

Solver Devloper Documentation

version 1.0

Last generated: December 07, 2018



© 2018 KISTI. This is a boilerplate copyright statement... All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law.

Table of Contents

Introduction

Introduction	4
Linux Foundation.....	5
Science App	11

Developer Account

Workspace	14
Bulb Server.....	19

Science App Programming

Science App Programming 개요	22
Input Programing 개요	23
Structured Data Editor (SDE).....	25
Simrc setiing.....	27
Ounput Programing 개요.....	29
Simpost Setting.....	30
OneD Format.....	31
Plotly Format	33
Gnuplot.....	34
Gif Animation.....	37

- C Programming

C Programing 개요	38
입력 파일이 1개인 경우.....	40
입력 파일이 여러개인 경우	47
SDE case study	50
Output programing.....	57
oneD	59
Gnuplot.....	67

- Fortran Programming

Fortran Programing 개요	69
입력 파일이 1개인 경우.....	71
입력 파일이 여러개인 경우	76

SDE case study	79
Output programing.....	88
oneD	90

- Python Programming

Python Programing 개요	97
입력 파일이 1개인 경우.....	99
입력 파일이 여러개인 경우	103
SDE case study	104
Output programing.....	111
oneD	112
Plotly.....	118
QnA	119

- R Programming

R Programing 개요	120
입력 파일이 1개인 경우.....	123
입력 파일이 여러개인 경우	127
SDE case study	128
Output programing.....	133

- Octave Programming

Octave Programing 개요	134
입력 파일이 1개인 경우.....	139
SDE case study	143
Output programing.....	148

Editor (편집기)

Editor 개요.....	149
Structured Data Editor (SDE).....	151
File Selector.....	163
Text Editor	164
CSV Editor.....	165

Analyzer (분석기)

Analyzer 개요.....	166
OSPPlotViewer	168
OSPIImageViewer	169
OSPHtmlViewer.....	170

JSmol	171
Paraview	173
ProteinViewer	175
Plotly.....	176
NGL	178

Sience App Management

데이터 타입 생성	179
사이언스 앱 생성	182
사이언스 앱 정보 입력.....	183
사이언스 앱 실행환경 정보 입력	185
입출력 포트 생성	187
레이아웃 설정	191
공개 데이터 등록	193
사이언스 앱 관리하기.....	195

EDISON 플랫폼 소개

Summary: EDISON 플랫폼은 웹 브라우저 접속만으로 C, C++, Fortran, MPI, Python등으로 코딩된 Science Apps(시뮬레이션 SW)을 활용할 수 있는 환경을 제공합니다. 본 문서는 Science App 개발자들을 위한 개발 문서로 EDISON 플랫폼에서 시뮬레이션 SW를 개발하는 방법에 대해서 소개하고자 합니다.

EDISON 플랫폼 소개

EDISON 플랫폼은 웹 브라우저 접속만으로 C, C++, Fortran, MPI, Python등으로 코딩된 Science Apps(시뮬레이션 SW)을 활용할 수 있는 환경을 제공합니다. 본 문서는 Science App 개발자들을 위한 개발 문서로 EDISON 플랫폼에서 시뮬레이션 SW를 개발하는 방법에 대해서 소개하고자 합니다.

Science App 등록 절차

개발자 권한 신청 및 발급

Bulb 서버 접속 - Code upload & Compile - 실행 파일 압축

사이언스 앱스토어 > 앱 관리 > 앱 등록 - 앱 정보 입력 - 실행환경 정보 입력 - 입/출력 포트 정보 입력 - 앱테스트

서비스 요청 -> 관리자 승인 -> 서비스!

리눅스 기초

Summary:

리눅스 기초

EDISON 플랫폼은 리눅스 기반으로 개발되어 있어서 시뮬레이션 SW의 경우에도 리눅스에서 컴파일과 실행이 되는지 확인해야 합니다. 이를 위해 필요한 기본적인 내용들을 정리하였습니다.

리눅스 기본 명령어

명령어	설명	예제
ls	파일 리스트 보기	
cd	디렉토리를 변경	cd cgi-bin : 하부 디렉토리인 cgi-bin으로 들어감. cd .. : 상위 디렉토리로 이동 cd ~ : 어느곳에서든지 자기 홈디렉토리로 바로 이동
mkdir	디렉토리를 생성	mkdir download : download 디렉토리 생성
rm	파일 삭제	rm test.html : test.html 파일 삭제 rm -rf <디렉토리> : 디렉토리 전체를 삭제
pwd	현재 경로 보기	
chmod	퍼미션 조정	chmod 755 a.sh : a.sh파일의 퍼미션을 755로 조정
vi	vi 에디터 실행	vi newfile : vi 편집기 상태로 들어감

리눅스 필수 명령어

명령어	예제
./x	x 프로그램 실행 (현재 디렉토리에 있는 것)
↑/↓	이전에(↑) / 다음에(↓) 입력했던 명령어

명령어	예제
cd x	디렉토리 X로 가기
cd ..	한 디렉토리 위로 가기
x 다음 [tab] [tab]	x 로 시작하는 모든 명령어 보기
ls (또는 dir)	디렉토리 내부 보여주기
mv x y	파일 x를 파일 y로 바꾸거나 옮기기
cp x y	파일 x를 파일 y로 복사하기
rm x	파일 지우기
mkdir x	디렉토리 만들기
rmdir x	디렉토리 지우기
rm -r x	디렉토리 x를 지우고 하위도 다 지우기
exit	시스템 종료

- [레퍼런스](#)

Vim Editor

리눅스 시스템에 설치되어 있는 기본적인 워드 프로세서로 기본적인 사용법을 숙지하고 있어야 합니다. 본 가이드에서는 Vim Editor에 대한 세부적인 사용법에 대해서는 설명하진 않겠습니다. 아래 사이트를 참고하여, Vim Editor에 대한 사용법을 습득하길 바랍니다.

- [웹을 통해 vim 사용법을 배울 수 있는 튜토리얼 제공](#)
- [vim 관련 사용법에 대해 정리되어 있는 사이트\(한글\)](#)

리눅스에서 컴파일 하기

- [리눅스에서 C언어 컴파일 하기](#)

Makefile

리눅스 상에서 소스코드 컴파일을 편리하게 해주는 쉘 스크립트 파일입니다. 관련 상세한 정보는 아래 링크 사이트를 참고 하시길 바라며, 본 가이드에서는 makefile 샘플 파일에 대한 설명만 하도록 하겠습니다.

- makefile 구조 이해

Sample 예제 파일 구성

파일 명	설명
main.c	main.c
sub1.c, sub1.h	라이브러리1
sub2.c, sub2.h	라이브러리2
Makefile	Makefile file

예제 파일

main.c

```
#include <stdio.h>
#include "sub1.h"
#include "sub2.h"

int main()
{
    myprint_sub1();
    myprint_sub2();
    return 1;
}
```

sub1.h

```
void myprint_sub1();
```

sub1.c

```
#include <stdio.h>
#include "sub1.h"
void myprint_sub1()
{
    printf("I am sub1\n");
}
```

sub2.h

```
void myprint_sub2();
```

sub2.c

```
#include <stdio.h>
#include "sub2.h"
void myprint_sub1()
{
    printf("I am sub2\n");
}
```

Makefile

```
# Makefile example
CC = gcc
CLAGS = -o

OBJS = main.o sub1.o sub2.o
SRCS = $(OBJS:.o=.c)

TARGET = test
all: $(TARGET)
$(TARGET) : $(OBJS)
    $(CC) $(CLAGS) $@ $(OBJS)
clean :
    rm -rf $(OBJS)
```

(1) `CC = gcc` 사용하고자 하는 컴파일러 명령어를 입력하면 된다. (icc, gcc, ifort, f90 등...)

(2) `CLAGS = -o` 컴파일 옵션을 넣어 주면 된다. 세부 옵션은 각 컴파일러 매뉴얼 참조 -o 옵션은 컴파일 결과 파일의 이름을 지정해주겠다는 명령어이다. 여기에서는 결과 파일명을 `TARGET =test`로 지정해주었다.

(3) `OBJS = main.o sub1.o sub2.o` 컴파일에 필요한 Object 파일과 Source 파일 명을 써주면 된다. 컴파일 과정에서 Object 파일은 자동으로 생성되며, (3)에 이름을 지정해 주어야 된다.

(4) `SRCS = $(OBJS:.o=.c)` 의 `$(OBJS:.o=.c)`는 `main.c sub1.c sub2.c`를 컴퓨터가 `OBJS`를 읽어 `.o`를 `.c`로 수정해 준 것이다.

fortran 의 경우에는 `$(OBJS:.o=.f90)`라고 입력하면 된다.

(5) `TARGET = test` 출력 파일 명 설정 CLASS에 -o 옵션을 지우면 안된다.

(6)

```
all: $(TARGET)
$(TARGET) : $(OBJS)
    $(CC) $(CLAGS) $@ $(OBJS)
```

make all 명령어를 설정하는 부분이며, 모든 Source 파일을 컴파일 하고 생성된 Object 파일들을 가지고 링크과정을 거쳐 실행 파일을 만드는 부분이다. \$(CC) 앞에 <tap> 키를 입력해야 한다.

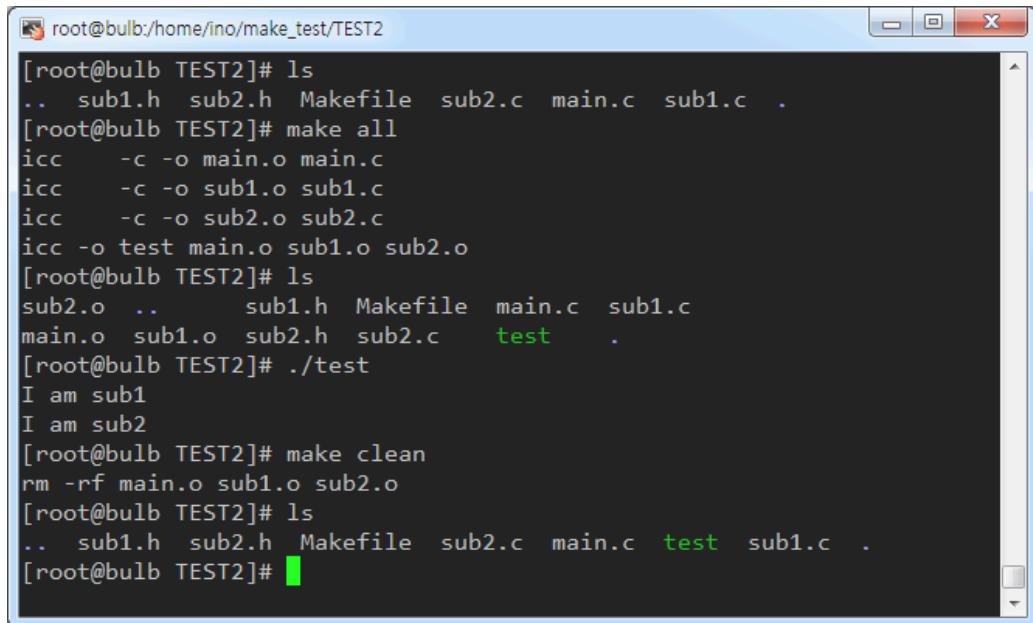
(7)

```
clean :  
        rm -rf $(OBJS)
```

make clean 명령어를 설정하는 부분이며, 컴파일 과정에서 생성된 Object 파일을 지우는 부분.

- Makefile 실행하기
- 실행 파일 생성 : make all
- 생성된 Object 파일 삭제 : make clean

완성된 Makefile과 소스 파일을 가지고 실행 파일을 만들어 보면 아래와 같다.



The screenshot shows a terminal window titled 'root@bulb:/home/ino/make_test/TEST2'. The terminal displays the following command-line session:

```
[root@bulb TEST2]# ls  
.. sub1.h sub2.h Makefile sub2.c main.c sub1.c .  
[root@bulb TEST2]# make all  
icc -c -o main.o main.c  
icc -c -o sub1.o sub1.c  
icc -c -o sub2.o sub2.c  
icc -o test main.o sub1.o sub2.o  
[root@bulb TEST2]# ls  
sub2.o .. sub1.h Makefile main.c sub1.c  
main.o sub1.o sub2.h sub2.c test .  
[root@bulb TEST2]# ./test  
I am sub1  
I am sub2  
[root@bulb TEST2]# make clean  
rm -rf main.o sub1.o sub2.o  
[root@bulb TEST2]# ls  
.. sub1.h sub2.h Makefile sub2.c main.c test sub1.c .  
[root@bulb TEST2]#
```

Science App

Summary:

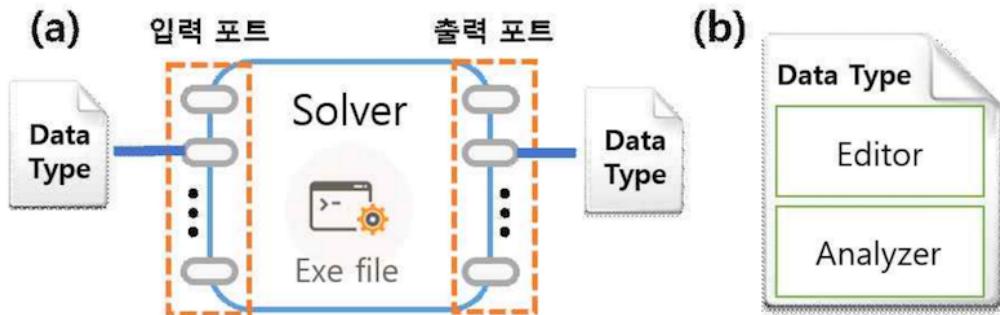
사이언스 앱 개요

EDISON에서 사용자가 특정 문제를 해석하기 위한 웹기반 시뮬레이션 소프트웨어를 사이언스 앱이라 정의합니다. 단일 시뮬레이션을 실행하는 경우와 사이언스 앱을 순차적으로 실행시키기 위한 워크플로우에서 사용자가 등록한 앱을 활용할 수 있도록 구성 요소를 5가지로 나누었습니다.

사이언스 앱 구성요소

요소	설명
해석기 (Solver)	과학/기술 분야의 문제를 해결하기 위해 개발한 리눅스 실행 파일이나 스크립트 파일로 입출력 포트를 설정하여 입출력 데이터를 설정 할 수 있습니다. 명령행 인자(Command Line Argument) 방식으로 입력 데이터를 파일 형태로 읽고 result 폴더를 생성해 결과 데이터를 파일 형태로 출력합니다.
입출력포트 (Input/ Output Port)	해석기의 입력 파일과 출력 파일의 형태를 정의해주는 요소로써 1개의 데이터 타입을 가지게 됩니다. 앱 등록시 데이터 타입 지정하여 생성해야 합니다.
데이터타입 (Data Type)	사이언스 앱에서 생성되는 데이터의 형태를 구분하는 요소로써 1개 이상의 편집기와 분석기를 가지고 있습니다.
편집기 (Editor)	웹 GUI를 통해 사용자의 입력을 받아 해석기의 입력 데이터를 만들어주는 요소로써 해석기의 입력 파일을 생성합니다. Liferay의 Portlet 형태로 개발되어야 합니다.
분석기 (Analyzer)	해석기를 통해 생성된 결과 데이터를 웹상에서 분석 작업을 할 수 있는 요소로써 해석기 결과 데이터를 파일형태로 읽어 웹상에서 표시해줍니다. Liferay의 Portlet 형태로 개발되어야 합니다.

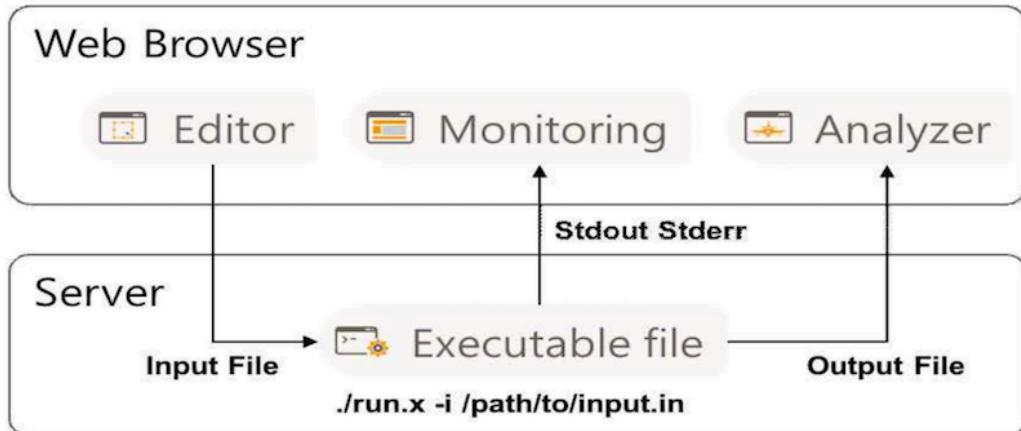
사이언스 앱은 웹을 통해 입력 데이터를 처리하는 편집기, 입력된 데이터를 통해 문제를 해석하는 해석기, 해석된 결과 데이터를 웹상에서 분석하는 분석기로 나누어집니다. 해석기는 출력 형태를 정의하기 위한 출력 포트를 가지고 있으며, 출력 포트의 데이터 형태를 지정하는데 이터 타입을 정할 수 있습니다.



사이언스 앱 구성요소 (a) 해석기 (b) 데이터 타입

위 그림은 사이언스 앱 구성요소 중 해석기와 데이터 타입의 구조를 나타낸 그림입니다. (a)해석기는 여러 개의 입출력 포트를 가지고 있으며, 각각의 입출력 포트는 1개의 데이터 타입을 가지게 됩니다. 데이터 타입의 경우 1개 이상의 편집기와 분석기를 가질수 있습니다. 사이언스 앱 개발자는 입출력 포트 등록 시 기존에 등록되어 있는 데이터 타입을 사용할 수 있습니다.

사이언스 앱 실행 방식

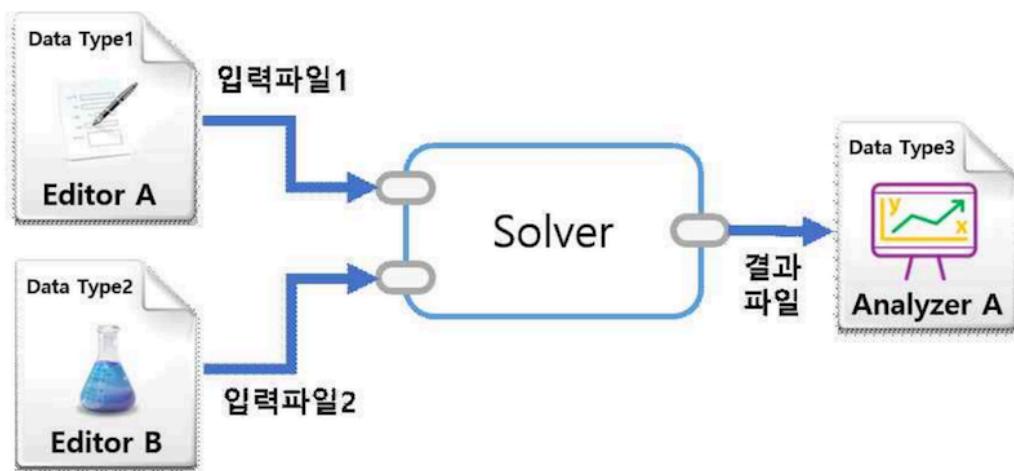


사이언스 앱 구성

사이언스 앱 실행 방식은 웹 브라우저에서 실행하는 부분과 서버에서 실행하는 부분으로 나눌 수 있습니다. 사용자가 EDISON에서 앱을 실행하게 되면, 웹 브라우저에서 편집기를 통해 시뮬레이션 실행에 필요한 입력데이터를 생성할 수 있습니다. 입력데이터는 파일형태로 저장하여 서버로 전송되고 해석기는 명령행 인자(Command Line Argument) 방식으로 입력 파일을 읽어 시뮬레이션을 수행하게 됩니다. 해석기가 실행 중 Sdtout, Stderr 형태로 정보를 출력하게되면

사용자가 웹에서 실행 도중 모니터링을 통해 중간 해석 정보를 확인할 수 있습니다. 해석 종료 이후 result 폴더를 생성해 파일 형태로 결과 데이터를 저장하게 됩니다. 웹브라우저에서 분석 기는 해석 결과 파일을 읽어 사용자에게 데이터를 가시화할 수 있습니다.

단일 앱 실행 구조



사이언스 앱 실행 시나리오

위 그림은 사이언스 앱을 실행하는 경우로 1개의 해석기가 2개의 입력 포트와 1개의 출력 포트를 갖는 경우를 도식화한 그림입니다. 2개의 입력 포트에서는 각 포트는 데이터 타입이 설정되어 있고, 데이터 타입이 가지고 있는 편집기를 통해 입력 파일을 생성할 수 있습니다. 출력 포트의 경우에도 포트는 데이터 타입이 설정되어 있고, 데이터 타입이 가지고 있는 분석기를 통해 결과 파일을 확인할 수 있습니다.

워크플로우 실행 구조



워크플로우 실행 시나리오

위 그림은 워크플로우를 이용하여 2개의 해석기를 연동하여 실행하는 경우를 도식화한 그림입니다. 워크플로우에서는 해석기 A의 출력 포트의 데이터 타입과 해석기 B의 입력 포트의 데이터 타입이 같은 경우 이를 연결할 수 있으며, 한 번에 실행할 수 있는 기능을 제공하고 있습니다.

개발자 계정 발급

Summary: EDISON 플랫폼에 Science App을 등록하기 위해서는 우선적으로 개발자 권한을 얻어야 합니다. 개발자 권한을 받은 아이디는 Science App을 등록 할 수 있으며, 앱 개발자를 위한 개발 서버인 Bulb 서버에 접근할 수 있는 계정을 제공해 드립니다. Science App 등록시 메타 정보를 입력해야 하며, 이는 웹 포털에서 할 수 있습니다.

개발자 계정 발급

The screenshot shows the EDISON website interface. At the top, there is a navigation bar with links for Search, Site Map, Language, and user account information (inojeon, 로그아웃, EDISON'S). Below the navigation bar, the main header reads "Everything for Computational Science&Engineering" and "BRIDGE TO COMPUTATIONAL SCIENCE FOR HIGHER EDUCATION AND ADVANCED RESEARCH". On the right side of the header, it says "My EDISON" and "/ My EDISON /".

The main content area has a sidebar on the left containing links for "즐겨찾기 앱", "강좌 현황", "수강 현황", "파일 관리", and "워크스페이스 >". The "워크스페이스 >" link is highlighted with a red box. The main content area has a table titled "워크스페이스" with columns for 요청 용도, 아이디, 임시 비밀번호, 요청일시, 사용희망일, 처리상태, and 처리결과. A message at the bottom of the table says "조회된 결과가 없습니다." (No results found). A red box highlights the "워크스페이스 요청" button at the bottom right of the table.

At the bottom of the page, there is a footer with copyright information: "COPYRIGHT(C) 2012~ NISN, KISTI ALL RIGHTS RESERVED.", a link to "개인정보처리방침", an email address "E-mail. edison@kisti.re.kr", and logos for "과학기술정보통신부", "NRF", and "KISTI".

EDISON 워크스페이스 메뉴

개발자 계정을 발급 받기 위해서는 EDISON 사이트에서 워크스페이스 요청을 해야합니다.
EDISON 사이트 접속 > **My EDISON** > 워크스페이스 에서 워크스페이스 요청 을 통해 신청 할 수 있습니다.

| My EDISON > 워크스페이스 요청

▣ 사용자 정보

아이디	████████	사용자명	████████
E-mail	████████	대학교/기관	████████

▣ 워크스페이스 요청정보

(1) 요청용도	솔버 개발	(2) 사용희망일*	2017-07-31	~ 2020-07-30
(3) 사용언어	<input type="checkbox"/> fortran <input type="checkbox"/> c/c++ <input checked="" type="checkbox"/> python <input type="checkbox"/> java <input checked="" type="checkbox"/> 기타 R			
(4) 접속 IP*	127.0.0.1	추가		
(5) 비고				

▣ 동의 정보

<p>보안 서약서</p> <p>상기 본인은 "첨단 사이언스·교육 허브 개발 사업 플랫폼 연구개발 및 사이버 인프라 기반 사용자 서비스" 연구과제 개발 인원으로 참여하면서 다음사항을 준수할 것을 서약합니다.</p> <p>(6) 보안 동의*</p> <p>1. 본 연구과제를 수행하는 과정에서 알 수 있었던 연구기밀에 대해 연구과제 수행중은 물론 종료후에도 연구원장 허락없이 자신 또는 제3자를 위하여 사용하지 않는다. 2. 본 연구과제 추진성과가 적법하게 공개된 경우라고 하여도 미공개 부분에 대해서는 앞에서와 같이 비밀유지의무를 부담한다. 3. 본 연구과제가 완료되거나 연구과제를 수행할 수 없게 된 경우, 그 시점에서 본인이 보유하고 있는 연구기밀을 포함한 관련 자료를 즉시 연구개발책임자에게 반납하며 앞에서</p> <p><input checked="" type="radio"/>동의 <input type="radio"/>동의 안함</p> <p>(7) 보안서약서*</p> <p><input type="checkbox"/> 파일선택 선택된 파일 없음</p>		(8) [security_oath.docx]
요청		

COPYRIGHT(C) 2012~ NISN, KISTI ALL RIGHTS RESERVED. | 개인정보처리방침 | E-mail. edison@kisti.re.kr



EDISON 워크스페이스 메뉴2

화면 설명

메뉴	설명
(1) 요청 용 도	워크스페이스 요청 용도에 맞게 선택하면 됩니다.
(2) 사용희망 일	워크스페이스를 사용하고자 하는 기간을 지정하면 됩니다.
(3) 사용언어	주로 사용하는 언어를 선택하시면 됩니다. (선택하지 않은 언어도 사용할 수 있습니다.)
(4) 접속 IP	접속하는 PC의 IP 주소를 입력하시면 됩니다. (작성한 IP이외의 다른 IP에 서는 접속이 불가 합니다.)

메뉴	설명
(5) 비고	워크스페이스 신청시 문의사항이나 요청사항이 있으면 여기에 작성하시면 됩니다.
(6) 보안 동의	보안 서약서 내용을 확인하시고 동의해주세요.
(7) 보안서약서	(8)에 있는 보안서약서 양식을 다운받아 양식에 맞게 서약서를 작성한 후 보안서약서 파일을 업로드하시면 됩니다.
(8) 보안서약서 양식	클릭하시면 보안서약서 양식을 다운 받을 수 있습니다.

워크스페이스 요청정보

사용하고자 하는 프로그램 언어와 접속하고자하는 IP 주소를 입력해야합니다. 여기에 입력한 IP 주소에서만 Bulb 서버에 접속이 가능합니다. 본인이 접속하고자 하는 IP를 정확하게 입력해 주셔야 하며, 추가 메뉴를 이용하면 여러 IP를 입력할 수 있습니다.

- 자신의 IP 주소 확인하기

보안 동의

보안 서약서 약관을 확인 후 사용자 동의를 체크 해야합니다. 보안 서약서 서명 파일을 다운로드 받습니다. 양식의 내용을 작성하고, 출력 후 자필 서명이 들어간 스캔 파일을 업로드 해주셔야 합니다. 작성한 보안서약서 파일을 업로드 합니다. 모든 작성이 완료된 이후 개발자 권한을 요청합니다.

워크스페이스 발급 확인

관리자 승인이 완료되면 개발자 권한을 얻게 되며, **EDISON 사이트 접속 > My EDISON > 워크스페이스** 발급받은 상세정보를 확인할 수 있습니다. Bulb서버 접속에 필요한 아이디와 초기 비밀번호를 확인할 수 있습니다.

The screenshot shows the My EDISON portal interface. At the top, there is a navigation bar with links for '통합검색', '앱스토어', '시뮬레이션', '콘텐츠', '교육', 'ABOUT', '통계', 'My EDISON' (which is highlighted with a red box), '경진대회', '전인호', '로그아웃', and 'EDISON'S'. Below the navigation bar, a banner reads 'Everything for Computational Science&Engineering' and 'BRIDGE TO COMPUTATIONAL SCIENCE FOR HIGHER EDUCATION AND ADVANCED RESEARCH'. To the right, it says 'My EDISON' and 'My EDISON /'. On the left, a sidebar menu includes '즐겨찾기 앱', '사이언스 앱', '데이터 탑재', '이력관리', '강좌 현황', '수강 현황', '파일 관리', '워크스페이스 >' (which is highlighted with a red box), '콘텐츠', and '사이트 가입/탈퇴'. The main content area displays a table titled '워크스페이스' with columns: 요청 용도 (App Development), 아이디 (Redacted), 임시 비밀번호 (Redacted), 요청일시 (2018-03-23), 사용희망일 (2015-07-21 ~ 2019-07-01), 처리상태 (Pending), and 처리결과 (Additional Request). At the bottom of the page, there is copyright information: 'COPYRIGHT(C) 2012~ NISN, KISTI ALL RIGHTS RESERVED.' and logos for '과학기술정보통신부', 'NRC', '한국연구재단', and 'KISTI'.

해당 내용을 클릭하게 되면 워크스페이스 신청과 관련한 상세 내용을 조회할 수 있습니다.

| My EDISON > 워크스페이스 상세정보

This screenshot shows the '워크스페이스 상세정보' (Workspace Detailed Information) page. It contains two main sections: '사용자 정보' (User Information) and '워크스페이스 요청정보' (Workspace Request Information).

사용자 정보:

아이디	Redacted	사용자명	Redacted
E-mail	Redacted@kisti.re.kr	대학교/기관	Redacted

워크스페이스 요청정보:

요청용도	앱 개발	사용희망일	2015-07-21 ~ 2019-07-01
사용언어	<input type="checkbox"/> fortran <input type="checkbox"/> c/c++ <input type="checkbox"/> python <input type="checkbox"/> java <input type="checkbox"/> 기타		
접속 IP	Redacted		
비고			

워크스페이스 상세정보 버튼은 '열기'로 표시되어 있습니다.

처리정보 섹션에는 '추가요청' 버튼이 포함된 테이블이 있습니다.

처리상태	추가요청	처리일자	2018-03-23
아이디	Redacted	임시 비밀번호	Redacted
요청용도	앱 개발		
요청내용			

추가 요청을 통해 접속 IP 추가등의 요청을 관리자에게 할 수 있습니다.



개발자 서버 접속

Summary:

개발자 서버 접속

개발자 계정 발급 절차를 통해 EDISON SW를 등록할 수 있는 웹페이지 권한과 개발자 서버 (bulb)에 접속할 수 있는 아이디와 최초 비밀번호를 발급 받게 됩니다.

개발자 서버 bulb 계정 확인

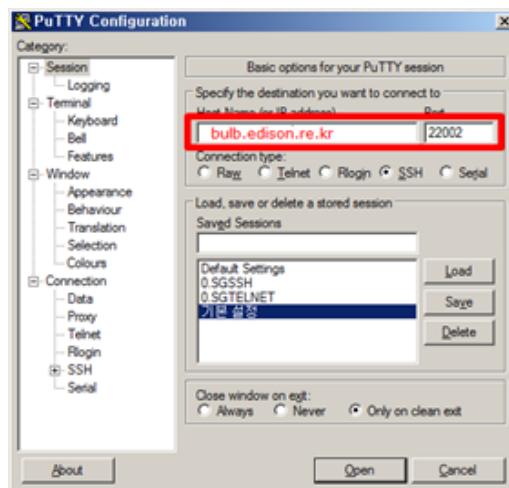
개발자 서버에 접속하기 위한 아이디와 비밀번호는 **EDISON 사이트 접속 > My EDISON > 워크스페이스**에서 확인할 수 있습니다.

