# EDISON 플랫폼용 코드 개발 절차

KISTI 전인호 연구원

inojeon@kisti.re.kr

042-869-1658

# 목차

- EDISON 플랫폼 탑재를 위한 코드 개발 절차 소개
- 사이언스 앱스토어 등록 절차 소개
- 셈플 코드 등록

# Science App 등록 절차

- Bulb 서버 접속
- Code upload  & Compile
- 실행 파일 압축 (tar –cvf )
- 사이언스 앱스토어 > 앱 관리 > 앱 등록
  - 앱 정보 입력
  - 실행환경 정보 입력
  - 입/출력 포트 정보 입력
  - 앱 테스트 -> 성공?
- 서비스 요청 -> 관리자 승인 -> 서비스!

# 계정 발급

- 각 분야 별 사이트 접속 **사이언스 앱스토어 > 워크스페이스** 클릭
  - 양식에 맞게 작성 (IP가 다른 경우 접속이 되지 않음!)
  - 워크 스페이스 요청 > EDISON 관리자의 승인
    - IP 접근 권한 획득, 접속 ID,  초기 비밀번호 획득

- 비밀번호 만료나 분실, IP 추가 등의 문의는 **워크스페이스 추가 요청**을 통해 신청
  - 장기간 미 접속시 비밀번호가 만료 됨

# 개발자(Bulb) 서버 접속

- 서비스 머신 환경과 동일한 환경인 Bulb 머신 제공
  bulb.edison.re.kr
- 개발자 서버에 접속 컴파일/테스트 작업 진행
- SSH, SFTP을 제공하며 Putty, FileZilla등의 클라이언트를 이용해 접속 가능
  - 프록시 설정 필수
  - 최초 접속시 `passwd` 명령어를 이용 비밀번호 변경
  - 실습 PC에서는 프록시 설정을 안해도 됨.

# 개발자(Bulb) 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
- 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

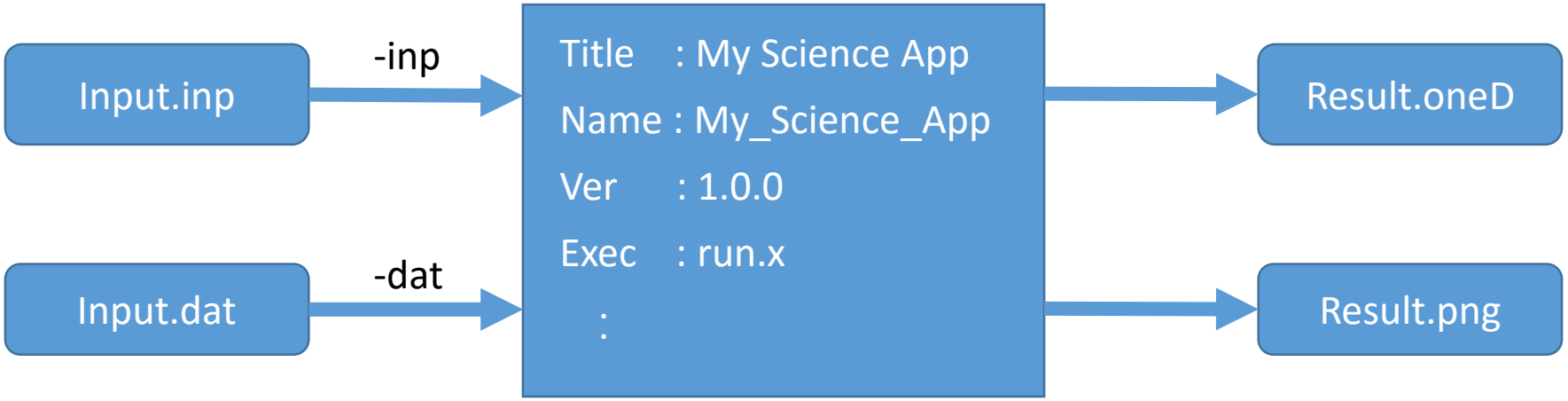
# 입출력 프로그래밍

- 시뮬레이션 SW의 실행 방식은 **./[실행 파일 명] [커맨드 옵션] [인풋 파일의 절대 경로]** 형태로 실행
  - ex) ./a.out –inp /home/user1/data/input.dat
  - **[인풋 파일의 절대 경로]** < 배열로 저장하는 경우 크기를 충분히 크게 잡아야 함
- 결과 파일은 실행 파일 디렉토리에서 **result 폴더를 생성**하여 그 안에 저장

