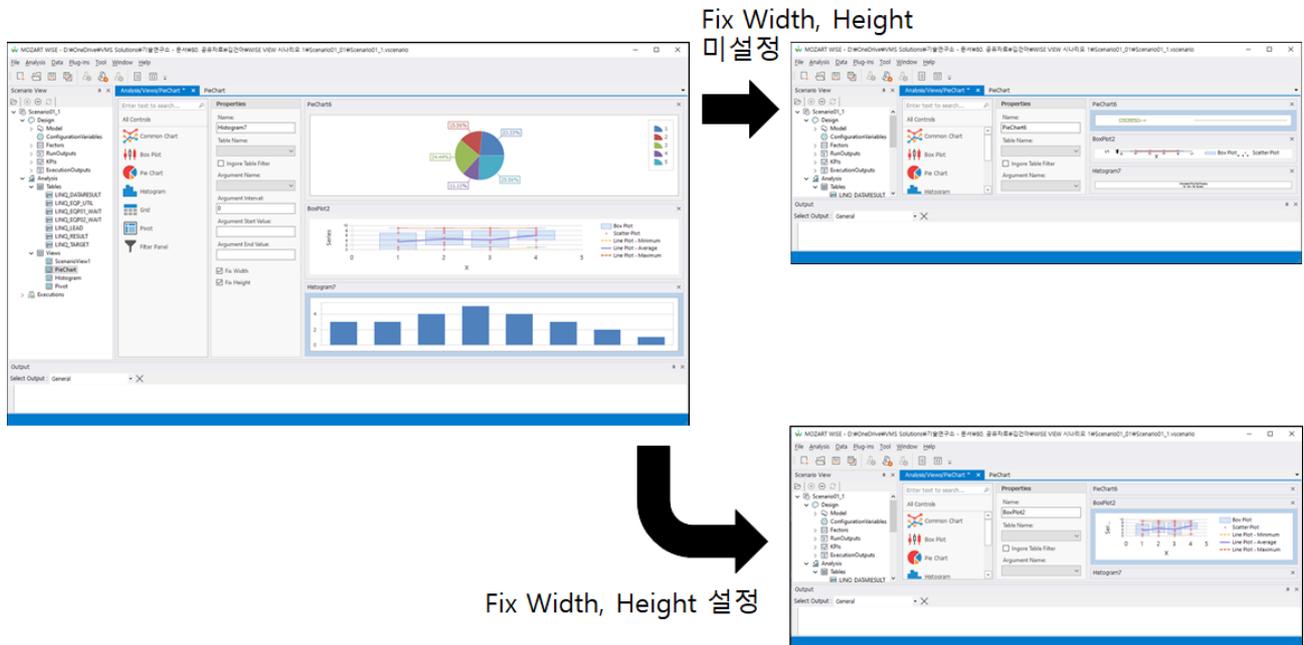


Fix Width

컨트롤 너비 고정

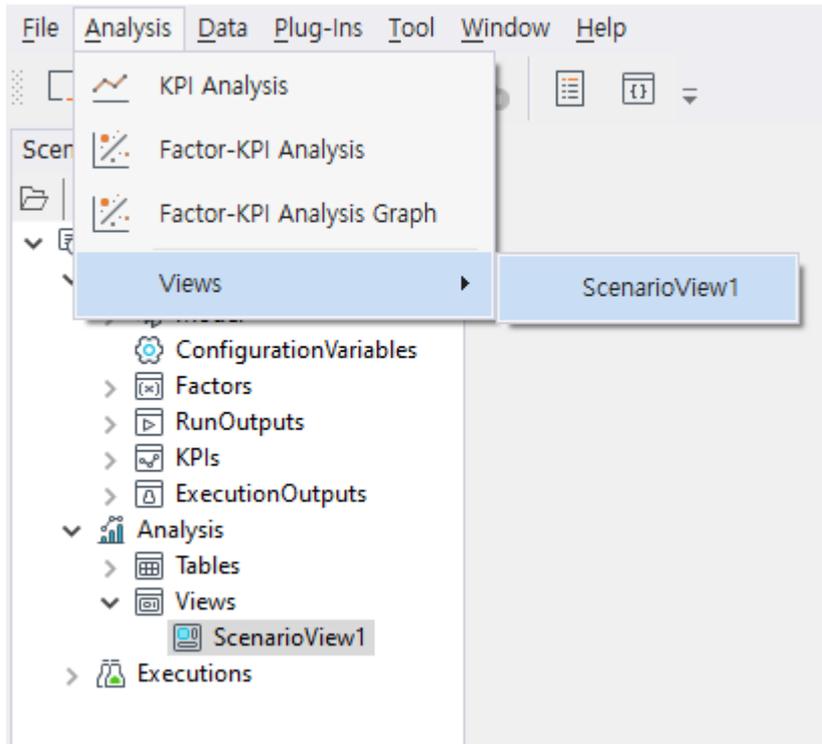
Fix Height

컨트롤 높이 고정

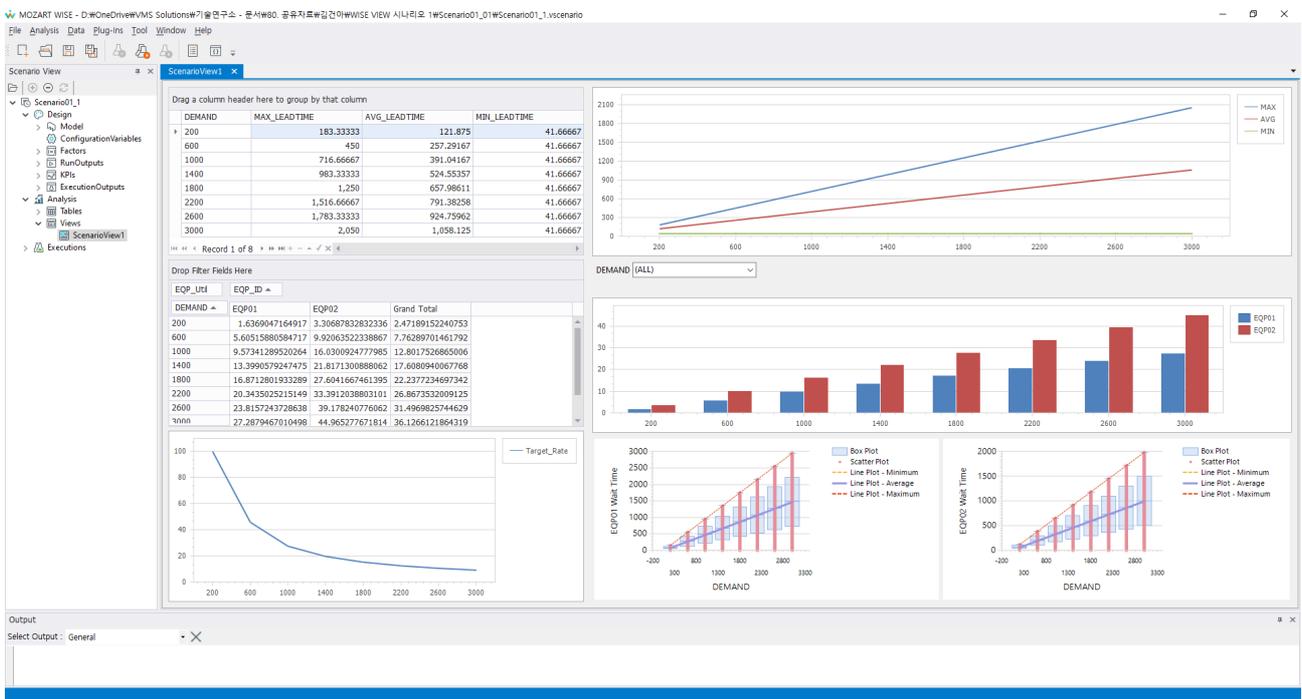


## Analysis View 메뉴

Analysis View를 생성하면, 메뉴의 Analysis/Views/[사용자 정의 분석 화면 이름]을 통해 분석 화면의 결과를 확인할 수 있습니다.



시나리오 편집기에서 Analysis View를 추가했을 경우 위의 메뉴 아래에 해당 View가 추가



메뉴를 통해 확인할 수 있는 View 화면 예시

# WISE 플러그인

## Multi Model Executor

WISE의 시나리오를 통해 다중모델과 다중 DLL을 사용하여 실행할 수 있도록 시나리오를 확장하는 플러그인입니다.

---

## Weight Optimizer

MOZART Simulation 엔진을 적용한 스케줄 모델을 대상으로 디스패칭 시점에 강화학습을 통해 스케줄시 사용하는 Preset Factor의 가중치를 최적화할 때 사용하는 플러그인입니다. 플러그인을 사용해서 가중치 최적화를 하기 위해 모델에 일정정도의 수정이 필요합니다.

---

## PEPG Optimizer (예정)

Parameter Exploring Policy Gradient를 사용하여 Simulation Argument 최적화를 지원하는 Optimizer입니다. 사용자가 Parameter와 KPI를 지정하면 일종의 강화학습을 통해 최적 파라미터를 찾을 수 있도록 실험을 반복 수행하도록 지원합니다. 실행대상 모델과 독립적으로 적용할 수 있어 범용적으로 사용할 수 있습니다.

---

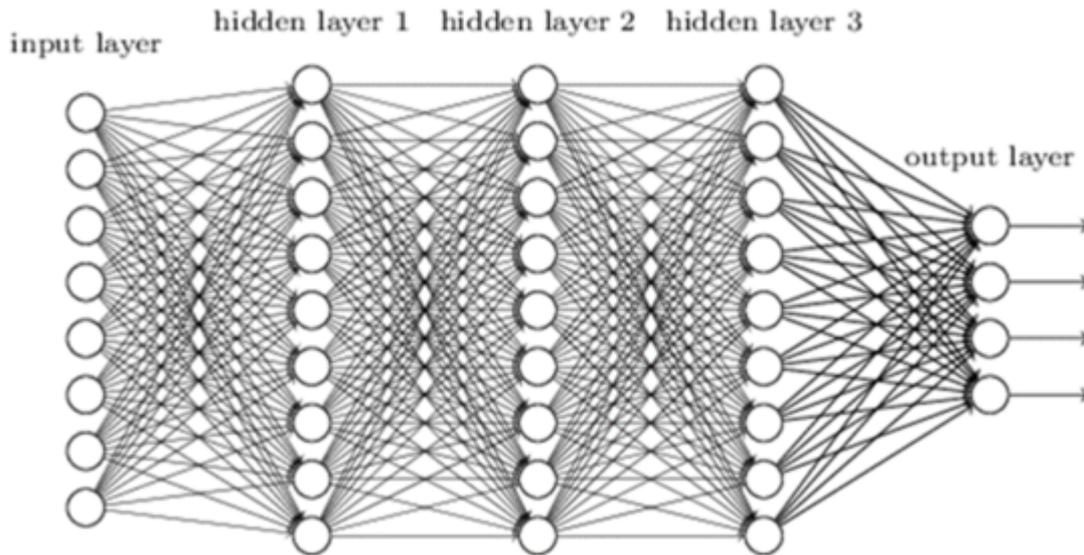
## AOWSA Optimizer(예정)

Advanced Optimal Weight Search Algorithm을 사용하여 Scheduler에 적용된 Preset Factor에 대한 최적 가중치를 찾아주는 Optimizer입니다. Decision Tree를 사용하여 가중치 영역을 효과적으로 분류하고 반복적으로 샘플링하여 최적의 가중치 값을 빠르게 찾을 수 있도록 지원합니다.

# Machine Learning Scheduler

## Machine Learning Scheduler 개요

Machine Learning Scheduler는 기존 MOZART에서 사용하던 선형 가중합방식(Linear Weighted Sum)을 벗어나 비선형 가중합방식(Non-Linear Weighted Sum)으로 의사결정을 내립니다.



비선형 구조로 인공지능 분야에서 사용하는 신경망(Neural Network)구조가 사용됩니다. 디스패칭 시, 각 작업물과 공장의 특성을 나타내는 입력값을 신경망에 통과시켜 비선형 합을 얻어내고, 선형 가중합 방식에서처럼, 더 높은 값을 지니는 작업물을 선택하는 방식으로 동작합니다.

---

## Machine Learning Scheduler 의 강점

Machine Learning Scheduler가 비선형 구조를 통해 가중합을 구함으로써 얻는 강점은 다음과 같습니다.

1. 의사결정에 사용할 입력 및 설명 변수 설계에 더 적은 노력이 들어가고, 자유도가 높습니다.
2. Weight Optimizer에서처럼 학습을 통해 더 나은 의사결정을 내릴 수 있습니다.

3. 선형 가중합 방식에 비해 의사결정의 자유도가 매우 높아, 학습을 통해 더 좋은 스케줄에 도달할 잠재력이 있습니다.
4. Weight Optimizer에 비해 학습 시 Hyper Parameter에 덜 민감하게 반응하여 사용이 편리합니다.