개발자 서버 접속

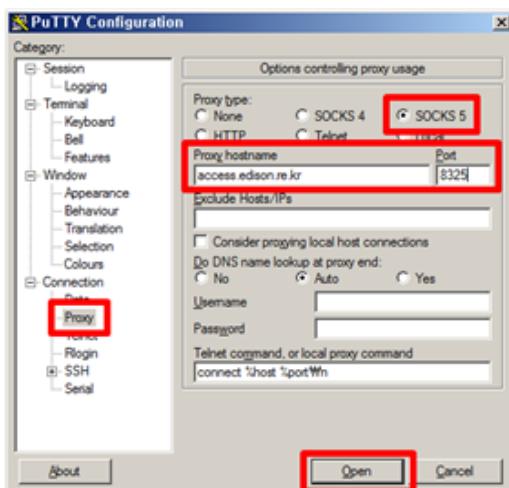
Putty로 Bulb 접속하기 (ssh)

[PuTTY 다운로드](#) 후 아래 그림과 같이 접속

- 호스트 : bulb.edison.re.kr, 포트 : 22002, 프로토콜 : SSH
- 프록시 : SOCK 5, 프록시 호스트 : access.edison.re.kr, 프록시 포트 8325



(1) 접속 서버의 IP, Port 지정



(2) Proxy 탭의 설정 변경

Putty로 Bulb 접속하기 (ssh)

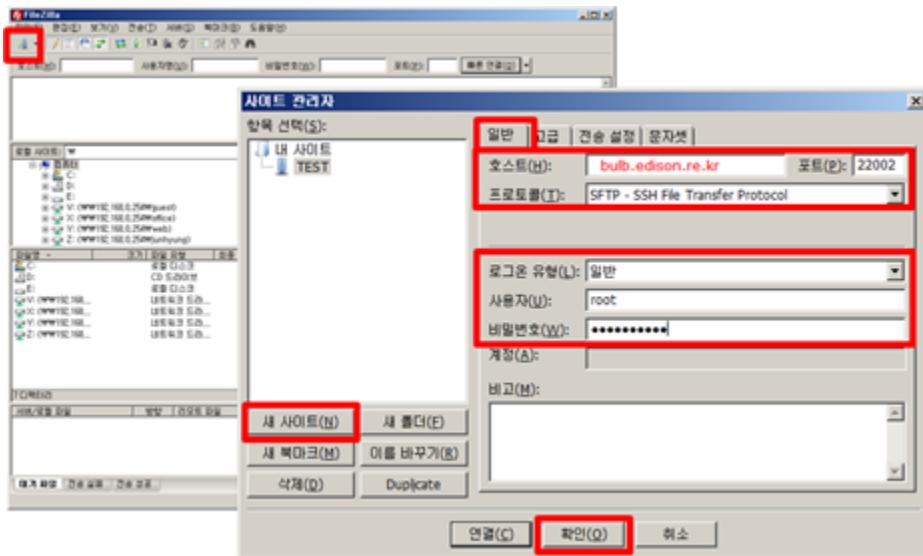
최초 접속 시 비밀번호를 꼭 변경해 주세요.

- 리눅스에서 비밀번호 변경하기

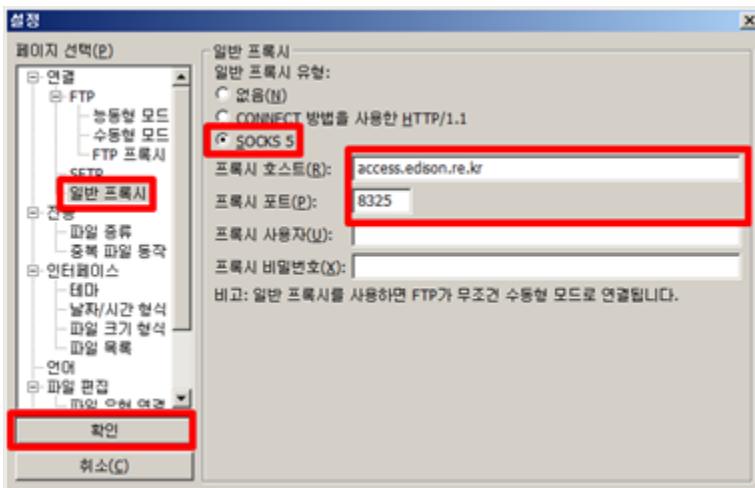
FileZilla Bulb 접속하기 (sftp)

File Zilla 다운로드 후 아래 그림과 같이 접속

- 호스트 : bulb.edison.re.kr, 포트 : 22002, 프로토콜 : SFTP
- FileZilla > 편집 > 설정 이동
- 프록시 : SOCK 5, 프록시 호스트 : access.edison.re.kr, 프록시 포트 8325



호스트 : bulb.edison.re.kr, 포트 : 22002, 프로토콜 : SFTP



프록시 : SOCK 5, 프록시 호스트 : access.edison.re.kr, 프록시 포트 8325

Mac OS Terminal로 bulb 접속하기 (ssh)

터미널 접속 후 아래 커멘드 입력

```
ssh -o ProxyCommand='nc -x access.edison.re.kr:8325 %h %p' [User_id]@bulb.edison.re.kr -p 22002
```

[User_id] 부분의 본인의 아이디를 입력하고 커멘드 실행

Linux Terminal로 bulb 접속하기 (ssh)

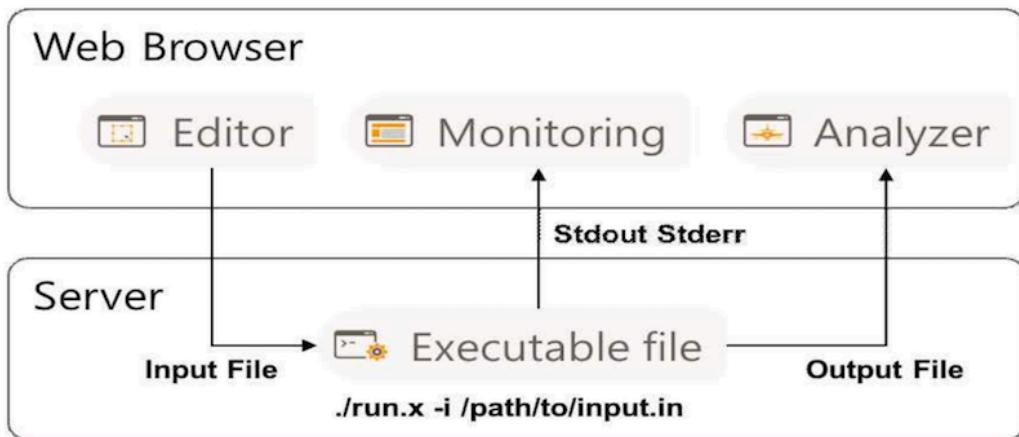
Cent OS 7

connect-proxy 설치 후 아래 커멘드 입력

```
ssh -o ProxyCommand="connect-proxy -S access.edison.re.kr:8325 %h %p" [User_id]@bulb.edison.re.kr -p 22002
```

Science App 프로그래밍 개요

EDISON 플랫폼은 리눅스(CentOS)기반에서 동작하고 있으며, 시뮬레이션 SW의 경우 리눅스 환경에서 동작해야 합니다. 컴파일이 필요한 언어인 경우 리눅스(CentOS)에서 컴파일이 완료 되어야 합니다. 이를 위해 시뮬레이션 SW 개발자들을 위한 개발자(Bulb) 서버를 제공하고 있으며, Bulb 환경에서 동작하는 실행 파일(or 스크립트)로 개발 되어야 합니다.



사이언스 앱 구성

- 입력 데이터를 받는 방법은 명령행 인자(Command Line Argument) 방식을 이용해 입력 데이터를 파일 형태로 읽어야 합니다.
- 해석 결과 데이터는 파일 형태로 출력해야 하며, result 폴더에 생성되어야 합니다.
- `stdout`, `stderr` 함수를 이용해 실행 사용자에게 제공하고자 하는 정보를 전달할 수 있습니다.

입출력 프로그래밍에 대한 기본적인 방법에 대해 알아보고, 각 언어별 프로그래밍 방법을 배워봅시다.

입력 프로그래밍

EDISON에서 정의한 입출력 형식을 준수해야 합니다. 시뮬레이션 SW의 실행 방식은 명령행 인자(Command Line Argument) 방식을 따르며, **./[실행 파일 명] [커맨드 옵션] [인풋 파일의 절대 경로]** 형태로 실행해야 합니다.

- 실행 파일 이름이 a.out이고 입력 옵션이 “-i”로 설정한 경우 실행되는 명령어는 다음과 같습니다.

```
$ ./a.out -i /home/user1/data/input.dat
```

언어별 예제 보기

- [C \(page 40\)](#)
- [Fortran \(page 71\)](#)
- [Python \(page 99\)](#)
- [R \(page 123\)](#)
- [Octave \(page 139\)](#)

입력 파일이 두개 이상인 경우 각각의 인풋 파일의 옵션은 서로 다른 값을 가져야 한다. 이 경우에서 실행 방식은 다음과 같다.

```
./[실행 파일 명] [인풋 옵션1] [인풋 파일 1의 절대 경로] [인풋 옵션2]  
[인풋 파일 2의 절대 경로] ...
```

- 입력 파일이 두개이며, input.dat, block.msh 파일을 각각 “-i”, “-m” 입력 옵션을 통해 파일을 받는 경우 실행되는 명령어는 다음과 같다.

```
$ ./a.out -i /home/user1/data/input.dat -m /home/user  
1/data/block.msh
```

언어별 예제 보기

- [C \(page 47\)](#)
- [Fortran \(page 76\)](#)
- [Python \(page 103\)](#)
- [R \(page 127\)](#)

3개 이상인 경우에도 서로 다른 입력 옵션을 정하고 이를 받을 수 있도록 코드를 작성하면 된다.

소스코드에서 [인풋 파일의 절대경로] 처리를 위해 파일 경로를 버퍼에 저장하는 경우, 버퍼 공간을 512byte 잡아야 합니다.

다음과 같이 프로그래밍 된 시뮬레이션 SW는 EDISON 플랫폼에서 서비스 될 수 있습니다.

- 입력 파일을 입력 받지 않고 특정 위치의 특정 이름인 입력 파일만 읽는 경우
- 사용자 키 입력을 통해 입력 파일의 이름이나 위치를 입력 받는 경우

Structured Data Editor (SDE)

EDISON 플랫폼에서는 SDE(Structured Data Editor)이라는 기능을 제공하여, 시뮬레이션 수행에 필요한 변수 값, 문자열, 벡터등의 데이터를 웹에서 바로 입력할 수 있는 기능을 제공하고 있습니다.

EDISON 사업 초창기 Inputdeck으로 불리던 데이터 형태를 Structured Data Editor로 명칭을 변경하였습니다.

SDE 데이터 타입 생성과 관련하여 아래 링크를 참고하시기 바랍니다.

- [데이터 타입 생성하기 \(page 151\)](#)

SDE를 자신의 시뮬레이션 SW에 활용하고 싶다면, SDE에서 생성되는 입력 파일을 읽을 수 있도록 프로그램을 작성해야 합니다. SDE 작성 시 입력 파일을 생성하는 규칙을 정할 수 있으며, 이 규칙에 따라 생성된 입력 파일을 읽어 올 수 있으면 됩니다. 프로그램 작성 시 유의 사항은 다음과 같습니다.

- SDE에서 생성된 파일은 text 파일 형태로 되어 있으며, 파일 한줄에 하나의 변수에 대한 이름(KEY)와 값(VALUE)로 구성되어 있다.
- SDE 생성 규칙(Value delimiter, Line delimiter 등)에 맞게 변수 값을 읽어야 함
- SDE 데이터의 생성 순서에 상관 없이도 동작해야 함
- 원하는 변수 값들이 정상적으로 입력되지 않았다면 에러 메시지를 발생 시켜야 함

Example

다음과 같이 숫자형 변수 2개(정수형 변수 1개, 실수형 변수 1개), 리스트형 변수 1개, 3차원 벡터 1개를 받는 SDE를 생성하고, 데이터 생성 방식을 아래 표와 같이 설정하면,

INPUTDECK 정의

value delimiter	SPACE ↴	Line delimiter	NULL ↴	Comment Char	<input type="text"/>	미리보기	KEY VALUE
Vector bracket	SQUARE_SPACE ↴	Vector Delimiter	SPACE ↴	미리보기	[A B C]		
Variable Name		Value	Description	타입 *	선택 ↴	※ 타입 선택 후 변수 추가	
INT1	42			변수명 *	<input type="text"/>		
REAL1	42.112						
LIST1	a ↴						
VECTOR1	1	1	1				

Case1

KEY	VALUE	KEY	VALUE
value delimiter	SPACE	Vector vracket	SQUARE_SPACE
line delimiter	NULL	Vector delimiter	SPACE

생성된 입력 파일은 다음과 같습니다.

```
INT1 42
REAL1 42.112
LIST1 a
VECTOR1 [ 1 0 0 ]
```

생성 방식을 달리하여 아래 표와 같이 설정하게 되면,

KEY	VALUE	KEY	VALUE
value delimiter	EQUAL	Vector vracket	SQUARE_SPACE
line delimiter	SEMICOLON	Vector delimiter	SPACE

생성된 입력 파일은 다음과 같습니다.

```
INT1 = 42 ;
REAL1 = 42.112 ;
LIST1 = a ;
VECTOR1 = [ 1 0 0 ] ;
```

해석기(Solver)는 입력 파일에서 해석에 필요한 조건을 찾아 해석시 활용해야 합니다. 이것에
해당 언어별 예제는 아래 링크에 있습니다.

- [C \(page 50\)](#)
- [Fortran \(page 79\)](#)
- [Python \(page 104\)](#)
- [R \(page 128\)](#)
- [Octave \(page 143\)](#)

simrc

simrc는 해석기(Solver)가 실행 되기전에 실행이 필요한 쉘스크립트를 저장하는 파일입니다. simrc에 작성된 쉘스크립트 명령어들은 해석기 실행되기 전에 실행됩니다. 앱 등록시 실행 파일과 같이 업로드해야 합니다.

쉘 프로그램은 유닉스에서 제공하는 명령어(인터페이스)를 그대로 사용할 수 있습니다.

\$PATH 환경변수

해석기로 등록한 실행 파일이나 스크립트의 환경변수 추가가 필요한 경우 simrc 파일에 환경 변수 등록 명령어를 활용할 수 있습니다.

gnuplot 명령어를 해석기에서 사용하려고 하는 경우 gnuplot을 실행하기 위해 실행 파일 위치를 다음과 같이 \$PATH 변수에 지정해야 합니다.

```
export PATH=/SYSTEM/gnuplot-4.6.3/bin/bin:$PATH
```

해석기 실행 파일과 위 명령어를 simrc 파일로 작성하고, 같이 업로드 하면 해석기에서 gnuplot 명령어를 사용할 수 있습니다.

\$LD_LIBRARY_PATH 환경변수

사이언스 앱 실행시 아래와 같이 공유 라이브러리(shared libraries)에 관련된 에러가 발생할 수 있습니다.

```
error while loading shared libraries: libblas.so.3: cannot open shared object file: No such file or directory
```

이경우 \$LD_LIBRARY_PATH 환경변수에 에러가 발생한 라이브러리의 경로를 지정해 주면 해결할 수 있습니다.

```
export LD_LIBRARY_PATH=/SYSTEM/lapack/3.8.0/lib:$LD_LIBRARY_PATH
```

l_{dd} 명령어를 통해 지정한 프로그램이 요구하는 공유 라이브러리 (shared libraries)를 출력하는 명령어입니다. 이 명령어를 통해 찾지 못한 공유 라이브러리를 확인 미리 확인 할 수 있습니다.

window에서 simrc 파일을 작성하는 경우 개행문자 방식이 unix 개열과 달라 에러가 발생할 수 있습니다.

윈도우 환경에서는 다음과 같은 방법으로 수정 가능합니다.

Sublime text3 의 경우 Preferences -> Setting -> User 에 아래 명령어 추가

“default_line_ending”: “unix”

출력 프로그래밍 개요

출력 프로그래밍 개요

해석 결과 데이터는 파일 형태로 출력해야 하며, result 폴더에 생성되어야 합니다.

result 폴더 생성하는 예제를 보시면 예제코드에서는 result 폴더를 생성하는 명령어를 실행하기 전 result 폴더를 삭제하는 명령어를 실행합니다. 프로그램 테스트를 반복하는 과정에서 결과 파일이 남아있는 경우 문제가 발생할 수 있습니다. 이를 방지하고자 result 폴더를 삭제하는 코드를 추가하였습니다.

이후 파일 입출력 함수 이용해 result 폴더안에 파일을 쓰기 모드로 생성하면 됩니다.

언어별 예제 보기

- [C \(page 57\)](#)
- [Fortran \(page 88\)](#)
- [Python \(page 111\)](#)
- [R \(page 133\)](#)
- [Octave \(page 148\)](#)

Simpot Setting

simpot

simpot는 해석기(Solver)의 해석이 종료된 이후 실행이 필요한 쉘스크립트를 저장하는 파일입니다. simpot에 작성된 쉘스크립트 명령어들은 해석기 실행이 끝난 이후 실행됩니다. 앱 등록시 실행 파일과 같이 업로드해야 합니다.

쉘 프로그램은 유닉스에서 제공하는 명령어(인터페이스)를 그대로 사용할 수 있습니다.

해석된 결과 데이터들을 폴더별로 정리하거나, 다른 실행파일로 후처리 과정이 필요한 경우 이 파일을 이용하시면 됩니다.

oneD file format

oneD file format

1차원 그래프를 EDISON 시스템에서 보여주기 원한다면, 아래와 같은 파일 형태로 데이터를 생성해야 합니다.

⌚ Data Structure

```
#NumField: nDatafield
#LabelX: xlabelname, LabelY: ylabelname
#Field1: FieldLegend, NumPoint: nPoint
    X11      Y11
    X21      Y21
    |
    XnPoint1  YnPoint1
#Field2: FieldLegend2, NumPoint: nPoint
    X12      Y12
    X22      Y22
    |
    XnPoint2  YnPoint2
.....
* printf("%10.3e %10.3e\n", x, y)
* printf("%10.3e %10.3d\n", (double)x, (double)y)
```

```
#NumField: 3
#LabelX: position(um), LabelY: energy(eV)
#Field1: Ec at V=0(V), NumPoint: 9283
    0.000e+00  9.338e-01
    6.464e-04  9.338e-01
    1.293e-03  9.338e-01
    1.939e-03  9.338e-01
    2.585e-03  9.338e-01
    3.595e+00  1.459e-01
    6.000e+00  1.459e-01
#Field2: Ev at V=0(V), NumPoint: 9283
    0.000e+00 -1.942e-01
    6.464e-04 -1.942e-01
    1.293e-03 -1.942e-01
    1.939e-03 -1.942e-01
    2.585e-03 -1.942e-01
```

- (1) nDataField <= 10
- (2) nPoint <= 100,000
- (3) Different fields in a same file should have same nPoint, and share labelnames.
- (4) All the strings (labelname, legends) should be <= 20 characters

oneD file

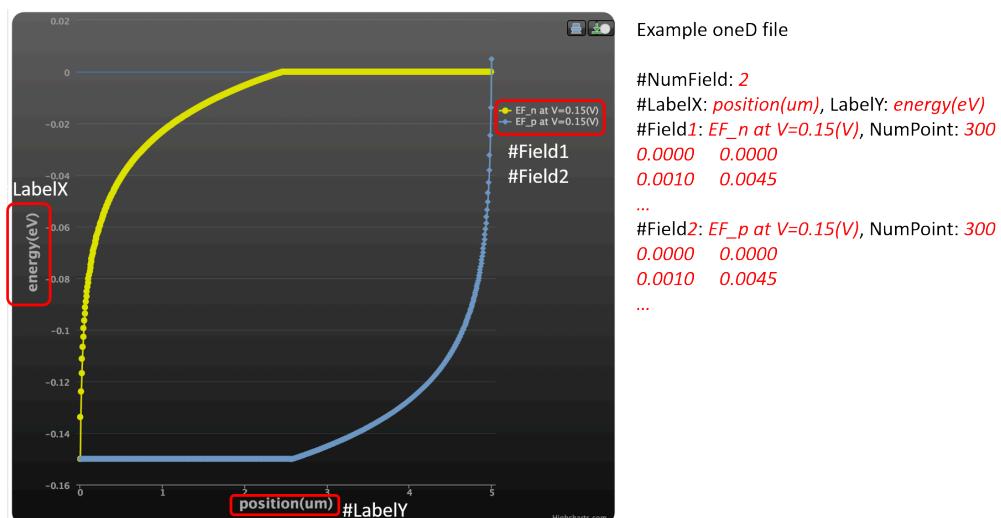
oneD 파일은 [OSPPlotViewer 분석기 \(page 31\)](#)를 통해 가시화 할 수 있습니다.

데이터 구조 설명

- #NumField: **nDatafield** : 한 화면에 출력할 그래프의 갯수를 입력해주는 부분으로 **nDatafield** 대신 생성하고자 하는 필드 값을 숫자로 입력해 주면 된다. 필드 값은 10 가 넘지 않도록 합니다.
 - ex) #NumField: 2 // 2개의 필드 생성
- #LabelX: **xlabelname**, LabelY: **ylabelname** : x축 y축 이름값을 넣어주는 부분으로 20자가 넘지 않도록 합니다.
 - ex) #LabelX: position(um), LabelY: energy(eV) // x축 이름을 position(um), y축 이름을 energy(eV)로 설정

- #Field1: **FieldLegnd**, NumPoint: **nPoint** : 첫번째 필드의 이름값과 데이터 갯수를 적는 부분
 - ex) #Field1: EF_n at V=0.15(V), NumPoint: 300 // Field1의 이름은 EF_n at V=0.15(V)이며, 300개의 데이터를 가지고 있음
- x축의 데이터 값과 y축의 데이터 값을 space 또는 tab 간격을 두고 위에서 정의한 것 수 만큼 데이터를 써주면 됩니다.
- #Field2도 Field1과 마찬가지로 작성해 주면 되며, 필수 갯수에 맞게 필드를 작성하면 됩니다.

샘플 데이터와 가시화 결과 화면



oneD result

언어별 예제 보기

- [C \(page 59\)](#)
- [Fortran \(page 90\)](#)
- [Python \(page 112\)](#)

Plotly file format

Plotly file 포맷은 Plotly 라이브러리를 활용하여 생성한 데이터를 json으로 저장한 파일을 의미합니다.

Python 예제 보기

- [Python \(page 118\)](#)

참고자료

[plotly.js figure reference](#)

[Plotly Python Document](#)

Gnuplot

Gnuplot

그누플롯은 2차원이나 3차원 함수나 자료를 그래프로 그려 주는 명령행 응용 소프트웨어입니다. EDISON 시스템에서 그누플롯을 사용하기 위해서는 `module load` 명령어를 통해 gnuplot 모듈을 불러와야 합니다. 설치되어 있는 gnuplot은 다음과 같습니다.

```
gnuplot/4.6.3  
gnuplot/5.0.6
```

[C 예제 보기 - oned file을 gnuplot을 이용 그림으로 저장하기 \(page 67\)](#)

Gnuplot을 활용 gif 파일 만들기

Gnuplot을 이용 3차원 3d bessel 함수가 시간에 따라 변화하는 그래프를 gif로 만드는 예제입니다.

run.plt

```
set term gif animate size 350,262
set output "animate.gif"

# color definitions
set palette rgb 3,9,9

unset key; unset colorbox; unset border; unset tics
set lmargin at screen 0.03
set bmargin at screen 0
set rmargin at screen 0.97
set tmargin at screen 1

set parametric
# Bessel function, which is moving in time
bessel(x,t) = besj0(x) * cos(2*pi*t)
# calculate the zeros for the bessel function (see Watson, "A Treatise on the
# Theory of Bessel Functions", 1966, page 505)
n = 6 # number of zeros
k = (n*pi-1.0/4*pi)
u_0 = k + 1/(8*k) - 31/(384*k)**3 + 3779/(15360*k)**5
set urange [0:u_0]
set vrange[0:1.5*pi]
set cbrange [-1:1]
set zrange[-1:1]

set isosamples 200,100
set pm3d depthorder
set view 40,200

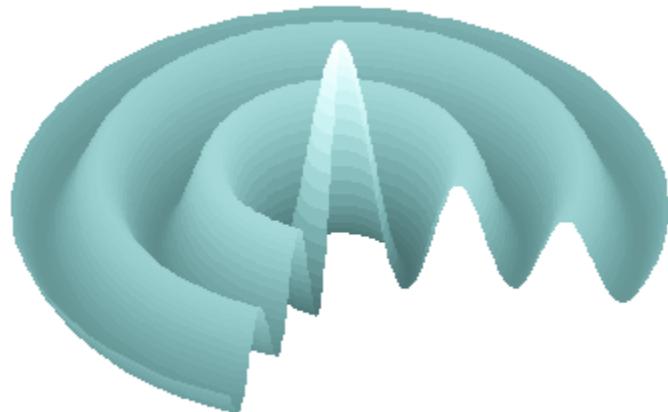
# initializing values for the loop and start the loop
t = 0
end_time = 1
load 'bessel.plt'
```

bessel.plt

```
# bessel loop
t = t + 0.02
#set output outfile
splot u*sin(v),u*cos(v),bessel(u,t) w pm3d ls 1
if(t<end_time) reread;
```

gnuplot 명령어를 통해 run.plt 파일을 실행하면, animate.gif 파일이 생성됩니다.

```
$ module load gnuplot
$ gnuplot run.plt
Could not find/open font when opening font "arial", using internal non-scalable font
End of animation sequence
```



gnuplot result

래퍼런스

GIF Animation 파일 만들기

imagemagick을 사용해 여러장의 그림 파일을 하나의 gif 파일로 생성할 수 있습니다. gif animation 생성하기 위해서는, 각 프레임별 그림파일이 생성되어 있어야 합니다. [bessel \[프레임넘버\].png](#) 인 그림파일들을 하나의 gif로 생성하는 경우

```
$ ls  
bessel001.png bessel008.png bessel015.png bessel022.png bessel029.png bessel036.png bessel043.png bessel050.png bessel002.png bessel009.png bessel016.png bessel023.png bessel030.png bessel037.png bessel044.png bessel003.png bessel010.png bessel017.png bessel024.png bessel031.png bessel038.png bessel045.png bessel004.png bessel011.png bessel018.png bessel025.png bessel032.png bessel039.png bessel046.png bessel005.png bessel012.png bessel019.png bessel026.png bessel033.png bessel040.png bessel047.png bessel006.png bessel013.png bessel020.png bessel027.png bessel034.png bessel041.png bessel048.png bessel007.png bessel014.png bessel021.png bessel028.png bessel035.png bessel042.png bessel049.png  
$ convert -delay 0.0001 -loop 0 bessel*.png result.gif  
$ ls -al result.gif  
-rw-r--r-- 1 edison edison 518738 2018-08-19 22:45 result.gif
```

그외 convert 명령어로는 다양한 그림 편집이 가능합니다.

[convert 명령어 사용법](#)

C 언어

리눅스에서 gcc로 c언어 컴파일하기

리눅스에서 hello world를 출력하는 간단한 C 코드를 컴파일하고 실행하는 예제 링크입니다.
[리눅스에서 gcc로 c언어 컴파일하기](#)

간단한 코드의 경우 gcc 명령어를 바로 사용하여 빌드 할 수 있습니다. 하지만 소스코드가 많은 경우 일일이 gcc 명령어를 통해 컴파일 하는 것은 무리가 있습니다. make 명령어를 사용하면 효과적으로 소스코드를 빌드 할 수 있습니다.

make를 이용한 빌드 방법

본 개발자 문서에서는 Makefile을 통해 C, Fortran 코드를 빌드 할 수 있는 예제 소스를 제공하고 있습니다.

gcc 컴파일러

EDISON 서버에 기본으로 설치되어 있는 gcc version은 4.4.7 입니다. `gcc -v` 명령어를 통해 버전을 확인할 수 있습니다.

```
[edison@edison ~]$ gcc -v
Using built-in specs.
Target: x86_64-redhat-linux
Configured with: .../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-bootstrap --enable-shared --enable-threads=posix --enable-checking=release --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-languages=c,c++,objc,obj-c++,java,fortran,ada --enable-java-awt=gtk --disable-dssi --with-java-home=/usr/lib/jvm/java-1.5.0-gcj-1.5.0.0/jre --enable-libgcj-multifile --enable-java-maintainer-mode --with-ecj-jar=/usr/share/java/ecj-ecj.jar --disable-libjava-multilib --with-ppl --with-cloog --with-tune=generic --with-arch_32=i686 --build=x86_64-redhat-linux
Thread model: posix
gcc version 4.4.7 20120313 (Red Hat 4.4.7-18) (GCC)
```

그 외 버전이 필요한 경우 `module avail` 명령어를 통해 이용 가능한 gcc 리스트를 확인하고, `module load gcc/[버전]` 명령어를 통해 사용할 수 있습니다.

```
[edison@bulb etc]# module avail
----- /usr/share/Modules/modulefiles -----
dot      module-git  module-info modules    null      us
e.own

----- /etc/modulefiles

R/3.3.2           gcc/5.5.0          intel/mk
l                 openmpi-1.8-x86_64
R/3.3.3(default)  gnuplot/4.6.3      mpi/gnu/openmp
i-1.6.5          openmpi-x86_64
gamess/gamess     gnuplot/5.0.6(default) mpi/intel/mpic
h-1.2.7p1         petsc/3.9.3
gcc/4.9.4          intel/2018        mpich-x86_6
4                 python/3.6.3
gcc/5.3.0          intel/2018-mpi   octave/4.0.3
gcc/5.4.0          intel/intel_11  openmpi/2.1.2
[edison@bulb ~]$ module load gcc/5.3.0
[edison@bulb ~]$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/SYSTEM/gcc-5.3.0/build/bin/../libexec/gcc/x86_64-unknown-linux-gnu/5.3.0/lto-wrapper
Target: x86_64-unknown-linux-gnu
Configured with: ./configure --prefix=/SYSTEM_BULB/gcc-5.3.0/build
Thread model: posix
gcc version 5.3.0 (GCC)
[edison@bulb ~]$
```

입력 파일이 1개인 경우

[링크] Github에서 보기

[링크] 소스코드 다운받기

예제코드 다운로드 및 실행

Github 사이트에 접속 해서 전체 소스가 압축된 파일을 다운 받거나, `git clone` 명령어를 이용해 소스 코드를 받을수 있습니다.

Bulb 서버에 접속 후 아래 명령어를 실행하면, 해당 소스코드를 다운 받을 수 있습니다.

Bulb 서버가 아닌 곳에서도 git이 설치되어 있다면, `git clone` 명령어를 통해 소스코드를 다운로드 받을 수 있습니다.

- `git 설치하기`

```
$ git clone https://github.com/sp-edison/c_example_input1.git
```

다운로드가 완료되면, `c_example_input1` 폴더가 생성됩니다. 폴더 구성은 다음과 같습니다.

```
c_example_input1
| README.md
|---src
|   | Makefile
|   | main.c
|---inp
|   | input.dat
```

`src` 폴더로 이동하여 `make all` 명령어를 사용하면, 컴파일이 완료됩니다.

```
$ cd c_example_input1/src
$ make all
gcc -c main.c -o main.o
Compiled main.c successfully!
gcc -o ../bin>Hello.x main.o
Linking complete!
```

컴파일이 완료되면 `c_example_input1` 폴더 안에 `bin` 폴더가 생성되며, `src/Makefile`에서 지정한 `TARGET` 명으로 실행 파일이 생성됩니다.

예제의 경우 `TARGET` 이 `Hello.x`로 설정되어 있습니다.

생성된 `bin` 폴더로 이동하여, `inp` 폴더에 있는 `input.dat`을 입력 파일로 넣고 실행시 여러 없이 실행 종료됨을 확인할 수 있습니다.

```
$ cd ../bin
$ ./Hello.x -i ../inp/input.dat
Succeed to open inputfile. Path: ../inp/input.dat
num
1000
```

이 예제에는 옵션 값이 맞는지 체크하고 옵션 뒤에 붙은 추가 파라미터 값이 파일 경로인지 확인하고 종료하는 프로그램입니다. 옵션 명이 잘못되었거나 입력 파일 경로가 잘못된 경우에는 에러가 발생합니다.

컴파일 된 파일을 삭제하고 싶은 경우에는 다시 `src` 폴더로 이동하여 `make clean` 명령어를 사용하면, 오브젝트 파일과 실행 파일을 삭제하게 됩니다.

```
$ cd ../src
$ make clean
rm -rf main.o ../bin>Hello.x
```

Source code

main.c

```

#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>

int main (int argc, char *argv[]) {
    int opt;
    FILE *fp_input;
    char buf_string[256];

    // Detect the end of the options.
    while((opt = getopt(argc, argv, "i:")) != -1) {
        switch(opt) {
            case 'i':
                if((fp_input = fopen(optarg
g,"r")) == NULL ) {
                    fprintf(stderr, "Error
opening inputfile Path: %s\n", optarg);
                    return -1;
                } else {
                    printf("Succeed to ope
n inputfile Path: %s\n", optarg);
                }
                break;
            default:
                printf("Usage: %s -i [filepat
h] \n", argv[0]);
                return -1;
        }
    }

    // Write an algorithm program in this section.

    while(1) {
        fscanf(fp_input, "%s", buf_string);
        if(feof(fp_input))
            break;
        printf("%s\n", buf_string);
    }

    // Input file close
    fclose(fp_input);
    return 0;
}

```

주요 함수 설명

```
int main (int argc, char *argv[]);
```

많은 프로그램들은 명령행 인자(Command Line Argument) 방식으로 프로그램 실행을 위한 입력 데이터를 받게됩니다. 이때 필요한 변수가 `argc`, `argv`입니다.

- `argc` 는, 프로그램을 실행할 때 지정해 준 “명령행 인자”의 “개수”가 저장되는 곳입니다.
- `argv` 는, 프로그램을 실행할 때 지정해 준 “명령행 인자의 문자열들”이 실제로 저장되는 배열입니다.

