

# Offline-блок ССМ 2.0

В. В. Калмыков

2012-2014

# 1 Введение

Данный блок предназначен для однопроцессорной довычислительной подготовки модели к запуску: построения сеток, файлов начальных условий и матриц интерполяционных весов. Более того, блок позволяет строить сетки моделей в унифицированном SCRIP-формате, благодаря чему данные могут быть обработаны многочисленными командами пакета CDO (например, интерполяционными).

## 2 Структура

Работа блока представляет вызовы различных процедур некоторого класса, который умеет хранить и обрабатывать данные сетки модели. Основная его часть реализована в базовом классе `GridConstructor`, который скрывает от пользователя внутренние процедуры работы с данными и представляет некоторые полезные функции (описание API приведено далее). Очевидно, базовый класс, написанный для общего случая, ничего не знает про конкретные сетки модели, поэтому некоторые методы (например, задания сетки) определены как *абстрактные*. Это значит, что для завершения класса и работы с блоком пользователь должен заполнить пробелы своими процедурами.

Сейчас, абстрактных метода три: построения сетки, добавления маски и добавления произвольных начальных условий в стартовый файл. Разделение первых двух функций обусловлено тем, что, например, для океана требуется сначала построить модельную сетку, переинтерполировать на нее топографию дна ЕТОРО и только потом, проанализировав данный файл и проведя чистку глубин, вычислить маску. От пользователя требуется заполнить поля экземпляра своего производного `center_lat`, `center_lon`, `corner_lat`, `corner_lon`, `imask` за счет вызова своих функций построения сеток. Методы добавления маски и начальных условий теоретически могут быть оставлены пустыми.

Например, в папке океана создаем файл `off_module.f90` в котором содержится модуль с производным классом `offline`-блока океана `off_ocn` (неважные строки пропущены):

```
module off_ocn_module

TYPE, EXTENDS(GridConstructor) :: off_ocn

CONTAINS

PROCEDURE :: make_grid => o_make_grid
```

```

PROCEDURE :: make_mask => o_make_mask
PROCEDURE :: make_ic => o_make_ic

END TYPE off_ocn

CONTAINS

! =====

subroutine o_make_grid(this)

call o_ini_set_grid

grid % z(1:grid % nz) = z(1:grid % nz)

do j = 2, jl-1
do i = 2, il-1
    grid % center_lat(i-1,j-1) = x2_t(i,j)
    grid % corner_lat(1,i-1,j-1) = x2_v(i,j)
    grid % corner_lat(2,i-1,j-1) = x2_v(i-1,j)
    grid % corner_lat(3,i-1,j-1) = x2_v(i-1,j-1)
    grid % corner_lat(4,i-1,j-1) = x2_v(i,j-1)

    grid % center_lon(i-1,j-1) = x1_t(i,j)
    grid % corner_lon(1,i-1,j-1) = x1_v(i,j)
    grid % corner_lon(2,i-1,j-1) = x1_v(i-1,j)
    grid % corner_lon(3,i-1,j-1) = x1_v(i-1,j-1)
    grid % corner_lon(4,i-1,j-1) = x1_v(i,j-1)
end do
end do

```

Правила именования остаются прежними: имена модуля и производного класса фиксированы, теперь с префиксом `off_`. Для работы с `off`-блоком, библиотека вашей компоненты должна содержать производный класс, наследующий базовый класс `GridConstructor`.

### 3 Команды блока

Работу блока регулирует скрипт `make` в папке `/off`:

```
bash make <команда> <имя компоненты> <аргументы>
```

Он попытается зайти в папку компоненты и подключить ее библиотеку к главному файлу программы `off_main`. Все необходимые подключения модулей и создание

экземпляра компоненты выполнится автоматически (для этого и требуются соглашения в именовании). Таким образом, и подготовительные работы с сеткой не требуют от пользователя изменений вне его папки – все необходимые действия контролируются за счет собственной реализации абстрактных интерфейсов.

Вся работа блока выполняется над объектом класса, наследуемого от `GridConstructor`. Переданные из командной строки аргументы и тип сетки записываются в поля `nx`, `ny`, `nz`, `gType` - их можно использовать.

На данный момент доступны следующие команды:

- **grd** – создание файла сетки и попытка добавления к нему маски. Вызываются методы `make_grid`, `make_mask`. Если маска пустая, пользователь получает предупреждение. Процедура дополняет объект полями `center_lat`, `center_lon`, `corner_lat`, `corner_lon`, `imask`.

```
bash make grd ocn 3600 1800 50 T
```

- **ics** – чтение входных файлов, их анализ и обработка, и добавление данных в стартовый файл. Вызываются методы `make_grid`, `make_mask`, `make_ic`.

```
bash make ics ocn 3600 1800 50 T
```

- **wts** – построение интерполяционных коэффициентов между двумя модельными сетками с помощью пакета CDO и SCRIP. Доступны все методы построения, определенные в SCRIP, например, консервативный, билинейный, бикубический, и т.д.

```
bash make wts bil ./data/src_cdo.nc ./data/dst_cdo.nc
```

- **int** – CDO-интерполяция двумерных и трехмерных полей. Необходима, например, для создания файлов начальных условий Левитуса. Данная опция пока работает в тестовом режиме.

```
bash make int ./data/ETOP05.nc topo ./data/OCN_720x360_V_cdo.nc.
```

## 4 Интерфейсы базового класса

Приведенные ниже методы наследуются от системного модуля и помогают пользователю работать с файловой системой полями, сетки которых определены объектами класса `GridConstructor`.

- `off_get_var()`

Считывает переменную из netCDF-файла.

#### Входные параметры:

`character(*) :: var`: имя переменной

`character(*) :: filename`: имя файла

`real(kind=4) :: arr_1D(:), arr_2D(:, :), arr_3D(:, :, :)`: необязательные аргументы, в которые будет считана переменная.

`real(kind=8) :: arr_2D_8(:, :), arr_3D_8(:, :, :)`: необязательные аргументы, в которые будет считана переменная.

`real(kind=4) :: misVal`: необязательный аргумент, при наличии которого будет попытка считывания потерянного значения (атрибут “\_FillValue”).

#### Пример использования:

Считывание файла топографии в функции построения маски океана:

```
call this % get_var(filename, "topo",
                   arr_2D = hv(iwest(myrank):ieast(myrank),
                               jsouth(myrank):jnorth(myrank)))
```

- `off_put_var()`

Записывает поле в netCDF-файл.

#### Входные параметры:

`character(*) :: var, long_name, units`: имя, расширенное имя и единицы измерения переменной

`character(*) :: filename`: имя файла

`real(kind=4) :: arr_1D(:), arr_2D(:, :), arr_3D(:, :, :)`: необязательные аргументы, из которых будет считана переменная.

`real(kind=8) :: arr_2D_8(:, :), arr_3D_8(:, :, :)`: необязательные аргументы, из которых будет считана переменная.

`character(*) :: filename`: необязательный аргумент, обозначающий файл, в который будут записаны данные (по умолчанию используется IC-файл).

#### Пример использования:

Добавление в IC-файл океана массива топографии `kv`:

```
call this % put_var(var = "kv", long_name = "kv-array",  
    units = "unitless",  
    arr_2D = REAL(kv(iwest(myrank):ieast(myrank),  
        jsouth(myrank):jnorth(myrank))))
```

## 5 Замечания

Работа блока требует установленного пакета CDO. Скрипт установки и инструкции находятся в удаленном git-репозитории. Для быстрого запуска на домашней машине можно обойтись упрощенной установкой:

```
sudo apt-get install cdo
```