# 이런 경우는 안돼요.

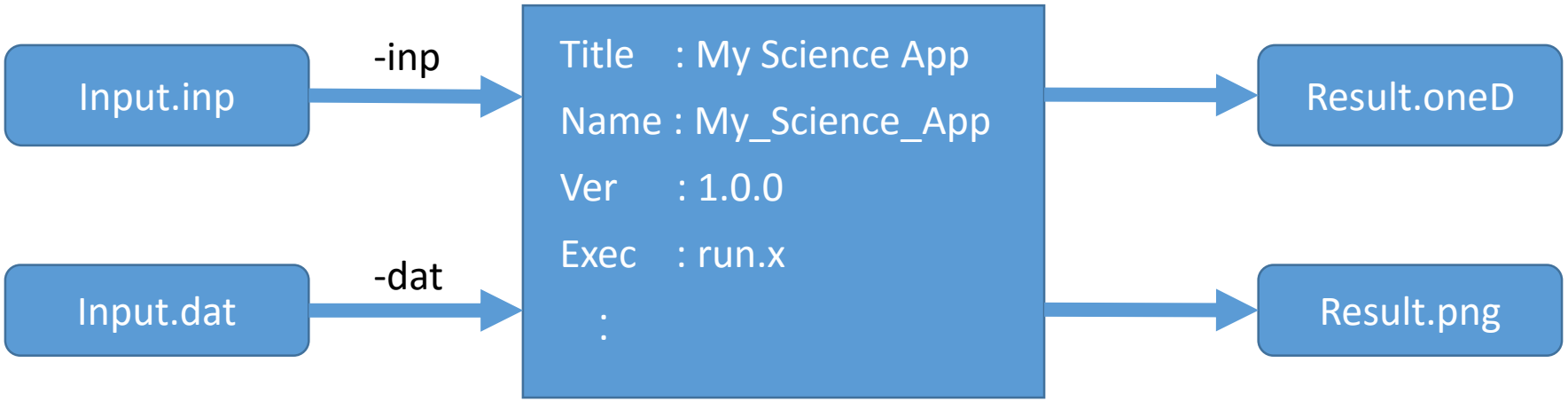

- Bulb 에서 컴파일과 테스트가 안된 경우
- 입력 파일을 입력 받지 않고 특정 위치의 특정 이름인 입력 파일만 읽는 경우
  - ex) f = open(“input.dat”, “r”);
- 사용자 키 입력을 통해 입력 파일의 이름이나 위치를 입력 받는 경우
  - C언어의 **scanf()** 함수, python의 **input()** 함수
- 결과 파일을 **result** 폴더가 아닌 다른 폴더에 결과 파일을 생성하는 경우

# Science App




```
[ino@bulb Sample_App]# ./run.x   -inp   /home/ino/Sample_App/Input.inp   -dat
/home/ino/Sample_App/Input.dat
[ino@bulb Sample_App]# ls result/
result.oneD  result.png
[ino@bulb Sample_App]#
```

# Science App




- 입력 파일은 변수 값 형태로 받는 경우와 특정 파일 포멧으로 받는 경우를 나누어 설계
  - EDISON 에서는 Inputdeck 이라는 편집기를 제공하고 있어요

# Science App

- 이렇게 해주면 참 좋은데...
  - 인풋 옵션을 체크해서 잘못 된 인풋 옵션이 들어온 경우 에러 발생
  - 인풋 파일이 잘못 입력된 경우 에러 발생
  - 1개 이상 입력 파일이 있는 경우, 입력 순서에 상관 없이 실행 가능
  - 실행 중 생성된 중간 파일들은 실행 종료시에 삭제