리눅스에서 쓰이는 많은 명령어들도 이러한 형태로 사용되고 있습니다. 예를들어 `cd result` 라는 명령어도 `cd`라는 디렉토리를 이동하는 프로그램과 이 프로그램을 실행하기 위한 입력 데이터 `result`로 나누어 지고, `result` 폴더로 이동하라는 명령이 됩니다.

```
int getopt(int argc, char * const argv[], const char *optstring);
```

입력 파라미터는 `main()` 함수가 받은 파라미터를 전달하는 `argc`, `argv` 와 처리해야하는 옵션 값을 가지고 있는 `optstring`로 구성되어 있습니다. `getopt()` 함수로 처리하고자 하는 옵션을 `optstring`에 저장해야 하며, 옵션 뒤에 추가 파라미터가 필요한 경우에는 옵션명 뒤에 “`:`”를 추가합니다. 옵션명은 1개의 문자로 구성되며, 문자열을 옵션으로 받는 경우는 `getoptlong()` 함수를 사용하면 됩니다.

예를 들어 단일 옵션 `-h`, `-v` 와 별도 파라미터가 필요한 옵션 `-i filename` 을 체크하고자 한다면 “`hvi:`” 값을 옵션 스트링에 넣어 주면 됩니다.

이 함수를 사용하기 위해서는 `#include <getopt.h>` 또는 `#include <unistd.h>` 선언을 해야하며, `getopt()` 함수에 필요한 전역 변수도 호출이 됩니다.

- `optarg` : 옵션 뒤에 별도의 파라미터 값이 오는 경우, 이를 파싱한 결과 파라미터 값은 `optarg`에 문자열로 저장됩니다.

- **optind** : 다음번 처리될 옵션의 인덱스이다. 만약 파싱한 옵션이후에 추가적인 파라미터를 받는다면 (예를 들어 입력 파일 이름 같이) 이 값을 활용할 수 있다.
`getopt()` 함수는 한 번 호출될 때마다 이 값을 업데이트 합니다.
- **opterr** : 옵션에 문제가 있을 때, 이 값은 0이 아닌 값이되며, `getopt()` 함수가 메시지를 표시하게 됩니다.
- **optopt** : 알 수 없는 옵션을 만났을 때 해당 옵션이 여기에 들어갑니다. (이 때 `getopt`의 리턴값은 '?'가 된다.)

주요 코드 설명

```

...
while((opt = getopt(argc, argv, "i:")) != -1) {
    switch(opt) {
        case 'i':
            if((fp_input = fopen(optarg
g,"r")) == NULL ) {
                fprintf(stderr, "Erro
r opening inputfile Path: %s\n", optarg);
                return -1;
            } else {
                printf("Succeed to op
en inputfile Path: %s\n", optarg);
            }
            break;
        default:
            printf("Usage: %s -i [filepat
h] \n", argv[0]);
            return -1;
    }
}
...

```

`while` 문 안에 있는 `opt = getopt(argc, argv, "i:")` 은 명령행 인자의 문자열들을 확인해 `-i filepath` 형태가 있는 경우 `i` 는 `opt` 변수에 `filepath`는 `optarg` 변수에 저장합니다. 정의한 `-i` 옵션 이외의 다른 옵션값(ex `-m`)이 들어오는 경우 `default` 문에서 예외 처리하였습니다.

만약 `-i, -m` 옵션 2개를 통해 입력 파일을 받기 원한다면, `"i:"`
 `-> "i:m:"` 으로 고치고, 아래 `switch` 문의 `m` case를 추가하면 됩니다.

-i 옵션 값이 입력된 경우 바로 뒤에 입력한 `filepath` 가 재대로 입력되었는지 확인합니다. 입력된 경로에 파일이 있다면, `fp_input = fopen(optarg, "r")` 값이 `NULL` 이 아닌 값 을 가지게 됩니다. `NULL` 인 경우에는 파일 경로가 잘못 입력된 경우로 이에 대한 에러 로그를 표준 에러(Standard error)를 통해 표시합니다.

```
fprintf(stderr, "Error opening inputfile Path: %s\n", optarg);
```

또한 `return -1;` 을 통해 `main` 함수의 리턴 값을 0이 아닌 값을 반환하여 정상적으로 종료 되지 않았다는 것을 알립니다.

```
...
while(1) {
    fscanf(fp_input, "%s", buf_string);
    if(feof(fp_input))
        break;
    printf("%s\n", buf_string);
}

// Input file close
fclose(fp_input);
return 0;
}
```

정상적으로 입력 파일을 오픈한 경우 오픈한 파일의 내용을 읽어 모니터에 출력합니다. 파일을 읽는 함수 중 `fscanf(fp_input, "%s", buf_string);` 함수를 통해 문자열 하나씩 읽어 `buf_string`에 저장하고 이를 표준출력으로 출력합니다. 입력 파일을 모두 읽고난 이후 `fclose(fp_input);`을 통해 파일을 닫고 `return 0;`을 통해 `main` 함수가 정상적으로 종료됨을 알립니다.

입력 파일이 여러개인 경우

입력 파일이 여러개인 경우는 1개인 경우 예제에서 `getopt()` 함수에 입력 받고자 하는 옵션을 추가하고, `switch` 문에서 추가한 옵션에 대한 코드를 추가하면 됩니다.

예를 들어 `-i, -m` 옵션 2개를 통해 입력 파일을 받기 원한다면, `"i:" -> "i:m:"` 으로 고치고, 아래 `switch` 문의 `m` case를 추가하면 됩니다.

```

#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>

int main (int argc, char *argv[]) {
    int opt;
    FILE *fp_input;
    FILE *fp_mesh;           //mesh 파일을 읽기 위한 파일 포인터
    추가 선언
    ...
    ...
    // Detect the end of the options.
    while((opt = getopt(argc, argv, "i:m:")) != -1) {
        switch(opt) {
            case 'i':
                if((fp_input = fopen(optarg
g,"r")) == NULL ) {
                    fprintf(stderr, "Error
opening inputfile Path: %s\n", optarg);
                    return -1;
                } else {
                    printf("Succeed to ope
n inputfile Path: %s\n", optarg);
                }
                break;
            case 'm':
                if((fp_mesh = fopen(optarg
g,"r")) == NULL ) {
                    fprintf(stderr, "Er
ror opening meshfile Path: %s\n", optarg);
                    return -1;
                } else {
                    printf("Succeed to
open meshfile Path: %s\n", optarg);
                }
                break;
            default:
                printf("Usage: %s -i [inputfil
e] -m [meshfile]\n", argv[0]);
                return -1;
        }
    ...
}

```

```
...
fclose(fp_input);
fclose(fp_mesh);
...
...
}
```

SDE case study

C언어로 SDE에서 생성된 입력 파일을 읽는 방법을 설명하고자 합니다.

관련 문서 링크

- [SDE 프로그래밍 방법 개요 \(page 25\)](#)
- [SDE 데이터 타입 생성하기 \(page 151\)](#)

다음과 같이 숫자형 변수 2개(정수형 변수 1개, 실수형 변수 1개), 리스트형 변수 1개, 3차원 벡터 1개를 받는 SDE를 생성했습니다.

INPUTDECK 정의

value delimiter	SPACE	Line delimiter	NULL	Comment Char	<input type="text"/>	미리보기	KEY VALUE
Vector bracket	SQUARE_SPACE	Vector Delimiter	SPACE	미리보기	[A B C]		
Variable Name	Value	Description	타입 * 선택 * ※ 타입 선택 후 변수 추가 변수명 *				
INT1	<input type="text" value="42"/>	<input type="button" value="?"/>					
REAL1	<input type="text" value="42.112"/>	<input type="button" value="?"/>					
LIST1	a	<input type="button" value="?"/>					
VECTOR1	<input type="text" value="1"/> <input type="text" value="1"/> <input type="text" value="1"/>	<input type="button" value="?"/>					

Case1

데이터 생성 방식은 다음과 같이 설정했습니다.

KEY	VALUE	KEY	VALUE
value delimiter	SPACE	Vector vracket	SQUARE_SPACE
line delimiter	NULL	Vector delimiter	SPACE

이렇게 설정되어 생성된 입력 파일은 다음과 같습니다.

```
INT1 42
REAL1 42.112
LIST1 a
VECTOR1 [ 1 0 0 ]
```

Example

명령행 인자(Command Line Argument) 방식으로 생성된 입력 파일을 읽고 입력된 변수 값을 출력하는 예제 코드입니다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>

typedef struct _inputparam {
    int int1;
    double real1;
    char list1;
    int vector1[3];
} INPUT;

int main (int argc, char* argv[])
{
    int opt;
    FILE *fp_input ;
    char buf_char[256];
    INPUT input;

    // Detect the end of the options.
    while((opt = getopt(argc, argv, "i:")) != -1 ) {
        switch(opt) {
            case 'i':
                if((fp_input = fopen(optarg,"r"))
== NULL ) {
                    fprintf(stderr, "Error open
ing inputfile Path: %s\n", optarg);
                    return -1;
                } else {
                    printf("Succeed to open inp
utfile Path: %s\n", optarg);
                }
                break;
            default:
                printf("Usage: %s -i [filepath]
\n", argv[0]);
                return -1;
        }
    }
    while(1) {
        fscanf(fp_input, "%s", buf_char);

        if(feof(fp_input))
            break;
    }
}
```

```
if(!strcmp(buf_char, "INT1")) {
    fscanf(fp_input, "%d", &input.int1);
} else if(!strcmp(buf_char, "REAL1")) {
    fscanf(fp_input, "%lf", &input.real1);
} else if(!strcmp(buf_char, "LIST1")) {
    fscanf(fp_input, "%s", &input.list1);
} else if(!strcmp(buf_char, "VECTOR1")) {
    fscanf(fp_input, "%*s %d %d %d %*s", &input.vector1[0], &input.vector1[1], &input.vector1[2]);
} else {
    printf("Error Invalid value name :: %s\n", buf_char);
    exit(1);
}

printf("int1: %d \n", input.int1);
printf("real1: %f \n", input.real1);
printf("list1: %c \n", input.list1);
printf("vector1 = %d %d %d \n", input.vector1[0], input.vector1[1], input.vector1[2]);

fclose(fp_input);

return 0;
}
```

위 예제코드는 입력 파일이 1개인 경우 예제 (page 40)의 코드에서 SDE 파일을 읽는 부분을 추가하였습니다.

주요 코드 설명

```

...
[1] while(1) {
[2]     fscanf(fp_input, "%s", buf_char);

[3]     if(feof(fp_input))
[4]         break;

[4]     if(!strcmp(buf_char, "INT1")) {
[5]         fscanf(fp_input, "%d", &input.int1);
[6]     } else if(!strcmp(buf_char, "REAL1")) {
[7]         fscanf(fp_input, "%lf", &input.real1);
[8]     } else if(!strcmp(buf_char, "LIST1")) {
[9]         fscanf(fp_input, "%s", &input.list1);
[10]    } else if(!strcmp(buf_char, "VECTOR1")) {
[11]        fscanf(fp_input, "%*s %d %d %d %*s", &input.vector
1[0], &input.vector1[1], &input.vector1[2]);
[12]    } else {
[13]        printf("Error Invalid value name :: %s\n", buf_cha
r);
[14]        exit(1);
[15]    }
}
...

```

[1] `while` 문을 이용해 파일을 처음부터 끝까지 읽습니다.

[2] `fscanf()` 함수를 이용해 변수 이름이 저장되어 있는 하나의 문자열(첫 단어)를 `buf_char`에 저장합니다.

[3] `feof()` 함수를 이용해 파일의 끝까지 읽으면 `break` 문을 이용해 `while` 문을 빠져 나오게 됩니다.

[4] 입력 파일에서 읽은 변수 이름을 확인하여 저장 합니다. 벡터 변수의 경우 변수 이름과 값 사이에 있는 [,] 문자를 `%*s` 를 사용해 읽기만 하고 따로 변수에 저장하지 않습니다.

[5] 원하지 않은 변수 값이 입력되는 경우 이에 대한 에러 메시지를 표시하고 프로그램을 종료합니다.

SDE case study 2

SDE 생성시 데이터 생성 방식을 아래와 같이 설정한다면, 생성되는 입력 파일의 모양이 약간 달라질 것입니다.

KEY	VALUE	KEY	VALUE
value delimiter	EQUAL	Vector vracket	SQUARE_SPACE
line delimiter	SEMICOLON	Vector delimiter	SPACE

생성된 입력 파일

```
INT1 = 42 ;
REAL1 = 42.112 ;
LIST1 = a ;
VECTOR1 = [ 1 0 0 ] ;
```

추가된 value delimiter와 line delimiter를 고려해 코딩을 해야 합니다. `%*s` 이용해 변수 이름과 변수 값 사이에 있는 `=`와 변수 끝에 있는 `;`을 파일에서 읽기만 하고, 따로 저장하지 않는 부분을 추가하면 됩니다. `fscanf(fp_input, "%d", &input.int1); ->`
`fscanf(fp_input, "%*s %d %*s", &input.int1);`

변경전 코드

```
...
[4]     if(!strcmp(buf_char, "INT1")) {
            fscanf(fp_input, "%d", &input.int1);
        } else if(!strcmp(buf_char, "REAL1")) {
            fscanf(fp_input, "%lf", &input.real1);
        } else if(!strcmp(buf_char, "LIST1")) {
            fscanf(fp_input, "%s", &input.list1);
        } else if(!strcmp(buf_char, "VECTOR1")) {
            fscanf(fp_input, "%*s %d %d %*s", &input.vector
1[0], &input.vector1[1], &input.vector1[2]);
        ...
    }
```

변경후 코드

```
...
[4]    if(!strcmp(buf_char, "INT1")) {
        fscanf(fp_input, "%*s %d %*s", &input.int1);
    } else if(!strcmp(buf_char, "REAL1")) {
        fscanf(fp_input, "%*s %lf %*s", &input.real1);
    } else if(!strcmp(buf_char, "LIST1")) {
        fscanf(fp_input, "%*s %s %*s", &input.list1);
    } else if(!strcmp(buf_char, "VECTOR1")) {
        fscanf(fp_input, "%*s %*s %d %d %d %*s %*s", &input.vector1[0], &input.vector1[1], &input.vector1[2]);
    ...
}
```

Output programing

result 폴더 생성

C 언어의 경우 `#include <stdlib.h>` 헤더를 선언하고 `int system (const char * string);` 함수를 이용하면 되며, `result` 폴더를 생성하는 예제는 아래와 같습니다.

```
#include <stdlib.h>

int main (int argc, char* argv[])
{
    ...
    system("rm -rf result");
    system("mkdir result");
    ...
    return 0
}
```

출력 파일 쓰기

C 언어에서 파일을 읽고 쓰기 위해서는 파일 입출력 함수를 사용한다. 출력 파일을 쓰기 위한 코드 구성은 다음과 같습니다.

```
#include <stdio.h>
...
int main (int argc, char *argv[]) {
    ...
    ...
    // 파일 포인터를 이용 result/result.txt을 쓰기 모드로 오픈
    fp_out = fopen("result/result.txt","w");
    ...
    ...
    // fprintf() 함수를 이용해 파일에 내용을 씁
    fprintf(fp_out,"Hello EDISON.\n");
    ...
    ...
    // 파일 쓰기가 끝났으면, 파일을 닫음.
    fclose(fp_out);
    ...
}
```

`fprintf()` 또는 `fputs()` 함수를 활용해 파일을 쓸 수 있습니다.

oneD

sin() 함수 결과를 oned로 출력하기

[링크] [Github에서 보기](#)

[링크] [소스코드 다운받기](#)

예제코드 다운로드 및 실행

1개의 입력 파일 읽어, sin 그래프를 그리는 C언어 예제 파일입니다.

다음 수식의 변수들을 입력으로 받으며, 입력 변수는 a,b,c,d 총 4개입니다.

입력 파일의 경우에는 Value delimiter를 '='을 사용하였으며, Line delimiter를 구분하기 위해 ';'를 사용하였습니다. 파일로 입력을 받으며, 샘플 입력 파일은 아래와 같으며, **inp** 폴더에 **input.dat**로 저장되어 있습니다.

```
a = 1 ;
b = 0.4 ;
c = -0.5 ;
d = 0.3 ;
```

본 예제는 ./[실행파일명] -[옵션] [입력 파일 경로]로 실행시 옵션 뒤에 입력된 경로의 파일을 열고 닫는 예제로 Makefile과 소스코드는 **src** 폴더에 저장되어 있으며, 컴파일이 완료되면 바이너리 파일은 **bin** 폴더에 저장됩니다.

설치하기

Github 사이트에 접속 해서 전체 소스가 압축된 파일을 다운 받거나, **git clone** 명령어를 이용해 소스 코드를 받을수 있습니다.

Bulb 서버에 접속 후 아래 명령어를 실행하면, 해당 소스코드를 다운 받을 수 있습니다.

Bulb 서버가 아닌 곳에서도 git이 설치되어 있다면, **git clone** 명령어를 통해 소스코드를 다운로드 받을 수 있습니다.

- **git 설치하기**

```
$ git clone https://github.com/sp-edison/c_example_oneD.git
```

```
c_example_oneD
├── README.md
└── src
    ├── Makefile
    └── main.c
└── inp
    └── input.dat
```

`src` 폴더로 이동하여 `make all` 명령어를 사용하면, 컴파일이 완료됩니다.

```
$ cd c_example_oneD/src
$ make all
gcc -c main.c -o main.o
Compiled main.c successfully!
gcc -o ../bin/Hello.x main.o
Linking complete!
```

컴파일이 완료되면 `c_example_oneD` 폴더 안에 `bin` 폴더가 생성되며, `src/Makefile`에서 지정한 `TARGET` 명으로 실행 파일이 생성됩니다.

예제의 경우 `TARGET`이 `Sin.x`로 설정되어 있습니다.

생성된 `bin` 폴더로 이동하여, `inp` 폴더에 있는 `input.dat`을 입력 파일로 넣고 실행시 여러 없이 실행 종료됨을 확인할 수 있습니다.

```
$ cd ../bin
$ ./Sin.x -i ../inp/input.dat
User input is :
A = 1.000000
B = 0.400000
C = -0.500000
D = 0.300000
```

해당 예제를 이용하여 EDISON 웹포털에서 자동으로 소스 컴파일을 하실수 있습니다. 자동으로 소스 컴파일을 하기위해 선행되어야하는 조건은 아래와 같습니다.

- 본 예제와 같이 Makefile과 소스코드는 **src** 폴더에 저장되어 있어야 합니다.
- **src** 폴더 안에서 `make all` 커맨드 입력시, 컴파일이 정상적으로 완료된 경우 바이너리 파일이 **bin** 폴더에 저장이 되면 됩니다.

Source code

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <math.h>

#define PI 3.140592
#define SIZE 128

typedef struct _inputparam {
//y = A sin(Bx + C) +D
    double a;          //A
    double b;          //B
    double c;          //C
    double d;          //D
} INPUT;

int main (int argc, char *argv[]) {
    int opt;
    FILE *fp_input, *fp_output;
    char buf_char[512];
    INPUT input;

    double x,y;
    int t;

    // Detect the end of the options.
    while((opt = getopt(argc, argv, "i:")) != -1 ) {
        switch(opt) {
            case 'i':
                fp_input = fopen(optarg,"r");
                break;
            default:
                printf("Usage: %s -i [Inputfile path]\n", argv[0]);
                exit(1);
        }
    }

    // Input file open
    if(fp_input == NULL){
        printf("Failed to open input file for -i \n");
        exit(1);
    }
```

```
// Write an algorithm program in this section.

while(1) {
    fscanf(fp_input, "%s", buf_char);

    if(feof(fp_input))
        break;

    if(!strcmp(buf_char, "a")) {
        fscanf(fp_input, "%*s %lf %*s", &input.a);
    } else if(!strcmp(buf_char, "b")) {
        fscanf(fp_input, "%*s %lf %*s", &input.b);
    } else if(!strcmp(buf_char, "c")) {
        fscanf(fp_input, "%*s %lf %*s", &input.c);
    } else if(!strcmp(buf_char, "d")) {
        fscanf(fp_input, "%*s %lf %*s", &input.d);
    } else {
        printf("Error Invalid value name :: %s\n", buf_char);
        exit(1);
    }
}

// Input file close
fclose(fp_input);

printf("User input is : \n");
printf("A = %f \n", input.a);
printf("B = %f \n", input.b);
printf("C = %f \n", input.c);
printf("D = %f \n", input.d);

system("rm -rf result");
system("mkdir result");

// Make oneD file
fp_output = fopen("result/result.oneD", "w");

fprintf(fp_output, "#NumField: 1\n");
fprintf(fp_output, "#LabelX: time, LabelY: a*sine(bx+c)+d\n");
fprintf(fp_output, "#Field1: a=%f b=%f c=%f d=%f, NumPoint:%d\n",
        input.a, input.b, input.c, input.d, SIZE);
```

```

for(t=0; t< SIZE; t++) {
    x = (4*PI * t)/SIZE -2*PI;
    y = input.a*(sin( input.b*x +input.c)) +input.d;
    fprintf(fp_output, "%10.3f %10.3f\n", x, y);
}

fclose(fp_output);
return 0;
}

```

###주요 코드 설명

입력 파일을 읽고 이를 각각의 변수로 저장하는 부분은 [SDE 프로그래밍 \(page 50\)](#) 예제를 참고하였습니다.

```

...
#include <math.h>

#define PI 3.140592
#define SIZE 128

```

- sin() 함수를 이용하기 위해 `math.h` 를 include 하였으며, 필요한 상수들을 define 하였습니다.

```

...
system("rm -rf result");
system("mkdir result");

fp_out = fopen("result/result.oneD", "w");
...

```

- 결과 데이터를 저장할 `result` 폴더를 생성하는 부분이며, `result` 폴더안에 `result.oneD` 파일을 생성하고 쓰기모드로 오픈하여 `fp_out` 파일포인터에 넣어주었습니다.

```

    fprintf(fp_output, "#NumField: 1\n");
    fprintf(fp_output, "#LabelX: time, LabelY: a*sine(bx+c)+d
\n");
    fprintf(fp_output, "#Field1: a=%f b=%f c=%f d=%f, NumPoint:%d\n",
            input.a, input.b, input.c, input.d, SIZE);

```

- oneD 파일의 헤더 부분의 내용을 생성합니다. [oneD 데이터 구조 \(page 31\)](#)에 대한 자세한 설명은 해당 페이지를 참조하시기 바랍니다.
- 필드가 1개이며, x라벨이 time y라벨이 $a * \sin(bx+c)+d$ 이고 필드의 이름은 사용자가 입력한 a,b,c,d 값을 나타내고 있습니다.

```

...
double x,y;
int t;
...
for(t=0; t< SIZE; t++) {
    x = (4*PI * t)/SIZE -2*PI;
    y = input.a*(sin( input.b*x +input.c)) +input.d;
    fprintf(fp_output, "%10.3f %10.3f\n", x, y);
}
fclose(fp_out);

```

- t=0 부터 SIZE 까지 for문을 수행하게됩니다. 잇대 에서 까지 SIZE등분으로 일정하게 나눈 값을 x에 저장하고, 이 x값에 대한 결과 값을 y에 저장한 뒤 `fprintf()` 를 이용해 result.oneD 파일에 쓰게됩니다.
- 이때 저장하는 데이터의 규칙은 `%10.3f` 값을 주어 10진수 형태로 총 10자리를 가지고, 소수점 아래로는 3자리를 가지도록 저장하였습니다.

Gnuplot을 이용해 oneD plot 파일을 그림으로 저장하기

[링크] [Github에서 보기](#)

[링크] [소스코드 다운받기](#)

코드는 [oneD 파일 출력하기 \(page 59\)](#)의 코드에서 oneD 파일을 Gnuplot을 통해 line 그래프를 그리고 이를 그림파일로 저장하는 예제입니다.

주요 코드

```

...
fp_output = fopen("plot.gnu","w");

fprintf(fp_output,"set term png\n");
fprintf(fp_output,"set output \"result.png\"\n");
fprintf(fp_output,"set xrange[-6:6]\n");
fprintf(fp_output,"set yrange[-1.5:1.5]\n");
fprintf(fp_output,"plot 'result.oneD' using 1:2 with line
s\n");

fclose(fp_output);

system("gnuplot plot.gnu");

system("rm -f plot.gnu");

...

```

fopen 함수를 통해 plot.gnu 파일에 Gnuplot 명령어들을 씁니다. 이를 통해 생성된 plot.gnu 파일은 다음과 같습니다.

```

set term png      // png 형태로 저장
set output "result.png" //저장하는 파일명은 result.png
set xrange[-6:6]    // x축의 범위는 -6~6까지
set yrange[-1.5:1.5] // y축의 범위는 -1.5~1.5까지
plot 'result.oneD' using 1:2 with lines // result.oneD 파일
을 읽어 첫번째 column을 x 데이터로, 두번째 column을 y 데이터로 사용해
line plot

```

gnuplot에서 2-dimensional data file을 읽는 경우 아래와 같은 형태로 저장된 데이터를 읽습니다. #은 주석으로 처리되며, 공백이나 탭으로 column을 구분합니다.

```
#  X      Y  
1.0  1.2  
2.0  1.8  
3.0  1.6  
...
```

Gnuplot Data load 방법

생성된 plot.gnu 파일을 gnuplot 명령어로 실행합니다.

Fortran 언어

리눅스에서 gcc로 fortran언어 컴파일하기

fortran 언어의 경우에도 gcc 컴파일러를 사용해 소스코드를 빌드할 수 있습니다. **gfortran** 명령어를 사용하며, C언어 빌드와 방식이 유사합니다. 포트란 언어도 역시 make 명령어를 통해 빌드하는것이 효과적입니다.

본 개발자 문서에서는 Makefile을 통해 C,Fortran 코드를 빌드 할 수 있는 예제 소스를 제공하고 있습니다.

gfortran 컴파일러

EDISON 서버에 기본으로 설치되어 있는 gcc version 4.4.7 버전입니다. **gfortran -v** 명령어를 통해 버전을 확인할 수 있습니다.

```
[edison@edison ~]$ gfortran -v
Using built-in specs.
Target: x86_64-redhat-linux
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-bootstrap --enable-shared --enable-threads=posix --enable-checking=release --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-languages=c,c++,objc,obj-c++,java,fortran,ada --enable-java-awt=gtk --disable-dssi --with-java-home=/usr/lib/jvm/java-1.5.0-gcj-1.5.0.0/jre --enable-libgcj-multifile --enable-java-maintainer-mode --with-ecj-jar=/usr/share/java/ecj-ecj.jar --disable-libjava-multilib --with-ppl --with-cloog --with-tune=generic --with-arch_32=i686 --build=x86_64-redhat-linux
Thread model: posix
gcc version 4.4.7 20120313 (Red Hat 4.4.7-18) (GCC)
```

그 외 버전이 필요한 경우 **module avail** 명령어를 통해 이용 가능한 gcc 리스트를 확인하고, **module load gcc/[버전]** 명령어를 통해 사용할 수 있습니다.

```
[edison@bulb etc]# module avail
----- /usr/share/Modules/modulefiles -----
dot      module-git  module-info modules    null      us
e.own

----- /etc/modulefiles

R/3.3.2          gcc/5.5.0           intel/mk
l               openmpi-1.8-x86_64
R/3.3.3(default)  gnuplot/4.6.3       mpi/gnu/openmp
i-1.6.5   openmpi-x86_64
gamess/gamess     gnuplot/5.0.6(default) mpi/intel/mpic
h-1.2.7p1 petsc/3.9.3
gcc/4.9.4         intel/2018          mpich-x86_6
4               python/3.6.3
gcc/5.3.0         intel/2018-mpi      octave/4.0.3
gcc/5.4.0         intel/intel_11     openmpi/2.1.2
[edison@bulb ~]$ module load gcc/5.3.0
[edison@bulb ~]$ gfortran -v
Using built-in specs.
COLLECT_GCC=gfortran
COLLECT_LTO_WRAPPER=/SYSTEM/gcc-5.3.0/build/bin/../libexec/gcc/x86_64-unknown-linux-gnu/5.3.0/lto-wrapper
Target: x86_64-unknown-linux-gnu
Configured with: ./configure --prefix=/SYSTEM_BULB/gcc-5.3.0/build
Thread model: posix
gfortran version 5.3.0 (GCC)
[edison@bulb ~]$
```

입력 파일이 1개인 경우

[링크] Github에서 보기

[링크] 소스코드 다운받기

예제코드 다운로드 및 실행

Github 사이트에 접속 해서 전체 소스가 압축된 파일을 다운 받거나, `git clone` 명령어를 이용해 소스 코드를 받을수 있습니다.

Bulb 서버에 접속 후 아래 명령어를 실행하면, 해당 소스코드를 다운 받을 수 있습니다.

Bulb 서버가 아닌 곳에서도 git이 설치되어 있다면, `git clone` 명령어를 통해 소스코드를 다운로드 받을 수 있습니다.

- [git 설치하기](#)

```
$ git clone https://github.com/sp-edison/fortran_example_input1.git
```

다운로드가 완료되면, `fortran_example_input1` 폴더가 생성됩니다. 폴더 구성은 다음과 같습니다.

```
fortran_example_input1
| README.md
└── src
    | Makefile
    └── main.f
```

```
$ cd fortran_example_input1/src
$ make all
gfortran -c main.f -o main.o
Compiled main.f successfully!
gfortran -o ../bin>Hello.x main.o
Linking complete!
```

컴파일이 완료되면 `fortran_example_input1` 폴더 안에 `bin` 폴더가 생성되며, `src/Mamkefile`에서 지정한 `TARGET` 명으로 실행 파일이 생성됩니다.

예제의 경우 `TARGET` 이 `Hello.x`로 설정되어 있습니다.

생성된 `bin` 폴더로 이동하여, 임의의 파일을 입력 파일로 넣고 실행시 에러 없이 실행 종료됨을 확인할 수 있습니다.

```
$ cd ../bin  
$ ./Hello.x -i /home/ino/test.input
```

이 예제에는 옵션 값이 맞는지 체크하고 옵션 뒤에 붙은 추가 파라미터 값이 파일 경로인지 확인하고 종료하는 프로그램입니다. 옵션 명이 잘못되었거나 입력 파일 경로가 잘못된 경우에는 에러가 발생합니다.

컴파일 된 파일을 삭제하고 싶은 경우에는 다시 `src` 폴더로 이동하여 `make clean` 명령어를 사용하면, 오브젝트 파일과 실행 파일을 삭제하게 됩니다.

```
$ cd ../src  
$ make clean  
rm -rf main.o ../bin/Hello.x
```

Source Code

main.f

```

program sample1

CHARACTER(len=16) :: cmd_option_name , value_name , temp
CHARACTER(len=512) :: inputdeck
INTEGER :: num_of_args, i, io_status
LOGICAL :: args_error_flag = .false.

num_of_args = iargc()

do i=1, num_of_args, 2
    call getarg(i,cmd_option_name)

    if( cmd_option_name .eq. "-i") then
        call getarg(i+1,inputdeck)
        write (*,*) inputdeck
    else
        args_error_flag = .true.
        write (*,*) "ERROR: INVALID COMAND OPTION: "
    ,
        +           cmd_option_name
    endif
enddo

if ( args_error_flag .eqv. .true. ) then
    write (*,*) "CHECK YOUR COMAND OPTION"
    stop
endif

write (*,*) "Input file path : ", inputdeck

end program

```

주요 변수 설명

- cmd_option_name : 커맨드 옵션(포트 명)을 저장하는 크기가 16인 character형

배열로 선언

- `inputdeck` : 입력 파일의 path를 저장하는 배열로 크기는 512인 character형 배열을 선언
- `num_of_args` : 실행 시 같이 입력된 argument의 개수를 저장하는 변수
- `args_error_flag = .false.` : 커맨드 옵션(포트 명)이 잘못 입력된 경우, 이 변수의 값을 `.true.` 로 변경

주요 코드 설명

```
...
[1] num_of_args = iargc()
...
```

[1] `iargc()` 함수를 이용하여 입력된 argument의 개수를 `num_of_args` 변수에 저장

```
...
[2] do i=1, num_of_args, 2
[3]   call getarg(i,cmd_option_name)

[4]   if( cmd_option_name .eq. "-i") then
[5]     call getarg(i+1,inputdeck)
      write (*,*) inputdeck
      else
        args_error_flag = .true.
        write (*,*) "ERROR: INVALID COMMAND OPTION: "
      ,
        + cmd_option_name
      endif
    enddo
...
```

[2] `do loop` 를 이용해 `num_of_args` 개수 까지 `i` 값을 2씩 증가하면서 `loop` 문 수행.
예제에서 `iargc()` 함수로 받은 값이 2 이므로 `loop` 문이 한번만 수행 됨 [3] `getarg()` 함수를 이용해 `i` 번째 arument 값을 `cmd_option_name` 변수에 저장

[4] 저장한 `cmd_option_name` 값을 `-i` 와 같은지 확인하여 같으면, `i+1`번째 arument 값을 읽어서 `inputdeck` 배열에 저장.

커맨드 옵션을 `-i` 가 아닌 다른 옵션 명으로 설정 하고 싶다면, 이 부분을 수정하면 된다.

