GRIB 데이터 다루기

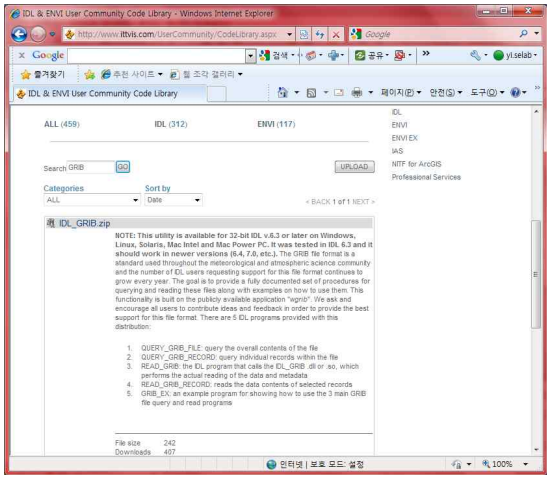
IDL

GRIB 포맷

GRIB(GRIdded Binary)는 WMO가 규격한 데이터 포맷으로, 기상 예보 자료를 저장하는 데에 많이 쓰입니다. 하나의 GRIB 파일 안에는 여러개의 레코드가 기록될 수 있으며, 각각의 레코드(배열이라고 생각하면 무난합니다)는 사실상 독립적입니다. 레코드 마다 스스로를 설명하는 헤더를 따로 가지고 있기 때문입니다.

IDL 기본 설치에는 GRIB 포맷 관련 루틴이 없습니다. 그렇지만, IDL_GRIB 이라는 패키지를 다운받아 설치함으로써 GRIB 파일을 다룰 수 있게 됩니다.

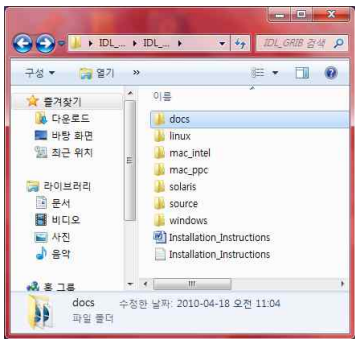
IDL_GRIB 다운로드



ITT VIS IDL 사용자 커뮤니티에 있는 Code Library 페이지. GRIB으로 검색하면 찾을 수 있습니다.

http://www.ittvis.com/UserCommunity/CodeLibrary.aspx 사이트에 들어가 Search 항목에서 GRIB으로 검색하면 관련 파일을 다운로드 받을 수 있습니다.

IDL_GRIB.zip 이라는 파일의 압축을 풀면 그림과 같은 파일/폴더 구조가 보입니다.



IDL_GRIB.zip 파일의 내용

DLM이 뭔가요?

IDL의 GRIB 라이브러리는 C로 만들어진 루틴을 IDL 명령으로 직접 Link 하게 되어 있습니다. 이러한 방식을 DLM(Dynamic Linking Module)이라고 하며 C/C++로 만든 프로그램을 IDL과 연결하는 가장 궁극적인 방법입니다. 우리가 항상 사용하는 표준 함수/프로시저 중에도 적지 않은 부분이 DLM으로 되어 있습니다.

```
IDL> read_jpeg, file_which('rose.jpg'), img
% Compiled module: FILE_WHICH.
% Loaded DLM: JPEG.
```

IDL이 처음 시작될 때에는 기본적인 기능만 적재된 아주 가벼운 상태입니다. 사용자가 필요한 모듈을 호출할 때에서야 관련 내용을 메모리에 적재합니다.

위 예문의 메시지를 보면, IDL로 만든 라이브러리(IDL 표준 라이브러리는 IDLDIR/lib에 위치)에서 FILE_WHICH를 찾아 컴파일하여 메모리에 올리고, C로 만든 DLM(IDL 표준 라이브러리는 IDLDIR/bin 의 하위 디렉토리에 있습니다) JPEG 모듈을 메모리에 적재하고 있습니다.