# Sample Code

- [사용자가이드 > 개발자 가이드] 예제 코드 및 설명
  - 인풋 파일이 1개인 경우 (Case1.*)
  - 인풋 파일이 2개인 경우 (Case2.*)
  - 인풋덱 에디터를 사용하는 경우 (Case3.*, Case4.*)
  - oneDplot 출력 예제 (Case5.*)
  - Gnuplot + oneDplot 출력 예제 (Case6.c, simrc, a.sh)
- Bulb : /home/ino/AppCode/ 에 각 언어 별로 압축 되어 있음
  - 복사하기 : cp /home/ino/AppCode/c_example.tar .
  - https://github.com/inojeon/edison_dev

# Inputdeck file read

- Inputdeck 파일은 한줄에 1개의 key 값과 value 값으로 구성
- 숫자, 스트링, 리스트, 벡터 타입을 선택 할 수 있음.
  - Value delimiter, Line delimiter, Comment Char, Vector bracket, Vector Delimiter 등을 정의 할 수 있음
- Ex) INT1 = 423 ;  INT1 112  INT1 = 231  …
- VECTOR = [ 1 2 3 ] ;  VECTOR [ 1 3 4 ]

INPUTDECK 정의

# Inputdeck file read

- Key 값을 읽어 필요한 key 값인지 확인 후 맞으면 value 값을 저장
- 원하지 않은 key 값이 입력되었을 경우 에러 발생
- Key 값들의 순서 상관 없이 값을 읽을 수 있어야 함

# 출력 프로그래밍

- 시스템 명령어를 사용 result 폴더를 생성해야 한다.
  - result 폴더 명은 소문자로만!

| C code | FORTRAN code | Python code |
| --- | --- | --- |
| #include <stdlib.h><br>…<br><br>system("rm -rf result");<br>system("mkdir result"); | CALL SYSTEM("rm -rf result")<br>CALL SYSTEM("mkdir result") | import os<br>…<br>os.system("rm -rf result");<br>os.system("mkdir result"); |