[5] `cmd_option_name` 값이 `-i` 와 다르면, `args_error_flag` 를 `.false.` 로 변경하고 잘못 입력한 커맨드 옵션을 출력

```
...  
[6]  if ( args_error_flag .eqv. .true. ) then  
      write (*,*) "CHECK YOUR COMAND OPTION"  
      stop  
endif  
...
```

[6] 커맨드 옵션이 잘못 입력 되었는지 확인하고. 잘못 입력 되었을 경우 프로그램 종료

입력 파일이 여러개인 경우

입력 파일이 여러개인 경우는 1개인 경우 예제에서 `getopt()` 함수에 입력 받고자 하는 옵션을 추가하고, `switch` 문에서 추가한 옵션에 대한 코드를 추가하면 됩니다.

예를 들어 `-i, -m` 옵션 2개를 통해 입력 파일을 받기 원한다면, `"i:" -> "i:m:"` 으로 고치고, 아래 `switch` 문의 `m` case를 추가하면 됩니다.

```

#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>

int main (int argc, char *argv[]) {
    int opt;
    FILE *fp_input;
    FILE *fp_mesh;           //mesh 파일을 읽기 위한 파일 포인터
    추가 선언
    ...
    ...
    // Detect the end of the options.
    while((opt = getopt(argc, argv, "i:m:")) != -1) {
        switch(opt) {
            case 'i':
                if((fp_input = fopen(optarg
g,"r")) == NULL ) {
                    fprintf(stderr, "Error
opening inputfile Path: %s\n", optarg);
                    return -1;
                } else {
                    printf("Succeed to ope
n inputfile Path: %s\n", optarg);
                }
                break;
            case 'm':
                if((fp_mesh = fopen(optarg
g,"r")) == NULL ) {
                    fprintf(stderr, "Er
ror opening meshfile Path: %s\n", optarg);
                    return -1;
                } else {
                    printf("Succeed to
open meshfile Path: %s\n", optarg);
                }
                break;
            default:
                printf("Usage: %s -i [inputfil
e] -m [meshfile]\n", argv[0]);
                return -1;
        }
    ...
}

```

```
...
fclose(fp_input);
fclose(fp_mesh);
...
...
}
```

SDE case study

[링크] Github에서 보기

[링크] 소스코드 다운받기

관련 문서 링크

- SDE 프로그래밍 방법 개요 (page 25)
- SDE 데이터 타입 생성하기 (page 151)

다음과 같이 숫자형 변수 2개(정수형 변수 1개, 실수형 변수 1개), 리스트형 변수 1개, 3차원 벡터 1개를 받는 SDE를 생성했습니다.

▣ INPUTDECK 정의

value delimiter	SPACE ↴	Line delimiter	NULL ↴	Comment Char	<input type="text"/>	미리보기	KEY VALUE															
Vector bracket	SQUARE_SPACE ↴	Vector Delimiter	SPACE ↴	미리보기	[A B C]																	
<table border="1"> <thead> <tr> <th>Variable Name</th> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>INT1</td> <td>42</td> <td><input type="button" value="?"/></td> </tr> <tr> <td>REAL1</td> <td>42.112</td> <td><input type="button" value="?"/></td> </tr> <tr> <td>LIST1</td> <td>a ↴</td> <td><input type="button" value="?"/></td> </tr> <tr> <td>VECTOR1</td> <td>1 1 1</td> <td><input type="button" value="?"/></td> </tr> </tbody> </table>				Variable Name	Value	Description	INT1	42	<input type="button" value="?"/>	REAL1	42.112	<input type="button" value="?"/>	LIST1	a ↴	<input type="button" value="?"/>	VECTOR1	1 1 1	<input type="button" value="?"/>	타입 *	선택 ↴	※ 타입 선택 후 변수 추가	
Variable Name	Value	Description																				
INT1	42	<input type="button" value="?"/>																				
REAL1	42.112	<input type="button" value="?"/>																				
LIST1	a ↴	<input type="button" value="?"/>																				
VECTOR1	1 1 1	<input type="button" value="?"/>																				
				변수명 *	<input type="text"/>																	

Case1

데이터 생성 방식은 다음과 같이 설정했습니다.

KEY	VALUE	KEY	VALUE
value delimiter	SPACE	Vector vracket	SQUARE_SPACE
line delimiter	NULL	Vector delimiter	SPACE

이렇게 설정되어 생성된 입력 파일은 다음과 같습니다.

```
INT1 42
REAL1 42.112
LIST1 a
VECTOR1 [ 1 0 0 ]
```

Example

명령행 인자(Command Line Argument) 방식으로 생성된 입력 파일을 읽고 입력된 변수 값을 출력하는 예제 코드입니다.

```
program sample1

CHARACTER(len=16) :: cmd_option_name , value_name , temp
CHARACTER(len=512) :: sde
INTEGER :: num_of_args, i, io_status
LOGICAL :: args_error_flag = .false.

INTEGER INT1
DOUBLE PRECISION REAL1
CHARACTER LIST1, tempchar
INTEGER :: VECT0R1(3)

num_of_args = iargc()

do i=1, num_of_args, 2
    call getarg(i,cmd_option_name)

    if( cmd_option_name .eq. "-i") then
        call getarg(i+1,sde)
        write (*,*)  sde
    else
        args_error_flag = .true.
        write (*,*) "ERROR: INVALID COMAND OPTION: " ,
+
        cmd_option_name
    endif
enddo

if ( args_error_flag .eqv. .true. ) then
    write (*,*) "CHECK YOUR COMAND OPTION"
    stop
endif

write (*,*) "Input file path : ", sde

open(1,file=trim(sde),iostat=io_status, status='old')
if (io_status /= 0) then
    write(*,*) 'File open error'
    stop
end if

do
    READ(1,* , IOSTAT=io) value_name
    if ( io < 0) then
        WRITE(*,*) "SDE file read end"
```

```

        EXIT
    end if

    BACKSPACE (1)

    if ( value_name .eq. "INT1" ) then
        READ(1,*) value_name, INT1
        WRITE(*,*) "INT1 = ", INT1
    else if ( value_name .eq. "REAL1" ) then
        READ(1,*) value_name, REAL1
        WRITE(*,*) "REAL1 = ", REAL1
    else if ( value_name .eq. "LIST1" ) then
        read(1,*) value_name, LIST1
        write(*,*) "list = ", LIST1
    else if ( value_name .eq. "VECTOR1" ) then
        read(1,*) value_name, tempchar, VECTOR1(1),
+ VECTOR1(2), VECTOR1(3)
        write(*,*) "Vector = ", VECTOR1(1), VECTOR1(2),
+ VECTOR1(3)
    else
        WRITE(*,*) "SDE value read error"
        stop
    endif
end do

CLOSE(1)

end program

```

위 예제코드는 입력 파일이 1개인 경우 예제 (page 71)의 코드에서 SDE 파일을 읽는 부분을 추가하였습니다.

주요 코드 설명

주요 변수 설명

- INT1, REAL1, LIST1, VECTOR1(3) : Inputdeck 파일에서 각각의 변수 값을

저장하는 변수

- `io_status` : 입력 파일 오픈 시 에러 발생 여부를 저장하는 변수

주요 코드 설명

```
...
[1] open(1,file=trim(inputdeck),iostat=io_status, status='old')
[2] if (io_status /= 0) then
    write(*,*) 'File open error'
    stop
end if
...
```

1. `open()` 함수를 이용해 장치번호(UNIT)을 1로 설정하여 `inputdeck` 배열에 저장된 PATH의 인풋 파일 읽는다. 이때 `trim()` 함수를 이용해 앞에서 받은 inputdeck 경로 뒤에 붙은 공백을 제거한다.
 - `iostat=io_status` 은 파일 오픈 시 에러 발생 여부를 확인하는 옵션으로, 정상적으로 파일 오픈시 0 값을 저장
 - `status='old'` 는 기존의 있는 파일을 오픈하는 경우 `old` 값을 입력한다.
 - 예를들어 `trim("Hello ")` 실행하면, “Hello”를 리턴하게 된다.
2. 입력된 path에 파일이 없거나 정상적으로 파일이 오픈이 안되는 경우 `io_status` 값이 0이 아닌 값을 리턴한다. 에러 메시지를 표시하고 프로그램을 종료한다.

```

...
    do
[1]      READ(1,*, IOSTAT=io) value_name
        if ( io < 0) then
            WRITE(*,*) "SDE file read end"
            EXIT
        end if

[2]      BACKSPACE (1)

[3]      if ( value_name .eq. "INT1") then
            READ(1,*) value_name, INT1
            WRITE(*,*) "INT1 = ", INT1
        else if ( value_name .eq. "REAL1") then
            READ(1,*) value_name, REAL1
            WRITE(*,*) "REAL1 = ", REAL1
        else if ( value_name .eq. "LIST1") then
            read(1,*) value_name, LIST1
            write(*,*) "list = ", LIST1
        else if ( value_name .eq. "VECTOR1") then
            read(1,*) value_name, tempchar, VECTOR1(1),
+ VECTOR1(2), VECTOR1(3)
            write(*,*) "Vector = ", VECTOR1(1), VECTOR
1(2),
+ VECTOR1(3)
[4]      else
            WRITE(*,*) "Inputdeck value read error"
            stop
        endif
    end do

CLOSE(1)
...

```

1. 장치 번호 1번을 사용해 앞서 open한 입력 파일의 한 줄을 `read()`를 사용해 읽는다. 이때 입력 파일의 첫 번째 문자열인 변수 이름을 `value_name`에 저장한다.
 - 여기서 파일을 끝까지 다 읽은 경우 파일 `read`를 하게 되면 `io` 값이 0보다 작게 된다. 이를 통해 파일을 끝까지 다 읽었는지 여부를 판단하고 파일을 끝까지 다 읽은 경우 loop문을 빠져나온다.
2. `read()` 함수를 이용해 이미 앞에서 읽었던 파일 라인의 변수의 값을 다시 읽기 위해서, `backspace()`를 이용 방금 읽었던 파일 라인을 다시 읽을 수 있도록 파일 포인터를 이동한다.

3. 입력 파일에서 읽은 변수 이름을 확인하여 저장 함. 벡터 변수의 경우 변수 이름과 값 사이에 있는 [문자를 **tempchar** 변수에 저장하고, **VECTOR1(1~3)** 에 각각의 벡터 원소들을 저장한다.
4. 원하지 않은 변수 값이 입력되는 경우 이에 대한 에러 메시지를 표시하고 프로그램을 종료

SDE case study 2

SDE 생성시 데이터 생성 방식을 아래와 같이 설정한다면, 생성되는 입력 파일의 모양이 약간 달라질 것입니다.

KEY	VALUE	KEY	VALUE
value delimiter	EQUAL	Vector vracket	SQUARE_SPACE
line delimiter	SEMICOLON	Vector delimiter	SPACE

생성된 입력 파일

```
INT1 = 42 ;
REAL1 = 42.112 ;
LIST1 = a ;
VECTOR1 = [ 1 0 0 ] ;
```

추가된 value delimiter와 line delimiter를 고려해 코딩을 해야 합니다. 위의 예제 코드와 크게 다르지 않으며, **tempchar** 변수를 이용해 변수 이름과 변수 값 사이에 있는 **=** 을 처리하는 부분을 추가하였다.

```
fscanf(fp_input, "%d", &input.int1); -> fscanf(fp_input, "%*s %d
%*s", &input.int1);
```

변경전 코드

```
...
do
    READ(1,*, IOSTAT=io) value_name
    if ( io < 0) then
        WRITE(*,*) "SDE file read end"
        EXIT
    end if

    BACKSPACE (1)

    if ( value_name .eq. "INT1") then
        READ(1,*) value_name, INT1
        WRITE(*,*) "INT1 = ", INT1
    else if ( value_name .eq. "REAL1") then
        READ(1,*) value_name, REAL1
        WRITE(*,*) "REAL1 = ", REAL1
    else if ( value_name .eq. "LIST1") then
        read(1,*) value_name, LIST1
        write(*,*) "list = ", LIST1
    else if ( value_name .eq. "VECTOR1") then
        read(1,*) value_name, tempchar, VECTOR1(1),
+ VECTOR1(2), VECTOR1(3)
        write(*,*) "Vector = ", VECTOR1(1), VECTOR1(2),
+ VECTOR1(3)
    else
        WRITE(*,*) "SDE value read error"
        stop
    endif
end do
...

```

변경후 코드

```
...
do
    READ(1,*, IOSTAT=io) value_name
    if ( io < 0) then
        WRITE(*,*) "SDE file read end"
        EXIT
    end if

    BACKSPACE (1)

    if ( value_name .eq. "INT1") then
        READ(1,*) value_name, tempchar, INT1
        WRITE(*,*) "INT1 = ", INT1
    else if ( value_name .eq. "REAL1") then
        READ(1,*) value_name, tempchar, REAL1
        WRITE(*,*) "REAL1 = ", REAL1
    else if ( value_name .eq. "LIST1") then
        read(1,*) value_name, tempchar, LIST1
        write(*,*) "list = ", LIST1
    else if ( value_name .eq. "VECTOR1") then
        read(1,*) value_name, tempchar, tempchar,
+ VECTOR1(1), VECTOR1(2), VECTOR1(3)
        write(*,*) "Vector = ", VECTOR1(1), VECTOR1(2),
+ VECTOR1(3)
    else
        WRITE(*,*) "SDE value read error"
        stop
    endif
end do
...

```

Output programing

result 폴더 생성

FORTRAN의 경우 `CALL SYSTEM(COMMAND [, STATUS])` 를 이용하면 되며, `result` 폴더를 생성하는 예제는 아래와 같습니다.

```
program sample
  ...
  ...
  !
  ! 결과 파일이 저장될 result 폴더 생성
  CALL SYSTEM("rm -rf result")
  CALL SYSTEM("mkdir result")
  ...
```

출력 파일 쓰기

```
...
  !
  ! result/result.txt을 오픈
  open(2,file="result/result.txt")
  ...

  ...
  !
  ! 파일안에 데이터를 씁
  write(2,*)"Hello EDISON."
  ...

  ...
  !
  ! 파일을 닫음
  CLOSE(2)
```

`open()` 함수를 이용해 장치번호(UNIT)을 2로 설정하여 result 폴더 안 result.txt 파일을 오픈한다. 여러개의 파일을 동시에 쓰거나 읽는 경우 장치번호가 겹치지 않게 조심해야합니다.

장치번호는 1부터 99 사이의 정수로 설정할 수 있으며, 5는 표준 입력 standard input, 6은 표준 출력 standard output으로 이미 설정되어 있습니다.

장치 번호와 `write()` 함수를 통해 파일에 데이터를 쓸 수 있으며, 파일쓰기가 완료된 이후 `close()` 함수를 이용해 파일을 닫습니다.

oneD

sin() 함수 결과를 oned로 출력하기

[링크] [Github에서 보기](#)

[링크] [소스코드 다운받기](#)

예제코드 다운로드 및 실행

1개의 입력 파일 읽어, sin 그래프를 그리는 C언어 예제 파일입니다.

다음 수식의 변수들을 입력으로 받으며, 입력 변수는 a,b,c,d 총 4개입니다.

입력 파일의 경우에는 Value delimiter를 ‘ ’을 사용하였으며, Line delimiter를 구분하기 위해 ‘\n’을 사용하였습니다. 파일로 입력을 받으며, 샘플 입력 파일은 아래와 같으며, **inp** 폴더에 **input.dat**로 저장되어 있습니다.

```
a 1  
b 0.4  
c -0.5  
d 0.3
```

본 예제는 ./[실행파일명] -[옵션] [입력 파일 경로]로 실행시 옵션 뒤에 입력된 경로의 파일을 열고 닫는 예제로 Makefile과 소스코드는 **src** 폴더에 저장되어 있으며, 컴파일이 완료되면 바이너리 파일은 **bin** 폴더에 저장됩니다.

설치하기

Github 사이트에 접속 해서 전체 소스가 압축된 파일을 다운 받거나, **git clone** 명령어를 이용해 소스 코드를 받을수 있습니다.

Bulb 서버에 접속 후 아래 명령어를 실행하면, 해당 소스코드를 다운 받을 수 있습니다.

Bulb 서버가 아닌 곳에서도 git이 설치되어 있다면, **git clone** 명령어를 통해 소스코드를 다운로드 받을 수 있습니다.

- [git 설치하기](#)

```
$ git clone https://github.com/sp-edison/fortran_example_oned.git
```

```
fortran_example_oned
| README.md
|---src
|   | Makefile
|   | main.f
|---inp
|   | input.dat
```

`src` 폴더로 이동하여 `make all` 명령어를 사용하면, 컴파일이 완료됩니다.

```
$ cd fortran_example_oned/src
$ make all
gfortran -c main.f -o main.o
Compiled main.f successfully!
fortran -o ../bin/Hollo.x main.o
Linking complete!
```

컴파일이 완료되면 `fortran_example_oned` 폴더 안에 `bin` 폴더가 생성되며, `src/Makefile`에서 지정한 `TARGET` 명으로 실행 파일이 생성됩니다.

예제의 경우 `TARGET`이 `Sin.x`로 설정되어 있습니다.

생성된 `bin` 폴더로 이동하여, `inp` 폴더에 있는 `input.dat`을 입력 파일로 넣고 실행시 여러 없이 실행 종료됨을 확인할 수 있습니다.

```
$ cd ../bin
$ ./Sin.x -i ../inp/input.dat
User input is :
A = 1.000000
B = 0.400000
C = -0.500000
D = 0.300000
```

해당 예제를 이용하여 EDISON 웹포털에서 자동으로 소스 컴파일을 하실수 있습니다. 자동으로 소스 컴파일을 하기위해 선행되어야하는 조건은 아래와 같습니다.

- 본 예제와 같이 Makefile과 소스코드는 **src** 폴더에 저장되어 있어야 합니다.
- **src** 폴더 안에서 `make all` 커맨드 입력시, 컴파일이 정상적으로 완료된 경우 바이너리 파일이 **bin** 폴더에 저장이 되면 됩니다.

Source code

main.f

```

program sample1

CHARACTER(len=16) :: cmd_option_name , value_name
CHARACTER(len=512) :: sde
INTEGER :: num_of_args, i, io_status, t
LOGICAL :: args_error_flag = .false.

DOUBLE PRECISION :: a, b, c, d

num_of_args = iargc()

do i=1, num_of_args, 2
    call getarg(i,cmd_option_name)

        if( cmd_option_name .eq. "-i") then
            call getarg(i+1,sde)
            write (*,*) sde
        else
            args_error_flag = .true.
            write (*,*) "ERROR: INVALID COMAND OPTION: " ,
+
            cmd_option_name
        endif
enddo

if ( args_error_flag .eqv. .true. ) then
    write (*,*) "CHECK YOUR COMAND OPTION"
    stop
endif

write (*,*) "Input file path : ", sde

open(1,file=trim(sde),iostat=io_status, status='old')
if (io_status /= 0) then
    write(*,") 'File open error'
    stop
end if

do
    READ(1,*, IOSTAT=io) value_name
    if ( io < 0 ) then
        WRITE(*,") "SDE file read end"
        EXIT
    end if

```

BACKSPACE (1)

```

if ( value_name .eq. "a" ) then
    READ(1,*) value_name, a
    WRITE(*,*) "a = ", a
else if ( value_name .eq. "b" ) then
    READ(1,*) value_name, b
    WRITE(*,*) "b = ", b
else if ( value_name .eq. "c" ) then
    read(1,*) value_name, c
    write(*,*) "c = ", c
else if ( value_name .eq. "d" ) then
    read(1,*) value_name, d
    write(*,*) "d = ", d
else
    WRITE(*,*) "SDE value read error"
    stop
endif
end do
CLOSE(1)

CALL SYSTEM("rm -rf result")
CALL SYSTEM("mkdir result")

open(2,file="result/result.oneD")
write(2,*)"#NumField: 1"
write(2,*)"#LabelX: time, LabelY: a*sine(bx-c)+d"
write(2,*)"#Field1: sine() , NumPoint:128"

PI = 3.140592

do t = 1, 128, 1
    x = (4*PI * t )/128 -2*PI
    y = a*sin(b*x - c) +d
    write(2,'(F10.3, F10.3)') x, y
enddo

CLOSE(2)
end program

```

####주요 코드 설명

입력 파일을 읽고 이를 각각의 변수로 저장하는 부분은 [SDE 프로그래밍 \(page 79\)](#) 예제를 참고하였습니다.

코드 앞부분에서 oneD 데이터 생성에 필요한 변수를 `INTEGER :: t DOUBLE PRECISION x, y` 를 선언 하였습니다.

```
CALL SYSTEM("rm -rf result")
CALL SYSTEM("mkdir result")

open(2,file="result/result.oneD")
```

- 결과 데이터를 저장할 `result` 폴더를 생성하는 부분이며, `result` 폴더안에 `result.oneD` 파일을 생성하고 쓰기모드로 오픈하였습니다.

```
write(2,*)"#NumField: 1"
write(2,*)"#LabelX: time, LabelY: a*sine(bx-c)+d"
write(2,*)"#Field1: sine(), NumPoint:128"
```

- oneD 파일의 헤더 부분의 내용을 생성합니다. [oneD 데이터 구조 \(page 31\)](#)에 대한 자세한 설명은 해당 페이지를 참조하시기 바랍니다.
- 필드가 1개이며, x라벨이 time y라벨이 $a * \sin(bx+c)+d$ 이고 필드의 이름은 사용자가 입력한 a,b,c,d 값을 나타내고 있습니다.

```
PI = 3.140592
```

```
do t = 1, 128, 1
    x = (4*PI * t )/128 -2*PI
    y = a*sin(b*x - c) +d
    write(2,'(F10.3, F10.3)') x, y
enddo
```

- PI값을 정의하고, $t=1$ 부터 128 까지 for문을 수행한다. 대체에서 까지 128등분으로 일정하게 나눈 값을 x에 저장하고, 이 x값에 대한 결과 값을 y에 저장한 뒤 각각 파일에 써주게 됩니다.
- 이때 저장하는 데이터의 규칙은 10진수 형태로 총 10자리를 가지고, 소수점 아래로는 3자리를 가지도록 저장합니다.

Python 언어

Python은 인터프리터 방식의 프로그래밍 언어. 별도의 빌드 과정이 없이 실행 할 수 있으며, 코드의 가독성이 좋은 장점을 가지고 있습니다. 또한 PyPI라는 패키지 저장소가 있어 원하는 패키지를 `pip` 명령어를 통해 쉽게 내려받을 수 있습니다. 이러한 장점으로 계산과학분야에서도 Python언어는 많이 활용되고 있습니다.

파이썬 스크립트를 리눅스 Command Line에서 실행하는 방법은 2가지 입니다.

- 실행 파일 앞에 `python`을 명시하고 실행하는 방법입니다. 예를들어 `main.py` 파일을 실행하기 위해서는 `python main.py` 명령어로 실행하면 됩니다.
- 파이썬 스크립트 파일 첫줄에 `#!` (Shebang)을 통해 스크립트를 실행시켜줄 프로그램의 경로 지정해주면 됩니다.
 - 일반적으로 `#!/usr/bin/env python` 로 선언하면 됩니다. 여기서 `env` 명령어가 환경 변수에서 지정한 언어의 위치를 찾아서 실행하게 됩니다.
 - `env` 명령어를 쓰지 않고 직접 `python` 경로를 지정해 주어도 됩니다.
 - ex) `#!/SYSTEM/python/2.7.11/bin/python`
 - 이 경우 `chmod +x filename.py` 명령어를 통해 해당 파일에 실행 권한을 줘야 합니다.

local path에 패키지 설치하기

Python으로 코드 개발시 추가 패키지 설치가 필요한 경우가 있습니다. Bulb에서 `pip`로 파이션 패키지 설치시 Permission denied에러가 발생합니다.

```
Could not install packages due to an EnvironmentError: [Errno 13] Permission denied: '/usr/local/lib/python2.7/site-packages/python_genutils-0.2.0.dist-info'
Consider using the `--user` option or check the permissions.
```

`pip` 명령어의 `-t` 옵션을 이용하여 `python`이 설치된 경로가 아닌 local 경로에 설치하면 문제를 해결할 수 있습니다.

```
pip install -t <설치할 경로> <설치할 패키지>
```

numpy 패키지를 lib 경로에 설치한다면, 아래 명령어를 사용하면 됩니다.

```
pip install -t ./lib numpy
```

파이썬에서 lib 경로에 설치한 패키지를 읽어올수 있도록 환경변수를 추가해줍니다.

```
export PYTHONPATH=lib
```

local path의 패키지를 사용하는 파이썬 코드를 사이언스 앱으로 등록하기

사이언스 앱으로 등록하고자 하는 파이썬 코드가 local path의 패키지를 사용하는 경우, 실행 파일에 있는 위치에 lib 폴더에 패키지를 설치하고, simrc 파일에 export PYTHONPATH=lib 명령어를 저장합니다. 이후 앱 등록시 lib 폴더와 simrc 파일을 실행 스크립트와 같이 압축하여 업로드 해주시면 됩니다.

python 3 버전 사용하기

EDISON 환경에서 python3가 설치되어 있는 경로는 다음과 같습니다.

```
/SYSTEM/Python/3.6.3
```

환경변수에 파이썬 3이 설치경로를 추가해주면 python3을 사용할 수 있습니다.

```
export PATH=/SYSTEM/Python/3.6.3/bin:$PATH
```

다른 방법으로는 파이썬3 스크립트 파일 첫줄에 #!을 통해 python3의 경로를 지정해주면 됩니다.

- \$PATH 환경변수에 python3 경로가 추가된 경우라면, #!/usr/bin/env python3
- 아니라면, #!/SYSTEM/Python/3.6.3/bin/python 을 스크립트 파일 첫줄에 적어주어야 합니다.

입력 파일이 1개인 경우

[링크] Github에서 보기

[링크] 소스코드 다운받기

예제코드 다운로드 및 실행

Github 사이트에 접속 해서 전체 소스가 압축된 파일을 다운 받거나, `git clone` 명령어를 이용해 소스 코드를 받을수 있습니다.

Bulb 서버에 접속 후 아래 명령어를 실행하면, 해당 소스코드를 다운 받을 수 있습니다.

Bulb 서버가 아닌 곳에서도 git이 설치되어 있다면, `git clone` 명령어를 통해 소스코드를 다운로드 받을 수 있습니다.

- [git 설치하기](#)

```
$ git clone https://github.com/sp-edison/python_example_input1.git
```

다운로드가 완료되면, `python_example_input1` 폴더가 생성됩니다. 폴더 구성은 다음과 같습니다.

```
python_example_input1
| README.md
└── main.py
```

Python의 경우 Script를 실행하는 형태로 별다른 컴파일 과정이 필요 없다. 하지만 리눅스 명령어 chmod 명령어를 이용해 그룹과 일반 사용자에게 읽기와 실행 권한을 주어야 합니다.

`main.py` 가 실행 파일인 경우 `chmod u+x main.py` 를 리눅스 상에서 실행하면 됩니다.

```
$ chmod u+x main.py
$ ./main.py -i /home/ino/test.input
input file = /home/ino/test.input
```

Source Code

main.py

```
#!/usr/local/bin/python

import sys
import getopt

try:
    opts, args = getopt.getopt(sys.argv[1:], "i:")
except getopt.GetoptError as err:
    print str(err)
    sys.exit(1)

for opt,arg in opts:
    if opt in ("-i"):
        f_inputdeck = open(arg, "r")

print "input file = " + f_inputdeck.name
inputdeck_lines = f_inputdeck.readlines()

f_inputdeck.close()
```

주요 코드 설명

```
#!/usr/local/bin/python

import sys
import getopt
```

`#!` 을 통해 이 스크립트를 실행시켜줄 파이썬 프로그램의 경로를 지정하여 필요한 모듈을 불러온다.

Shebang과 env에 대한 설명

- sys 모듈 : 파이썬 인터프리터와 관련된 정보와 기능을 제공하는 모듈이며, 해당 스크

립트로 넘어온 입력인자(argv)를 확인하기 위해 사용한다.

- getopt 모듈 : 입력인자를 보다 편리하게 처리하게 해주는 모듈이다.

```
...
try:
    opts, args = getopt.getopt(sys.argv[1:],"i:")
except getopt.GetoptError as err:
    print str(err)
    sys.exit(1)
...
```

`getopt.getopt()` 함수를 이용해 입력 인자를 옵션과 옵션에 대한 추가 값으로 분류합니다. 추가 옵션 값을 가지는 “`-i`” 옵션을 받을 수 있으며, 이때 입력된 옵션 값은 `opts`에 저장되고, 추가 옵션 값은 `args`에 저장합니다.

- `getopt.getopt(sys.argv[1:],"i:")`에서 `getopt.getopt()` 함수의 첫 번째 인자는 프로그램 실행시 입력되는 파라미터 문자열 입력 받습니다. 기본적으로 실행 파일 이름을 제외한 문자열인 `sys.argv[1:]`을 입력 받으며, 두 번째 인자는 입력 받을 단일 문자 옵션, 마지막 인자는 입력 받을 긴 문자 옵션이 입력됩니다.
- 옵션 뒤에 추가 옵션 값이 필요한 경우 단일 문자 옵션인 경우 옵션 명 뒤에 `:`를 긴 문자 옵션인 경우 `=`를 붙여 줍니다.
- `./main.py -i /home/ino/test.input`로 실행 시 `opts`에 옵션 명과 옵션 값이 저장된 튜플 형태의 배열인 `[('i', 'input.dat')]`가 저장됩니다.

입력된 옵션 값 이외에 다른 옵션 값이 입력되는 경우 `getopt.GetoptError` 에러가 발생하며, 이에 대한 에러 메시지는 `err`에 저장됩니다.

`try ... except` 구문을 이용해 잘못된 옵션 값이 입력된 경우 이에 대한 에러 값을 출력하고 프로그램을 종료합니다.

영어자료, getopt 함수 사용하기 한글자료, getopt 함수 사용하기

```
...
for opt,arg in opts:
    if opt in ("-i", "--inp"):
        f_inputdeck = open(arg, "r")
...
...
```

`opts`에 저장된 옵션 명을 `opt`에 옵션 값을 `arg`에 저장하고 이를 체크하는 부분입니다.

입력된 옵션 명이 `-i` 이면 이때의 옵션 값을 입력 파일의 경로라 판단하여 `open()` 함수를 통해 파일을 열고 이에 대한 정보를 `f_inputdeck` 이라는 파일 객체에 저장합니다.

```
...
print "input file = " + f_inputdeck.name
inputdeck_lines = f_inputdeck.readlines()

f_inputdeck.close()
```

입력 파일이 여러개인 경우

입력 파일이 여러개인 경우는 1개인 경우 예제 (page 99)에서 `getopt()` 함수에 입력 받고자 하는 옵션을 추가하고, `for opt,arg in opts:` 문에서 추가한 옵션에 대한 `if` 문을 추가하면 됩니다.

예를들어 `-i, -m` 옵션 2개를 통해 입력 파일을 받기 원한다면, `"i:" -> "i:m:"` 으로 고치고, 아래 `for opt,arg in opts:` 문의 `-m` 에 대한 `elif` 문을 추가하면 됩니다.

```
#!/usr/local/bin/python

import sys
import getopt

try:
    opts, args = getopt.getopt(sys.argv[1:],"i:m:")
except getopt.GetoptError as err:
    print str(err)
    sys.exit(1)

for opt,arg in opts:
    if opt in ("-i"):
        f_input = open(arg, "r")
    elif opt in ("-m"):
        f_mesh = open(arg, "r")

print "input file = " + f_input.name
f_input.close()

print "mesh file = " + f_mesh.name
f_mesh.close()
```

SDE case study

[링크] Github에서 보기

[링크] 소스코드 다운받기

관련 문서 링크

- SDE 프로그래밍 방법 개요 (page 25)
- SDE 데이터 타입 생성하기 (page 151)

다음과 같이 숫자형 변수 2개(정수형 변수 1개, 실수형 변수 1개), 리스트형 변수 1개, 3차원 벡터 1개를 받는 SDE를 생성했습니다.

INPUTDECK 정의

value delimiter	SPACE ↴	Line delimiter	NULL ↴	Comment Char	<input type="text"/>	미리보기	KEY VALUE
Vector bracket	SQUARE_SPACE ↴	Vector Delimiter	SPACE ↴	미리보기	[A B C]		
Variable Name	Value	Description	타입 * <input type="button" value="선택"/> ※ 타입 선택 후 변수 추가 변수명 * <input type="text"/>				
INT1	<input type="text" value="42"/>	<input type="button" value="?"/>					
REAL1	<input type="text" value="42.112"/>	<input type="button" value="?"/>					
LIST1	a ↴	<input type="button" value="?"/>					
VECTOR1	<input type="text" value="1"/> <input type="text" value="1"/> <input type="text" value="1"/>	<input type="button" value="?"/>					

Case1

데이터 생성 방식은 다음과 같이 설정했습니다.