DLM은 두 개의 파일로 이루어져 있습니다.

- 실행파일 : 독립적으로 실행할 수는 없지만 외부에서 필요시에 호출할 수 있는 형태. 유닉스 계열은 확장자 .so, 윈도우 계열은 확장자 .dll. 일반적으로 C/C++ 코드를 컴파일하여 생성된 결과물.
- DLM 파일 : IDL이 .dll이나 .so를 호출할 때 참조할 내용. 아스키 파일이어서 메모장 등으로 열어볼 수 있음.

이 둘은 확장자만 다르고 파일 이름은 같으며, 항상 같은 디렉토리에 존재해야 합니다(예를 들면, idl_grib.dll 과 idl_grib.dlm 한쌍).

IDL_GRIB 설치

IDL GRIB은 32bit C 코드로 작성되어 있어 IDL이 32bit 모드로 실행되어야 사용 가능합니다. 64비트 운영체제에 IDL을 설치하였더라도 -32 옵션을 주어 32비트 모드로 실행하는 것이 가능합니다. 설치는 간단합니다.

1. 자신의 시스템에 맞는 디렉토리에서 .so 또는 .dll과 .dlm 한쌍을 다음 디렉토리에 복사해 넣습니다.
   윈도우 : IDL 설치 디렉토리/bin/bin.x86
   유닉스계열 : IDL 설치 디렉토리/bin/bin.os종류.x86
2. source 디렉토리에는 실제 IDL 사용자가 쓰게 될 다섯 개의 프로그램이 있습니다. 이 파일들을 IDL 경로 상에 옮겨 놓습니다.
3. IDL을 다시 시동하세요. 설치가 완료되었습니다.

GRIB_EX

grib_ex.pro는 IDL GRIB 루틴들을 어떻게 쓰는지 보여주는 예제 코드입니다. 일단 실행을 시키면 파일을 하나 고르라는 창이 뜹니다. 가지고 있는 GRIB 파일을 하나 선택하여 보세요.

이 프로시저는 GRIB 루틴들의 사용 예제이므로 소스코

드를 자세히 한번 확인하는 것이 좋습니다. 현재 코드는 Quantity 항목이 tmp(온도)로 매칭되는 모든 레코드를 열어 iImage로 연속하여 보여줍니다.
READ_GRIB_RECORD 함수를 이용하여 각각의 레코드를 어떻게 읽는지 확인할 수 있습니다(정말 간단합니다).
아래 설명될 QUERY_GRIB_FILE, QUERY_GRIB_RECORD 함수의 사용예도 확인할 수 있습니다.
records=[1,3,5,7] 이나 records=[1,5,8,17]과 같은 부분이 있습니다. 가지고 있는 GRIB 데이터에 17개 이상의 레코드가 있는지 확인하고, 만일 그렇지 않다면 이 배열의 값을 수정하여 실행해 보세요.

READ_GRIB_RECORD

아마도 최종적으로 가장 많이 쓰이게 될 함수는 READ_GRIB_RECORD일 것입니다. 다음과 같이 씁니다.

```
file='C:/Users/yi/Desktop/sample.grb'
data=read_grib_record(file, [1,2,3])
help, data
        DATA   FLOAT = Array[192, 97, 3]
```

매우 간단합니다. GRIB 파일에서 [1,2,3] 번 3개의 레코드를 읽었고, 그 결과는 3개의 층을 가지는 3차원 배열이 됩니다. 3번 레코드에 해당하는 데이터를 iImage를 이용하여 디스플레이 하는 것은 다음과 같습니다.

```
iImage, data[*,*,2]     ;배열 번호는 0부터 시작하므로
```

만일, GRIB 파일에 포함된 레코드 개수가 93개이고 모든 레코드를 읽고자 한다면 다음과 같이 쓰면 됩니다.

```
data=read_grib_record(file, indgen(93)+1)
help, data
        DATA   FLOAT = Array[192, 97, 93]
```

IDL 배열 번호는 0번부터 시작하지만, read_grib_record의 레코드 번호는 1번부터 시작한다는 것을 기억해 두십시오(그래서 indgen(93) 더하기 1입니다).