# OneDplot – 데이터 구조

➲ **Data Structure**

#NumField: *nDatafield*

#LabelX: *xlabelname*, LabelY: *ylabelname*

#Field*1*: *FieldLegend*, NumPoint: *nPoint*

$X_{11}$ $Y_{11}$

$X_{21}$ $Y_{21}$

| |

$X_{nPoint1}$ $Y_{nPoint1}$

#Field*2*: *FieldLegend2*, NumPoint: *nPoint*

$X_{12}$ $Y_{12}$

$X_{22}$ $Y_{22}$

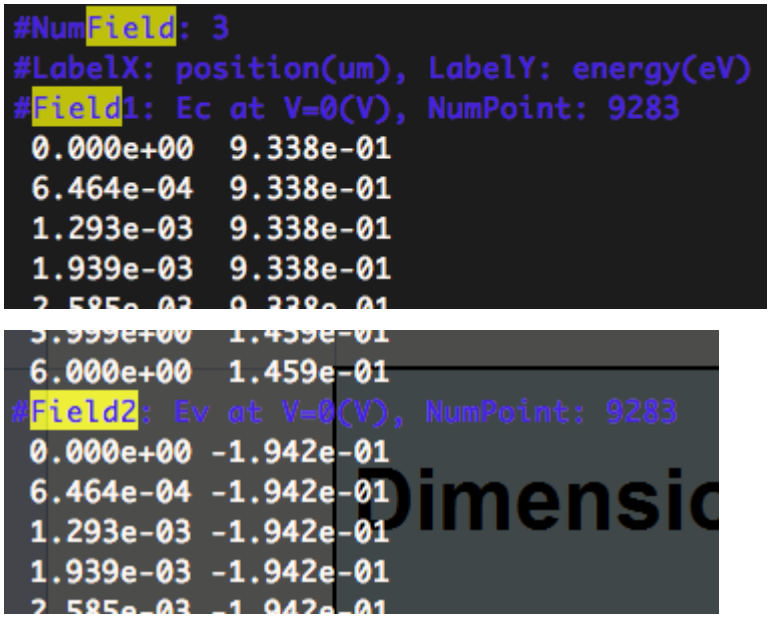
| |

$X_{nPoint2}$ $Y_{nPoint2}$

……

*** printf("%10.3e %10.3e\n", x, y)**

*** printf("%10.3e %10.3d\n", (double)x, (double)y)**



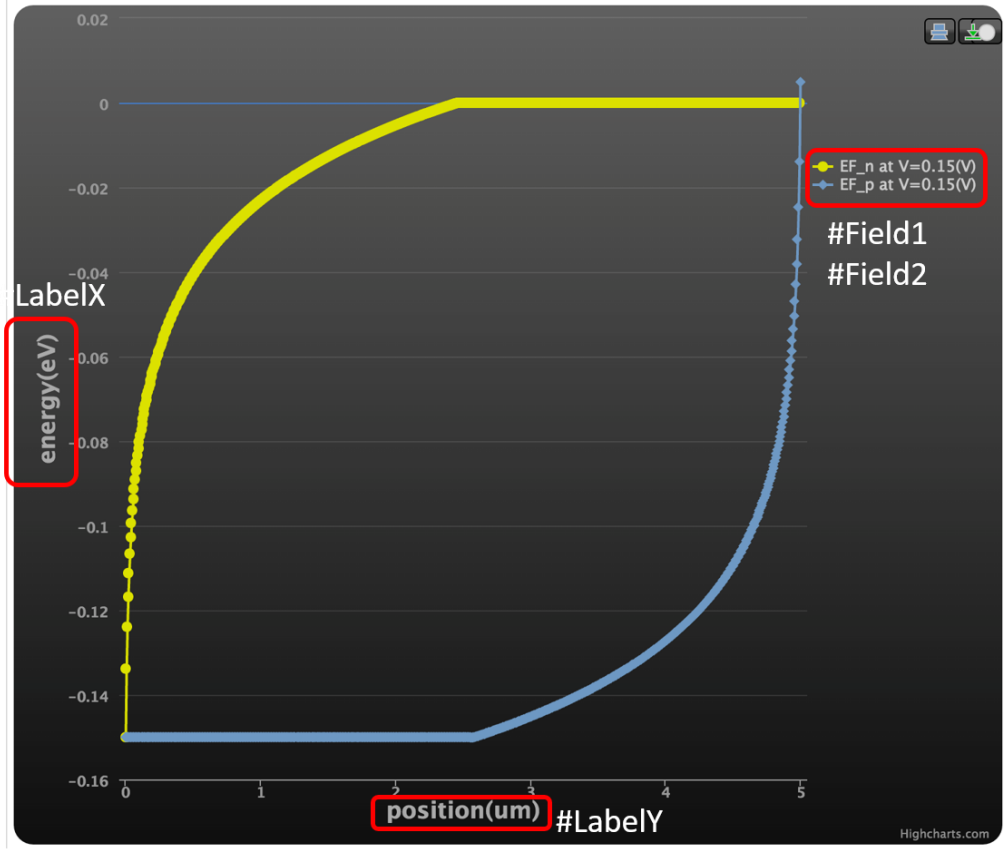

(1) *nDataField* <= 10

(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

# 샘플 데이터와 가시화 결과 화면




Example oneD file

#NumField: *2*
#LabelX: *position(um)*, LabelY: *energy(eV)*
#Field*1*: *EF_n at V=0.15(V)*, NumPoint: *300*
*0.0000 0.0000*
*0.0010 0.0045*
*…*
#Field*2*: *EF_p at V=0.15(V)*, NumPoint: *300*
*0.0000 0.0000*
*0.0010 0.0045*
*…*