KEY	VALUE	KEY	VALUE
value delimiter	SPACE	Vector vracket	SQUARE_SPACE
line delimiter	NULL	Vector delimiter	SPACE

이렇게 설정되어 생성된 입력 파일은 다음과 같습니다.

```
INT1 42
REAL1 42.112
LIST1 a
VECTOR1 [ 1 0 0 ]
```

Example

명령행 인자(Command Line Argument) 방식으로 생성된 입력 파일을 읽고 입력된 변수 값을 출력하는 예제 코드입니다.

```
#!/usr/local/bin/python
""" EDISON python sample code"""

import sys
import os
import getopt


try:
    otps, args = getopt.getopt(sys.argv[1:], "i:")
except getopt.GetoptError as err:
    print(str(err))
    sys.exit(1)

for opt, arg in otps:
    if opt in "-i":
        f_sde = open(arg, "r")

print("input file = " + f_sde.name)
sde_lines = f_sde.readlines()

for line in sde_lines:
    opt = line.split()[0]
    if opt in "INT1":
        int1 = int(line.split()[1])
        print "init1 :" + str(int1)
    elif opt in "REAL1":
        real1 = float(line.split()[1])
        print "real1 :" + str(real1)
    elif opt in "LIST1":
        list1 = line.split()[1]
        print "list1 :" + list1
    elif opt in "VECTOR1":
        vector1 = map(int, line.split("[")[1].split("]")[0].split())
        print "vector1 :" + str(vector1)
    else:
        print "SDE value read error. your input key is"
        " + str(opt)
        sys.exit(1)

f_sde.close()
```

위 예제코드는 [입력 파일이 1개인 경우 예제 \(page 99\)](#)의 코드에서 SDE 파일을 읽는 부분을 추가하였습니다.

주요 코드 설명

#####주요 코드 설명

```

...
sde_lines = f_sde.readlines()

for line in sde_lines:
    opt = line.split()[0]
    if opt in "INT1":
        int1 = int(line.split()[1])
        print "init1 :" + str(int1)
    elif opt in "REAL1":
        real1 = float(line.split()[1])
        print "real1 :" + str(real1)
    elif opt in "LIST1":
        list1 = line.split()[1]
        print "list1 :" + list1
    elif opt in "VECTOR1":
        vector1 = map(int, line.split("[")[1].split("]")[0].split())
        print "vector1 :" + str(vector1)
    else:
        print "SDE value read error. your input key is "
        " + str(opt)
        sys.exit(1)
...

```

- `f_sde.readlines()` 함수를 이용해 입력 파일을 한줄씩 `sde_lines` 리스트에 저장합니다.
- `for ... in ... :` 문을 이용해 `sde_lines` 원소를 하나씩 `line`에 저장하고 `sde_lines`의 길이 만큼 `for` 문을 반복합니다.
- 문자열을 나누기 위해 `split()` 함수를 사용합니다. 예제에서 처럼 괄호 안에 아무런 값도 넣어 주지 않으면 공백을 기준으로 문자열을 나눕니다.
 - `line = "INT1 42"` 인 경우 이 문자열을 나눈 결과 `line.split()` 는 `['INT1', '42']` 가 되고, 각각의 요소들을 지정하기 위해 뒤에 [숫자]

자] 추가해 요소를 지정합니다. `line.split()[0] = 'INT1'` 이며,
`line.split()[1] = 42` 이 됩니다..

- 공백으로 나눈 값의 첫 번째 요소를 `opt`에 저장해 해석에 필요한 변수 이름과 비교에 각각 저장합니다.
- 입력 파일에 저장된 값들은 문자열 이므로 숫자를 저장해야 하는 경우 이에 맞게 형 변환을 해주어야 합니다. 정수 형으로 변환하는 경우 `int()`, 실수로 저장하는 경우 `float()`을 이용하면 됩니다.
- `vector1 = map(int,`
`line.split("[")[1].split(']')[0].split())` 을 정리하면
 - 초기 `line`은 `VECT0R1 [1 3 0]`이며, 이를 `[`로 나누면, 나눠서 저장된 배열의 2번째 값 `line.split("[")[1]`은 `1 3 0`이 됩니다.
 - 이를 다시 `]`나눈 값의 첫 번째 값 `.split("]") [0]`은 `1 3 0`이 됩니다.
 - 이를 다시 공백으로 나누어 배열에 저장하고 `map(int, [배열])` 함수를 통해 문자 값인 각각의 원소를 정수 형으로 변환 시킵니다.

SDE case study 2

SDE 생성시 데이터 생성 방식을 아래와 같이 설정한다면, 생성되는 입력 파일의 모양이 약간 달라질 것입니다.

KEY	VALUE	KEY	VALUE
value delimiter	EQUAL	Vector vracket	SQUARE_SPACE
line delimiter	SEMICOLON	Vector delimiter	SPACE

생성된 입력 파일

```
INT1 = 42 ;
REAL1 = 42.112 ;
LIST1 = a ;
VECT0R1 = [ 1 0 0 ] ;
```

추가된 value delimiter `=` 와 line delimiter `;`를 고려해 코딩을 해야 합니다.

변경전 코드

```
...
for line in sde_lines:
    opt = line.split()[0]
    if opt in "INT1":
        int1 = int(line.split()[1])
        print "init1 :" + str(int1)
    elif opt in "REAL1":
        real1 = float(line.split()[1])
        print "real1 :" + str(real1)
    elif opt in "LIST1":
        list1 = line.split()[1]
        print "list1 :" + list1
    elif opt in "VECTOR1":
        vector1 = map(int, line.split("[")[1].split("]")[0].split())
        print "vector1 :" + str(vector1)
    else:
        print "SDE value read error. your input key is"
        " + str(opt)
        sys.exit(1)
...
```

변경후 코드

```

...
for line in sde_lines:
    opt = line.split()[0]
    if opt in "INT1":
        int1 = int(line.split('=')[1].split(';')[0])
        print "init1 : " + str(int1)
    elif opt in "REAL1":
        real1 = float(line.split('=')[1].split(';')[0])
        print "real1 : " + str(real1)
    elif opt in "LIST1":
        list1 = line.split('=')[1].split(';')[0]
        print "list1 : " + list1
    elif opt in "VECTOR1":
        vector1 = map(int, line.split("[")[1].split("]")[0].split())
        print "vector1 :" + str(vector1)
    else:
        print "error"
        sys.exit(1)
...

```

주요코드설명

```
int1 = int(line.split('=')[1].split(';')[0])
```

- `line = 'INT1 = 42 ;'` 이며 이를 `=`로 나눈 `line.split('=')[1]` 값의 두 번째 값은 `'42 ;'` 가 됩니다. 이를 다시 `;`로 나눈 첫번째 값은 `'42'` 인 문자열이 되며, 이를 `int()` 를 통해 정수로 형 변환을 시킵니다.

Output programing

result 폴더 생성

Python의 경우 `import os` 를 선언하고 `os.system (const char * string);` 함수를 이용하면 되며, result 폴더를 생성하는 예제는 아래와 같습니다.

```
import os

...
os.remove("result");
os.mkdir("result");
...
```

결과 파일 생성

```
# open()함수를 통해 result/result.txt을 쓰기 모드로 오픈
f_out = open("result/result.txt","w")
...
...
# f_out.write() 함수를 이용해 파일에 내용을 씀
f_out.write("Hello EDISON.\n")

...
# f_out.close() 함수를 통해 파일을 닫음
f_out.close()
```

oneD

sin() 함수 결과를 oned로 출력하기

[링크] [Github에서 보기](#)

[링크] [소스코드 다운받기](#)

예제코드 다운로드 및 실행

1개의 입력 파일 읽어, sin 그래프를 그리는 Python언어 예제 파일입니다.

다음 수식의 변수들을 입력으로 받으며, 입력 변수는 a,b,c,d 총 4개 입니다.

입력 파일의 경우에는 Value delimiter를 ‘SPACE’을 사용하였으며, Line delimiter를 구분하기 위해 ‘NULL’를 사용하였습니다. 파일로 입력을 받으며, 샘플 입력 파일은 아래와 같으며, **inp** 폴더에 **input.dat** 로 저장되어 있습니다.

```
a 1  
b 0.4  
c -0.5  
d 0.3
```

본 예제는 ./[실행파일명] -[옵션] [입력 파일 경로]로 실행시 옵션 뒤에 입력된 경로의 파일을 열고 닫는 예제는 **bin** 폴더에 main.py 파일을 바로 실행합니다.

설치하기

Github 사이트에 접속 해서 전체 소스가 압축된 파일을 다운 받거나, **git clone** 명령어를 이용해 소스 코드를 받을수 있습니다.

Bulb 서버에 접속 후 아래 명령어를 실행하면, 해당 소스코드를 다운 받을 수 있습니다.

Bulb 서버가 아닌 곳에서도 git이 설치되어 있다면, **git clone** 명령어를 통해 소스코드를 다운로드 받을 수 있습니다.

- [git 설치하기](#)

```
$ git clone https://github.com/sp-edison/python_example_oned.git
```

```
python_example_oned
| README.md
|---bin
|   | main.py
|---inp
|   | input.dat
```

bin 폴더로 이동하여, **inp** 폴더에 있는 **input.dat**을 입력 파일로 넣고 실행시 에러 없이 실행 종료됨을 확인할 수 있습니다.

```
$ cd ../bin
$ ./main.py -i ../inp/input.dat
input file = ../inp/input.dat
a : 1.0
b : 2.0
c : 0.5
d : 2.3
```

Source code

[main.py](#)

```
#!/usr/local/bin/python
""" EDISON python sample code"""

import sys
import os
import getopt
import math
import numpy as np

try:
    otps, args = getopt.getopt(sys.argv[1:], "i:")
except getopt.GetoptError as err:
    print(str(err))
    sys.exit(1)

for opt, arg in otps:
    if opt in "-i":
        f_sde = open(arg, "r")

print("input file = " + f_sde.name)
sde_lines = f_sde.readlines()

for line in sde_lines:
    opt = line.split()[0]
    if opt in "a":
        a = float(line.split()[1])
        print("a : " + str(a))
    elif opt in "b":
        b = float(line.split()[1])
        print("b : " + str(b))
    elif opt in "c":
        c = float(line.split()[1])
        print("c : " + str(c))
    elif opt in "d":
        d = float(line.split()[1])
        print("d : " + str(d))
    else:
        print("SDE value read error. your input key is " + str(opt))
        sys.exit(1)

f_sde.close()

os.system("rm -rf result")
```

```

os.system("mkdir result")

f_out = open("result/result.oned", "w")

times = np.arange(-2.0*math.pi, 2.0*math.pi, 0.1)

f_out.write("#NumField: 1\n")
f_out.write("#LabelX: time, LabelY: a*sine(x+b) \n")
f_out.write("#Field1: a=%f b=%f c=%f d=%f, NumPoint:%i\n" % (a,
b, c, d, len(times)))

for time in times:
    y = a*math.sin(b*time-c)+d
    f_out.write("%10.3f      %10.3f\n" % (time, y))

f_out.close()

```

####주요 코드 설명

####주요 코드 설명

입력 파일을 읽고 이를 각각의 변수로 저장하는 부분은 [SDE 프로그래밍 \(page 104\)](#) 예제에서 가져왔습니다.

```

import sys, os
import getopt
import math
import numpy as np
...

```

- `os.system()`, `sin()`, `arange()`을 사용하기 위해 `math` 와 `numpy` 를 import 하였습니다.

```

...
os.system("rm -rf result")
os.system("mkdir result")
...

```

- 결과 데이터를 저장할 `result` 폴더를 생성합니다.

```
f_out = open("result/result.oneD", "w")
times = np.arange(-2.0*math.pi, 2.0*math.pi, 0.1)
```

- result.oneD 파일을 result 폴더에 쓰기 모드로 오픈하였으며, 이를 f_out에 저장하였습니다.
- numpy에서 제공하는 `arange()` 함수를 이용해 -2π 부터 2π 까지 0.1 간격의 배열인 `times`을 생성하였습니다.

```
f_out.write("#NumField: 1\n")
f_out.write("#LabelX: time, LabelY: a*sine(x+b) \n")
f_out.write("#Field1: a=%f b=%f c=%f d=%f, NumPoint:%i\n" % (a,
b, c, d, len(times)))
```

- oneD 파일의 헤더 부분의 내용을 생성합니다. [oneD 데이터 구조 \(page 31\)](#)에 대한 자세한 설명은 해당 페이지를 참조하시기 바랍니다.
- 필드가 1개이며, x라벨이 time y라벨이 $a * \sin(bx+c)+d$ 이고 필드의 이름은 사용자가 입력한 a,b,c,d 값을 나타내고 있습니다.

```
for time in times:
    y = a*math.sin(b*time-c)+d
    f_out.write("%10.3f      %10.3f\n" % (time, y))

f_out.close()
```

- for문을 이용해 앞서 생성한 `times`의 배열 값을 하나씩 읽고 이를 이용하여 예 대한 결과 y를 계산합니다. 이후 x에 저장할 값인 time과 y에 저장할 값인 y를 각각 파일에 쓰게됩니다.
- 이때 저장하는 데이터의 규칙은 10진수 형태로 총 10자리를 가지고, 소수점 아래로는 3자리를 가지도록 저장하였습니다.

Plotly

Python을 활용하여 ply 파일 생성하기

[링크] [Github에서 보기](#)

[링크] [소스코드 다운받기](#)

python을 활용하여 입력 데이터를 Plotly json 형태로 변환해주는 예제입니다. 폴더별로 ply로 변환해주는 python 코드와 octave 실행 코드 그리고 입력 파일이 있습니다.

각 예제별 설명

1_line_plot : dat 파일을 읽어 line plot을 출력

실행 명령어 : `python convert.py -i Freq_S11Abs.dat` 결과 result 폴더에 Freq_S11Abs.ply 생성

2_line_plot_holdon : dat 파일을 일거 2개의 line plot을 동시에 출력

실행 명령어 : `python convert.py -i Time_Voltage_Current.dat` 결과 result 폴더에 Time_Voltage_Current.ply 생성

3_image_2D : dat 파일을 읽어 2D contour plot을 출력

실행 명령어 : `python convert.py -i Hy5500.dat` 결과 result 폴더에 Hy5500.ply 생성

예제 이외에도 plotly 라이브러리에서 제공하는 다양한 plot을 가시화 할 수 있습니다.

`go.Figure()` 함수를 통해 plotly 데이터를 생성하고 `plotlyfig2json()` 함수를 통해 Plotly json 파일 형태로 저장해야합니다.

```
fig = go.Figure(data=data, layout=layout);
...
plotlyfig2json(fig, 'result/' + output_name + '.ply');
```

- [Plotly Python doc 문서보기](#)

QnA

matplotlib 사용시 아래와 같은 에러가 발생한다면

The main problem is that (on your system) matplotlib chooses a non-x-using backend by default. I just had the same problem on one of my servers. The solution for me was to add the following code in a place that gets read before any other pylab/matplotlib/ pyplot import:

matplotlib import 이후 `matplotlib.use('Agg')` 코드를 추가해 에러를 해결할 수 있습니다.

```
import numpy, scipy, math, matplotlib
matplotlib.use('Agg') <-- 코드 추가
import matplotlib.pyplot as plt
```

R 언어

R은 인터프리터 방식의 프로그래밍 언어입니다. 파이썬 언어와 마찬가지로 별도의 빌드 과정이 없이 실행 할 수 있으며, 코드의 가독성이 좋은 장점을 가지고 있습니다. R언어의 장점은 다음과 같습니다.

- 오픈소스, 무료 소프트웨어
- 포괄적인 통계플랫폼 : 다양한 라이브러리, 다양한 분석기법, 정형/비정형
- 시각화 기능으로 수학 기호를 포함할 수 있는 출판물 수준의 그래프를 제공
 - 출처: <http://sjh836.tistory.com/110> [빨간색코딩]

EDISON에서 R을 사용하기 위해서는 `module load R/<버전>` 명령어를 통해 사용할 수 있습니다.

파이썬 스크립트와 마찬가지로 R 스크립트를 리눅스 Command Line에서 실행하는 방법은 2 가지입니다.

- 실행 파일 앞에 Rscript을 명시하고 실행하는 방법입니다. 예를들어 main.py 파일을 실행하기 위해서는 `Rscript run.r` 명령어로 실행하면 됩니다.
- R 스크립트 파일 첫줄에 `#!` (Shebang)을 통해 스크립트를 실행시켜줄 프로그램의 경로 지정해주면 됩니다.
 - 일반적으로 `#!/usr/bin/env Rscript`로 선언하면 됩니다. 여기서 `env` 명령어가 환경 변수에서 지정한 언어의 위치를 찾아서 실행하게 됩니다.
 - 이 경우로 작성된 R 스크립트를 사이언스 앱으로 등록하는 경우, `simrc` 파일에 사용하는 R 경로 환경변수로 추가하거나, `module load` 명령어를 통해 R 모듈을 추가해주어야 합니다.
 - `env` 명령어를 쓰지 않고 직접 R 경로를 지정해 주어도 됩니다.
 - ex) `#!/SYSTEM/R/3.3.3/bin/Rscript`
 - 이 경우 `chmod +x run.r` 명령어를 통해 해당 파일에 실행 권한을 줘야 합니다.

local path에 패키지 설치하기

설치하고자 하는 위치에 폴더(ex: libs)를 생성합니다.

```
> mkdir libs
```

`module load R/<버전>` 통해 R 모듈을 불러옵니다. 이후 `R` 명령어로 R을 실행합니다.

```
$ R
```

```
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
```

```
Natural language support but running in an English locale
```

```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
```

```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

```
>
```

`install.packages("패키지이름", repos="저장소 위치", lib="설치위치")` 명령어를 통해 패키지를 설치합니다. `lib="libs"`로 지정해 `libs` 경로에 설치합니다.

`lattice` 패키지를 `libs` 폴더에 설치하는 경우

```
> install.packages("lattice", repos="http://cran.r-project.org",
  lib="libs")
```

R 스크립트에서 로컬 경로에 설치된 패키지 경로를 추가해야 합니다. 패키지 경로를 추가는 `.libPaths("<패키지 설치 경로>")` 명령어를 실행하면 됩니다.

`libs` 폴더의 설치된 패키지를 추가하는 경우

```
.libPaths("./libs")
```

local path의 패키지를 사용하는 R 스크립트를 사이언스 앱으로 등록하기

사이언스 앱으로 등록하고자 하는 R 스크립트가 local path의 패키지를 사용하는 경우, 실행 R 스크립트가 있는 위치에 `libs` 폴더에 패키지를 설치하고, 앱 등록시 `libs` 폴더와 `simrc` 파일을 실행 스크립트와 같이 압축하여 업로드 해주시면 됩니다.

입력 파일이 1개인 경우

[링크] Github에서 보기

[링크] 소스코드 다운받기

예제코드 다운로드 및 실행

Github 사이트에 접속 해서 전체 소스가 압축된 파일을 다운 받거나, `git clone` 명령어를 이용해 소스 코드를 받을수 있습니다.

Bulb 서버에 접속 후 아래 명령어를 실행하면, 해당 소스코드를 다운 받을 수 있습니다.

Bulb 서버가 아닌 곳에서도 git이 설치되어 있다면, `git clone` 명령어를 통해 소스코드를 다운로드 받을 수 있습니다.

- [git 설치하기](#)

```
$ git clone https://github.com/sp-edison/octave_example_input  
1.git
```

다운로드가 완료되면, `r_example_input1` 폴더가 생성됩니다. 폴더 구성은 다음과 같습니다.

```
r_example_input1  
| README.md  
| simrc  
└── run.r
```

실행하기

source 명령어를 통해 simrc에 있는 명령어를 실행합니다.

simrc에는 R 모듈을 추가하는 명령어가 있습니다.

```
$ source simrc
```

R의 경우 Script를 실행하는 형태로 별다른 컴파일 과정이 필요 없습니다. 하지만 리눅스 명령어 chmod 명령어를 이용해 실행 권한을 주어야 합니다. `run.r` 가 실행 파일인 경우 `chmod +x run.r` 를 리눅스 상에서 실행하면 됩니다.

```
$ chmod +x run.r  
$ ./run.r -i /home/ino/test.input  
input file = /home/ino/test.input
```

Source Code

`simrc`

```
module load R/3.3.2
```

`module load` 명령어를 통해 R 모듈을 추가합니다.

run.r

```
#!/usr/bin/env Rscript

library(optparse)

option_list <- list (
  make_option(c("-i","--inp"), type='character', help="Input
file path", default=NULL ,metavar="character"))
);

opt_parser <- OptionParser(option_list=option_list);
opt <- parse_args(opt_parser);

if (is.null(opt$inp)){
  print_help(opt_parser)
  stop("At least one argument must be supplied (input fil
e).n", call.=FALSE)
}

inputfile = opt$inp;

print(inputfile);
```

주요 코드 설명**입력 인자 읽기**

```
#!/usr/bin/env Rscript
```

`#!` 을 통해 이 스크립트를 실행하는 파일이 `Rscript` 임을 지정합니다.

EDISON 서버에서의 `R` 설치 위치는 `/SYSTEM/R`에 버전 별로 정리되어 있습니다. `module load` 명령어를 통해 원하는 버전의 R 모듈을 추가할 수 있습니다.

```
library(optparse)
```

`optparse` 모듈을 이용하여 커맨드 라인에서 스크립트 실행시 입력 옵션과 입력 파일을 받을 수 있도록 구성합니다.

```
option_list <- list (
  make_option(c("-i", "--inp"), type='character', help="Input
file path", default=NULL , metavar="character")
);
```

`"-i", "-inp"` 옵션을 생성해 `option_list`에 저장합니다. `type='character'`을 통해 옵션 뒤에 받을 추가 값을 `character`로 지정합니다. `default=NULL`을 통해 기본값을 따로 지정하지 않으면, `help`와 `metavar`을 설정해 help 커맨드 실행시 필요한 정보를 입력합니다.

입력 옵션이 여러개인 경우 `make_option` 부분을 추가하여 작성합니다.

```
opt_parser <- OptionParser(option_list=option_list);
opt <- parse_args(opt_parser);

inputfilepath = opt$inp;
print(inputfilepath);
```

스크립트 실행시 입력된 입력 인자들을 `option_list`와 비교하여 옵션에 맞게 입력 되어 있다면, `opt <- parse_args(opt_parser);`을 통해 `opt$[옵션명]`에 해당 옵션을 통해 받은 추가 문자열(입력파일 경로)을 저장합니다.

이후 `print()` 함수를 통해 입력 파일의 경로를 출력합니다.

입력 파일이 여러개인 경우

입력 파일이 여러개인 경우는 [1개인 경우 예제 \(page 123\)](#)에서 option_list 리스트에 make_option함수를 통해 입력 받고자 하는 옵션을 추가 해주면 됩니다.

```
#!/usr/bin/env Rscript

library(optparse)

option_list <- list (
  make_option(c("-i", "--inp"), type='character', help="Input
file path", default=NULL ,metavar="character"),
  make_option(c("-m", "--mesh"), type='character', help="Inpu
t mesh path", default=NULL ,metavar="character")
);

opt_parser <- OptionParser(option_list=option_list);
opt <- parse_args(opt_parser);

if (is.null(opt$inp) & is.null(opt$mesh) ){
  print_help(opt_parser);
  stop("At least one argument must be supplied (input fil
e).n", call.=FALSE);
}

inputfilepath = opt$inp;
meshfilepath = opt$mesh;

print(inputfilepath);
print(meshfilepath);
```

SDE case study

[링크] Github에서 보기

[링크] 소스코드 다운받기

관련 문서 링크

- SDE 프로그래밍 방법 개요 (page 25)
- SDE 데이터 타입 생성하기 (page 151)

다음과 같이 숫자형 변수 2개(정수형 변수 1개, 실수형 변수 1개), 리스트형 변수 1개, 3차원 벡터 1개를 받는 SDE를 생성했습니다.

▣ INPUTDECK 정의

value delimiter	SPACE ↴	Line delimiter	NULL ↴	Comment Char	<input type="text"/>	미리보기	KEY VALUE															
Vector bracket	SQUARE_SPACE ↴	Vector Delimiter	SPACE ↴	미리보기	[A B C]																	
<table border="1"> <thead> <tr> <th>Variable Name</th> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>INT1</td> <td>42</td> <td><input type="button" value="?"/></td> </tr> <tr> <td>REAL1</td> <td>42.112</td> <td><input type="button" value="?"/></td> </tr> <tr> <td>LIST1</td> <td>a ↴</td> <td><input type="button" value="?"/></td> </tr> <tr> <td>VECTOR1</td> <td>1 1 1</td> <td><input type="button" value="?"/></td> </tr> </tbody> </table>				Variable Name	Value	Description	INT1	42	<input type="button" value="?"/>	REAL1	42.112	<input type="button" value="?"/>	LIST1	a ↴	<input type="button" value="?"/>	VECTOR1	1 1 1	<input type="button" value="?"/>	타입 *	선택 ↴	※ 타입 선택 후 변수 추가	
Variable Name	Value	Description																				
INT1	42	<input type="button" value="?"/>																				
REAL1	42.112	<input type="button" value="?"/>																				
LIST1	a ↴	<input type="button" value="?"/>																				
VECTOR1	1 1 1	<input type="button" value="?"/>																				
				변수명 *	<input type="text"/>																	

Case1

데이터 생성 방식은 다음과 같이 설정했습니다.

KEY	VALUE	KEY	VALUE
value delimiter	SPACE	Vector vracket	SQUARE_SPACE
line delimiter	NULL	Vector delimiter	SPACE

이렇게 설정되어 생성된 입력 파일은 다음과 같습니다.

```
INT1 42
REAL1 42.112
LIST1 a
VECTOR1 [ 1 0 0 ]
```

Example

명령행 인자(Command Line Argument) 방식으로 생성된 입력 파일을 읽고 입력된 변수 값
을 출력하는 예제 코드입니다.

```
#!/usr/bin/env Rscript

library(optparse)

option_list <- list (
  make_option(c("-i","--inp"), type='character', help="Input
file path", default=NULL ,metavar="character")
);

opt_parser <- OptionParser(option_list=option_list);
opt <- parse_args(opt_parser);

if (is.null(opt$inp)){
  print_help(opt_parser);
  stop("At least one argument must be supplied (input fil
e).n", call.=FALSE);
}

inputfile = opt$inp;

## input param ##
lines <- readLines(inputfile);

sde <- list()

for (line in lines) {
  data <- unlist(strsplit(line, " "))
  if (data[1] == 'INT1') {
    sde$int1 = as.integer(data[2]);
  } else if (data[1] == 'REAL1') {
    sde$real1 = as.double(data[2]);
  } else if (data[1] == 'LIST1') {
    sde$list1 = data[2];
  } else if (data[1] == 'VECTOR1') {
    sde$vector1 = c(as.integer(data[3]),
                    as.integer(data[4]),
                    as.integer(data[5]));
  }
}

cat('INT1 is ', sde$int1, '\n');
cat('REAL1 is ', sde$real1, '\n');
cat('LIST1 is ', sde$list1, '\n');
cat('VECTOR1 is ', sde$vector1, '\n');
```

위 예제코드는 [입력 파일이 1개인 경우 예제 \(page 123\)](#)의 코드에서 SDE 파일을 읽는 부분을 추가하였습니다.

SDE case study 2

SDE 생성시 데이터 생성 방식을 아래와 같이 설정한다면, 생성되는 입력 파일의 모양이 약간 달라질 것입니다.

KEY	VALUE	KEY	VALUE
value delimiter	EQUAL	Vector vracket	SQUARE_SPACE
line delimiter	NULL	Vector delimiter	SPACE

생성된 입력 파일

```
INT1 = 42
REAL1 = 42.112
LIST1 = a
VECT0R1 = [ 1 0 0 ]
```

추가된 value delimiter `=` 를 고려해 코딩을 해야 합니다.

```
## input param ##
con <- file(inputfile, "r");
lines <- readLines(con);
close(con);

sde <- list();

for (line in lines) {

    data <- unlist(strsplit(line, " = "))
    if (data[1] == 'INT1') {
        sde$int1 = as.integer(data[2]);
    } else if (data[1] == 'REAL1') {
        sde$real1 = as.double(data[2]);
    } else if (data[1] == 'LIST1') {
        sde$list1 = data[2];
    } else if (data[1] == 'VECTOR1') {
        sde$vector1 = c(as.integer(data[3]),
                        as.integer(data[4]),
                        as.integer(data[5]));
    }
}
```

Output programing

result 폴더 생성

R의 경우 별도의 선언없이 `dir.create()` 함수를 이용하면 되며, `result` 폴더를 생성하는 예제는 아래와 같습니다.

```
unlink("result", recursive=TRUE)
dir.create("result");
...
```

결과 파일 생성

```
fileOut<-file("result/output.txt")
...
writeLines(c("Hello EDISON."), fileOut)
close(fileOut)
```

Octave 언어

Octave은 인터프리터 방식의 프로그래밍 언어입니다. Matlab과 호환성이 높은 GNU 소프트웨어로써 계산과학을 위한 도구로 사용됩니다.

EDISON에서 R을 사용하기 위해서는 `module load octave/4.0.3` 명령어를 통해 사용할 수 있습니다.

Octave 스크립트를 리눅스 Command Line에서 실행하는 방법은 2가지입니다.

- 실행 파일 앞에 `octave` 명시하고 실행하는 방법입니다. 예를들어 `main.py` 파일을 실행하기 위해서는 `octave run.m` 명령어로 실행하면 됩니다.
- R 스크립트 파일 첫줄에 `#!` (Shebang)을 통해 스크립트를 실행시켜줄 프로그램의 경로 지정해주면 됩니다.
 - 일반적으로 `#!/usr/bin/env octave`로 선언하면 됩니다. 여기서 `env` 명령어가 환경 변수에서 지정한 언어의 위치를 찾아서 실행하게 됩니다.
 - 이 경우로 작성된 `octave` 스크립트를 사이언스 앱으로 등록하는 경우, `simrc` 파일에 사용하는 `octave` 경로 환경변수로 추가하거나, `module load` 명령어를 통해 `octave` 모듈을 추가해주어야 합니다.
 - `env` 명령어를 쓰지 않고 직접 `octave` 경로를 지정해 주어도 됩니다.
 - ex) `#!/SYSTEM/octave-4.0.3/bin/octave`
 - 이 경우 `chmod +x run.m` 명령어를 통해 해당 파일에 실행 권한을 줘야 합니다.

Octave 메뉴얼 번역 프로젝트 링크

Octave 스크립트 시작

Octave 스크립트를 EDISON에 올리고자 한다면 `octave` 스크립트 파일 맨 처음에 아래 3줄이 포함 되어있어야 합니다.

```
#!/usr/bin/env octave  
  
clear, clc, close all  
texi_macros_file("/dev/null");  
  
...
```

`clear, clc, close all`을 통해 스크립트 실행전 변수, 화면등을 초기화합니다.
`texi_macros_file("/dev/null");`을 통해 makeinfo와 관련된 에러메시지가 발생하지 않도록 설정합니다.

패키지 설치하기

`module load octave/4.0.3` 명령어를 통해 octave 모듈을 추가하면, octave 명령어를 사용할 수 있습니다.

```
$ module load octave/4.0.3
$ octave
octave: X11 DISPLAY environment variable not set
octave: disabling GUI features
warning: docstring file '/SYSTEM/octave-4.0.3/share/octave/
4.0.3/etc/built-in-docstrings' not found
GNU Octave, version 4.0.3
Copyright (C) 2016 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. For details, type 'warranty'.
Octave was configured for "x86_64-pc-linux-gnu".
Additional information about Octave is available at http://www.octave.org.
Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html
Read http://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.
octave:1>
```

pkg 명령어를 통해 사용하고자 하는 패키지를 설치 할 수 있습니다.

- **pkg** 명령어 설명

패키지를 설치하고자 하는 경우 **pkg install -local -forge <패키지이름>** 명령어를 통해 설치 할 수 있습니다. **-local** 옵션을 통해 **\$HOME/octave** 폴더에 패키지를 설치합니다.

pkg prefix 명령어를 통해 패키지 설치 위치를 변경할 수 있습니다.
local의 패키지를 설치하면 이와 관련한 정보는
\$HOME/.octave_packages 위치에 저장됩니다.

dataframe 패키지를 local에 설치하는 명령어는 다음과 같습니다.

```

octave:1> pkg install -local -forge dataframe
warning: creating installation directory /home/edison/octave
warning: called from
      install at line 30 column 5
      pkg at line 405 column 9
For information about changes from previous versions of the da
taframe package, run 'news dataframe'.
octave:2> pkg list
Package Name | Version | Installation directory
-----+-----+-----
control    | 3.0.0 | /SYSTEM/octave-4.0.3/share/octave/pa
ckages/control-3.0.0
dataframe   | 1.2.0 | /home/edison/octave/dataframe-1.2.0
signal     | 1.3.2 | /SYSTEM/octave-4.0.3/share/octave/pa
ckages/signal-1.3.2
octave:3>

```

`pkg list` 명령어를 통해 설치되어있는 패키지 확인이 가능합니다. 설치된 패키지를 사용하
기 위해서는 `pkg load <패키지 이름>` 명령어를 사용합니다.

간단한 예제

`module load octave/4.0.3`을 통해 octave 모듈을 추가하고, `main.m` 파일을 만들어 아래와 같이 작성합니다.