# 적용 절차

## 1. 필요한 참조 DLL 추가

Weight Optimizer 와 마찬가지로 구동을 위해 필요한 DLL 을 추가 설치하고 Project 에 참조를 추가합니다. 추가 파일은 아래의 8개 파일입니다.

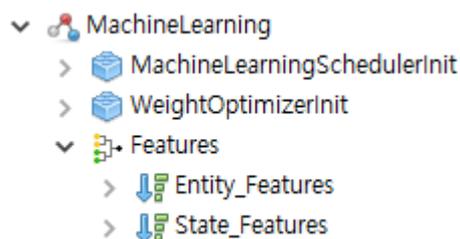
- Mozart.ML.dll
- tensorflow.dll
- TensorFlow.NET.dll
- NumSharp.Core.dll
- Google.Protobuf.dll
- System.Memory.dll
- System.RunTime.CompilerServices.Unsafe.dll
- System.ValueTuple.dll

상기 파일들은 별도 제공됩니다. Mozart ML Package 를 통해 설치할 수 있습니다. 파일이 설치된 이후에는 대상 Project 에 상기 파일 중 아래의 3개 파일을 참조 추가합니다.

- Mozart.ML.dll
- System.RunTime.CompilerServices.Unsafe.dll
- System.ValueTuple.dll

상기 파일중 하위 2개의 파일(System.RunTime.CompilerServices.Unsafe.dll, System.ValueTuple.dll) 은 WISE 에서 Machine Learning Scheduler 실행시 필요한 파일들입니다. 해당 파일이 없어도 모델의 빌드 오류가 발생하지 않지만 WISE 에서 시나리오별로 지정한 실행파일 폴더에 항상 포함되어 있어야 합니다.

## 2. ML Module Extension 추가



Module Extension 기능을 통해 Simulation 모듈에 Machine Learning 기능을 사용하기 위한 Component를 추가 할 수 있습니다.

## 2. 필수 Output 추가

Machine Learning Scheduler는 Tensorflow.Net에서 제공하는 저장기능을 통해 비선형 구조를 별도의 파일로 저장하는 방식을 채택하고 있습니다. 따라서 Weight Optimizer에서처럼 학습된 Preset Weight를 출력해주는 별도의 Output Schema는 필요없습니다. 단, KPI 테이블은 동일하게 가져가야 합니다. 테이블을 모델에 추가한 후 WISE 에서 Machine Learning Scheduler 플러그인에 테이블을 지정하면 지정된 테이블에 Run 중 발생하는 데이터를 자동으로 기록합니다.

- **KPI 추출을 위한 Output** : WISE 연계시 KPI 결과 참조를 위해 사용할 Output 입니다. WISE 에서 해당 테이블을 매핑시키는 작업이 추가적으로 필요합니다.

### KPI 추출을 위한 Output Schema

Column	Type	Description
KPI_ID	string	KPI 명칭
KPI_TYPE	string	KPI 종류(Sum, Count, Avg, Std)
KPI_VALUE	float	KPI
TIME	string	KPI 집계 시점. Simulation clock 기

Model안에서 KPI를 수집하는 별도의 함수를 호출해주면, 자동으로 해당 KPI의 합, 횟수, 평균, 편차를 수집하고 시뮬레이션 종료시 지정된 테이블에 각 타입별로 자동 기록됩니다.

 KPI\_ID,KPI\_TYPE,TIME 세가지 Column은 반드시 Primary Key로 지정되어 있어야 합니다.

## 3. 설명 변수 정의

Weight Optimizer에서는 Weight Factor 로 사전에 구현된 함수의 출력물을 학습의 설명 변수로 사용했습니다. Machine Learning Scheduler에서는 입력 및 설명 변수를 Feature 라 칭하고, 2가지 타입으로 구분하여 사용합니다.

- State Feature : 작업물과는 무관한 공장의 상태를 나타내기 위한 설명 변수 입니다. 출력 형태는 FeatureValue 타입이고, 동시에 여러 값을 Vector 형태로 출력 할 수 있습니다. 예를 들어 장비가 10대인 공장에서 각 장비 앞에 대기중인 작업물들이 전체 작업물에서 얼마만큼의 비율을 차지하는지를 계산하여 1x10 Vector에 넣어 출력해줄 수 있습니다.
- Entity Feature : Dispatching 시점에 고려되는 작업물들의 상태를 나타내기 위한 설명 변수입니다. 기존 Weight Factor와 정확하게 동일한 기능을 하지만, 출력 형태가 FeatureValue이기에 복수개의 출력 값을 줄 수 있습니다. 예를들어 공장에서 생산하는 모든 제품의 종류가 5종류인데 현재 고려되는 작업물의 종류가 3번 종류라면, [0 0 1 0 0] 같은 방식으로 OneHotCoding 된 출력물을 줄 수 있습니다.

```
1 //1xN의 Vector 형태로 출력할 수 있습니다.
2 public FeatureValue FIFO(ActiveObject target, ISimEntity entity, DateTime
3 {
4     float[] vector = new float[]{0,0,0,1,0};
5
6     return new FeatureValue(vector); //{0,0,0,1,0}
7 }
```

추가된 ML Module Extension의 Feature Component에서 Add Item을 통해 Feature 를 추가하고 사용할 Feature를 정의 할 수 있습니다. 정의된 Feature 들은 Machine Learning Scheduler 신경망의 입력값으로 사용됩니다.

 Entity Feature 에 기존 Weight Factor에서 개발되었던 로직을 그대로 복사한 뒤 출력값 형태를 WeightValue에서 FeatureValue 로 바꾼 뒤 사용 할 수 있습니다.

```
1 //FIFO Weight Factor를 Feature로 옮긴 예
2 public FeatureValue FIFO(ActiveObject target, ISimEntity entity, DateTime
3 {
4     FabLot lot = (entity as IHandlingBatch).Sample as FabLot;
5     double max = 10f; //days
6 }
```

```

7     TimeSpan waitTime = now - lot.DispatchInTime;
8
9     float weight = 0f;
10    if (waitTime.TotalDays > max)
11        weight = 1;
12    else
13        weight = (float)(waitTime.TotalDays / max);
14
15    return new FeatureValue(weight);
16 }

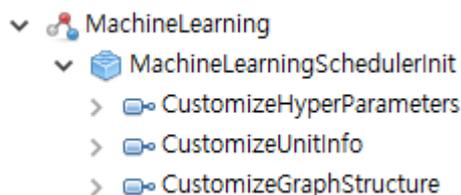
```

**i** 복잡한 로직이 포함된 Feature보다는 단순하지만 상황,작업물의 특성을 잘 나타내는 특성치가 좀 더 권장됩니다.

예) Tact Time, Processing Time, Due Date, Arrival Time, Wip Ratio 등

## 4. 학습 단위와 그래프 구조 초기화

Weight Optimizer에서는 하나의 장비가 하나의 학습 단위로 구성되어 있었습니다. Machine Learning Scheduler에서는 하나의 학습 단위에 여러개의 장비가 포함될 수 있고, 하나의 신경망을 통해 의사결정을 내리도록 할 수 있습니다. 이를 위해 학습 단위와 장비 사이의 연결관계를 만들어 줄 필요가 있습니다.



MachineLearningSchedulerInit/CustomizeUnitInfo 에서 AddItem을 누르고 사용자의 학습단위를 설정해 줍니다. 먼저 사용할 Feature와 학습대상이 될 장비를 구상한 뒤, 학습 단위인 Unit을 생성하여 연결해 줍니다.

```

1 //학습과 의사결정의 단위를 설정하는 예시입니다.
2 public IEnumerable<UnitInfo> CUSTOMIZE_UNIT_INFO0(ref bool handled, IEnumer
3 {
4     List<EntityFeatureInfo> eFeatures = new List<EntityFeatureInfo>();

```

```

5     string[] eNames = new string[] { "FIFO", "LAYER_BALANCE", "MAX_MOVE",
6     //정의된 Feature Method의 이름을 등록해두고
7
8     foreach (var name in eNames)
9     {
10        EntityFeatureInfo eInfo = new EntityFeatureInfo(name, 1);
11        eFeatures.Add(eInfo);
12    }
13    //그 이름과 출력 Vector의 크기를 넣어 FeatureInfo 타입으로 만들어줍니다.
14
15    List<UnitInfo> units = new List<UnitInfo>();
16    //Return 할 Unit의 Collection을 생성합니다.
17
18    foreach (var eqp in InputMart.Instance.FabEqp.Values)
19    {
20        if (eqp.ResGroup.Contains("PHT") && eqp.ResGroup.Contains("TF"))
21        {
22            NonSharingUnitInfo currentUnit = new NonSharingUnitInfo(eqp.EqpID, eqp.EqpName);
23            //Unit을 생성합니다.
24            currentUnit.AddEntityFeature(eqp.EqpID, eFeatures);
25            //Unit에 포함될 장비와 그 장비가 사용할 Feature Method를 등록해줍니다.
26            units.Add(currentUnit);
27        }
28    }
29
30    return units;
31 }

```

학습 단위를 초기화 했다면 학습 단위에서 사용할 신경망의 구조를 만들어 주어야합니다.

MachineLearningSchedulerInit/CustomizeGraphStructure 에서 AddItem을 누르고 수정할 수 있습니다. 이전 MachineLearningSchedulerInit/CustomizeUnitInfo 에서 초기화 된 Unit들의 정보가 전달되고 출력시 Unit과 사용할 Graph Pair의 Collection을 Return해줍니다.

```

1 //이전에 설정된 UnitInfo에서 사용할 Graph 구조를 만들어주는 예시입니다.
2 public Dictionary<UnitInfo, GraphInfo> CUSTOMIZE_GRAPH_STRUCTURE0 (IEnumerable<UnitInfo> units)
3 {
4     GraphInfo graphinfo = new GraphInfo();
5     //먼저 GraphInfo를 생성합니다.
6     graphinfo.AddLayerInfo(LayerType.Leaky_ReLu, true, 20);
7     //Graph에 신경망 Layer를 (Type, Bias 사용여부, Node 갯수) 를 지정하여 추가합니다.
8     graphinfo.AddLayerInfo(LayerType.Leaky_ReLu, true, 10);
9     graphinfo.AddLayerInfo(LayerType.Leaky_ReLu, true, 5);
10    graphinfo.AddLayerInfo(LayerType.Leaky_ReLu, true, 1);
11    //가장 마지막 Layer의 Node 갯수는 반드시 1이어야 합니다.
12

```

```
13     Dictionary<UnitInfo, GraphInfo> result = new Dictionary<UnitInfo, GraphInfo>();
14
15     foreach(var unit in unitInfos)
16     {
17         result.Add(unit, graphinfo);
18         //여기에서는 모든 Unit이 동일한 위의 Graph 구조를 이용하도록 연결해주었습니다.
19     }
20
21     return result;
22 }
```

## 5. 강화학습을 위한 Reward 정의

Weight Optimizer 와 마찬가지로 Machine Learning Scheduler도 장비가 디스패칭을 하는 시점에 선택에 대한 적절한 보상을 합니다. 보상을 통해 의사결정을 수행한 학습 단위의 신경망 내부 가중치를 업데이트 합니다. 신경망 내부 가중치는 사용자가 지정한 보상의 값을 높아지는 의사결정을 내릴 수 있도록 변화합니다. 따라서 디스패칭의 결과를 평가하여 적절한 보상을 할 수 있어야 합니다.

# Functions

## 공통 적용 방법

아래의 함수들은 Mozart.ML.MLSBinding 에 포함되어 있으며, 이는 Static Class 입니다. 사용을 위해 다음과 같이 사용 할 수도 있고.

```
1 using static Mozart.ML.MLSBinding
2 ///
3 Feed_Reward(eqp,"Setup",1.0f);
```

직접 클래스를 명시하는 방식으로도 사용할 수 있습니다.

```
1 using Mozart.ML;
2 ///
3 MLSBinding.Feed_Reward(eqp,"Setup",1.0f);
```

Method Name	Description
Initialize	MLS 기능을 사용하기 위한 객체의 초기화 함수
Feed_Reward	학습 시 의사결정에서 발생한 보상을 강화학습 객체에 전달하는 함수
Feed_KPI	학습 과정에서 발생하는 KPI를 수집하는 함수
Evaluation	의사결정시 사전에 정의된 Feature를 계산한 뒤 학습 단위의 신경망을 통과시켜 작업물 별 점수를 계산하는 함수
Select	Evaluation을 통해 점수가 계산된 작업물을 사전에 설정된 선택방법으로 선택해주는 함수

Learn	Evaluation 계산과 Select 결과를 통해 학습 단위의 신경망을 강화학습 하는 함수
IsLearnTarget	고려 중인 장비가 학습 대상인지 알려주는 함수
ToString	시뮬레이션 도중 학습 단위가 수집한 정보를 사용자가 지정한 요약수준으로 출력하는 함수

## Initialize :

함수 형태 :

```
public static void Initialize()
```

상세 :

```
MachineLearningSchedulerInit/CustomizeUnitInfo  
MachineLearningSchedulerInit/CustomizeGraphInfo  
MachineLearningSchedulerInit/CustomizeHyperParameters
```

해당 Component 에 사전에 구현된 세 함수를 실행하여 Machine Learning Scheduler 강화학습 Agent 를 초기화 합니다. Factory/FactoryEvents/OnEndInitialize 에서 공장이 초기화된 후 불러주는 것이 권장 됩니다.

```
1 public void ON_END_INITIALIZE0(AoFactory aoFactory, ref bool handled)  
2 {  
3     MLSBinding.Initialize()  
4 }
```

## Feed\_Reward :

함수 형태 :

```
public static void Feed_Reward(string source, string rewardName, float value)
```

- **string** source : Reward가 발생한 장비입니다.
- **string** rewardName : Reward의 이름입니다.
- **float** value : Reward의 양입니다.