# OneDplot 출력 예제 C code

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

fprintf(fp_out,"#NumField: 1\n");
fprintf(fp_out,"#LabelX: time, LabelY: a*sine(x+b) \n");
fprintf(fp_out,"#Field1: a=%d b=%f,NumPoint:%d\n", int1, real1, SIZE);
double x,y;
int t;
for(t=0; t< SIZE; t++) {
      x = (4*PI * t )/SIZE -2*PI;
      y = int1*sin( x - real1 );
      fprintf(fp_out,"%10.3f  %10.3f\n",x, y);
}

fclose(fp_out);
```

# Gnuplot 사용하기

- C 나 Fortran을 이용하여 그림 파일을 생성 시 Gnuplot를 사용하면 쉽게 그림으로 된 그래프를 그릴 수 있다.
  - Gnuplot 를 사용하기 전에 아래와 같이 path를 추가해 주어야 한다.
  - export PATH=$PATH:/SYSTEM/gnuplot-4.6.3/bin/bin

# 코드 작성 이후 컴파일

- gcc, gfortran 등 기본적인 명령어는 기본 제공, 이외 mpich, mkl 등의 명령어들은 module 명령어를 통해 사용가능
  - module avail , module load [modulefiles]

```
[root@bulb AppCode]# module avail

------------------------------------------------ /usr/share/Modules/modulefiles ------------------------------------
dot         module-git  module-info modules     null        use.own
------------------------------------------------------- /etc/modulefiles -------------------------------------------