```

#!/usr/bin/env octave

clear, clc, close all
texi_macros_file("/dev/null");

pkg load dataframe;

truc={"Id", "Name", "Type";1, "onestring", "bla"; 2, "somestrin
g", "foobar";}
tt=dataframe(truc)

```

이후 `chmod +x main.m` 명령어를 통해 실행권한을 부여하고, `./main.m` 실행합니다.

```
$ module load octave/4.0.3
$ chmod +x main.m
$ ./main.m
octave: X11 DISPLAY environment variable not set
octave: disabling GUI features
warning: docstring file '/SYSTEM/octave-4.0.3/share/octave/
4.0.3/etc/built-in-docstrings' not found
truc =
{
    [1,1] = Id
    [2,1] = 1
    [3,1] = 2
    [1,2] = Name
    [2,2] = onestring
    [3,2] = somestring
    [1,3] = Type
    [2,3] = bla
    [3,3] = foobar
}
tt = dataframe with 2 rows and 3 columns
_1      Id      Name   Type
Nr double      char   char
 1      1  onestring    bla
 2      2 somestring  foobar
```

local 패키지를 사용하는 octave 스크립트를 사이언스 앱으로 등록하고자 하는 경우, 사용하는 패키지를 관리자에게 설치해 달라고 요청해야 합니다.

입력 파일이 1개인 경우

[링크] Github에서 보기

[링크] 소스코드 다운받기

예제코드 다운로드 및 실행

Github 사이트에 접속 해서 전체 소스가 압축된 파일을 다운 받거나, `git clone` 명령어를 이용해 소스 코드를 받을수 있습니다.

Bulb 서버에 접속 후 아래 명령어를 실행하면, 해당 소스코드를 다운 받을 수 있습니다.

Bulb 서버가 아닌 곳에서도 git이 설치되어 있다면, `git clone` 명령어를 통해 소스코드를 다운로드 받을 수 있습니다.

- [git 설치하기](#)

```
$ git clone https://github.com/sp-edison/octave_example_input1.git
```

다운로드가 완료되면, `octave_example_input1` 폴더가 생성됩니다. 폴더 구성은 다음과 같습니다.

```
r_example_input1
|   README.md
|   simrc
└── run.m
```

실행하기

source 명령어를 통해 simrc에 있는 명령어를 실행합니다.

simrc에는 R 모듈을 추가하는 명령어가 있습니다.

```
$ source simrc
```

R의 경우 Script를 실행하는 형태로 별다른 컴파일 과정이 필요 없습니다. 하지만 리눅스 명령어 chmod 명령어를 이용해 실행 권한을 주어야 합니다. `run.m` 가 실행 파일인 경우 `chmod +x run.m` 를 리눅스 상에서 실행하면 됩니다.

```
$ chmod +x run.m
$ ./run.m -i /home/ino/test.input
/home/ino/test.input
```

Source Code

simrc

```
module load octave/4.0.3
```

`module load` 명령어를 통해 octave 모듈을 추가합니다.

run.m

```
#!/usr/bin/env octave

clear, clc, close all
texi_macros_file("/dev/null");

args = argv() ;
i = 1 ;
while i <= length(args)
    option = args{i} ;
    switch option
    case {"-i" "--inp"}
        inputfile = args{++i} ;
    otherwise
        disp("err") ;
        exit(1) ;
        break ;
    endswitch
    i++ ;
endwhile

disp(inputfile)
```

주요 코드 설명

입력 인자 읽기

```
#!/usr/bin/env octave
```

#! 을 통해 이 스크립트를 실행하는 파일이 octave임을 지정합니다.

```
args = argv() ;
i = 1 ;
while i <= length(args)
    option = args{i} ;
    switch option
    case {"-i" "--inp"}
        infile = args{++i} ;
    otherwise
        disp("err") ;
        exit(1) ;
        break ;
    endswitch
    i++ ;
endwhile
```

`argv()` 함수를 통해 매개변수 리스트를 args 변수에 저장합니다. 이후 while loop를 통해 -i 또는 -inp 값을 확인해 그 다음 매개변수를 infile 변수에 저장합니다.

이후 `disp()` 함수를 통해 입력 파일의 경로를 출력합니다.

SDE case study

[링크] Github에서 보기

[링크] 소스코드 다운받기

관련 문서 링크

- SDE 프로그래밍 방법 개요 (page 25)
- SDE 데이터 타입 생성하기 (page 151)

다음과 같이 숫자형 변수 2개(정수형 변수 1개, 실수형 변수 1개), 리스트형 변수 1개, 3차원 벡터 1개를 받는 SDE를 생성했습니다.

▣ INPUTDECK 정의

value delimiter	SPACE ↴	Line delimiter	NULL ↴	Comment Char	<input type="text"/>	미리보기	KEY VALUE															
Vector bracket	SQUARE_SPACE ↴	Vector Delimiter	SPACE ↴	미리보기	[A B C]																	
<table border="1"> <thead> <tr> <th>Variable Name</th> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>INT1</td> <td>42</td> <td><input type="button" value="?"/></td> </tr> <tr> <td>REAL1</td> <td>42.112</td> <td><input type="button" value="?"/></td> </tr> <tr> <td>LIST1</td> <td>a ↴</td> <td><input type="button" value="?"/></td> </tr> <tr> <td>VECTOR1</td> <td>1 1 1</td> <td><input type="button" value="?"/></td> </tr> </tbody> </table>				Variable Name	Value	Description	INT1	42	<input type="button" value="?"/>	REAL1	42.112	<input type="button" value="?"/>	LIST1	a ↴	<input type="button" value="?"/>	VECTOR1	1 1 1	<input type="button" value="?"/>	타입 *	선택 ↴	※ 타입 선택 후 변수 추가	
Variable Name	Value	Description																				
INT1	42	<input type="button" value="?"/>																				
REAL1	42.112	<input type="button" value="?"/>																				
LIST1	a ↴	<input type="button" value="?"/>																				
VECTOR1	1 1 1	<input type="button" value="?"/>																				
				변수명 *	<input type="text"/>																	

Case1

데이터 생성 방식은 다음과 같이 설정했습니다.

KEY	VALUE	KEY	VALUE
value delimiter	SPACE	Vector vracket	SQUARE_SPACE
line delimiter	NULL	Vector delimiter	SPACE

이렇게 설정되어 생성된 입력 파일은 다음과 같습니다.

```
INT1 42
REAL1 42.112
LIST1 a
VECTOR1 [ 1 0 0 ]
```

Example

명령행 인자(Command Line Argument) 방식으로 생성된 입력 파일을 읽고 입력된 변수 값
을 출력하는 예제 코드입니다.

```
#!/usr/bin/env octave

clear, clc, close all
texi_macros_file("/dev/null");

args = argv() ;
i = 1 ;
while i <= length(args)
    option = args{i} ;
    switch option
    case {"-i" "--inp"}
        inputfile = args{++i} ;
    otherwise
        disp("err") ;
        exit(1) ;
        break ;
    endswitch
    i++ ;
endwhile

disp(inputfile)

fid = fopen(inputfile, 'r') ;

lines = textscan(fid, "%s", 'delimiter', ';' \n');

i=1;
while i <= length(lines{1})
    line = strsplit(lines{1}{i}, ' ') ;
    i++;
    switch line{1}
    case {"INT1"}
        int1 = str2num(line{2}) ;
    case {"REAL1"}
        real1 = str2double(line{2}) ;
    case {"LIST1"}
        list1 = line{2} ;
    case {"VECTOR1"}
        vector_tmp = strsplit(line{2}, " ") ;
        vector1 = { str2num(vector_tmp{1}),
                    str2num(vector_tmp{2}),
                    str2num(vector_tmp{3}) } ;
    endswitch
endwhile
```

```
disp(int1)
disp(real1)
disp(list1)
disp(vector1)

fclose ("all");
```

위 예제코드는 [입력 파일이 1개인 경우 예제 \(page 139\)](#)의 코드에서 SDE 파일을 읽는 부분을 추가하였습니다.

SDE case study 2

SDE 생성시 데이터 생성 방식을 아래와 같이 설정한다면, 생성되는 입력 파일의 모양이 약간 달라질 것입니다.

KEY	VALUE	KEY	VALUE
value delimiter	EQUAL	Vector vracket	SQUARE_SPACE
line delimiter	NULL	Vector delimiter	SPACE

생성된 입력 파일

```
INT1 = 42 ;
REAL1 = 42.112 ;
LIST1 = a ;
VECTOR1 = [ 1 0 0 ] ;
```

추가된 value delimiter `=` 를 고려해 코딩을 해야 합니다.

```
...
while i <= length(lines{1})
    line = strsplit(lines{1}{i}, ' = ');
    i++;
    switch line{1}
        case {"INT1"}
            int1 = str2num(line{2}) ;
        case {"REAL1"}
            real1 = str2double(line{2}) ;
        case {"LIST1"}
            list1 = line{2} ;
        case {"VECTOR1"}
            vector_tmp = strsplit(line{2}, " ");
            vector1 = { str2num(vector_tmp{2}),
                         str2num(vector_tmp{3}),
                         str2num(vector_tmp{4}) } ;
    endswitch
endwhile
...
```

Output programing

result 폴더 생성

Octave의 경우 별도의 선언없이 `mkdir` 함수를 이용하면 되며, `result` 폴더를 생성하는 예제는 아래와 같습니다.

```
...
rmdir result
mkdir result
...
```

결과 파일 생성

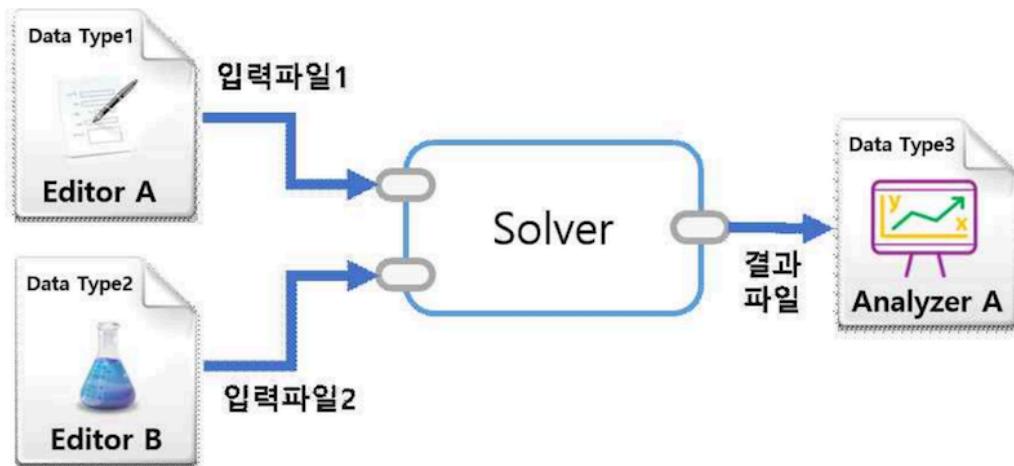
```
file_out = fopen('result/result.txt', 'w');
...
...
fdisp(file_out, 'Hello EDISON.');
...
...
fclose(file_out);
```

Octave plot을 그림파일로 저장하기

`print -dpng <저장할 파일명>` 명령어를 통해 `figure()`로 생성한 그래프를 그림으로 저장할 수 있다.

