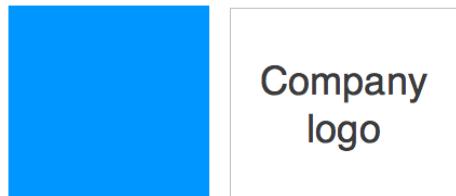


Solver Devloper Documentation

version 1.0

Last generated: December 06, 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	2
Linux Foundation.....	3
Science App	9

Developer Account

Workspace	11
Bulb Server.....	13

Science App Programming

Science App Programming 개요	15
Input Programing 개요	16
Structured Data Editor (SDE).....	18
Simrc setiing.....	20
Ounput Programing 개요.....	22
Simpost Setting.....	23
OneD Format.....	24
Plotly Format	26
Gnuplot.....	27
Gif Animation.....	30

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.

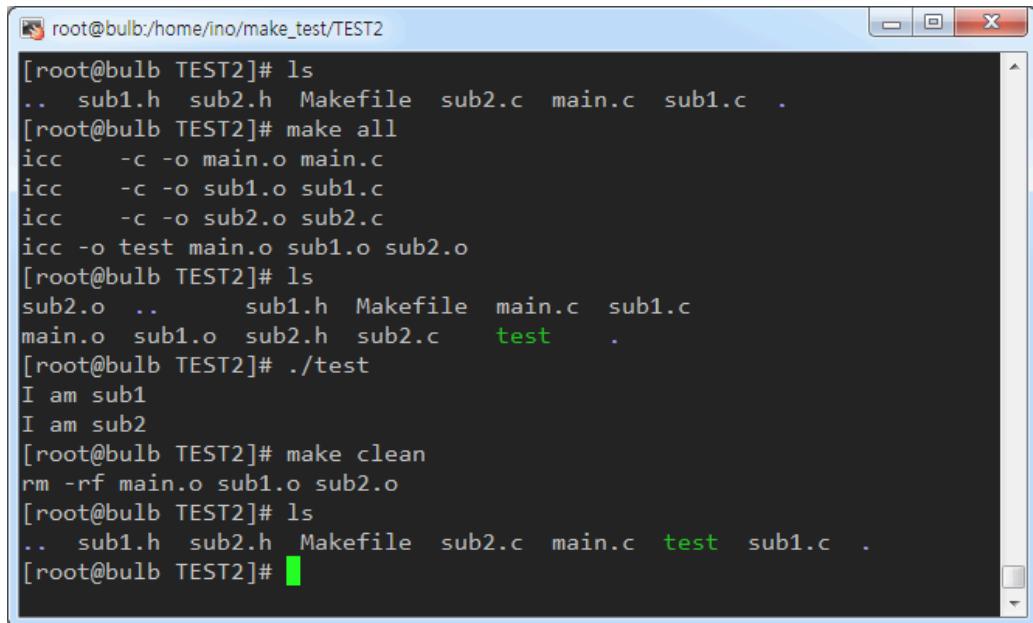
(7)

```
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 입출력 형태를 정의하기 위한 입출력 포트를 가지고 있으며, 입출력 포트의 데이터 형태를 지정하는 데 데이터 탑입을 정할 수 있습니다.

사이언스 앱 구성요소 (a) 해석기 (b) 데이터 탑입

위 그림은 사이언스 앱 구성요소 중 해석기와 데이터 탑입의 구조를 나타낸 그림입니다. (a) 해석기는 여러 개의 입출력 포트를 가지고 있으며, 각각의 입출력 포트는 1개의 데이터 탑입을 가지게 됩니다. 데이터 탑입의 경우 1개 이상의 편집기와 분석기를 가질 수 있습니다. 사이언스 앱 개발자는 입출력 포트 등록 시 기준에 등록되어 있는 데이터 탑입을 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 분석기는 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 탑입이 설정되어 있고, 데이터 탑입이 가지고 있는 편집기를 통해 입력 파일을 생성할 수 있습니다. 출력 포트의 경우에도 포트는 데이터 탑입이 설정되어 있고, 데이터 탑입이 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 탑입과 해석기 B의 입력 포트의 데이터 탑입이 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

EDISON 워크스페이스 메뉴

개발자 계정을 발급 받기 위해서는 EDISON 사이트에서 워크스페이스 요청을 해야합니다.

EDISON 사이트 접속 > My EDISON > 워크스페이스에서 **워크스페이스 요청**을 통해 신청 할 수 있습니다.

EDISON 워크스페이스 메뉴

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 IP에서는 접속이 불가 합니다.)
(5) 비고	워크스페이스 신청시 문의사항이나 요청사항이 있으면 여기에 작성하시면 됩니다.
(6) 보안 동의	보안 서약서 내용을 확인하시고 동의해주세요.

메뉴	설명
(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.

EDISON 워크스페이스 메뉴

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

EDISON 워크스페이스 메뉴

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

EDISON 워크스페이스 메뉴

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

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 \(page 0\)](#)
- [Fortran \(page 0\)](#)
- [Python \(page 0\)](#)
- [R \(page 0\)](#)
- [Octave \(page 0\)](#)

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 \(page 0\)](#)
- [Fortran \(page 0\)](#)
- [Python \(page 0\)](#)
- [R \(page 0\)](#)

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 \(page 0\)](#)

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 읽어와야 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,

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
해단 언어별 예제는 아래 링크에 있습니다.

- [C \(page 0\)](#)
- [Fortran \(page 0\)](#)
- [Python \(page 0\)](#)
- [R \(page 0\)](#)
- [Octave \(page 0\)](#)

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 \(page 0\)](#)
- [Fortran \(page 0\)](#)
- [Python \(page 0\)](#)
- [R \(page 0\)](#)
- [Octave \(page 0\)](#)

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.

oneD file

oneD 파일은 [OSPPlotViewer 분석기 \(page 24\)](#)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 갯수 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 \(page 0\)](#)

- [Fortran \(page 0\)](#)
- [Python \(page 0\)](#)

Plotly file format

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

Python 예제 보기

- [Python \(page 0\)](#)

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 \(page 0\)](#)

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 T
reatise 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

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[프래임넘버].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](#)