gamess/gamess          intel/intel_11         mpi/gnu/openmpi-1.6.5   mpi/intel/mpich-1.2.7p1
[root@bulb AppCode]# module load mpi/intel/mpich-1.2.7p1
```

# 코드 작성 이후 컴파일

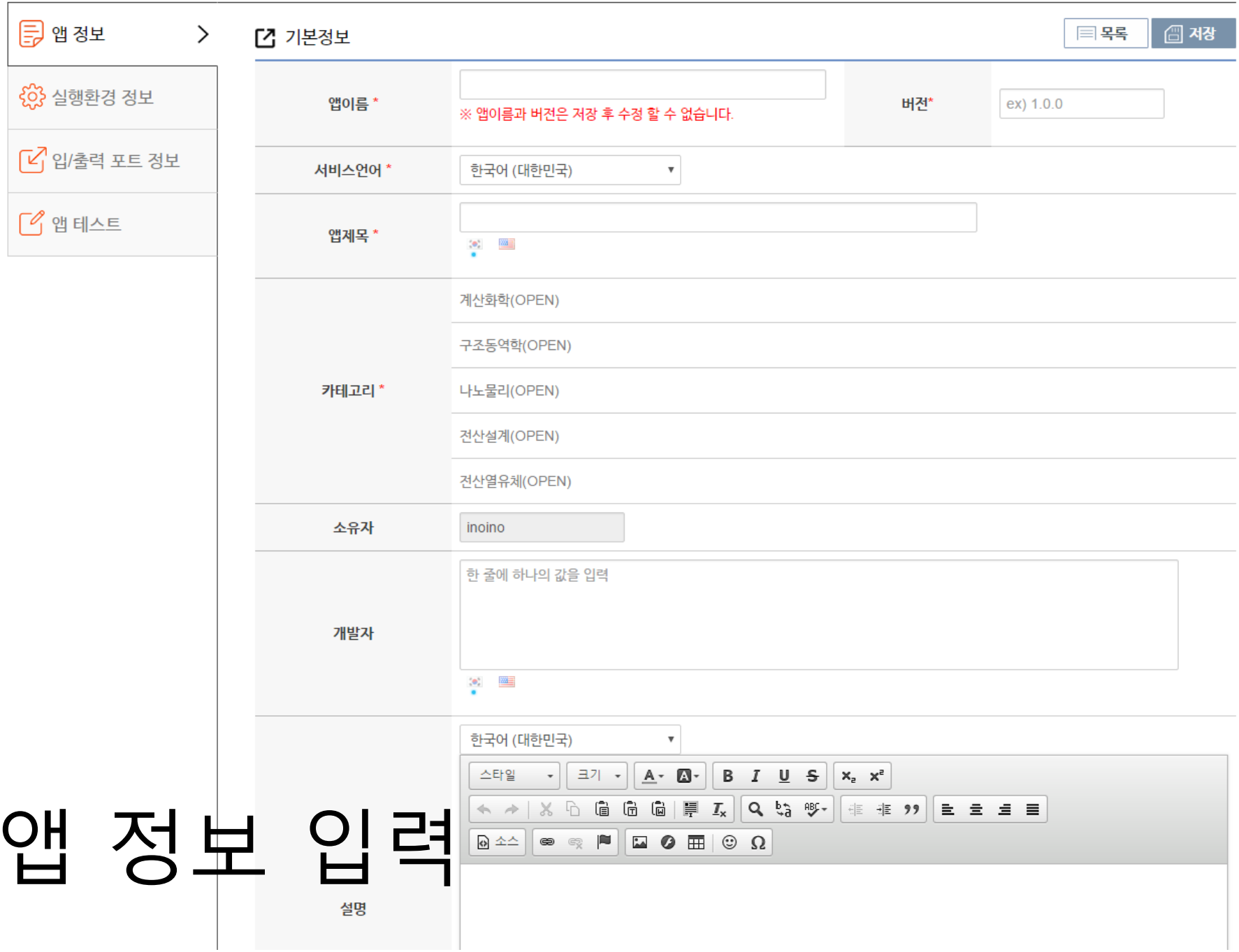
- 그외 오픈소스 명령어 등은 /SYSTEM/ 폴더에서 PATH와 LD_LIBRARY_PATH 를 추가
  - export PATH=$PATH:[추가하고자 하는 경로]
  - export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:[추가하고자 하는 경로]
- example
  - export PATH=$PATH:/SYSTEM/gnuplot-4.6.3/bin/bin
  - export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/SYSTEM/gcc-5.3.0/build/lib64/
- 이렇게 추가된 명령어들은 simrc 라는 파일 안에 작성하여 저장하고 실행파일 압축 시 같이 압축

# 컴파일 이후 압축

- tar 명령어를 이용해 실행에 필요한 파일을 압축
  - tar –cvf [압축 파일명] [압축할 파일 1 … 2]
  - ex) tar -cvf my_sci_app.tar run.x simrc input.py
- input.dat result/ 등 실행에 필요 없는 파일이나 폴더를 같이 압축하면 안됨

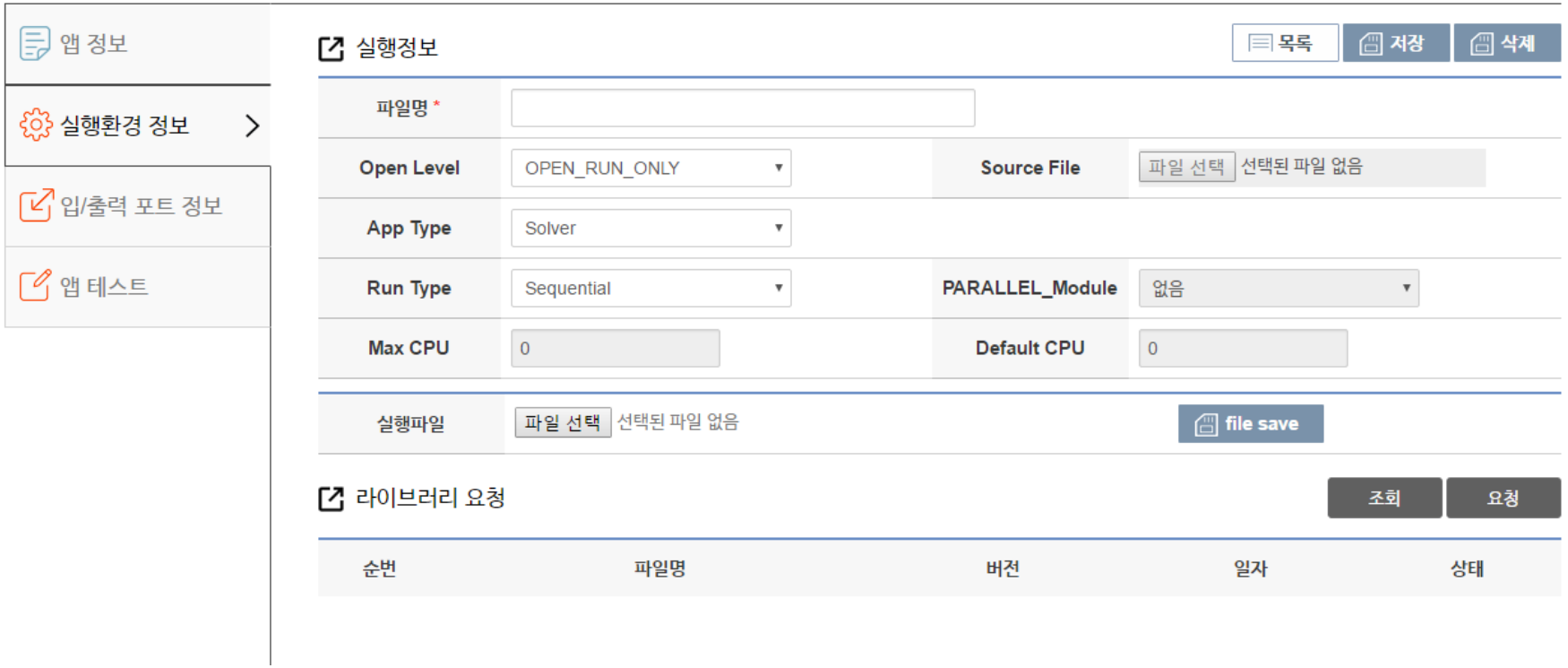
# 신규 사이언스 앱 등록 과정

- 앱 정보 입력
  - 앱 기본 정보 입력
- 실행 정보 입력
  - 실행 파일 업로드 및 실행 방식 설정
- 입출력 포트 정보 입력
  - 입출력 파일에 대한 정의
- 앱 테스트
  - 정상 동작하는지 테스트


앱 정보
실행환경 정보