```
x = -10:0.1:10;
figure();
plot (x, sin (x));
print -dpng result/Image_Power.png;
```

Editor (편집기)



사이언스 앱 실행 시나리오

편집기는 EDISON에서 앱 실행시 입력 데이터를 편집할 수 있는 구성 요소를 말합니다. 현재 등록되어 있는 에디터 목록은 다음과 같습니다.

범용 편집기

이름	설명
SDE (Structured Data Editor) (page 151)	시뮬레이션 수행에 필요한 변수 값, 문자열, 벡터등의 데이터를 웹에서 바로 입력할 수 있는 UI를 제공 하는 편집기입니다. 데이터 타입 생성시 데이터 구조를 설계할 수 있습니다.
TEXT_EDITOR (page 164)	텍스트 파일을 편집할 수 있는 편집기 입니다.
FILE_SELECTOR (page 163)	데이터 편집 기능은 없지만, 입력 파일을 선택하는 기능을 제공합니다.
CSVEditor (page 165)	CSV file 형태를 편집할 수 있는 편집기 입니다.

입력 데이터 설계시 간단한 변수 입력이 필요한 경우 SDE를 활용하고, SDE로 표현 불가능한 데이터 형태의 경우 TEXT_EDITOR를 활용하시기 바랍니다. 그외 파일이 크거나 TEXT형식이 아닌 입력 파일의 경우 FILE_SELECTOR를 활용하시면 됩니다.

그외 특정 앱에서 활용하는 편집기가 있습니다.

- SIESTA_editor
- EPF_EDITOR
- runWTE_editor

Structured Data Editor (SDE)

The screenshot shows a web-based Structured Data Editor (SDE) interface. At the top right is a blue "Sample" button. Below it is a table of input fields:

Parameter	Value	Help
Flow_Type :	Inviscid Flow	[?]
2D_Axi_Flow :	2D Flow	[?]
Mach_Number :	0.85	[?]
Reynolds_Number :	1000000	[?]
AOA :	0	[?]
Steadiness :	Steady Flow	[?]
Total_Iteration :	10000	[?]
Physical_Time_Step :	0.1	[?]
Write_File_Interval :	10	[?]
Xpc :	0	[?]
Ypc :	0	[?]
TOL :	0.00001	[?]
CFL :	1	[?]
Flux_Scheme :	RoeM	[?]
Limiter :	Van Albada	[?]
Time_Integration :	LU-SGS	[?]
Turbulence_Model :	Menter's k-omega SST	[?]

EDISON 플랫폼에서는 SDE(Structured Data Editor)이라는 기능을 제공하여, 시뮬레이션 수행에 필요한 변수 값, 문자열, 벡터등의 데이터를 웹에서 바로 입력할 수 있는 기능을 제공하고 있습니다.

EDISON 사업 초기 Inputdeck으로 불리던 데이터 형태를 Structured Data Editor로 명칭을 변경하였습니다.

SDE을 이용해 Datatype 만들기

SDE Editor 선택

생성 시 Datatype의 이름과 설명 그리고 사용하고자 하는 Editor를 선택할 수 있습니다.
Editor 목록에서 SDE을 선택하고 **다음** 버튼을 누르면 SDE을 Editor로 사용하는 Datatype를 생성할 수 있습니다.

| My EDISON > DataType 등록

이름 *	AppName_SDE_Input	버전*	1.0.0
설명			
Editor 목록	<input type="checkbox"/> SIESTA_editor <input type="checkbox"/> CSVeditor <input type="checkbox"/> EPF_EDITOR <input type="checkbox"/> runWTE_editor <input type="checkbox"/> TEXT_EDITOR <input type="checkbox"/> FILE_SELECTOR <input checked="" type="checkbox"/> SDE <input type="checkbox"/> STRING_EDITOR	Default Editor	SDE
Analyzer 목록	<input type="checkbox"/> SIESTA_analyzer <input type="checkbox"/> ParaviewGlance <input type="checkbox"/> EPF_VIEWER <input type="checkbox"/> CFDPostViewer <input type="checkbox"/> NGLViewer <input type="checkbox"/> PlotlyViewer <input type="checkbox"/> runWTE_modeler <input type="checkbox"/> runWTE_analyzer <input type="checkbox"/> OSPPlotViewer <input type="checkbox"/> OSPTextViewer <input type="checkbox"/> OSPHtmlViewer <input type="checkbox"/> OSPImageViewer <input type="checkbox"/> ProteinViewer <input type="checkbox"/> JSMol <input type="checkbox"/> X3Dom <input type="checkbox"/> ParaView	Default Analyzer	
<input type="button" value="목록"/> <input type="button" value="▶ 다음"/>			

Datatype의 Editor 지정

SDE Datatype 편집

샘플 데이터 등록

Sample file 등록

우선 해당 Datatype의 샘플 파일을 업로드 해야합니다.

데이터 생성 방식 설정

SDE 정의 화면을 통해 변수와 벡터의 데이터가 생성되는 규칙을 설정할 수 있습니다. 해당 규칙으로 입력 파일이 생성됩니다. 각 기능에 대한 설명은 다음과 같습니다.

기능	설명
value delimiter	변수 이름과 변수 값을 구분해 주는 기호를 설정하는 부분이다. EQUAL 과 SPACE 를 선택할 수 있다.
Line delimiter	하나의 변수가 종료 됨을 알려주는 문자를 선택할 수 있다. 기본적으로 변수간에 자동 줄바꿈이 들어가 있으며, SEMICOLON , COLON , NULL 중에 하나를 선택할 수 있다.
Comment Char	입력 파일에 주석 처리를 하고자 하는 경우 주석 문의 시작 문자를 설정

기능	설명
Vector bracket	벡터 변수 사용시 괄호의 종류를 선택할 수 있다. SQUARE , ROUND , SQUARE_SPACE , ROUND_SPACE 선택 가능하며 _SPACE가 붙은 경우에는 괄호와 벡터 원소 사이에 space가 들어가 있는 상태로 생성된다.
Vector Delimiter	벡터 원소간 구분해 주는 기호를 설정 COMMA 와 SPACE 중에 하나를 설정 할 수 있다.

미리보기를 통해 설정한 기능에 따라 생성되는 결과 형태를 확인 할 수 있습니다.

value delimiter 를 **EQUAL** , **Line delimiter** 를 **SEMICOLON** 로 지정한 경우 변수는 **KEY = VALUE ;** 형태로 생성 되게 됩니다. 여기에 **Vector bracket** 를 **SQUARE_SPACE** , **Vector Delimiter** 를 **SPACE** 로 지정한 경우 3차원 벡터는 **KEY_VECTOR = [VALUE1 VALUE2 VALUE3] ;** 형태로 생성 되게 됩니다.

변수 생성

변수 생성 화면은 다음과 같습니다. 오른쪽 메뉴를 통해 변수를 생성할 수 있습니다. 생성된 변수들로 실제 데이터를 입력받는 화면은 왼쪽에 표시되게 됩니다.

Variable Name	Value	Description
upper_t :	5 ≤ 50 ≤ 500	
k :	1	
mass :	2	
damping :	0.05	
omega_out :	1	
F_out :	0	

타입 *

선택
※ 타입 선택 후 변수 추가

변수명 *

변수 타입은 numeric, string, vector, string, group, comment로 나누어 지며 각 타입별 생성 방법은 다음과 같습니다.

numeric

변수가 숫자 값을 입력 받는 경우 이 타입을 설정하면 됩니다. 최대 최소 값 설정이 가능 및 sweep 기능을 사용할 수 있습니다.

sweep 기능 설정시 한번에 여러 개의 시뮬레이션 작업을 생성 할 수 있습니다.

타입 *	numeric		
※ 타입 선택 후 변수 추가			
변수명 *			
Active ?	true		
설명			
Unit ?			
최소값	>=		
최대값	<=		
기본값 *			
	최소값	>=	
<input type="checkbox"/> Sweep ?	최대값	<=	
	By Value		
Group 선택	--empty--		
<input checked="" type="checkbox"/> 활성화조건 <input type="button" value="x 취소"/> <input type="button" value="Q 추가"/>			

Numeric 타입 생성 확인

기능	설명
변수명	생성하는 변수의 이름을 지정할 수 있습니다. 입력 데이터 생성시 이 값이 KEY 값으로 생성됩니다.
Active	true, false를 설정할 수 있으며, 해당값이 false일 경우 데이터 입력창이 보이지 않게 됩니다. 사용자에게는 입력 받을 필요가 없는 고정 변수 값을 생성할때 false 지정하여 사용하면 됩니다.
설명	해당 변수의 대한 설명을 입력할 수 있습니다. 입력창 아래 국기를 선택해 다국어를 입력 할 수 있습니다.
Unit	변수의 단위가 있는 경우 해당 단위를 입력합니다.
최소값, 최대값	변수가 입력되는 최대값과 최소값 범위를 지정할 수 있습니다. 지정한 최대 최소값을 포함할지 안할지 결정할 수 있습니다.
기본값	기본으로 표시되는 값을 입력합니다.
Sweep	sweep 기능 사용 유무를 체크할 수 있으며, 체크시 기본 값을 지정할 수 있습니다. 입력받는 값으로는 sweep 최소값과 sweep 최대값이 있으며, 작업을 생성하는 방식으로는 By Value와 By Slice가 있습니다.
Group	Group 변수가 생성되어 있는 경우 원하는 Group 변수에 포함시킬 수 있습니다.

Sweep 기능 설명

- By Value : 최소값(시작값), 최대값(종료값)을 일정한 간격으로 입력한 값(step)으로 나누어 작업 생성
- By Slice : 최소값(시작값), 최대값(종료값)을 입력한 갯수로 나누어 작업 생성

string

한 줄의 문자열을 입력 받을 때 사용하는 타입입니다.

타입 *	<input type="text" value="string"/> 
※ 타입 선택 후 변수 추가	
변수명 *	<input type="text"/>
Active ?	<input type="text" value="true"/> 
설명	<input type="text"/>  
기본값 *	<input type="text"/>
Group 선택	<input type="text" value="--empty--"/> 
<input checked="" type="checkbox"/> 활성화조건  취소  추가	

String 타입 생성 화면

기능	설명
변수명	생성하는 변수의 이름을 지정할 수 있습니다. 입력 데이터 생성시 이 값이 KEY 값으로 생성됩니다.
Active	true, false를 설정할 수 있으며, 해당값이 false일 경우 데이터 입력창이 보이지 않습니다. (사용자에게는 입력 받을 필요가 없는 고정 변수 값을 생성할때 false 지정하여 사용하면 됩니다.)
설명	해당 변수의 대한 설명을 입력할 수 있습니다. 입력창 아래 국기를 선택해 다국어를 입력 할 수 있습니다.
기본값	기본으로 표시되는 값을 입력합니다.
Group	Group 변수가 생성되어 있는 경우 원하는 Group 변수에 포함시킬 수 있습니다.

list

list 항목을 미리 생성하고, 생성된 list 중 선택한 입력을 받을 때 사용하는 타입입니다.

타입 *	<input type="text" value="list"/> ▼
※ 타입 선택 후 변수 추가	
변수명 *	<input type="text"/>
Active ?	<input type="text" value="true"/> ▼
설명	<input type="text"/> 

※ 첫번째 입력 값은 list 타입에서 기본값으로 설정 됩니다.

Name	<input type="text"/> ▼		<input type="checkbox"/> 생성	<input type="button" value="삭제"/> ▼
Value	<input type="text"/>			
Group 선택	<input type="text" value="--empty--"/> ▼			
<input checked="" type="checkbox"/> 활성화조건 <input type="button" value="취소"/> <input type="button" value="추가"/>				

list 타입 생성 화면

기능	설명
변수명	생성하는 변수의 이름을 지정할 수 있습니다. 입력 데이터 생성시 이 값이 KEY 값으로 생성됩니다.
Active	true, false를 설정할 수 있으며, 해당값이 false일 경우 데이터 입력창이 보이지 않게 됩니다. (사용자에게는 입력 받을 필요가 없는 고정 변수 값을 생성할때 false 지정하여 사용하면 됩니다.)
설명	해당 변수의 대한 설명을 입력할 수 있습니다. 입력창 아래 국기를 선택해 다른국어를 입력 할 수 있습니다.
기본값	기본으로 표시되는 값을 입력합니다.

기능	설명
Name과 Value	입력 받을 리스트를 생성할 수 있습니다. Name은 Editor에서 표시되는 부분이며 다국어로 입력합니다. Value는 해당 List를 선택할 때 입력 데이터에 생성되는 KEY 값입니다. 갱신과 삭제를 통해 list를 추가하고 삭제할 수 있습니다.
Group	Group 변수가 생성되어 있는 경우 원하는 Group 변수에 포함시킬 수 있습니다.

vector

vector 형태의 입력 값을 받을 때 사용하는 타입입니다.

The screenshot shows a configuration dialog for a 'vector' type. The fields are as follows:

- 타입 ***: vector
- ※ 타입 선택 후 변수 추가**: (Red text message)
- 변수명 ***: (Empty input field)
- Active ?**: true
- 설명**: (Empty input field) with Korean and English flags below it.
- Dimension**: 2차원
- 기본값 ***: (Two empty input fields)
- Group 선택**: --empty--

At the bottom right are three buttons: 활성화조건, ✖ 취소, and ⌂ 추가.

Vector 타입 생성 화면

기능	설명
변수명	생성하는 변수의 이름을 지정할 수 있습니다. 입력 데이터 생성 시 이 값이 KEY 값으로 생성됩니다.

기능	설명
Active	true, false를 설정할 수 있으며, 해당값이 false일 경우 데이터 입력 창이 보이지 않게 됩니다. (사용자에게는 입력 받을 필요가 없는 고정 변수 값을 생성할때 false 지정하여 사용하면 됩니다.)
설명	해당 변수의 대한 설명을 입력할 수 있습니다. 입력창 아래 국기를 선택해 다른국어를 입력 할 수 있습니다.
Dimension	Vector의 차원 값을 설정할 수 있습니다. 차원 값에 따라 각각의 기본값을 설정해야 합니다.
기본값	기본으로 표시되는 값을 입력합니다.
Group	Group 변수가 생성되어 있는 경우 원하는 Group 변수에 포함시킬 수 있습니다.

group

UI에서 변수들을 그룹화하고 싶은경우 사용하는 변수입니다.

타입 *	<input type="text" value="group"/> ▼ ※ 타입 선택 후 변수 추가
변수명 *	
Active ?	<input type="text" value="true"/> ▼
별칭*	<input type="text"/> [] []
변수 선택	<input type="checkbox"/> upper_t <input type="checkbox"/> k <input type="checkbox"/> mass <input type="checkbox"/> damping <input type="checkbox"/> omaga_out <input type="checkbox"/> F_out
✖ 취소 ✚ 추가	

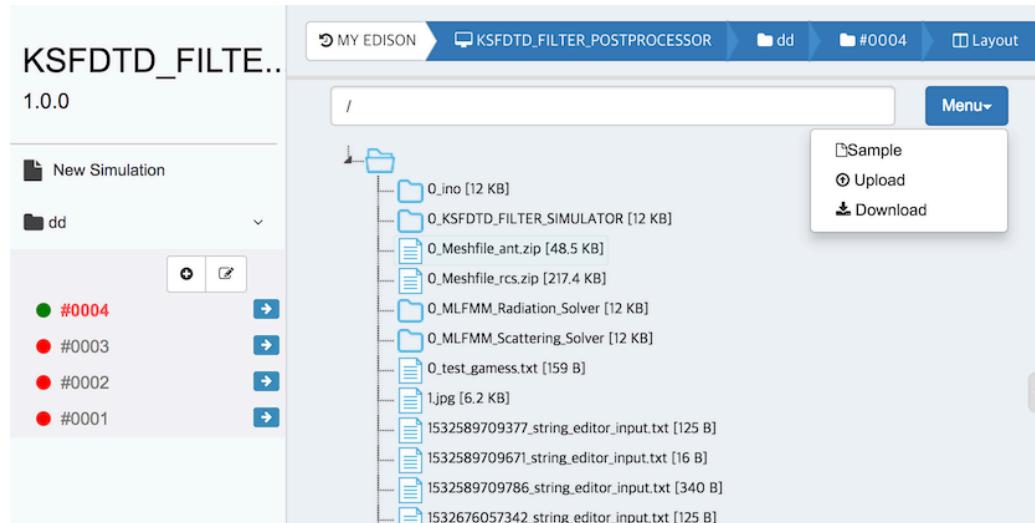
Group 타입 생성 화면

기능	설명
변수명	생성하는 변수의 이름을 지정할 수 있습니다. 입력 데이터 생성시 이 값이 KEY 값으로 생성됩니다.
Active	true, false를 설정할 수 있으며, 해당값이 false일 경우 데이터 입력 창이 보이지 않게 됩니다. (사용자에게는 입력 받을 필요가 없는 고정 변수 값을 생성할때 false 지정하여 사용하면 됩니다.)
설명	해당 변수의 대한 설명을 입력할 수 있습니다. 입력창 아래 국기를 선택해 다른국어를 입력 할 수 있습니다.
Dimension	Vector의 차원 값을 설정할 수 있습니다. 차원 값에 따라 각각의 기본값을 설정해야 합니다.
기본값	기본으로 표시되는 값을 입력합니다.

기능	설명
Group	Group 변수가 생성되어 있는 경우 원하는 Group 변수에 포함시킬 수 있습니다.

File Selector

사이언스 앱의 입력 파일을 선택할 수 있는 편집기입니다.



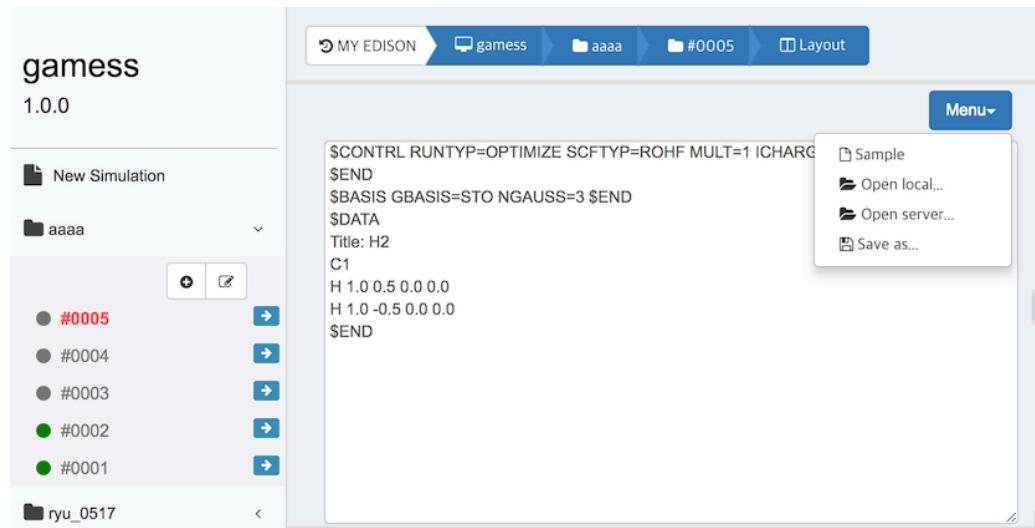
File Selector

메뉴 버튼을 클릭하면, 다음과 같은 기능을 수행할 수 있습니다.

기능	설명
Sample	개발자가 등록한 샘플 파일을 에디터로 불러옵니다.
Open local	사용자 PC로부터 텍스트 에디터로 편집하고자 하는 파일을 불러올 수 있습니다.
Open Server	EDISON 서버에 등록된 파일을 불러올 수 있습니다.
Download	선택한 파일을 PC 저장할 수 있습니다.

Text Editor

사이언스 앱의 입력 파일을 텍스트로 작성할 수 있는 편집기입니다.



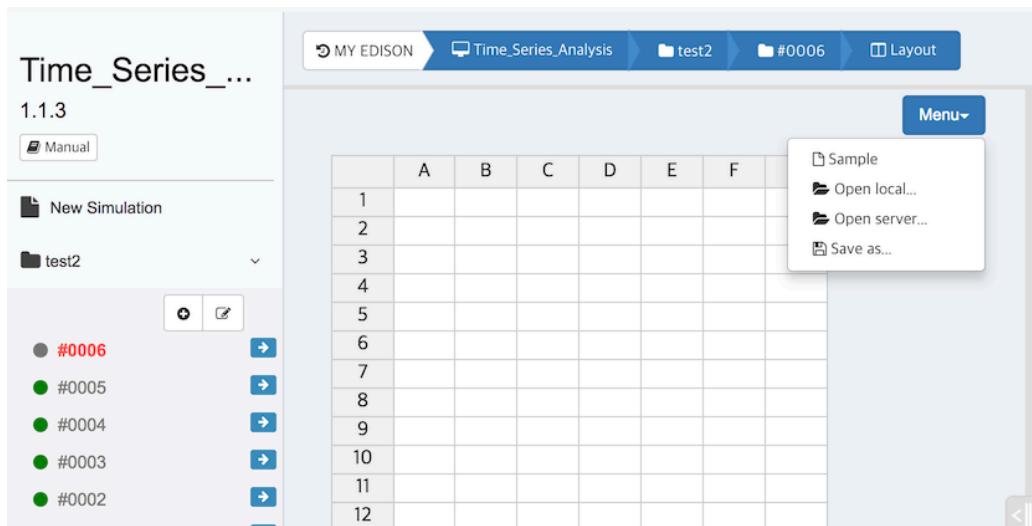
텍스트 에디터

메뉴 버튼을 클릭하면, 다음과 같은 기능을 수행할 수 있습니다.

기능	설명
Sample	개발자가 등록한 샘플 파일을 에디터로 불러옵니다.
Open local	사용자 PC로부터 텍스트 에디터로 편집하고자 하는 파일을 불러올 수 있습니다.
Open Server	EDISON 서버에 등록된 파일을 불러올 수 있습니다.
Save as	작성한 파일을 EDISON 서버에 저장합니다.

CSV Editor

사이언스 앱의 입력 파일로 CSV 파일을 읽어와 Spread Sheet 형태로 편집할 수 있는 편집기입니다.

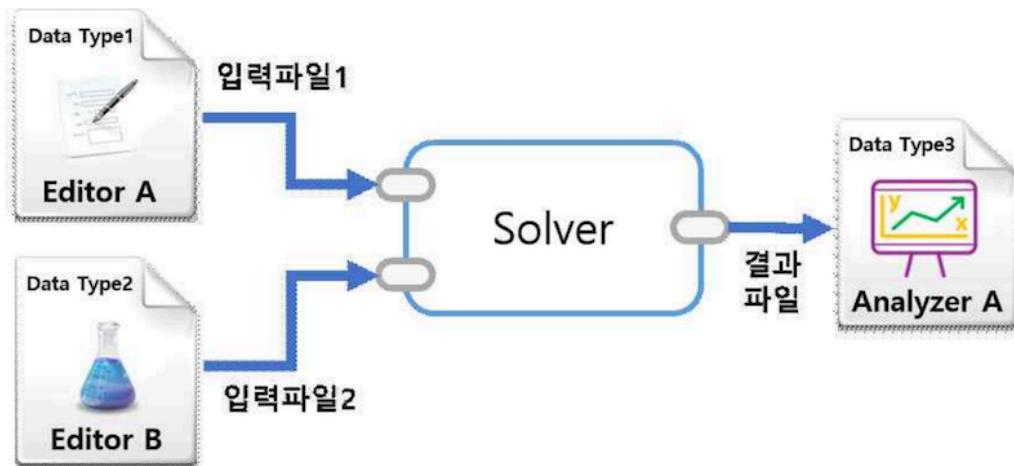


CSV Editor

메뉴 버튼을 클릭하면, 다음과 같은 기능을 수행할 수 있습니다.

기능	설명
Sample	개발자가 등록한 샘플 파일을 에디터로 불러옵니다.
Open local	사용자 PC로부터 텍스트 에디터로 편집하고자 하는 파일을 불러올 수 있습니다.
Open Serv- er	EDISON 서버에 등록된 파일을 불러올 수 있습니다.
Save as	작성한 파일을 EDISON 서버에 저장합니다.

Analyzer (분석기)



사이언스 앱 실행 시나리오

분석기는 EDISON에서 앱 실행 결과 데이터를 분석/조회 할 수 있는 구성 요소를 말합니다. 현재 등록되어 있는 분석기 목록은 다음과 같습니다.

범용 분석기

이름	설명
OSPPlotViewer (page 168)	oneD (page 31) 파일 형식을 가시화하는 뷰어입니다.
OSPTextViewer (page 0)	Text 결과 파일을 보여주는 뷰어입니다.
OSPIImageViewer (page 169)	JPG, PNG등 이미지 파일을 보여주는 뷰어입니다.
OSPHtmlViewer (page 170)	Html 파일을 보여주는 뷰어입니다.
JSMol (page 171)	Jmol의 자바스크립트 버전입니다.
ParaView (page 173)	PLOT3D등의 포맷을 가시화 해주는 3D 가시화 도구입니다.

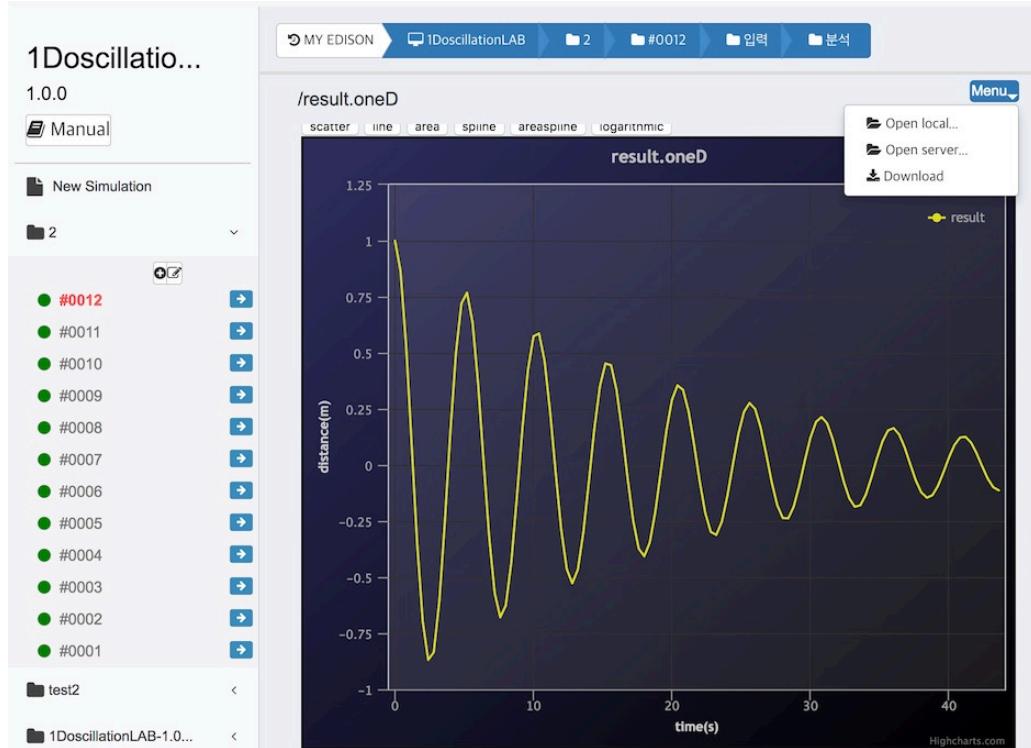
이름	설명
ProteinViewer (page 175)	3D 단백질 뷰어입니다.
PlotlyViewer (page 176)	Plotly (page 33) 라이브러리를 사용하여 다양한 Plot을 제공하는 뷰어입니다.
NGLViewer (page 178)	Proteins and DNA/RNA 등의 분자 구조를 보여주는 뷰어입니다.

그외 특정 앱에서 활용하는 분석기가 있습니다.

- SIESTA_analyzer
- runWTE_modeler
- runWTE_analyzer

OSPPlotViewer

[oneD \(page 31\)](#) 파일 형식을 가시화 해주는 뷰어입니다.



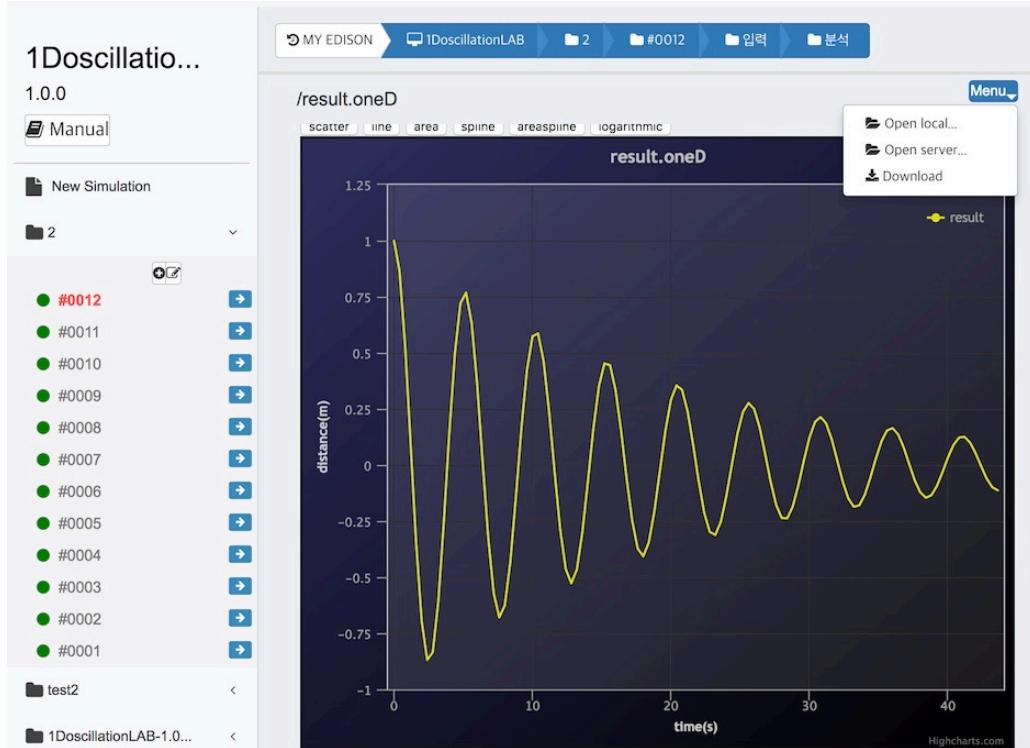
OSPPlotViewer

메뉴 버튼을 클릭하면, 다음과 같은 기능을 수행할 수 있습니다.

기능	설명
Open local	사용자 PC로부터 oneD 파일을 불러올 수 있습니다.
Open Server	출력포트 설정시 Extention과 Folder로 설정한 경우 다른 결과 파일을 불러 올 수 있습니다.
Download	선택한 파일을 PC 저장할 수 있습니다.

OSPIImageViewer

text 파일 형식을 가시화 해주는 뷰어입니다.



OSPIImageViewer

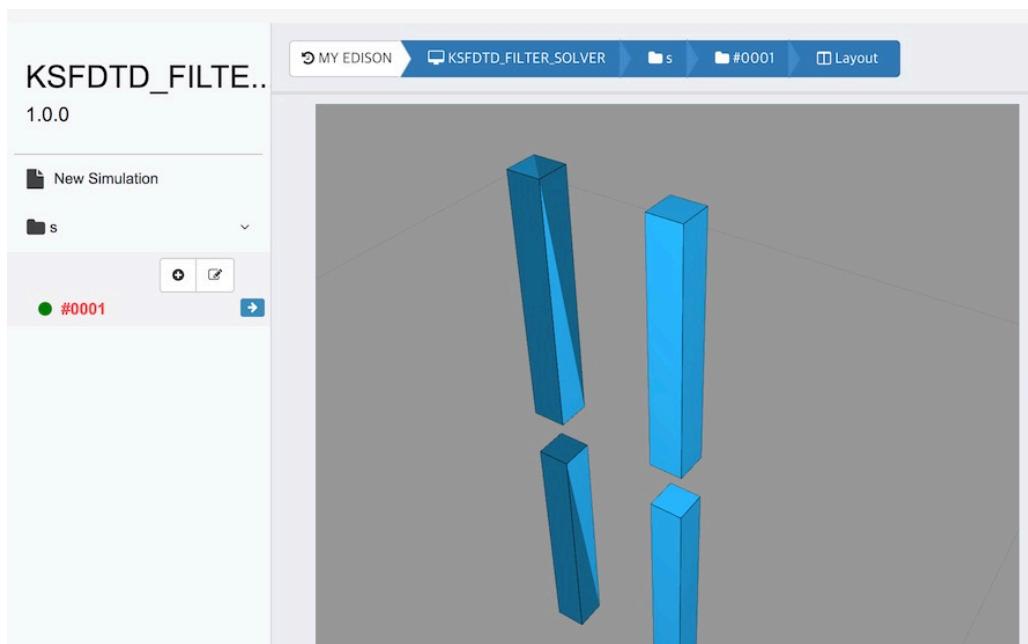
메뉴 버튼을 클릭하면, 다음과 같은 기능을 수행할 수 있습니다.

기능	설명
Open local	사용자 PC로부터 text 파일을 불러올 수 있습니다.
Open Server	출력포트 설정시 Extention과 Folder로 설정한 경우 다른 결과 파일을 불러 올 수 있습니다.
Download	선택한 파일을 PC 저장할 수 있습니다.

OSPHtmlViewer

html 파일 형식을 가시화 해주는 뷰어입니다. 출력포트의 포트 탑입을 File로 설정한 경우에만 동작합니다. Three.js등의 그래픽 자바스크립트 라이브러리를 활용하여, 결과 데이터를 가시화 할 수 있습니다. 해석기에서 생성된 결과 데이터를 읽어오는 경우에는 읽어오는 데이터의 확장자를 js로 변경해야 합니다. (브라우저 보안 정책상 js등의 일부 파일 확장자를 제외한 다를 확장자를 브라우저에서 불러오는 경우 이를 차단하고 있습니다.)

html 뷰어를 통한 가시화가 아닌 Analyzer를 개발하여 등록하면 이러한 문제를 해결할 수 있습니다.

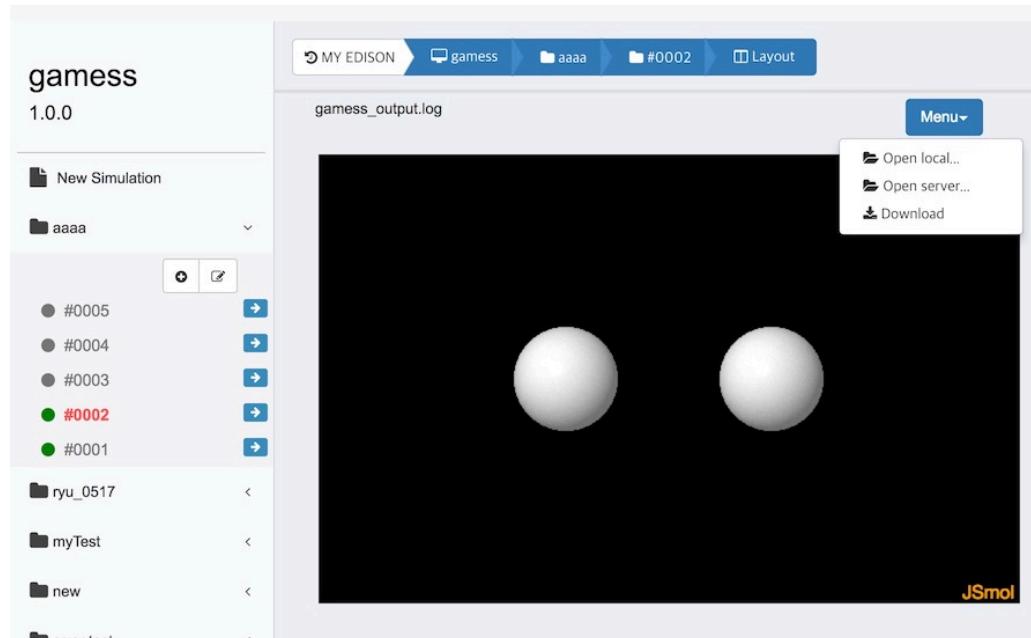


OSPHtmlViewer

JSmol

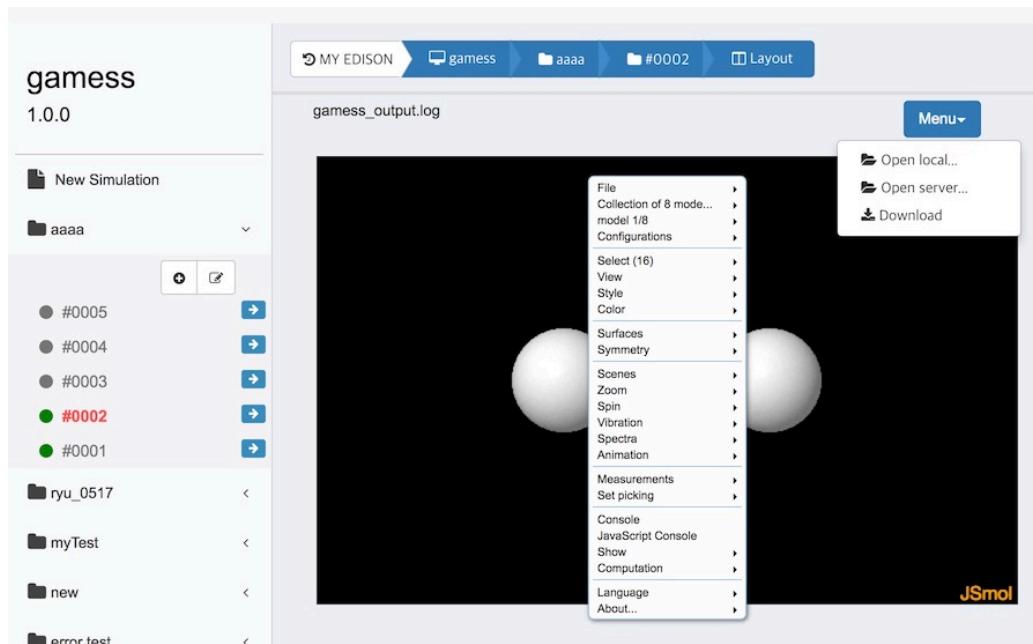
계산화학 분야에서 사용하는 jmol의 웹 버전으로 Javascript로 제작된 3D 분자 구조를 보여주는 오픈소스입니다.

가시화 가능 파일 포맷 보기

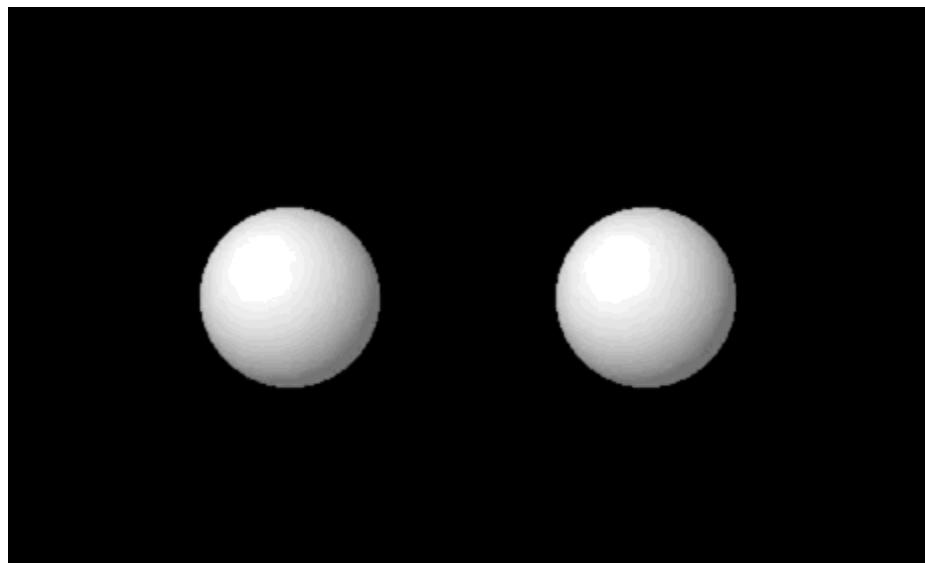


메뉴 버튼을 클릭하면, 다음과 같은 기능을 수행할 수 있습니다.

기능	설명
Open local	사용자 PC로부터 분자구조 파일을 불러올 수 있습니다.
Open Server	출력포트 설정시 Extention과 Folder로 설정한 경우 다른 결과 파일을 불러 올 수 있습니다.
Download	선택한 파일을 PC 저장할 수 있습니다.



우클릭을 통해 jsmol에서 제공하는 다양한 기능을 활용할 수 있습니다.

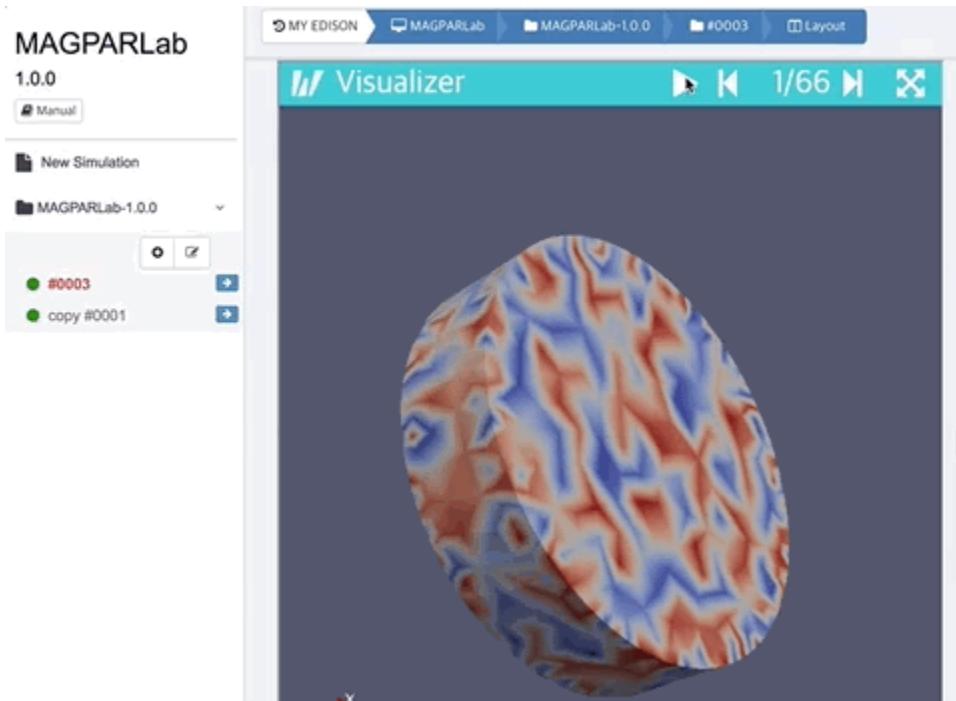


jsmol에서 제공하는 animation 기능을 통해 gamess 해석 결과를 가시화 한 예시입니다.

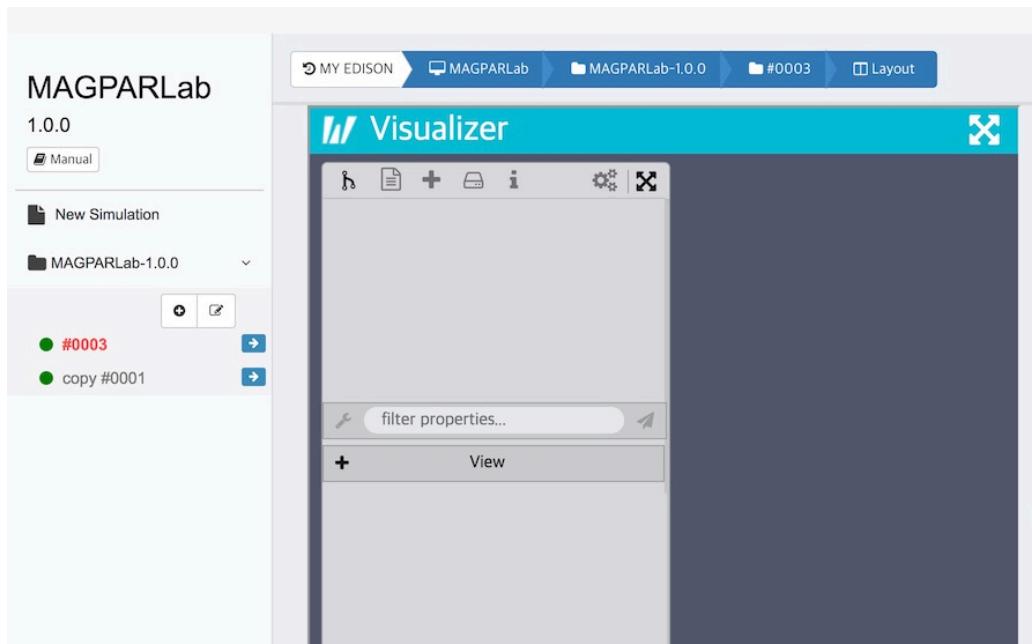
Paraview

3D 과학 데이터 가시화 도구로써 [The ParaViewWeb Visualizer application](#) 을 제공하고 있습니다. [VTK file](#) 포맷 형식의 데이터를 가시화 할 수 있으며, 그외 다른 데이터도 가시화 가능합니다.

- 가시화 가능 파일 포맷 확인하기

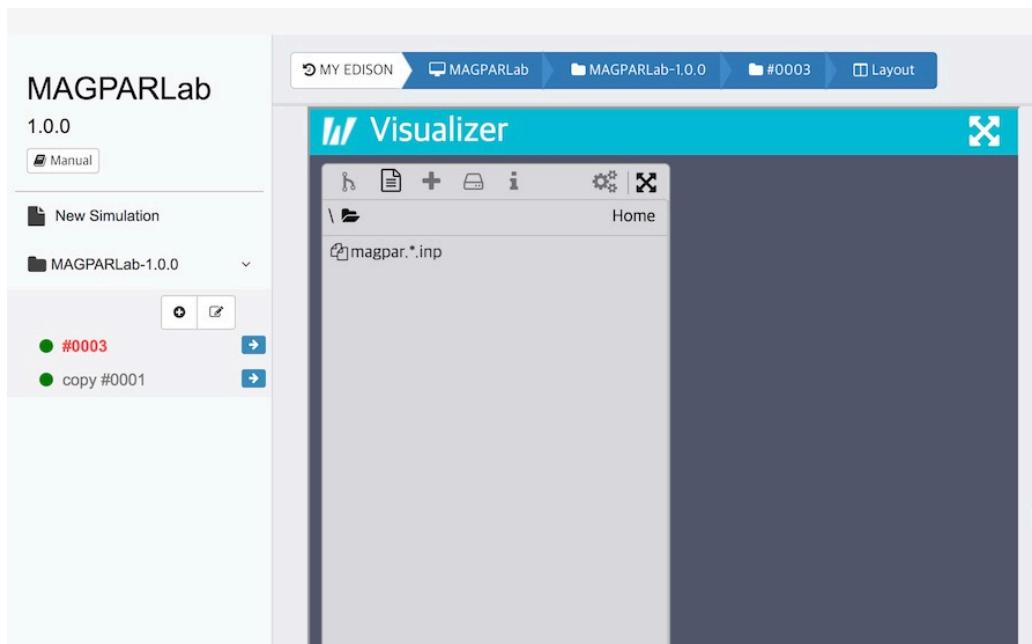


에디슨 Paraview을 실행한 화면입니다. 왼쪽 컨트롤 메뉴가 위치해 있으며, 두번째 파일 모양의 아이콘을 선택하면 가시화 할 수 있는 파일을 확인할 수 있습니다.



Animation 을 활용하기 위해서는 각 프레임 별로 vtk 파일을 생성해야하며, <파일명>.<프레임>.<확장자> 형태의 파일 이름을 가져야 합니다.

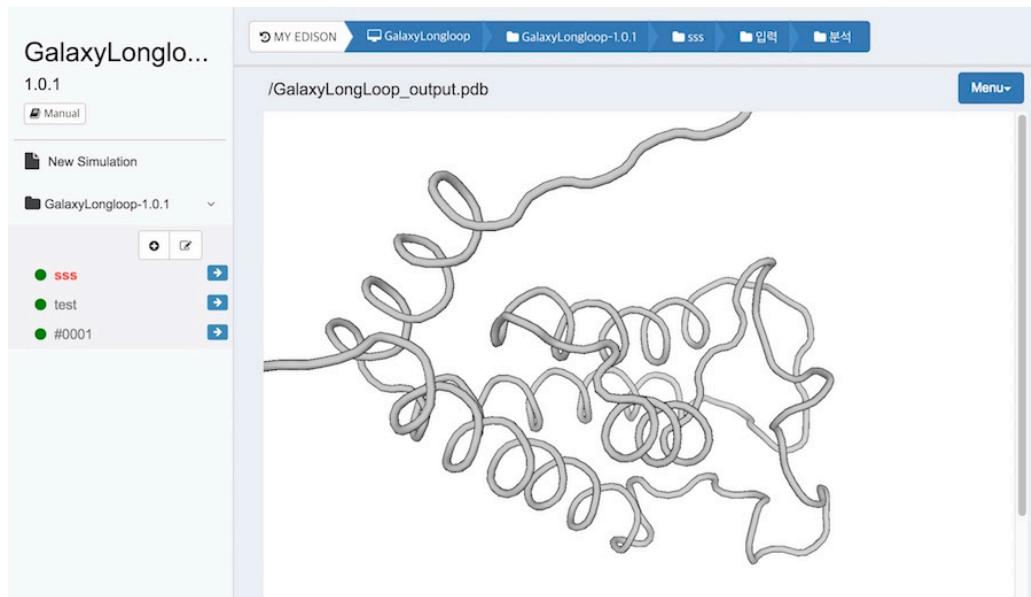
- Animation 기능을 사용하기 위한 파일이름 규칙



상단 플레이 버튼을 통해 애니메이션 결과를 확인 할 수 있습니다.

ProteinViewer

WebGL 기반 분자 가시화 라이브러리로 pdb, sdf의 3D 단백질 가시화를 제공합니다.



메뉴 버튼을 클릭하면, 다음과 같은 기능을 수행할 수 있습니다.

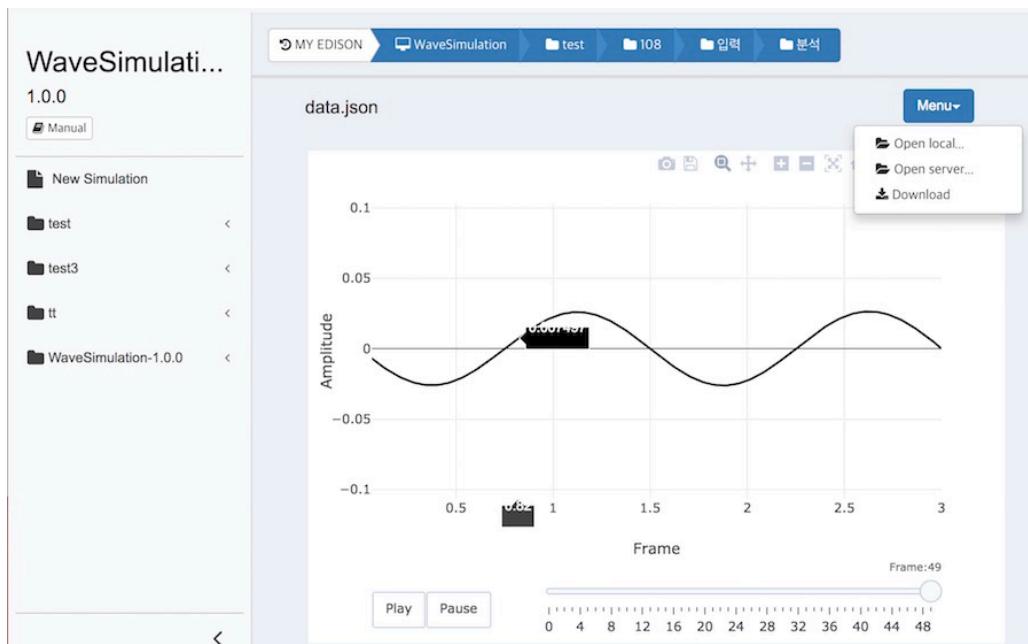
기능	설명
Open local	사용자 PC로부터 분자구조 파일을 불러올 수 있습니다.
Open Server	출력포트 설정시 Extention과 Folder로 설정한 경우 다른 결과 파일을 불러 올 수 있습니다.
Download	선택한 파일을 PC 저장할 수 있습니다.

Plotly Viewer

다양한 Plot을 제공하는 Plotly 라이브러리를 활용한 뷰어 입니다. 이 라이브러리는 Python, R, Javascript등 다양한 언어로 제공되고 있습니다. plotly 라이브러리를 활용하여 생성한 Plot 데이터를 json 포맷으로 저장해야 해야합니다.

- [Plotly 활용 프로그래밍 \(page 33\)](#)

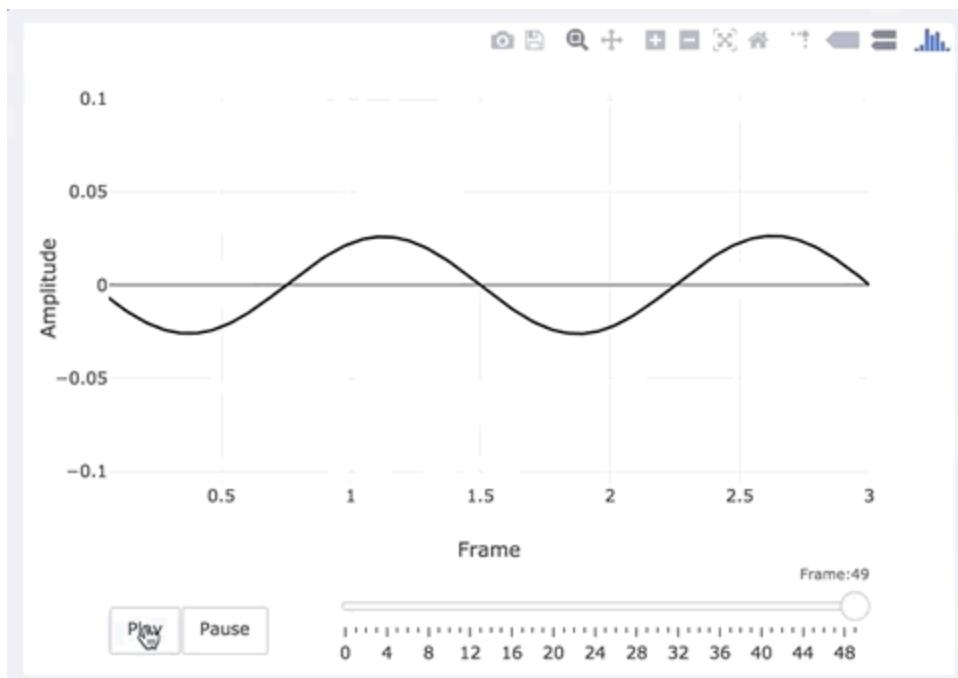
EDISON에서는 plotly json 파일의 확장자를 ply로 지정하였습니다



메뉴 버튼을 클릭하면, 다음과 같은 기능을 수행할 수 있습니다.

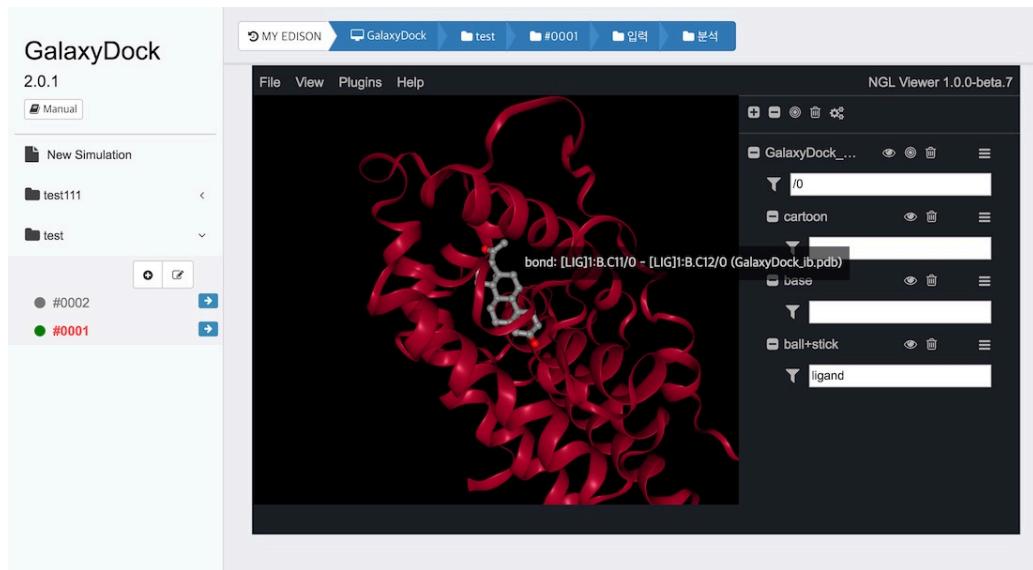
기능	설명
Open local	사용자 PC로부터 문자구조 파일을 불러올 수 있습니다.
Open Server	출력포트 설정시 Extention과 Folder로 설정한 경우 다른 결과 파일을 불러올 수 있습니다.
Download	선택한 파일을 PC 저장할 수 있습니다.

Plotly 활용 데이터 가시화 예시



NGL

WebGL 기반 분자 가시화 라이브러리로 pdb, sdf, mol2, pqr, gro, cif, mmtf등의 3D 분자 가시화를 제공합니다. jsmol 보다 빠른

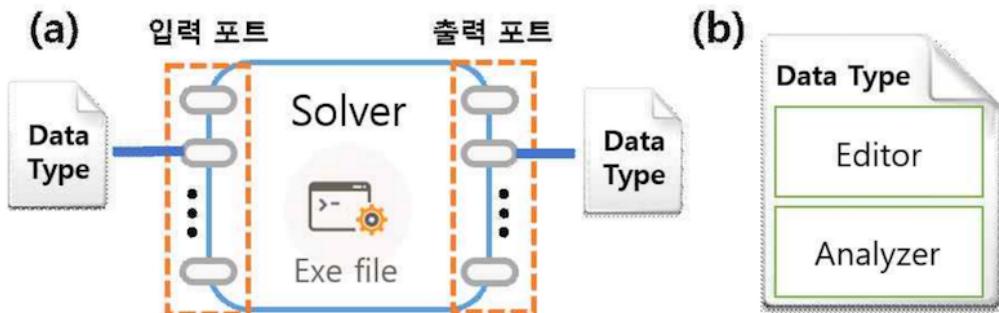


메뉴 버튼을 클릭하면, 다음과 같은 기능을 수행할 수 있습니다.

기능	설명
Open local	사용자 PC로부터 분자구조 파일을 불러올 수 있습니다.
Open Server	출력포트 설정시 Extention과 Folder로 설정한 경우 다른 결과 파일을 불러 올 수 있습니다.
Download	선택한 파일을 PC 저장할 수 있습니다.

Datatype

사이언스 앱에서 생성되는 다양한 데이터의 형태를 구분하는 요소입니다. 데이터 형태를 편집할 수 있는 편집기(Editor)와 분석할 수 있는 분석기(Analyzer)를 지정할 수 있으며, 앱을 등록하는 과정에서 입출력 포트를 생성하는 경우, 각 포트별로 데이터 타입을 지정해야 합니다.