상세 :

의사결정에 대한 Reward를 강화학습 Agent에 전달하는 함수입니다. 각 보상이 발생하는 조건 단위로 개별 함수를 만들어 그 내부에서 호출하는 것이 권장됩니다. 생성된 Reward 함수는 의사결정을 내리고 학습을 한 뒤 호출 해주어야 합니다.

```
1 [CustomRewardTiming(Category = "SETUP")]
2 //WISE Wizard 상에서 표시되기 위해서 추가해주어야합니다.
3 public static void SETUP(AoEquipment aeqp, IHandlingBatch selected)
4 {
5 //해당 함수는 Setup Reward를 발생시키는 함수 입니다.
6     if (!IsLearnTarget(aeqp.EqpID))
7         return;
8
9     float reward = MLSBinding.GetWISERewardValue("SETUP");
10 //WISE 상에서 정의된 Reward 값이 있다면 가져오는 함수 입니다.
11
12     if (float.IsNaN(reward))
13         reward = -600;
14
15 //만약 현재 장비에서 선택된 작업물이 Setup을 발생시킨다
16     if (aeqp.IsNeedSetup(selected))
17         MLSBinding.Feed_Reward(aeqp.EqpID, "SETUP", reward);
18 //Feed_Reward 함수를 통해 Reward를 발생시켜줍니다.
19 }
```

Feed\_Reward 함수의 부가 기능으로 KPI 자동 수집 기능이 있습니다. Feed\_Reward를 통해 수집된 Reward는 발생 한 장비와 종류 별로 합, 횟수, 평균, 편차가 자동으로 수집되어 시뮬레이

션 종료시 KPI 테이블에 자동으로 기록됩니다.

## Feed\_KPI :

함수 형태 :

```
public static void Feed_KPI(string name, float value)
```

```
public static void Feed_KPI(string name, float value, params string[] sources)
```

- **string name** : 수집하고자하는 KPI의 명칭입니다.
- **float value** : KPI 의 증가량입니다.
- **params string[] sources** : KPI 가 발생한 수준을 입력하는 부분 입니다. 예를들어 "Factory", "PhotoGroup", "PhotoEqp01" 과 같이 입력하게 되면 왼쪽부터 오른쪽으로 높은순에서 낮은순으로 수준별 자동 수집이 됩니다.

상세 :

공장에서 발생하는 Reward 이외의 KPI를 수집하기 위한 함수입니다. Feed\_Reward와는 다르게 의사결정 직후 호출 될 필요가 없이, 임의의 공장 Event 시점에서 호출되어도 무관합니다.

## Evaluation :

함수 형태 :

```
public static IList<IHandlingBatch> Evaluation(DispatcherBase db, AoEquipmen
```

- **DispatcherBase db** : DoSelect 함수에서 전달되는 DispatcherBase 입니다.
- **AoEquipment aeqp** : 현재 의사결정을 내리는 장비입니다.
- **IList<IHandlingBatch> lots** : 현재 의사결정을 내리는 장비에서 고려되는 작업물입니다.
- **IDispatchContext ctx** : DoSelect 함수에서 전달되는 DispatchContext입니다.
- **Func<DispatcherBase, List<IHandlingBatch>, IDispatchContext, IList<HandlingBatch>> defEvalFunc** : DoSelect에서 학습 대상이 아닌 장비에서 의사결정을 내리기 위해 사용될 기본 함수 입니다. Control.Evaluate 를 입력하면 됩니다.

## 상세 :

기존 DoSelect에서 작업물을 WeightFactor 에 따라 점수를 계산 하는 Control.Evaluate 함수와 같은 기능을 하지만, MachineLearning 기능에서 정의된 Feature 함수에 따라 점수를 계산하고 신경망을 통해 가중합을 수행하는 독립적인 Evaluation 함수입니다. Feature 계산 뒤 신경망을 통과한 가중합은 작업물 별로 부여되고, 작업물은 가중합이 높은 순서대로 정렬되어 List형태로 출력됩니다. 학습 대상이 아닌 장비에 대해서는 기존 WeightFator에 기반한 의사결정을 내리기 위해, Evaluation 시 기본함수를 입력값으로 전달합니다.

```

1 //Evaluation은 DoSelect 함수 안에서 Select전에 불러야합니다.
2 public IHandlingBatch[] DO_SELECT(DispatcherBase db, AoEquipment aeqp, ILi
3 {
4     var control = DispatchControl.Instance;
5     //var lotList = control.Evaluate(db, wips, ctx);
6     //기존 Evaluate 함수
7     var lotList = MLSBinding.Evaluation(db, aeqp, wips, ctx, control.Evalu
8     //Feature 기반 신경망 가중합을 기준으로 정렬된 Lot List
9
10    //Evaluation이 호출된 이후에 Select를 수행하면
11    //학습 대상 장비는 신경망에 기반한 의사결정을 할 수 있게됩니다.
12    var selected = control.Select(db, aeqp, lotList);
13
14    return selected;
15 }

```

## Select :

함수 형태 :

```
public static IHandlingBatch Select(string eqpID, IList<IHandlingBatch> lots
```

```
public static int Select_Index(string eqpID, int minIndex, int maxIndex)
```

- **string eqpID** : 현재 의사결정을 내리는 장비의 ID입니다.
- **IList<IHandlingBatch> lots** : 의사결정의 대안이 되는 가중합이 계산되고 정렬된 작업물 들입니다.
- **int minIndex, maxIndex** : 주어진 대안들을 선택할 때 Index로 접근해야하는 경우 사용하는 Index의 최소값과 최대값입니다.

### 상세 :

Evaluation 함수를 통해 신경망 가중합 점수가 계산된 작업물들 중 강화학습 Agent가 사용하는 Selection Method에 따라 작업물을 선택하는 함수 입니다. DispatchControl 의 Select 함수에 구현한 뒤 DoSelect에서 호출 해주는 방식이 권장됩니다.

```
1 //DoSelect에서 control.Select 로 불리는 Action 함
2 public IHandlingBatch[] SELECT1(DispatcherBase db, AoEquipment aeqp, IList
3 {
4     var selected = MLSBinding.Select(aeqp.EqpID, wips);
5     //강화학습 Agent에서 HyperParameter에 지정된 방식으로 선택합니다.
6     Learn(aeqp.EqpID, selected);
7     //선택 후 학습 함수를 호출해 주면 학습을 진행합니다.
8
9     return new IHandlingBatch[] { selected };
10 }
```

입력된 장비가 학습 대상에 포함되지 않을 경우, Agent가 평가 모드로 설정 되어 있을 경우 대안 중 0번 Index의 작업물을 Return합니다. 학습 대상일 경우 사전에 설정된 선택방법으로 선택하여 작업물을 Return 합니다. 현재 구현된 선택 방법은 3가지입니다.

- **Greedy** : 주어진 대안 중 최대 가중합을 가진 작업물을 선택합니다.
- **e-Greedy** : e의 확률로 최대 가중합이 아닌 대안 중 랜덤하게 선택합니다.

- **SoftMax** : 주어진 가중합으로 지수 가중 확률을 만든 뒤 해당 확률에 기반하여 선택합니다.

## Learn :

함수 형태 :

```
public static void Learn(string eqpID, IHandlingBatch selected)
```

- **string eqpID** : 현재 의사결정을 내리는 장비의 ID입니다.
- **IHandlingBatch selected** : 강화학습 Agent가 사용하는 선택 방법에 따라 선택된 작업물입니다.

상세 :

Evaluation을 마친 작업물들이 주어진 선택 방법에 따라 선택되었다면, 학습을 진행 할 수 있습니다. Select 함수가 작업물을 선택한 직후 호출 하는 방식이 권장됩니다. 학습 대상이 아닌 장비의 경우 아무 동작을 수행하지 않습니다. Agent가 평가 모드로 설정된 경우 신경망 학습을 수행하지 않고, 평가만 진행합니다.

## IsLearnTarget :

함수 형태 :

```
public static bool IsLearnTarget(string targetID)
```

- **string targetID** : 학습 대상인지 알아보기 위한 장비의 ID입니다.

상세 :

강화학습 Agent에서 학습대상으로 고려하고 있는 장비인지 True/False로 Return해주는 함수입니다. 기본적으로 MLSBinding 함수들은 내부에서 이 함수를 통해 학습 대상이 아닌 경우, 동작하지 않도록 구현되어있습니다. 사용자가 필요한 시점에 학습대상인지 알기 위해 임의 시점에 사용 할 수 있도록 제공되었습니다.

## ToString :

함수 형태 :

```
public static string ToString_KPI(AggregationType aType, int showDepthLevel,
```

```
public static string ToString_MachineLearningScheduler(AggregationType aType
```

- **AggregationType aType** : 호출 되는 시점에 Agent가 수집한 KPI를 어떤 방식으로 보여 줄지에 대한 Type입니다.
- **int showDepthLevel** : KPI가 복수 Level에 걸쳐 수집되었을 경우, 지정된 Level 까지만 표시해주기 위한 입력값입니다. Agent 단위까지만의 표시를 원할 경우 0을 입력하면 됩니다.
- **string kpiName** : 수집한 KPI중 해당 이름을 지니는 KPI에 대한 정보만 가져옵니다. 초기 값은 "ALL" 이고, "ALL"로 되어있을 경우 수집한 모든 KPI를 출력합니다.

상세 :

Agent 에서 학습이 일어나는 과정, 시뮬레이션이 진행되는 과정에서 수집된 KPI를 임의 시점에 살펴보기 위한 함수입니다. Simulation Module의 CustomEvent를 통해 일정 주기마다 MonitorInfo에 출력하여 학습 경과를 살펴보는데 유용합니다. 2가지 종류가 주어지는데 ToString\_KPI는 Feed\_KPI 함수를 통해 수집된 사용자 지정 KPI를 표시하는데 사용되고, ToString\_MachineLearningScheduler는 Feed\_Reward를 통해 수집된 학습과정에서 나오는 KPI(Reward, TD Error, Value Function)을 출력해줍니다.

# HyperParameters

## 공통 적용 방법

```
1 //MachineLearningSchedulerInit/CustomizeHyperParameters
2 public void CUSTOMIZE_HYPER_PARAMETERS0(HyperParameter hp, ref bool handle
3 {
4     hp[HyperParameterKeys.Run_Mode] = OptimizerRunMode.LEARN;
5     hp[HyperParameterKeys.Learning_Rate] = 0.001;
6     hp[HyperParameterKeys.Selector_Type] = SelectorTypes.eGreedy;
7     hp[HyperParameterKeys.Half_Life] = 480;
8     hp[HyperParameterKeys.Epsilon] = 0.01;
9     //hp[HyperParameterKeys.원하는 항목] = 원하는 값(Object)
10 }
```

MachineLearningSchedulerInit/CustomizeHyperParameters 함수를 생성하면 입력값으로 HyperParameter hp 가 주어집니다. HyperParameter는 사전에 주어진 Key를 이용한 Get/Set 을 허용하고 있으며, 주어진 `enum HyperParameterKeys` 에 포함되지 않은 값은 추가하거나 변경 할 수 없습니다.