입/출력 포트 정보
앱 테스트
기본정보
목록
저장
앱이름 *
※ 앱이름과 버전은 저장 후 수정 할 수 없습니다.
버전*
ex) 1.0.0
서비스언어 *
한국어 (대한민국)
앱제목 *
카테고리 *
계산화학(OPEN)
구조동역학(OPEN)
나노물리(OPEN)
전산설계(OPEN)
전산열유체(OPEN)
소유자
inoino
개발자
한 줄에 하나의 값을 입력
한국어 (대한민국)
스타일
크기
소스
설명

# 앱 정보 입력

# 실행환경 정보 입력




- 압축한 실행 파일 업로드 및 실행 파일 명 입력
- Run type에 맞게 값 설정

# 입출력 포트 정보 입력

- 입력 포트 생성시 포트 명은 입력 옵션 명 입력
- 기존에 생성된 포트 선택 가능
- 신규 포트 생성시 Editor와 Analyzer 선택 가능
- 포트에 대한 샘플 파일 등록 가능
- 입출력 포트의 포트 타입 명이 동일한 경우 Workflow에서 연동 가능
- 설정된 언어에서만 포트 정보가 출력

# Inputdeck 생성

- Inputdeck 정의를 통해 key 값과 value값 생성 규칙을 정의
  - 미리보기를 참고하며 생성
- Numeric, List , String, Vector(2~4차원) 등의 변수 생성 가능
- Group 기능 제공 : UI상에서 묶음 형태로 보여줌 -> 입력 파일에는 영향 없음

# Inputdeck 생성

- 변수 복사 기능
  - 변수 저장 이후 선택 시 변수 복사 기능 사용 가능
  - 변수명_숫자 형태로 생성
  - 뒤에 숫자만 다른 변수 생성시 이용
- 활성화 조건
  - 해당 변수가 활성화 or 값 할당되는 조건을 설정

# 출력 포트 설정

- 출력 파일을 워크플로우에 연동하거나 Analyzer로 보기 위해서는 출력 포트를 설정해야 함.
- 출력 포트가 없는 경우 모니터링 페이지에서 파일 다운로드만 가능

# 앱 테스트

- 업로드한 실행 파일과 생성된 입력포트를 기반으로 앱 테스트
- 입력 포트가 재대로 작성 되었는지 확인.
- 테스트 중간에 시뮬레이션 SW log 확인 가능
- 테스트가 성공하면 서비스 요청

# 실습 해보기