사이언스 앱 구성요소 (a) 해석기 (b) 데이터 타입

예를 들어 흔히 사용하는 텍스트 파일을 하나의 데이터 타입으로 등록 할 수 있습니다. 아래 그림은 편집기는 TEXT_EDITOR를 지정하고, 분석기는 OSPTextViewer로 설정한 **text** 데이터 타입을 조회한 결과입니다.

The screenshot shows the 'Data Type Editor' interface with a search bar containing 'text'. Below the search bar is a table listing data types:

데이터타입명	버전
fds_surf_text_editor	V1.0.0
GalaxyGen_inputText	V1.0.0
OSPText	V1.0.0
TEXT	V1.0.0
TextViewer	V1.0.0

To the right of the table is a sidebar titled '미리보기' (Preview) which lists:

- SAMPLE FILE
[ReadMe.txt](#)
- COMMENT
TEXT
- EDITOR
TEXT_EDITOR
- ANALYZER
OSPTextViewer

text 데이터 타입

Datatype 생성

My EDISON > Datatype 관리 > 추가를 통해 신규 Datatype을 생성할 수 있습니다.

The screenshot shows the 'Data Type Editor' interface. On the left is a sidebar with various application icons. The main area displays a table of data types with their versions. A red box highlights the '추가' (Add) button at the bottom right. A dropdown menu labeled '미리보기' (Preview) is visible.

데이터타입명	버전
fileImport_zip	V1.0.0
1DBoundStateLAB_input	V1.0.0
1DoscillationLAB_input	V2.0.0
1DoscillationLAB_input	V1.0.0
1dPhononLab_inputDect	V1.0.0
1Dphonon_gif	V1.0.0
1d_Isingmodel_MC_inp	V1.0.0
1d_shocktube	V1.0.0
1D_Shocktube_2Fluid_inputdeck	V1.0.0
1D_Shocktube_InputData	V1.0.0
1_D_Phonon	V1.0.0
1_D_TAT_modeler_input	V1.0.0
2016_2	V1.0.0
2DS_uComp	V1.0.0
2DS_uComp_3_0_0_InputDeck	V1.0.0

1 2 3 4 5 »

추가

Datatype 생성

데이터 타입 생성전 검색을 통해 생성하고자 하는 데이터 타입을 찾아
볼 수 있습니다.

| My EDISON > DataType 등록

이름 *	AppName_SDE_Input	버전 *	1.0.0
설명			
Editor 목록	<input type="checkbox"/> SIESTA_editor <input type="checkbox"/> CSVEditor <input type="checkbox"/> EPF_EDITOR <input type="checkbox"/> runWTE_editor <input type="checkbox"/> TEXT_EDITOR <input type="checkbox"/> FILE_SELECTOR <input checked="" type="checkbox"/> SDE <input type="checkbox"/> STRING_EDITOR	Default Editor	SDE
Analyzer 목록	<input type="checkbox"/> SIESTA_analyzer <input type="checkbox"/> ParaviewGlance <input type="checkbox"/> EPF_VIEWER <input type="checkbox"/> CFDPostViewer <input type="checkbox"/> NGLviewer <input type="checkbox"/> PlotlyViewer <input type="checkbox"/> runWTE_modeler <input type="checkbox"/> runWTE_analyzer <input type="checkbox"/> OSPPlotViewer <input type="checkbox"/> OSPTextViewer <input type="checkbox"/> OSPHtmlViewer <input type="checkbox"/> OSPImageViewer <input type="checkbox"/> ProteinViewer <input type="checkbox"/> JSMol <input type="checkbox"/> X3Dom <input type="checkbox"/> ParaView	Default Analyzer	
<input type="button" value="목록"/> <input style="border: 2px solid red;" type="button" value="다음"/>			

Datatype의 Editor 지정

데이터 타입 생성은 데이터타입의 이름과 설명 그리고 사용하려고 하는 편집기(Editor)와 분석기(Analyzer)지정하여 생성할 수 있습니다. 여러개의 편집기와 분석기를 선택할 수 있으며, 기본 값을 지정해 주어야 합니다.

| My EDISON > DataType 수정

이름	1DoscillationLAB_Input	버전	1.0.0
설명			
Editor 목록	SDE	Default Editor	SDE
샘플 데이터 추가 *	<input type="button" value="파일 선택"/> 선택된 파일 없음 <input type="button" value="file save"/>		1DoscillationLAB_Input.xls
<input type="button" value="◀ 이전"/> <input type="button" value="저장"/> <input type="button" value="목록"/>			

Sample file 등록

마지막으로 샘플파일을 등록하면 데이터타입 등록이 완료됩니다.

Science App 생성

My EDISON > 사이언스 앱 > APP 등록를 통해 신규 App을 생성할 수 있습니다.

앱 등록은 5단계로 나누어 등록합니다.

단계	설명
앱 정보 (page 183)	앱 이름과 버전등의 앱 생성에 필요한 최소정보를 입력합니다.
실행환경 정보 (page 185)	앱 실행에 필요한 해석기(Solver)의 정보를 입력하고, 해석기를 업로드 합니다.
입/출력 포트 정보 (page 187)	입출력 파일에 대한 데이터 타입을 정의합니다.
Layout (page 191)	시뮬레이션 화면 구성을 정의합니다.
공개 데이터 (page 193)	공개에 필요한 정보를 입력합니다.

Science App 정보 입력

기본정보

앱 이름 *	<input type="text"/>	※ 앱이름과 버전은 저장 후 수정 할 수 없습니다.	버전 *	<input type="text"/> ex) 1.0.0
서비스언어 *	<input type="text"/> 한국어 (대한민국)			
앱제목 *	<input type="text"/>			
카테고리 *	<input type="text"/> 가스터빈 블레이드(OPEN) <input type="text"/> 계산화학(OPEN) <input type="text"/> 구조동역학(OPEN) <input type="text"/> 나노물리(OPEN) <input type="text"/> 도시환경(OPEN) <input type="text"/> 비행공기역학(OPEN) <input type="text"/> 선박유동해석(OPEN) <input type="text"/> 전산설계(OPEN) <input type="text"/> 전산열유체(OPEN) <input type="text"/> 전산의학(OPEN) <input type="text"/> 전파위성(OPEN)			
소유자	<input type="text"/> inoino			

사이언스앱 등록

앱 이름과 버전을 입력합니다.

- 앱 이름 항목은 공백 없이 영문, 숫자, 특수기호(- , . , _) 문자만 허용됩니다.
- 앱 이름, 버전 항목은 저장 후 수정할 수 없습니다.

서비스 언어를 선택합니다. 서비스 언어를 선택하면 해당 언어에서만 앱이 노출 됩니다. 앱 제목을 입력합니다. 선택한 서비스 언어의 데이터를 입력해야 합니다. 등록하는 앱에 적합한 카테고리를 선택합니다. 선택된 카테고리로 분류되어 앱을 조회 할 수 있습니다. 앱 정보를 저장한 후 다음 단계로 이동 할 수 있습니다.

* 로 표시된 항목은 필수 입력 값입니다.

저장 버튼을 누르게 되면 앱 정보 하단에 앱 소유자를 변경하거나 앱 관리자를 등록할 수 있는 메뉴가 나타나게 됩니다. 검색창에 소유자로 변경하거나 관리자로 등록하고자 하는 아이디를 검색해 등록하시면 됩니다.

소유자	inoino	아이디	<input type="button" value="▶ 검색"/>		
앱관리자	아이디				
순번	아이디	사용자명	E-mail	일자	삭제
조회된 결과가 없습니다.					

관리자 등록

Science App 실행환경 정보 입력

(1) 파일명 *

(2) Open Level OPEN_RUN_ONLY Source File 파일 선택 선택된 파일 없음

(3) App Type Solver

(4) Run Type Sequential PARALLEL_Module 없음

(5) 컴파일
not found directory

업로드 옵션 Upload

업로드 케이스 Update

실행파일 파일 선택 선택된 파일 없음 file save

실행환경정보 등록

(1) 업로드 한 실행 파일 명을 입력 이 실행파일명을 바탕으로 앱 테스트 및 시뮬레이션 실행을 하므로 틀리지 않게 작성해야 한다. 실행 파일명을 파일 명으로 작성합니다.

(2) Open Level을 설정 해주는 부분입니다. Open Level은 OPEN_RUN_ONLY와 OPEN_SOURCE, OPEN_BINARY, DOWNLOAD_ONLY 중 하나를 선택할 수 있습니다.

메뉴	설명
OPEN_RUN_ONLY	소스코드나, 바이너리 파일을 올리지 않고 사이언스 앱으로만 서비스를 원하는 경우 이 옵션을 선택하시면 됩니다.
OPEN_SOURCE	사용한 소스코드를 업로드하고 싶으신 경우, 이 옵션을 선택하고 Source File에 파일을 업로드 해주시면 됩니다.
OPEN_BINARY	사용한 실행파일을 업로드하고 싶으신 경우, 이 옵션을 선택하고 Binary File에 파일을 업로드 해주시면 됩니다.
DOWNLOAD_ONLY	웹사에서 실행하는 앱이 아닌 PC에 다운 받아 사용하는 앱의 경우 이 옵션을 선택하면 됩니다.

(3) 앱의 타입을 설정합니다. Solver, Converter 를 선택 할 수 있습니다.

Converter는 워크플로우에서 앱과 앱사이 연결시 전달되는 파일의 수정이 필요한 경우 사용하는 모듈입니다.

(4) 앱의 실행 타입을 설정 할 수 있습니다. 실행 타입은 Sequential, Parallel 을 선택 할 수 있습니다. 앱실행 타입이 Parallel 이면 PARALLEL_Module, Max CPU, Default CPU 를 설정 부분이 활성화 되어 설정이 가능해 집니다.

(5) Bulb에서 실행 테스트가 완료된 실행 파일이나 스크립트를 업로드하는 메뉴입니다. zip이나 tar등의 압축된 형태로 실행 파일 부분에 업로드 해주시면 됩니다. Upload는 빌드가 완료된 실행파일이나 실행 스크립트를 올리는 경우 사용하면 됩니다. Upload 선택시 업로드 캐이스를 Update와 Clean 두가지 옵션을 선택할 수 있습니다.

업로드 옵션	업로드 케 이스	설명
Upload	Update	빌드가 완료된 실행파일이나 실행 스크립트를 업로드(기존에 업로드된 파일에 덮어씀)
Upload	Clean	빌드가 완료된 실행파일이나 실행 스크립트를 업로드(기존에 업로드된 파일들은 삭제됨)

입출력 포트 생성

등록하는 사이언스 앱의 입출력 포트를 정의합니다.



입출력포트 정보

(1) 입력 포트 값에 따른 간략한 Command Line을 조회 할 수 있습니다.

- 입력한 입력 포트 정보와 앞에서 입력한 파일 명 정보를 토대로 커멘드 라인을 생성합니다. 실제 앱이 실행되는 경우 해당 커멘드 라인으로 생성되니, 실행 파일이 해당 커멘드 라인으로 실행되는지 확인해야 합니다.

(2) 입력포트를 생성하거나, 생성된 입력 포트 정보 리스트를 확인 할 수 있습니다. [입력 포트 추가](#) 버튼을 통해 추가할 수 있습니다. 실행 커멘드의 인풋 옵션을 포트 명으로 지정해 주어야 합니다.



The screenshot shows a 'SEARCH' window titled 'Data Type Editor'. At the top right is a search bar with placeholder text '데이터타입명' and a magnifying glass icon. To its right are 'Clear' and 'X' buttons. Below the search bar is a table with two columns: '데이터타입명' (Data Type Name) and '버전' (Version). The table lists various input types with their corresponding versions. On the right side of the table is a panel with a title '미리보기' (Preview) and a small preview area. At the bottom of the table are navigation buttons (1, 2, 3, 4, 5, >) and action buttons ('추가' - Add, '선택' - Select).

데이터타입명	버전
fileImport_zip	V1.0.0
1DBoundStateLAB_input	V1.0.0
1DoscillationLAB_input	V2.0.0
1DoscillationLAB_input	V1.0.0
1dPhononLab_inputDeck	V1.0.0
1Dphonon_gif	V1.0.0
1d_Isingmodel_MC_inp	V1.0.0
1d_shocktube	V1.0.0
1D_Shocktube_2Fluid_inputdeck	V1.0.0
1D_Shocktube_InputData	V1.0.0
1_D_Phonon	V1.0.0
1_D_TAT_modeler_input	V1.0.0
2016_2	V1.0.0
2DS_uComp	V1.0.0
2DS_uComp_3_0_0_InputDeck	V1.0.0

포트 명을 지정하면, 데이터 타입을 선택할 수 있습니다. 검색을 통해 원하는 데이터 타입을 선택할 수 있으며, 추가 버튼을 통해 [데이터 타입을 생성 \(page 0\)](#) 할 수 있습니다. 데이터 타입을 선택하면 관련 정보를 미리보기를 통해 확인 할 수 있으며, 선택 버튼을 통해 입력 포트를 생성 할 수 있습니다.

The screenshot shows the 'Input Port' configuration dialog. On the left is a sidebar with 'Layout' and '공개 데이터' (Public Data) selected. The main area has tabs for '입력 포트' (Input Port) and '입력 포트 추가' (Add Input Port). A table lists input ports with columns: 순번 (Index), 포트명 (Port Name), 데이터타입명 (Data Type Name), Sample File, 필수 (Required), Default, and 삭제 (Delete). One row is shown with index 1, port name '-i', data type 'sinc_input', sample file 'sinc_input.sinc', required checked, default checked, and delete button. Below the table is a table of options with descriptions:

옵션	설명
순번	생성한 입력 포트의 순번을 표시합니다. 실행 커멘드 생성시 이 순서대로 생성됩니다.
포트명	설정된 포트명을 표시합니다. 입력포트명의 경우 실행 커멘드의 인풋 옵션으로 사용됩니다. ex) 등록한 솔버의 실행 커멘드가 <code>./run.x -i [input file path]</code> 인 경우 <code>-i</code> 를 포트명으로 입력해야 합니다.
데이터 타입명	생성된 입력 포트의 데이터 타입명을 표시합니다.

옵션	설명
Sample File	클릭시 해당 입력포트의 샘플파일을 지정할 수 있습니다. 데이터 탑입의 샘플 파일을 선택하거나 해당 포트의 샘플파일을 직접 업로드 할 수 있습니다.
필수	앱을 실행시 해당 입력포트에 값이 입력되었는지 확인 반드시 해야하는지에 대한 여부를 결정할 수 있습니다. Y인 경우 해당 포트의 입력 파일을 생성해야 하며, N인 경우 입력 파일을 만들지 않아도 앱을 실행할 수 있습니다.
Default	기본 편집기(Editor)를 선택합니다. 데이터타입에 등록된 편집기 중 하나를 선택 할 수 있습니다.
삭제	해당 포트를 삭제하고자 하는 경우 삭제 버튼을 클릭하여 삭제 할 수 있습니다.

(3) Log 포트를 생성하거나 생성된 Log 포트 정보 리스트를 확인 할 수 있습니다. 설정 방법은 출력포트와 동일합니다.

(4) 출력 포트를 생성하거나 생성된 출력 포트 정보 리스트를 확인 할 수 있습니다.

출력 포트							Q. 출력 포트 추가
순번	포트명	데이터타입명	포트 탑입	File Path	필수	Default	삭제
1	oneD	oneDplot	file	result/res1	Y		

레이아웃 설정

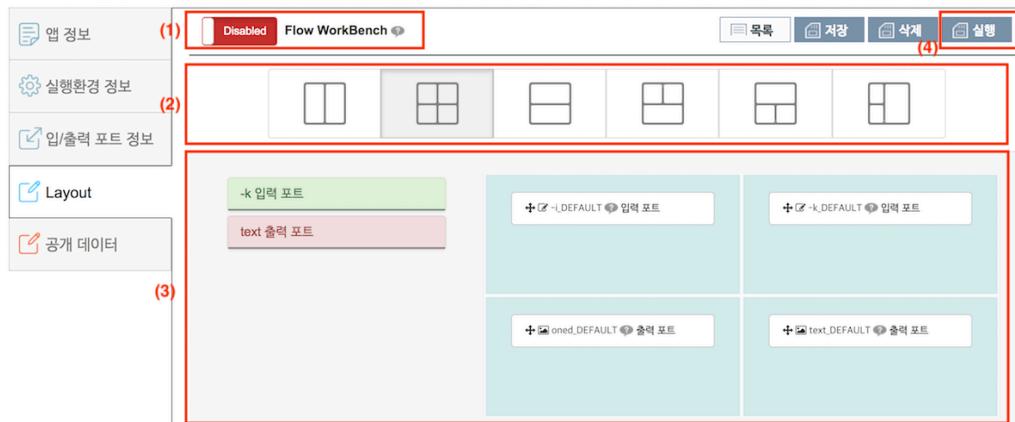
옵션	설명
순번	생성한 입력 포트의 순번을 표시합니다. 실행 커멘드 생성시 이 순서대로 생성됩니다.
포트명	설정된 포트명을 표시합니다. 입력포트명의 경우 실행 커멘드의 인풋 옵션으로 사용됩니다. ex) 등록한 솔버의 실행 커멘드가 <code>./run.x -i [input file path]</code> 인 경우 <code>-i</code> 를 포트명으로 입력해야 합니다.
데이터 탑입명	생성된 입력 포트의 데이터 탑입명을 표시합니다.
Sample File	클릭시 해당 입력포트의 샘플파일을 지정할 수 있습니다. 데이터 탑입의 샘플 파일을 선택하거나 해당 포트의 샘플파일을 직접 업로드 할 수 있습니다.

옵션	설명
포트 타입	출력 포트의 데이터 형태를 설정할 수 있다. 분석기(Analyzer)에서 결과 파일을 표시하는 방법을 지정할 수 있으며, File, Extension, Folder 등으로 선택 할 수 있습니다.
File Path	데이터 타입이 File인 경우 파일명을, Extension인 경우 확장자 이름을, Folder 인 경우 생성된 폴더 명을 입력해 줍니다. - File로 설정하고, result 폴더에 result.dat를 지정하고자 한다면, result/result.dat 로 작성해야 합니다. - Extension로 설정하고, 확장자가 oneD인 파일들을 지정하고자 한다면, result/oned 로 작성해야 합니다. - Folder로 설정하고, result/rlt인 폴더에 있는 파일들을 지정하고자 한다면, result/rlt 로 작성해야 합니다.
Default	기본 분석기(Analyzer)를 선택합니다. 데이터타입에 등록된 분석기 중 하나를 선택할 수 있습니다.
삭제	해당 포트를 삭제하고자 하는 경우 삭제 버튼을 클릭하여 삭제 할 수 있습니다.

레이아웃 설정

각 포트의 편집기와 분석기를 원하는 레이아웃에 배치할 수 있습니다. Flow 방식과 Non-flow 방식으로 나누어지며, 원하는 레이아웃을 선택해 입출력 포트 정보에서 생성한 포트들을 배치합니다. 입력 포트의 경우 편집기를 배치하며, 출력 포트의 경우 분석기를 배치합니다.

레이아웃 설정 화면은 다음과 같습니다.

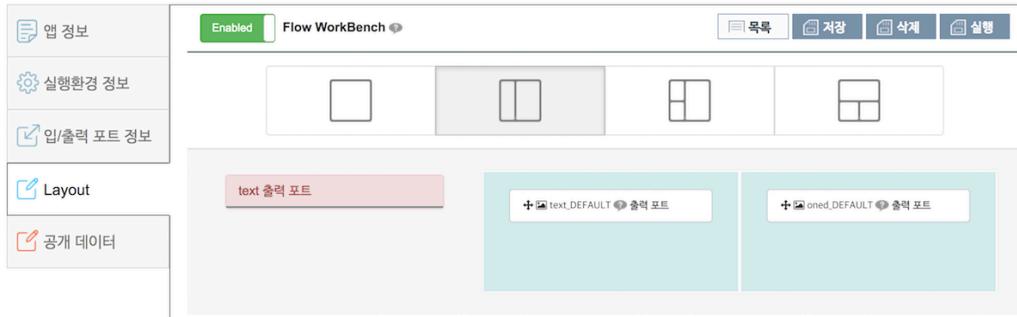


레이아웃 설정

(1) 앱 실행시 화면 구성 방식을 설정합니다.

옵션	설명
Flow Workbench Enabled	Flow Workbench Enabled의 경우 입력->로그->분석으로 나누어 레이아웃을 구성합니다. 각 단계별로 하나의 레이아웃을 가지며, 단계 이동시 레이아웃이 변경됩니다. - 분석에 대한 레이아웃만 선택하여 구성할 수 있으며, 입력과 로그의 경우 단일 레이아웃으로 설정됩니다. 시뮬레이션 수행시간이 길거나 결과 파일이 많은 경우 사용하기 적합합니다.
Flow Workbench Disabled	입력, 로그, 분석포트의 분석기 편집 하나에 레이아웃에 배치합니다. 시뮬레이션 수행시간이 짧고 입력과 출력 데이터가 간단한 경우 사용하기 적합합니다.

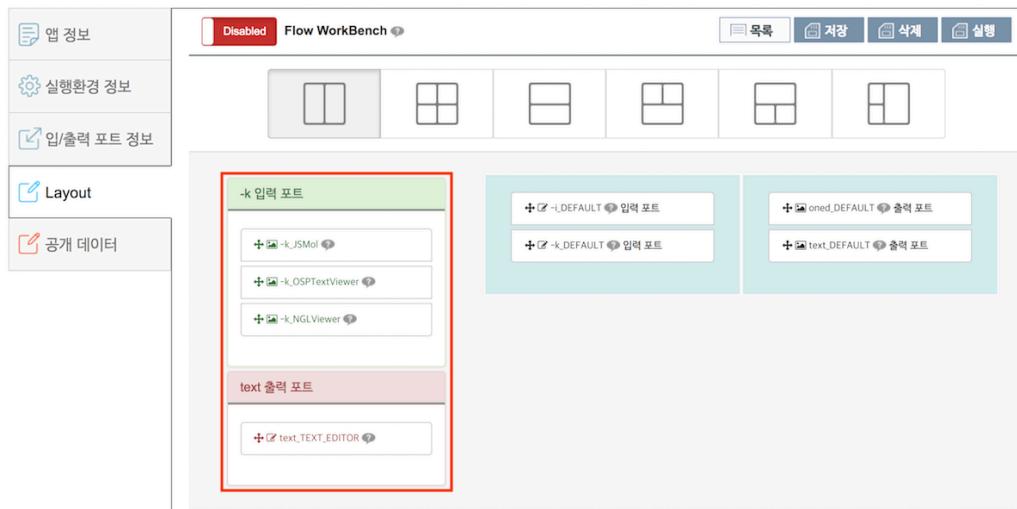
Flow Workbench Enabled인 경우의 레이아웃 편집 화면 예시입니다. 이 경우 출력 단계의 레이아웃만 편집 가능합니다.



(2) 레이아웃의 종류를 선택합니다. 선택한 레이아웃에 따라 (3) 구성이 변경됩니다.

(3) 레이아웃에 입출력 포트를 배치합니다. 입력포트를 배치하는 경우 해당 포트의 Default 편집기가 배치되며, 출력포트의 경우 해당 포트의 Default 분석기를 배치하게 됩니다.

입력 포트의 선택된 데이터 타입이 분석기를 가지고 있는 경우 분석기를 배치할 수 있습니다. 또한 출력 포트의 선택된 데이터 타입이 편집기를 가지고 있는 경우 편집기를 배치할 수 있습니다. 편집기와 분석기를 모두 가지는 포트의 경우 아래 그림과 같이 왼쪽에 표시됩니다. 해당 포트를 클릭하면 배치 가능한 편집기 분석기가 표시되며 이를 원하는 레이아웃에 배치하면 됩니다.



(4) 레이아웃을 구성하고 저장을 누르면, 구성된 레이아웃을 기반으로 앱을 실행할 수 있습니다.

공개 데이터 등록

구성된 레아이웃으로 앱 테스트를 완료하였다면, 공개 요청에 필요한 정보를 작성한 뒤 관리자에게 공개 요청해야합니다. 공개 요청을 하기 위해서 작성해야하는 내용은 설명, 매뉴얼, 아이콘, 개발자입니다.

The screenshot shows the 'App Information' registration page. On the left sidebar, there are tabs: 'App Information' (selected), 'Execution Environment Information', 'Input/Output Port Information', 'Layout', and 'Public Data'. The main area is titled 'Basic Information' and contains the following fields:

- (1) **설명 ***: A rich text editor window with Korean input enabled. It has a toolbar at the top with buttons for bold, italic, underline, etc. The text area is empty.
- (2) **매뉴얼 ***: A file selection field labeled 'English (United States)' with the message '선택된 파일 없음' (No file selected). Below it is another field for '한국어 (대한민국)' with the same message.
- (3) **아이콘 ***: A file selection field labeled '파일 선택' with the message '선택된 파일 없음' (No file selected).
- (4) **개발자 ***: A text input field with placeholder '한 줄에 하나의 값을 입력' (Enter one value per line). It contains two small icons: a blue square with a white triangle and a red square with a white triangle.
- (5) **라이센스**: An empty text input field.

공개데이터 등록

(1) 한국어(대한민국) 영어(미국)을 선택해 각각의 언어로 설명을 입력합니다. (2) 각 언어별로 매뉴얼을 파일로 업로드 합니다. (pdf 파일 권장) (3) 아이콘을 등록합니다 200x200 이하의 앱과 어울리는 이미지를 업로드 해주시면 됩니다. (4) 개발자 정보를 한글과 영어로 입력하면 됩니다. (5) 공개하고자 하는 앱의 라이센스를 입력하시면 됩니다.

The screenshot shows the 'Related Data' management page. At the top right is a button labeled '관련정보관리' (Manage Related Data). The main area displays a list of related assets:

- 사이언스앱 >
- 콘텐츠 >
- 시뮬레이션 프로젝트 >
- 커뮤니티 >

관련정보 관리

관련정보 관리 버튼을 통해 앱과 관련된 관련 Asset을 등록할 수 있습니다. 이미 등록된 앱, 콘텐츠, 프로젝트 등을 검색하여 등록할 수 있습니다.

관련정보

사이언스앱	콘텐츠	시뮬레이션 프로젝트
-------	-----	------------

선택 정보

앱이름	버전	앱체목
-----	----	-----

텍스트를 입력하여 검색하세요.

검색

관련정보 등록

사이언스 앱 관리하기

자신이 등록한 사이언스 앱은 My EDISON > 사이언스 앱에서 관리할 수 있습니다. 소유 앱과 관리 앱을 분류하여 조회할 수 있으며, 탭을 통해 목록을 조회할 수 있습니다. 앱의 상태는 작성 중, 서비스요청, 공개, 비공개 4단계로 나누어지며,

소유 앱		관리 앱			
사이언스 앱					
Clear Filter					
순번	타입	앱이름(앱제목)	상태	이름	기관명
9	Solver	2D_Percolation_WF_1.0.1 (2차원 스미기 모형)	공개	██████	KISTI
8	Solver	mergeApp_1.0.0 (mergeApp)	작성중	██████	KISTI
7	Solver	My_first_app_1.0.0 (나의 첫번째 EDISON App)	작성중	██████	KISTI
6	Solver	Curate_test_1.0.0 (Curate_test)	작성중	██████	KISTI
5	Converter	tec2vtp_1.0.0 (tec2vtp)	작성중	██████	KISTI
4	Solver	sin_func_1.0.1 (sin 함수 그리기)	비공개	██████	KISTI
3	Solver	2D_Percolation_WF_1.0.0 (2차원 스미기 모형)	비공개	██████	KISTI
2	Solver	test_app_1.0.0 (test app)	작성중	██████	KISTI
1	Solver	sin_func_1.0.0 (sin 함수 그리기)	비공개	██████	KISTI

APP 등록 편집기, 분석기 등록

앱 검색이 필요한 경우 Filter 버튼을 클릭하고, 검색하고자 하는 내용을 입력하거나 리스트를 선택하면 됩니다.

공개된 앱의 경우 앱 상세보기 페이지로 이동하면, 앱을 업그레이드하거나 비공개로 전환할 수 있습니다. 업그레이드를 클릭하면 현재 버전의 앱 관리 정보는 비공개로 변경되며, 버전의 마지막 자릿수가 하나 늘어난 앱 관리 정보가 생성됩니다. 이때 생성된 앱 관리 정보는 실행 파일 정보를 제외한 나머지 정보가 기존 버전과 동일합니다.