Parameter Name	Type	Description
Loop_Index	int	WISE 상에서 몇 번째 Loop 인지를 나타내는 Index
Run_Index	int	WISE 상에서 몇 번째 Run 인지를 나타내는 Index
Run_Mode	OptimizerRunMode	Run의 모드 3가지 중 하나 -평가(Evaluation) -학습(Learn) -None
Lambda	float	현재의 의사결정이 얼마나 미래의 의사결정에 영향을 줄 것인지에 대한 Parameter. 1에 가까울수록 큰 영향력. 초기값, 권장값 = 0

Half_Life	float	Reward의 가치가 0.5배 되는데 걸리는 시간 (hour). 작을 수록 빠르게 감가상각됨. 권장값 = Simulation Period의 1/10~1/30
Random_Seed	int	Agent에서 Random한 값을 사용할 경우 (Selection, Neural Net Initialize) 사용하는 Random Stream의 Seed. 초기값 = 12345.
Selector_Type	SelectorTypes	선택 방법의 종류 -Greedy -e-Greedy -SoftMax
Epsilon	float	e-Greedy 선택방법에서 사용하는 확률 Parameter. 초기값 = 0.4. -WO 권장 값 : 0.0~0.4 -MLS 권장 값 : 0.0~0.05
Temperature	float	SoftMax 선택방법에서 사용하는 민감도 Parameter
Optimizer_Type	OptimizerType	SGD 방법의 종류 -Adam(권장) -RMSProp
Learning_Rate	float	SGD에서 사용하는 학습률. 값이 클수록 Preset Weight/신경망의 가중치 변화가 크다. Reward의 절대적인 크기와 밀접한 관련이 있다. -WO 권장 사항 : Reward 100단위에서 Learning Rate = 0.001~0.1 -MLS 권장 사항 : Reward 100단위에서 Learning Rate = 0.00001~0.001
First_Momentum_Beta	float	Adam/RMSProp 에서 사용하는 Gradient 관성 Parameter. 권장값 = 0.9~0.99

Second_Momentum_Beta	float	Adam에서 사용하는 Gradient 관성 Scaler. 권장값 = 0.99~0.999
Use_Immediate_Update	bool	WO에서 1회의 Learn 함수 호출 후 바로 WeightFactor를 Update 할지 여부. 기본값 = True
Parallel_Evaluation	bool	MLS에서 Evaluation 함수 호출 시 Lot 별 Feature 계산을 병렬로 할지 유무. 기본값 = True(권장, Evaluation Time 기준 약 20~40%의 성능개선)
Initial_Node_Min_Max_Range	float	MLS에서 첫 학습을 진행 할 시 신경망 Node Parameter 최초값 범위. 초기값 = 0.0000001 권장값 = 1/신경망 총 Node 수
CheckPointPath	string	MLS에서 학습/평가에 사용되는 신경망 정보가 담긴 meta 파일이 저장될 경로. 기본값 = C:\NN Graph

# Weight Optimizer

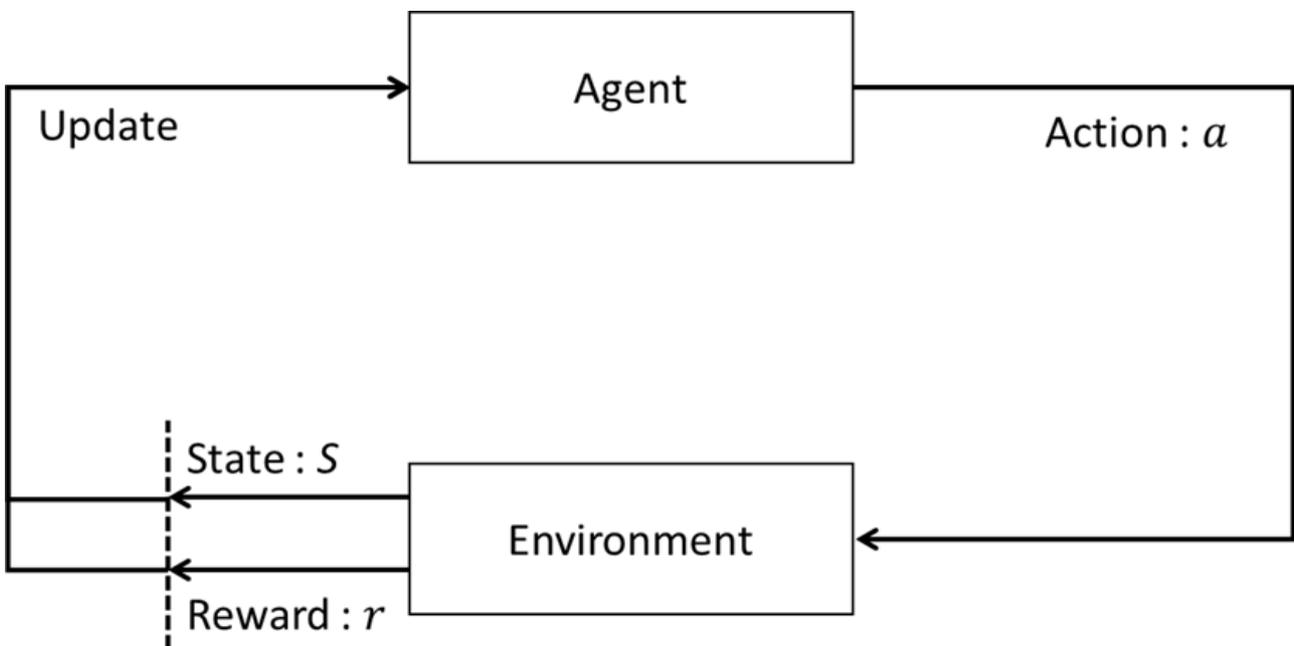
## Weight Optimizer 개요

Weight Optimizer 는 MOZART 로 생성된 모델 중 가중합방식의 Weight Preset 을 사용하여 스케줄 결과를 생성하는 모델에 적용될 수 있는 가중치 최적화 솔루션입니다.

Weight Optimizer 는 디스패칭 시점마다 의사결정에 대한 보상을 통해 최적의 팩터의 가중치를 찾도록 강화학습을 수행하는 구조이기 때문에 시뮬레이션 모델 수행중에 학습을 하도록 모델 자체의 수정이 필요합니다. 모델을 학습 가능한 구조로 만들고 WISE 를 통해 실행방법과 학습에 필요한 여러 파라미터를 설정하여 실행합니다.

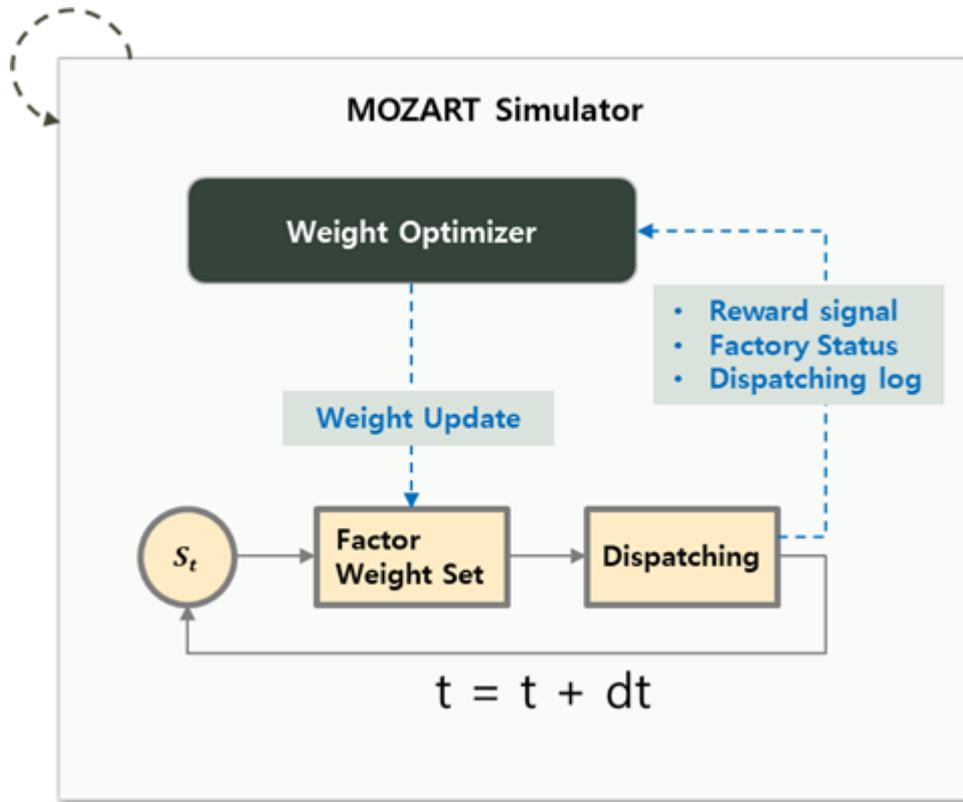
## Weight Optimizer의 실행 구조

Weight Optimizer는 기계 학습(Machine Learning)의 한 분야인 강화 학습(Reinforcement Learning)을 이용하여 디스패칭 의사결정시 사용되는 가중치를 조절해주는 도구입니다.

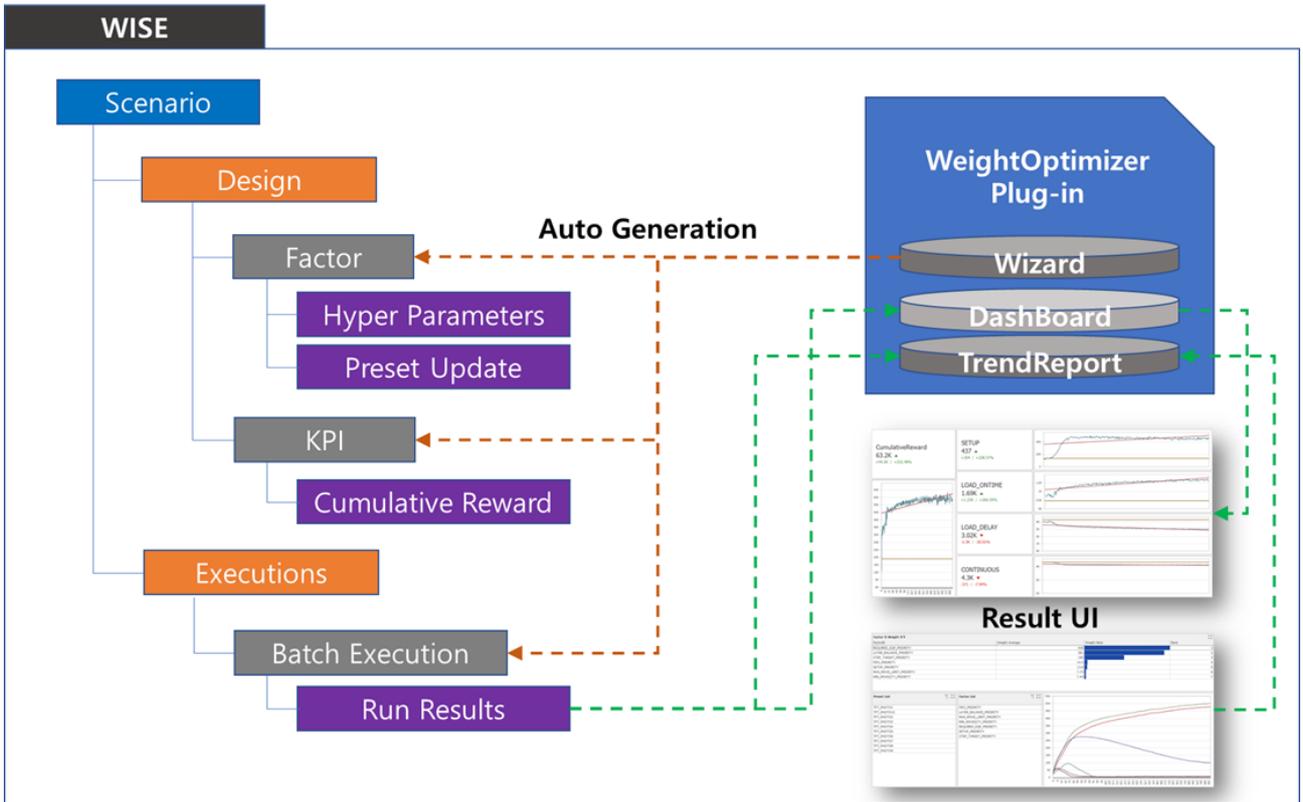


강화 학습을 구성하는 요소에는 의사결정자(Agent), 환경(Environment)이 있습니다. 의사결정자는 주어진 상태에서 의사결정(Action)을 내리고, 해당 의사결정은 환경에 영향을 줍니다. 영향을 받은 환경은 상태(State)가 변하며, 의사결정에 대한 보상(Reward)를 내어줍니다. 의

사결정자는 환경의 새로운 상태와 보상을 참고하여 자신의 의사결정 방식을 바꿉니다. 이러한 과정이 반복적으로 일어나며, 의사결정자는 더 나은 의사결정을 내릴 수 있게 됩니다.



공장 시뮬레이션에서의 디스패칭 의사결정은 여러 Factor들의 가중합이 가장 높은 작업물을 선택하는 방식으로 이루어집니다. 이러한 Discrete Event Simulation 모델에 강화학습을 적용시켜 보면, 의사결정자는 가중합 방식의 Dispatcher, 환경은 공장전체가 될 것입니다. Dispatcher가 가중합 방식으로 작업물을 선택하고나면, 해당 의사결정에 대한 보상을 얻고, 공장의 상태가 변하게 됩니다. Weight Optimizer는 이런 정보들을 전달받아 의사결정에 관여하는 가중치를 조절합니다. 반복적으로 가중치 조절이 이루어지면서, 더 높은 보상을 받을 수 있는 의사결정을 내릴 수 있는 가중치를 찾게 됩니다. 이때 의사결정을 내리는 경향은 사용자가 사전에 지정한 보상구조에 크게 영향을 받습니다. 사용자는 보상구조를 변경함으로써 원하는 경향의 정책을 학습을 통해 얻을 수 있습니다.