QUERY_GRIB_FILE

```
result=query_grib_file(file, info)
Number of Parameters in File:          93
```

실행시에 파일 조회가 성공적인 경우는 1, 실패인 경우는 -1이 리턴됩니다. 실제 파일에 대한 정보는 두 번째 인자(위 예에서는 info 변수)에 저장됩니다. 물론 이는 구조체입니다.
위와 같이 사용하면 info 변수에 중요한 몇 가지의 정보만 받아 오지만, /FULL 키워드를 사용하면 모든 정보를 다 읽을 수 있습니다.

QUERY_GRIB_RECORD

QUERY_GRIB_FILE과 유사합니다. 모든 레코드를 조회하지 않고 지정한 레코드의 메타데이터만 읽어옵니다.

```
result=query_grib_record(file, info, [1,2])
help, info[0], /struct
** Structure <dc9cce0>, 2 tags, length=284, data...
   INFO      STRUCT    -> <Anonymous> Array[1]
   PROJECTION  STRUCT    -> <Anonymous> Array[1]
```

메타데이터의 필드가 또 다시 구조체이므로 계속해서 다음과 같이 파헤쳐 볼 수 있습니다.

```
help, info[0].info, /struct
help, info[0].projection, /struct
```

READ_GRIB

READ_GRIB 함수는 앞서 설명된 루틴들이 하는 모든 일을 수행할 수 있습니다. 하지만 사용자 친화적이라고 보기는 어렵습니다. 벌써 리턴값부터 포인터 배열입니다.

```
pdata=read_grib(file)
help, pdata
        PDATA   POINTER  = Array[93]
```

포인터 배열의 각 요소는 구조체이며 여기에는 IMAGE 필드(레코드 배열 데이터)와 HEADER 필드(레코드 메타데이터)가 들어 있습니다.

```
help, *pdata[0], /struct
** Structure <b030f00>, 2 tags, length=74780,...
   IMAGE       FLOAT     Array[192, 97]
   HEADER      STRUCT    -> <Anonymous> Array[1]
help, (*pdata[0]).header, /struct
** Structure <b030dd8>, 2 tags, length=284, data...
   INFO        STRUCT    -> <Anonymous> Array[1]
   PROJECTION  STRUCT    -> <Anonymous> Array[1]
```

READ_GRIB은 다음 키워드를 사용할 수 있습니다.

- RECORDS : 특정 레코드만 읽고자 할 경우에 사용합니다.
- /HEADER_ONLY : 헤더만 읽습니다.
- /DATA_ONLY : 데이터만 읽습니다.
- MISSING : missing 값을 지정할 수 있습니다.

앞서 소개한 READ_GRIB_RECORD, QUERY_GRIB_FILE, QUERY_GRIB_RECORD 소스코드를 열어보면 내부적으로는 모두 READ_GRIB을 쓰고 있다는 것을 확인할 수 있습니다. 하지만 READ_GRIB은 포인터/구조체의 형태로 데이터를 받기 때문에 일반 사용자들에게는 좀 거추장스러운 면이 있습니다. 그래서 사용자 친화적인 루틴들로 복잡한 내용을 감추고 있는 것이며 사용자들은 굳이 READ_GRIB을 쓰지 않아도 됩니다.

(예제) 파일의 내용 조사하기.

```
result=query_grib_file(file, info)
n=n_elements(info)       ;레코드 개수 확인
for i=0, n-1 do print, i+1,' ', $
        info[i].quantity,':', info[i].gridid
```

```
  1 PRMSL:255 MSL
  2 UGRD:255 10 m above gnd
  3 VGRD:255 10 m above gnd
  4 TMP:255 2 m above gnd
  5 DEPR:255 2 m above gnd
  ....
```