사용중인 스케줄러 혹은 플래닝모델에 Weight Optimizer 를 적용하기 위해서는 상기 설명한 바와 같이 모델이 가중합방식의 디스패칭 방식으로 구성되어야 합니다. 또한 모델에 맞는 적절한 보상구조를 설정하기 위해서는 모델에 일정한 수정을 해주어야 합니다. 강화학습을 하기 위해 모델에 준비가 끝났다면 WISE 환경에서 Weight Optimizer Plug-In 을 통해 준비된 모델을 사용하여 강화학습을 통한 가중치 최적화를 수행할 수 있습니다.

Weight Optimizer Plug-In은 WISE의 확장 기능 중 하나로써, 학습을 수행하고 결과를 보는데 유용한 기능을 제공합니다. 크게 Wizard, Dash Board, Trend Report 로 구성되어 있습니다. Wizard는 학습을 위해 수정된 모델을 반복 실행하는 Scenario 설정과 학습에 사용될 파라미터 값을 자동생성해주는 역할을 합니다. DashBoard는 KPI가 학습이 진행됨에 따라 어떻게 변화하였는지 한 눈에 보여주는 기능이고, TrendReport는 학습이 진행됨에 따라 Preset Weight가 어떻게 변화하였는지 추세를 보여주는 기능입니다.

## 플러그인 설치

Weight Optimizer 플러그인의 설치 는 WISE 의 다른 플러그인과 마찬가지로 [Plug-Ins]>[Plug-In Manager] 을 사용하여 설치합니다. [여기](#) 를 참조합니다.

# Weight Optimizer 적용 절차

1단계로 가중치 최적화 대상 시뮬레이션 모델에 학습이 가능하도록 준비합니다. 2단계로 MOZART WISE 상에서 Weight Optimizer 를 구동하기 위한 설정을 합니다. 이상의 과정을 통해 학습을 위한 준비는 마무리 됩니다. 준비가 끝나면 마지막단계로 반복적인 실험과 결과 분석을 통해 최적의 러닝 파라미터와 리워드 세팅 값을 찾습니다.

---

## 1단계 : 학습 모델 설정

학습을 위해 다음 작업을 진행합니다.

### 1. Machine Learning 에 필요한 참조 DLL 추가

Mozart 기반 Weight Optimizer 구동을 위해 필요한 DLL 을 추가 설치하고 Project 에 참조를 추가합니다. Weight Optimizer 구동을 위한 추가 파일은 아래의 8개 파일입니다.

- Mozart.ML.dll
- tensorflow.dll
- TensorFlow.NET.dll
- NumSharp.Core.dll
- Google.Protobuf.dll
- System.Memory.dll
- System.Runtime.CompilerServices.Unsafe.dll
- System.ValueTuple.dll

상기 파일들은 별도 제공됩니다. Mozart ML Package 를 통해 설치할 수 있습니다. 파일이 설치된 이후에는 대상 Project 에 상기 파일 중 아래의 3개 파일을 참조 추가합니다.

- Mozart.ML.dll
- System.Runtime.CompilerServices.Unsafe.dll
- System.ValueTuple.dll

상기 파일중 하위 2개의 파일(System.Runtime.CompilerServices.Unsafe.dll, System.ValueTuple.dll) 은 WISE 에서 Weight Optimizer 실행시 필요한 파일들입니다. 해당

파일이 없어도 모델의 빌드 오류가 발생하지 않지만 WISE 에서 시나리오별로 지정한 실행파일 폴더에 항상 포함되어 있어야 합니다.

## 2. 필수 Output 추가

Weight Optimizer 는 WISE 환경에서 구동되며, WISE 에서 실행결과를 실험간 연계하거나 KPI 를 추가로 정의하기 위해서는 모델에서 사전 정의된 두가지 Output 을 제공해야 합니다. 따라서 아래의 두 가지 Output 을 Schema 에 맞게 모델 Output 으로 추가해야 합니다. 테이블을 모델에 추가한 후 WISE 에서 Weight Optimizer 플러그인에 테이블을 지정하면 지정된 테이블에 Run 중 발생하는 해당 데이터를 기록합니다.

- **KPI 추출을 위한 Output** : WISE 연계시 KPI 결과 참조를 위해 사용할 Output 입니다. WISE 에서 해당 테이블을 매핑시키는 작업이 추가적으로 필요합니다.
- **Weight 학습결과 Output** : Weight Optimizer 수행시 동일 모델을 반복해서 학습하게 되며, 매 실행 Run 별 최종 학습된 Preset Weight 값을 기록하기 위한 테이블입니다. WISE 에서 해당 테이블을 매핑시키는 작업이 추가적으로 필요합니다.

### KPI 추출을 위한 Output Schema

Column	Type	Description
KPI_ID	string	KPI 명칭
KPI_VALUE	float	KPI 값
TIME	string	KPI 집계 시점. Simulation clock 기준 시점

### Weight 학습결과 Output Schema

Column	Type	Description
PRESET_ID	string	Preset 정보에 사용된 PRESET_ID
FACTOR_ID	string	Preset 정보에 포함된 Factor ID
FACTOR_VALUE	float	Preset 정보에 설정된 Factor 별 학습된 가중치 값
TIME	string	학습 종료시점

### 3. 학습 대상 설정

가중치 최적화는 기본적으로 장비단위 학습을 하도록 구성됩니다. 따라서 학습을 하기 위해서는 시뮬레이션 모델 중 대상이 되는 장비와 해당 장비가 사용할 Preset 을 지정해 주어야 합니다. 기본 모델에서 장비별 Preset 이 정의 되지 않은 경우(대부분의 경우 장비별로 지정하지는 않음)에는 장비별로 Preset 을 만들어주어야 합니다.

#### 1) 학습 대상 장비 지정

일반적으로 전체 장비에 적용을 하기보다는 주요한 스케줄 대상 장비그룹에 대해서만 학습을 하도록 함에 따라 상기 함수에서 장비 ID 리스트를 반환하도록 코드를 작성합니다. 아래는 장비그룹이 "EG01"인 장비들만 대상으로 학습을 하도록 하는 예제입니다.

```
1 public IEnumerable<string> CUSTOMIZE_TARGET_EQP_LIST0(ref bool handled, IE
2 {
3     List<string> eqps = new List<string>();
4
5     foreach (var eqp in InputMart.Instance.EduEqp.Rows)
6     {
7         if (eqp.ResGroup.Contains("EG01"))
8             eqps.Add(eqp.EqpID);
9     }
10
11     return eqps;
12 }
```

#### 2) 학습 대상 장비에 대한 개별 Preset 정보 작성 및 입력

일반적으로 스케줄 로직을 설정할 때는 주로 장비그룹단위로 Preset 을 설정합니다. 즉, 동일 장비그룹내의 장비들은 모두 같은 Preset 을 사용하게 됩니다. 그러나 Weight Optimizer 에서는 개별 장비별로 학습을 하게 됨으로 모든 장비가 독립적인 Preset 을 사용하도록 입력값을 만들어 주어야 합니다. 즉 사용자가 수작업으로 장비별로 사용할 Preset 을 만들어 입력하고, 장비정보에서 장비별 적용 Preset Mapping 해주어야 합니다.

 Preset 정보를 저장하는 테이블은 반드시 Primary Key 설정이 되어 있어야 합니다. WISE 에서 해당 테이블의 Weight 값을 업데이트하면서 실행을 해야 함으로 만

일 기존 사용하는 Model 에 Key 설정이 안되어 있는 경우 이를 추가해 주어야 합니다.

### 3) 학습 대상 Preset Factor 지정

학습 대상 장비에서 사용하는 Preset Factor 를 지정합니다. 대상 Preset 에 포함된 모든 Factor 를 학습에 고려할 수도 있고, 본 함수에서 일부 Factor 를 제외하여 학습에 반영할 대상만 지정할 수 있습니다. 구현 대상 함수는

WeightOptimizer/OptimizerInit/CustomizeTargetFactorList 함수 입니다. 본 함수는 상기 지정된 대상 장비별로 호출 됩니다.

```
1 public IEnumerable<Mozart.SeePlan.DataModel.WeightFactor> CUSTOMIZE_TARGET
2 {
3     List<WeightFactor> factors = new List<WeightFactor>();
4
5     EduEqp eqp = InputMart.Instance.EduEqp.Rows.Where(x => x.EqpID == targ
6     if (eqp == null)
7         return factors;
8
9     foreach (var f in eqp.Preset.FactorList)
10    {
11        factors.Add(f);
12    }
13
14    return factors;
15 }
```

## 4. 강화학습을 위한 Reward 시점 정의

Weight Optimizer 는 장비가 디스패칭을 하는 시점을 기반으로 해당 선택에 대해 적정한 보상을 함으로써 디스패칭을 통해 선택된 Lot 이 받은 점수를 평가하고 이를 통해 당시 Preset Factor Weight 를 조정하여 더 많은 보상을 받을 수 있는 Factor 가중치를 설정하도록 학습합니다. 따라서 디스패칭의 결과를 평가하여 적정할 보상을 할 수 있어야 합니다.

Weight Optimizer 는 작업물과 장비의 상태변경시점에 디스패칭에 대한 보상을 정의할 수 있도록 구성되어 있습니다. 아래의 두가지 함수의 구현을 통해 보상시점과 보상의 정도를 정의할 수 있는 지점만들면 WISE 의 Weight Optimizer 플러그인과 연동되어 WISE 상에서 보상지점과 보상의 정도를 조정해가면서 다양한 실험을 수행할 수 있습니다.

`WeightOptimizer/RewardControl/GetEqpReward` : 장비 상태 변경 시점에 보상을 줄 수 있도록 설정 가능 함. `LoadingStates` = {**SETUP, BUSY, IDLERUN, IDLE, PM, DOWN, WAIT\_SETUP**}.

`WeightOptimizer/RewardControl/GetEntityReward` : 작업물 상태 변경 시점에 보상을 줄 수 있도록 설정가능한 함수. `EntityState` = {**HOLD, MOVE, WAIT, RUN, OUT\_WAIT**}. 본 함수의 구현시 작업물의 현재 상태와 `nextState` 를 확인하여 보상여부를 판별할 수 있습니다.

상기 함수 외에 시뮬레이션의 임의 시점에 보상을 할 수 있도록 설정할 수 있습니다. 상기 함수의 정의를 추가하는 것을 포함하여 임의 시점에 보상을 정의하는 함수를 만들 때 지정된 확장 속성을 추가합니다. 확장 속성명은 `CustomRewardTiming` 입니다. 설정 예시는 아래와 같습니다. 해당 속성을 가진 `public` 함수의 경우 WISE 에서 Weight Optimizer 플러그인을 통해 대상 보상의 적용여부와 보상정도(값)을 지정할 수 있습니다. 이를 위해서 사용해야 하는 함수들은 아래에 설명합니다. `Category` 및 `Description` 속성은 Weight Optimizer 플러그인에서 Reward Setting 화면에 Reward 별로 표시되는 값입니다.

```
1 [CustomRewardTiming(Category = "CUSTOM", Description = "SETUP 발생시 PENALTY")
2 public void SETUP_REWARD(AoEquipment aeqp, LoadingStates nextState, ref bool isReward)
3 {
4     // 장비가 학습대상 장비인지 판별함
5     if (!IsLearnTarget(aeqp.EqpID))
6         return;
7
8     // WISE 에서 설정한 REWARD 값을 찾습니다.
9     float setupPenalty = GetWISERewardValue("SETUP");
10    if (float.IsNaN(setupPenalty))
11        return;
12
13    Feed_Reward(aeqp.EqpID, setupPenalty);
14    Feed_KPI("SETUP", 1.0f);
15 }
```

## WISE 바인딩을 위해 필요한 보상 체계 설정 함수

*Namespace : Mozart.ML.WoBinding*

`static` 함수 집합임으로 `static` 한정자를 사용하여 `using` 문을 구성해야 합니다. 아래는 설정 예시입니다.

```
using static Mozart.ML.WoBinding;
```

- **LearningTarget**: 학습대상 장비로 지정된 장비인지 여부 확인
- **GetWISERewardValue**: WISE의 Weight Optimizer 플러그인을 통해 설정된 Reward 설정 값을 반환합니다.
- **Feed\_Reward**: Weight Optimizer에 Reward를 등록합니다. 장비별로 학습을 함에 따라 함수 호출시 대상 장비를 지정해주어야 합니다.
- **Feed\_KPI**: 모델에 추가한 KPI Output Schema에 KPI를 기록합니다. 마지막 Argument인 `incremental = false`인 경우에는 입력된 값이 그대로 기록되며, 기본값(true)을 사용하는 경우 누적 값을 기록합니다.

## 5. 학습 함수 호출코드 추가

학습을 위해서 디스패칭 시점에 선택대상 작업물의 평가 점수(Feature)를 추출하는 함수와 최종 작업물 선택 함수를 호출하도록 코드를 추가해주어야 합니다.

### 1) 작업물별 평가점수 추출 코드 추가방법

시뮬레이션 모듈에서 디스패칭 대상 작업물을 평가하는 부분은

`DispatcherControl/DoSelect` 함수입니다. 디스패칭 시점에 각 대기 작업물의 평가한 후 우선순위 팩터별 점수를 학습을 위한 입력 값으로 추출하기 위해

`Mozart.ML.WoBinding.Feed_FactorEvaluation` 함수를 호출합니다. 본 함수는 기본 정의에서 평가가 끝난 지점에서 호출하도록 처리합니다. 아래는 예시입니다. 만일

`DispatcherControl/DoSelect`를 자체적으로 구현한 모델을 사용하는 경우에도 함수 호출 순서는 모든 평가가 끝난 후입니다.

```
1 public IHandlingBatch[] DO_SELECT1(DispatcherBase db, AoEquipment aeqp, IL
2 {
3     var control = DispatchControl.Instance;
4
5     if (wips.Count == 0)
6         return null;
7
8     var lotList = control.Evaluate(db, wips, ctx);
9
10    var evalLots = new List<IHandlingBatch>((int)(lotList.Count * 1.5));
11    foreach (var entity in lotList)
```

```

12     {
13         if (entity is LotGroup<ILot, Step>)
14         {
15             evalLots.AddRangeCast(entity.Contents);
16         }
17         else
18         {
19             evalLots.Add(entity);
20         }
21     }
22
23     // Extract Learning Input(Factor Value)
24     Mozart.ML.WoBinding.Feed_FactorEvaluation(aeqp, evalLots);
25
26     var selected = control.Select(db, aeqp, evalLots);
27
28     if (control.IsWriteDispatchLog(aeqp))
29         aeqp.EqpDispatchInfo.AddDispatchInfo(evalLots, selected, aeqp.Target);
30
31     return selected;
32 }

```

## 2) 작업물 선택 및 학습 코드 추가방법

평가가 끝난 작업물을 선택하는 함수는 `DispatcherControl/Select` 함수입니다. 본 함수에서 `Mozart.ML.WoBinding.Select` 함수를 사용하여 작업물을 선택하고, 선택한 작업물과 앞서 입력된 작업물별 평가점수 (Feature Value) 를 사용하여 학습을 하도록 처리합니다. 학습을 수행하는 함수는 `Mozart.ML.WoBinding.Learn` 함수 입니다. 아래 코드는 함수 구현 예시입니다. 만일 `DoSelect` 함수 전체를 사용자가 재정의해서 사용하는 경우에는 선택하는 부분에 예시와 같은 방식을 사용하여 학습을 할 수 있는 코드를 삽입하면 됩니다.

```

1 using static Mozart.ML.WoBinding ;
2 public IHandlingBatch[] SELECT1(DispatcherBase db, AoEquipment aeqp, IList
3 {
4     IHandlingBatch selected = Select(aeqp.EqpID, wips);
5     Learn(aeqp.EqpID, selected);
6     return new IHandlingBatch[] { selected };
7 }

```

## 2단계 : 학습 실행 준비

준비된 Weight Optimizer 적용대상 모델은 MOZART WISE 환경에서 Learning 을 실행합니다. 따라서 기본적으로 MOZART WISE 를 설치해야 하며, Weight Optimizer 적용을 위한 플러그인을 추가적으로 설치해야 합니다. 설치방법은 [플러그인 설치](#) 를 참조합니다.

### 시나리오 생성

WISE 의 시나리오를 새로 만듭니다. 시나리오명과 시나리오 폴더를 지정하고, 상기 1단계 과정에서 생성한 모델을 지정합니다. 모델을 실행시킬 수 있는 엔진 dll 위치를 지정하면 기본 설정은 끝납니다. 해당 모델의 구동 후 WISE 상에서 모델별 UI 를 직접 실행시키고자 하는 경우 "Model UI Assembly" 와 "Model UI Config" 파일을 지정합니다.

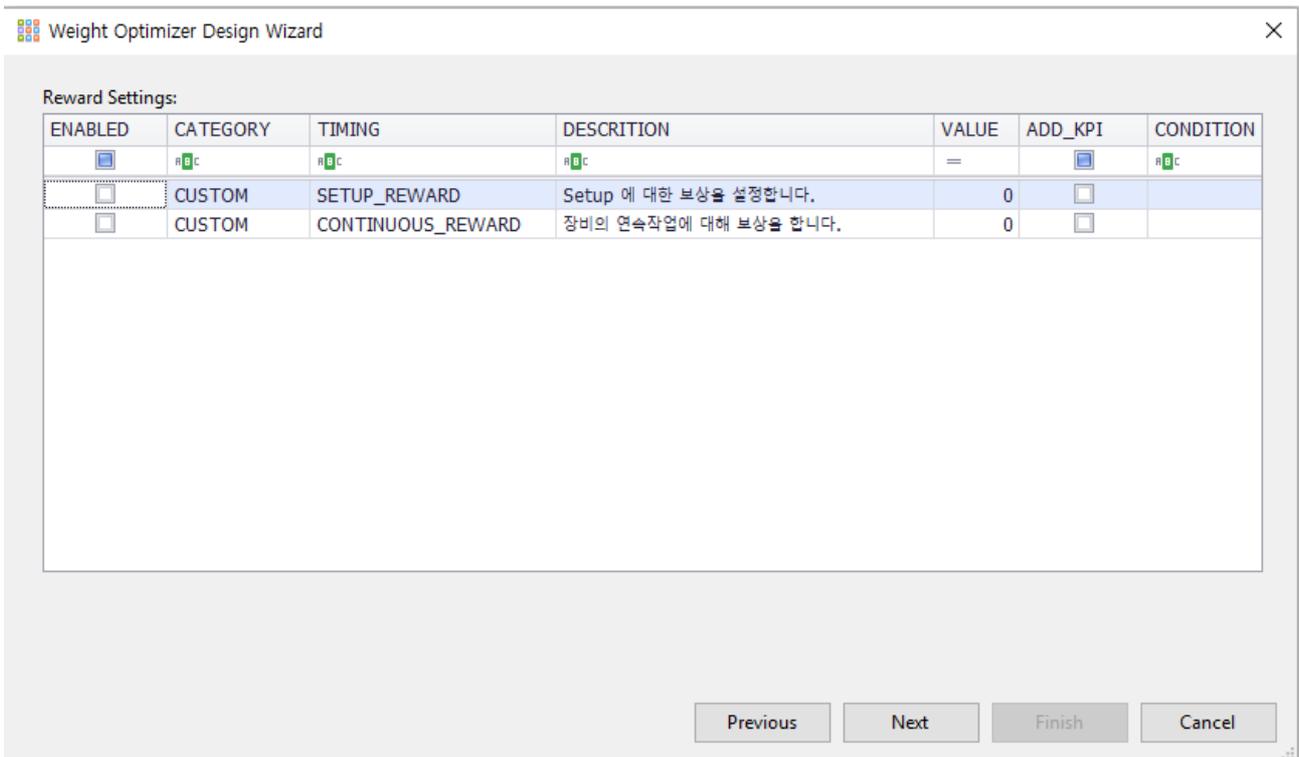
### 학습조건 설정

Weight Optimizer 를 통한 학습을 실행하기 위해 러닝 파라미터, 리워드 등의 설정이 필요합니다. 플러그인에 포함된 Design Wizard 는 단계별로 필요한 설정을 할 수 있도록 지원합니다. 먼저 `Plug-Ins/WeightOptimizer/Scenario Design Wizard` 메뉴로 마법사 기능을 사용할 수 있습니다.

1) "Create New Execution" 을 선택합니다.

#### 2) Reward Settings

모델에 설정한 가용한 Reward 중 학습에 사용할 Reward 를 선택하고, 해당 보상 값을 입력합니다. 아래 그림은 상기 예제 모델을 사용할 때의 Reward Settings 화면 예시입니다.



Weight Optimizer Wizard 화면(Reward Settings)

### 3) WISE와 Model 연결을 위한 Mapping

앞서 모델에 설정한 두가지 Output (KPI, Preset) 을 연결해주어야 하며, 실험에서 사용하는 Preset 정보가 담긴 테이블을 지정해주어야 합니다.

### 4) Learning Parameter 설정

- **Random Seed**: 실험시 사용할 Random Stream 의 Seed 입니다.
- **Lambda**: 강화학습의 Eligibility Trace 에 사용되는 매개변수입니다. 이전까지 발생한 의사결정이 Reward 발생에 어느정도 영향을 주었는지를 결정하는 파라미터입니다. 값이 0 에 가까울 수록 최근의 의사결정에 영향도가 커지며 1에 가까울 수록 이전에 발생한 의사결정이 현재 발생할 Reward에 영향을 주도록 학습이 됩니다. 0인 경우 최종 의사결정에 의해서만 Reward가 발생했다는 의미입니다. 일반적으로 0으로 설정하고 이벤트 발생 시점에 즉각적으로 Reward를 줄때 학습효과가 빠르게 나타납니다.
- **Half Life**: 발생한 Reward는 시간이 지날수록 가치가 줄어들게 됩니다. 본 파라미터는 Reward의 가치가 50% 로 감가상각되는데 걸리는 시간을 정의하는 파라미터입니다.
- **Epoch**: 모델 수행 횟수를 정의하는 파라미터입니다. 1Epoch 에 학습과 평가목적으로 두 번의 모델 실행을 하게 됩니다.

### 5) Computation Graph 설정

현재 Linear Graph 만을 지원함으로 Linear 를 선택합니다.

## 6) Selector 설정

Selector 의 경우 최종 산출된 Lot 중 Lot 을 선택하는 방식을 결정합니다. 두가지 방식을 지원합니다.

- **eGreedy** : 의사결정을 내릴 시, 가장 높은 가치를 주는 의사결정을 선택하는 것이 일반적이지만,  $e$  의 확률로 Random 하게 선택을 하는 방법입니다. 학습시  $e$  값이 클 경우 Random 하게 선택하는 경향이 높아 짐으로 Exploration 을 더 많이 할 수 있다고 볼 수 있습니다. 본 방식을 선택하는 경우 **Epsilon** 파라미터를 추가적으로 설정해주어야 합니다. 값은 0~1 값으로 설정합니다. 기본값은 0으로 사용합니다.
- **SoftMax** : 주어진 값들을 합이 1이되는 정규화된 지수함수로 변환한 뒤, 정규화 된 수치에 비례한 확률로 선택하는 방법입니다. 본 방식을 사용하기 위한 **Temperature** 는 Softmax에서 정규화시, 주어진 값들의 민감도를 조절하는 변수입니다. 양의 무한대에 가까울수록 정규화 결과물이 균등한 값을 지니게 되고, 0에 가까울 수록 가장 높은 값의 정규화 값이 1, 나머지는 0에 가깝도록 합니다.

## 7) SDG Method 설정

MOZART Weight Optimizer 의 경우 SGD(Stochastic Gradient Descent) 방식으로 학습을 하며 이중 지원하는 방식은 Adam 입니다. 지원방식은 이후 추가가 될 수 있습니다. Adam 은 SGD 방법들 중 가장 일반적으로 사용되는 Momentum 기반의 방법으로 학습속도가 빠르고, Learning Rate 에 영향을 크게 받지 않으며, Gradient element 사이의 Scale 이 크게 달라도 잘 적용되는 장점을 가지고 있습니다.

Adam 을 사용하기 위해 설정할 수 있음 파라미터는 다음과 같습니다.

- **Learning Rate** : 학습 속도를 제어하기 위한 파라미터입니다. 일반적인 값의 설정 범위는 0.001 ~ 0.1 입니다. Learning Rate 가 클 수록 의사결정 파라미터(factor weight) 의 변화 폭이 커집니다. 기본값은 0.1을 사용합니다.
- **Learning Rate Decay Parameter** : 학습 수행중 Learning Rate 를 줄여가기 위해 사용되는 파라미터입니다. 현재는 사용되지 않으며, 기본값은 0.0000001을 사용합니다.
- **First Momentum Beta** : 설정범위는 0~1 사이의 값입니다. 1에 가까울 수록 이전 Gradient 의 방향을 유지하려는 관성이 커지게 됩니다. 기본값으로 0.99 를 설정합니다.
- **Second Modentum Beta** : 설정범위는 0~1 사이의 값이며, First Momentum Beta 값보다 큰 것을 권장합니다. 기본값으로 0.999 를 설정합니다.

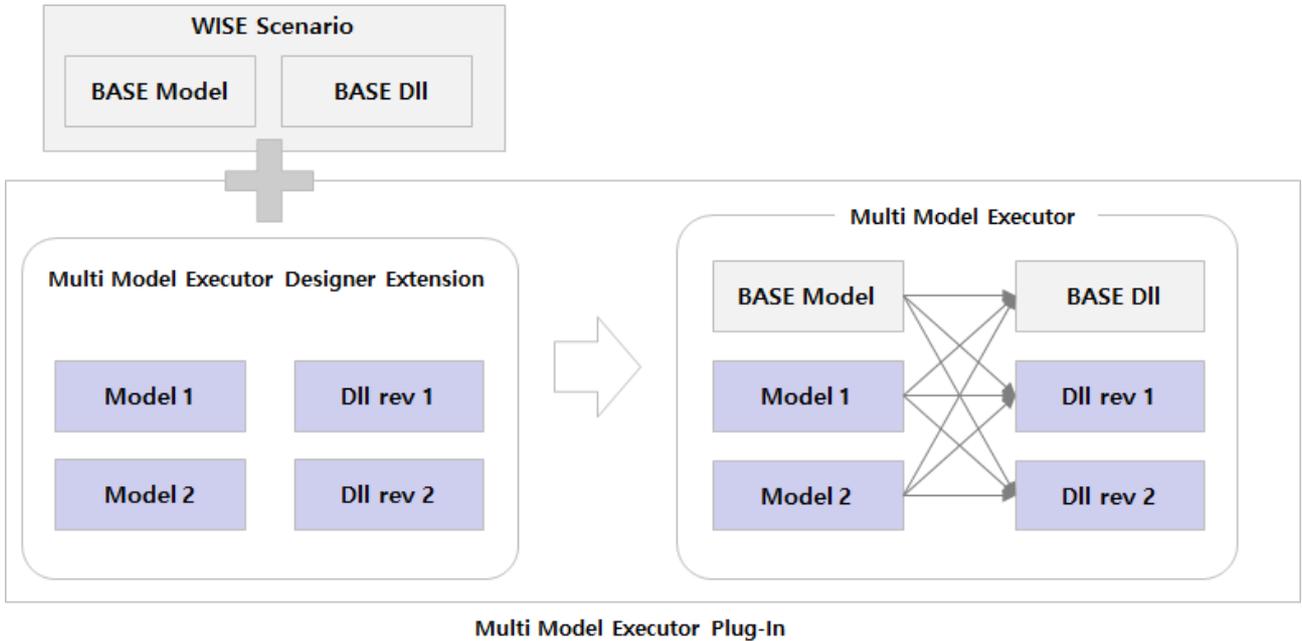
## 3단계 : 학습 수행 및 최적 파라미터 설정

1,2 단계의 절차를 모두 마무리 하면 학습을 할 수 있는 준비는 완료됩니다. 이제는 생성된 WISE Execution 을 실행합니다.

# Multi Model Executor

## 개요

다중모델 실행 플러그인(Multi Model Executor)은 시나리오에 실행가능한 모델과 DII 을 추가 하고, 이를 조합하여 실행하는 기능을 제공하는 플러그인입니다.



다중모델 실행 플러그인의 개념

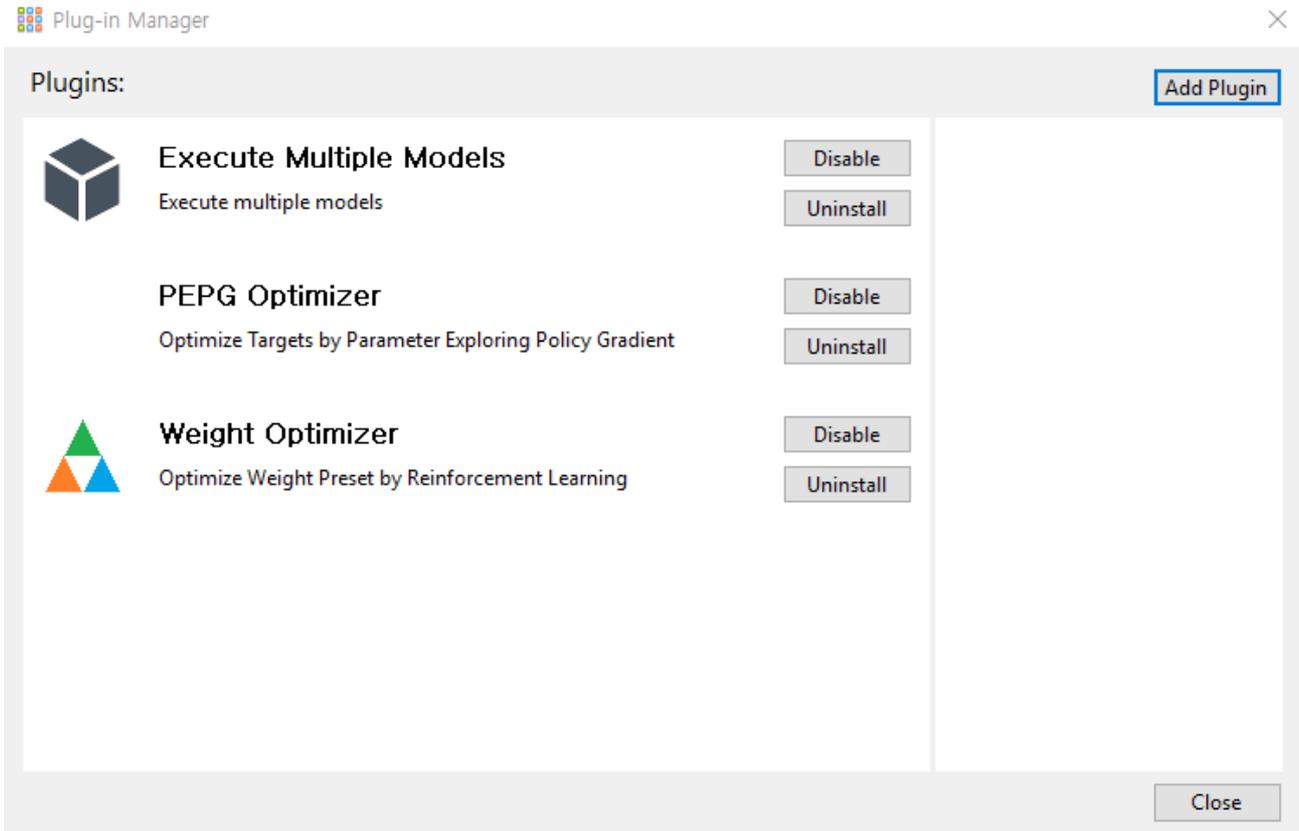
다중모델 실행 플러그인은 위의 그림과 같이 플러그인의 마법사 기능을 통해 시나리오에서 실행할 수 있는 모델과 DII 을 추가할 수 있습니다. 또한 추가된 Executor 를 통해 추가된 모델 과 DII 을 선택적으로 지정하여 실행 할 수 있습니다. 즉 여러모델을 여러 DII 로 실행할 수 있도록 기존의 시나리오를 확장 합니다.

## 플러그인 설치

WISE 가 설치된 상태에서 Plug-Ins>Plug-In Manager 를 사용하여 Multi Model Executor 플러그인을 설치합니다.

1. MOZART WISE를 실행하고, 상단 Plug-Ins > Plug-In Manager 메뉴를 클릭합니다.
2. Plug-In Manager 대화상자의 우측 상단에 있는 Add Plugin 버튼을 클릭합니다.

3. Select Plugin File 대화상자에서 다운로드 받은 "MultiModelExecutor.vplugin" 파일을 선택하고, 열기 버튼을 클릭합니다.
4. 설치가 되면 아래 그림과 같이 설치된 Plug-In 이 표시됩니다. 플러그인 우측의 Enable 버튼을 클릭하여 플러그인을 활성화합니다.
5. Plug-in Manager 대화상자의 Close 버튼을 클릭합니다.
6. 단 Plug-Ins > MultiModelExecutor > Model and DLL Settings 메뉴가 존재하는지 확인합니다.



Plug-In Manager 화면

## 사용 방법

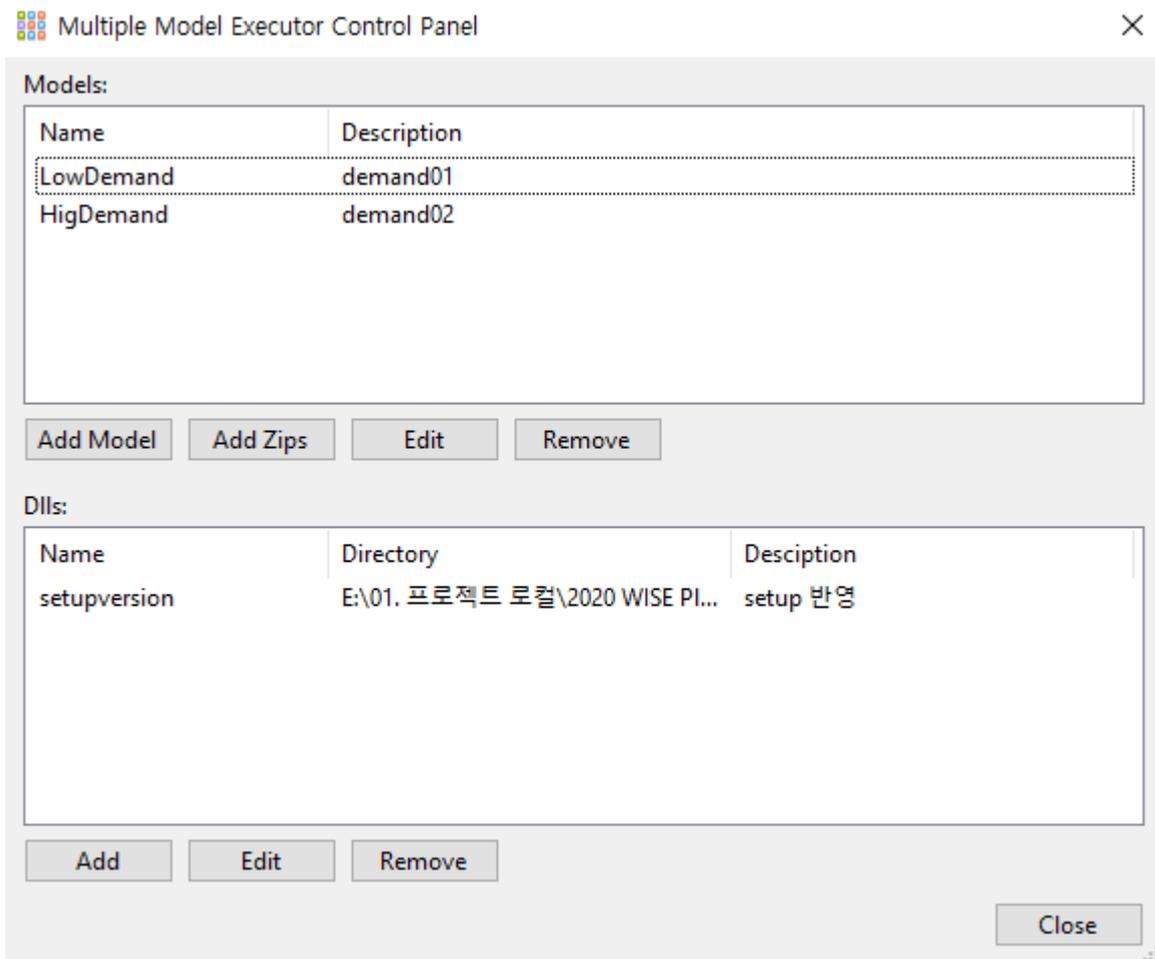
본 장에서 Multi Model Executor 를 사용방법을 1) 기본 시나리오 생성, 2) 실행 대상 모델/DLL 등록, 3) KPI 등록, 4) 실행대상 지정 및 실행의 총 4단계로 설명합니다.

### 1) 기본 시나리오 생성

File Menu>New Scenario 혹은 메뉴바의 아이콘을 통해 신규 시나리오를 생성합니다. 시나리오를 생성할때는 기본 모델과 모델을 실행할 수 있는 실행 DLL 이 포함된 폴더를 지정합니다. 시나리오에 지정된 모델은 시나리오 생성폴더로 복사가 되며, 실행 DLL 폴더는 절대경로로 저장됩니다.

## 2) 실행대상 모델 및 DLL 등록

시나리오를 연 상태로 Plug-Ins>MultiModelExecutor>Model & DLL Setting 메뉴를 사용하여 실행할 모델과 실행대상이 되는 DLL을 등록할 수 있습니다. 아래 그림은 설정화면입니다.



Multiple Model Executor Control Panel

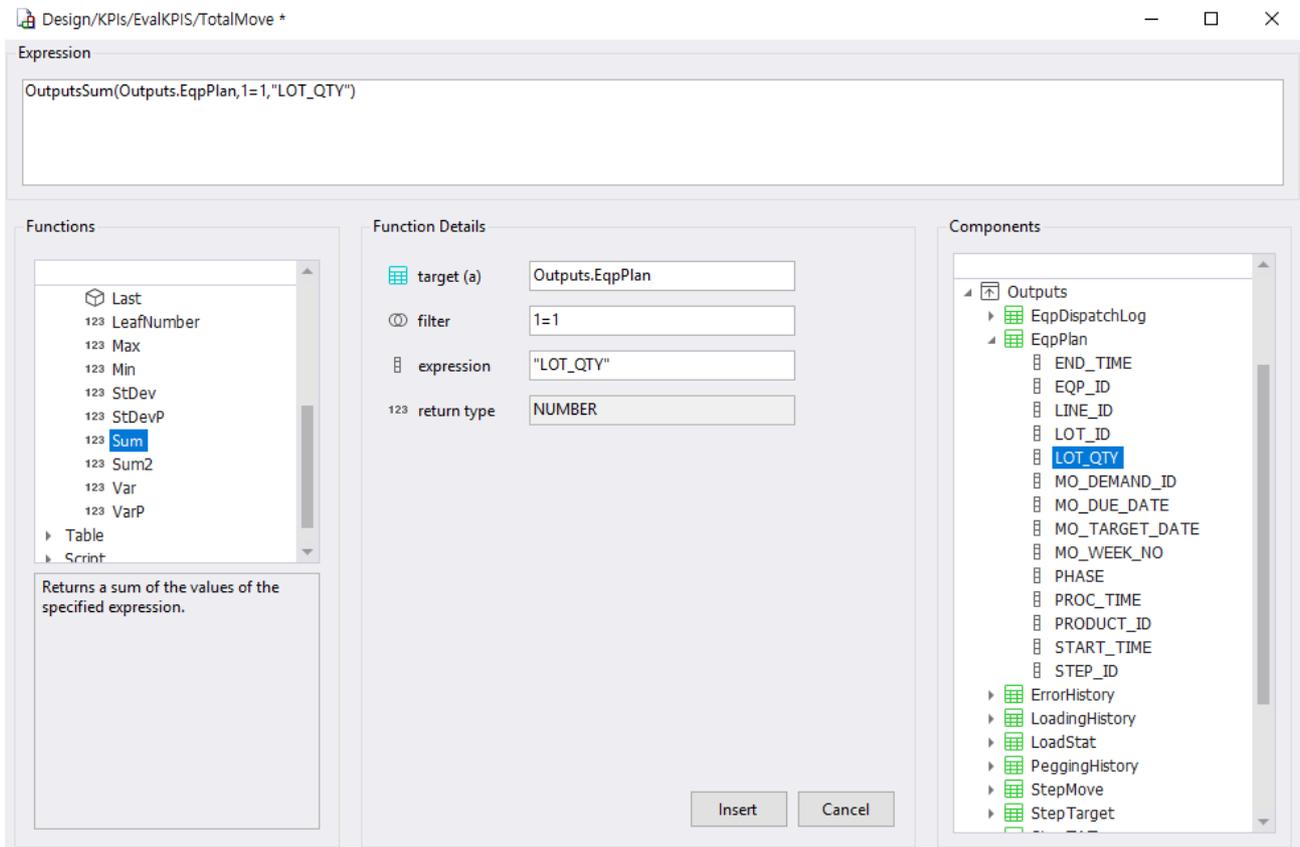
추가로 모델을 등록할때는 압축되지 않은 모델 파일을 선택하여 추가하는 방식과 압축된 Mozart 모델 파일을 배치로 선택하여 추가하는 두 가지 방식을 지원합니다. 모델을 추가하면 대상 모델을 시나리오 폴더 하위의 Models 폴더에 복사됩니다. 파일을 추가한 후 리스트의 항목을 더블클릭하거나 하단의 "Edit" 버튼을 사용하여 모델의 명칭과 Description 을 수정할 수 있습니다. 표시되는 모델의 명칭(Name)이 최종 Report 의 "Model Name" 컬럼에에 표시됨

니다. 따라서 여러개의 모델을 추가하여 수행하고자 하는 경우 모델 결과 비교시 확인하기 용이한 별칭을 설정하는 것이 좋습니다.

하단의 리스트 박스에는 추가된 DLL 이 표시됩니다. DLL 을 추가하기 위해서는 하단의 "Add" 버튼을 사용합니다. 대화상자에서 추가할 dll 이 포함된 폴더를 선택하면 DLL 명칭과 Description 을 입력할 수 있는 대화상자가 활성화됩니다. 모델과 마찬가지로 실행결과 Report 의 "Dll Name" 컬럼에 여기서 정의된 dll 명칭이 표시됩니다. 결과를 쉽게 알아보기 위해서는 구분 가능한 별칭을 사용하여 dll 을 등록하는 것이 좋습니다.

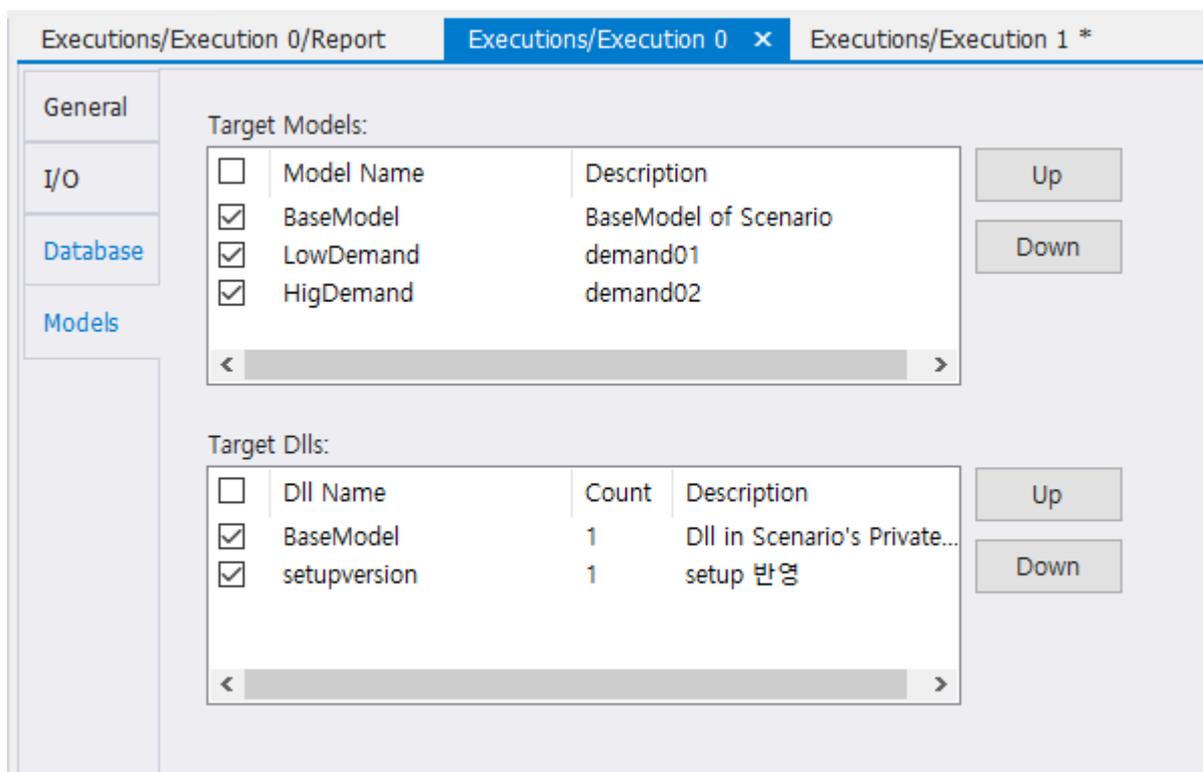
### 3) KPI 등록

KPI의 등록은 WISE 의 일반 사용법과 동일합니다. WISE 에서 제공하는 Function 을 사용하여 정의할 수 있으며, 별도의 LINQPad 용 Linq 파일을 사용할 수도 있습니다. 통상 Model 의 Output 을 사용한 간단한 통계수치(Sum, Average, Min, Max 등)를 KPI 로 할 경우에는 WISE 에서 제공하는 함수를 사용하고, 복잡한 처리를 필요로 하는 경우에는 LinqPad 를 사용하여 Linq Stagement 혹은 Program 등으로 Scalar Value 를 반환하는 함수를 만들어 사용합니다. 아래 그림은 Table Aggregation/Sum 함수를 사용하여 전체 Move 량을 KPI 로 정의한 예제입니다. Linq 파일을 사용하기 위해서는 Functions 에서 "Script/RunScript" 를 선택하고 Linq 파일을 선택합니다. Linq 파일은 Scalar 값을 반환하도록 구성해야 합니다.



#### 4) 실행

1~3의 과정을 통해 필요한 모든 과정이 마무리 되면 Executions 노드에서 "Popup Menu>Add" 메뉴를 사용하여 Execution 을 추가합니다. 추가된 Execution 에는 WISE 기본의 3개 탭외에 "Targets" 탭이 추가됩니다. 본 탭에서 실행대상이 되는 모델과 dll 을 선택하여 실행 할 수 있습니다.



Multi Model Executor 의 Experiment 설정 확장기능(Model 탭)

**Target Models :** 실행대상이 되는 Model 을 선택합니다. 실행 가능한 모델에는 시나리오 생성시 지정한 BaseModel 과 플러그인을 통해 추가된 모델이 모두 포함됩니다. 이중 실행할 모델만 선택합니다.

**Target Dlls :** 실행대상이 되는 DLL 을 선택합니다. 시나리오 생성시 지정한 실행 dll 은 Base Model 이라는 명칭으로 대상에 추가되어 있습니다. 모델과 마찬가지로 실행대상 dll 만 선택을 합니다. 만일 특정 dll 을 사용하여 모델 수행 횟수를 정의하고자 하는 경우에는 Count 컬럼의 값을 직접 조정할 수 있습니다.

이렇게 대상 모델과 dll 을 선택한 후 General 탭의 실행버튼 혹은 메뉴의 실행아이콘을 통해 실행하면 선택된 모든 모델마다 선택된 dll 로 지정된 횟수만큼씩 실행합니다. 만일 대상모델이 3개이고, 대상 dll 이 2개이며 각 dll 별로 1회씩 실행(기본설정)하도록 설정한 경우 총 6번이 실행됩니다. 동일한 설정으로 첫번째 dll 의 Count = 4로 설정하면 총  $15(= 3 * 4 + 3 * 1)$  회 실행이 됩니다.

실행된 결과는 Execution 하단의 Report 기능을 통해 실행 조합별 KPI 를 확인할 수 있습니다. 아래 그림은 KPI 를 두개 정의한 경우 Report 예시입니다.

## Execution Analysis Report

Execution Start Time: 2020-01-13 19:38:04  
 Execution End Time: 2020-01-13 19:38:32  
 Execution Elapsed Time: 00:00:28  
 Total Run Count: 15  
 Mozart Studio Path: C:\Program Files (x86)\vms\mozart\client\Wbin\WLSE\_Studio.exe

Execution		Performance			Model Info		Factors	KPIs		Model
Run Index	Run Name	Start Time	End Time	Elapsed Time	Model Name	Dll Name	Models	Total Move	Utilization	Model
0	Run 0	2020-01-13 19:38:04	2020-01-13 19:38:05	0:00:01	BaseModel	BaseModel	0	7025	87.5826717558361	Open in Studio
1	Run 1	2020-01-13 19:38:06	2020-01-13 19:38:08	0:00:01	BaseModel	BaseModel	0	7025	87.5826717558361	Open in Studio
2	Run 2	2020-01-13 19:38:08	2020-01-13 19:38:10	0:00:01	BaseModel	BaseModel	0	7025	87.5826717558361	Open in Studio
3	Run 3	2020-01-13 19:38:10	2020-01-13 19:38:11	0:00:01	BaseModel	BaseModel	0	7025	87.5826717558361	Open in Studio
4	Run 4	2020-01-13 19:38:11	2020-01-13 19:38:13	0:00:01	BaseModel	setupversion	0	7025	87.5826717558361	Open in Studio
5	Run 5	2020-01-13 19:38:13	2020-01-13 19:38:15	0:00:01	LowDemand	BaseModel	0	6875	85.5820109049479	Open in Studio
6	Run 6	2020-01-13 19:38:15	2020-01-13 19:38:17	0:00:01	LowDemand	BaseModel	0	6875	85.5820109049479	Open in Studio
7	Run 7	2020-01-13 19:38:17	2020-01-13 19:38:19	0:00:01	LowDemand	BaseModel	0	6875	85.5820109049479	Open in Studio
8	Run 8	2020-01-13 19:38:19	2020-01-13 19:38:21	0:00:01	LowDemand	BaseModel	0	6875	85.5820109049479	Open in Studio
9	Run 9	2020-01-13 19:38:21	2020-01-13 19:38:23	0:00:01	LowDemand	setupversion	0	6875	85.5820109049479	Open in Studio
10	Run 10	2020-01-13 19:38:23	2020-01-13 19:38:24	0:00:01	HigDemand	BaseModel	0	7025	87.5826717558361	Open in Studio
11	Run 11	2020-01-13 19:38:24	2020-01-13 19:38:26	0:00:01	HigDemand	BaseModel	0	7025	87.5826717558361	Open in Studio

다중 모델 실행시 Report 예시