RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

METEO FRANCE

Field API

Judicaël Grasset, MétéoFrance
5th All Staff Workshop, 31 March - 4 April 2025, Zalakaros & hybrid

What is field API ?

Field API is a library developped jointly by Météo-France and ECMWF.

The aim of the library is to ease the porting of the code on GPU by providing a light abstraction over the directives used to transfer the data to and from the GPU and abstract the set of directives used (OpenACC, OpenMP, CUDA).

Made in object-oriented fortran, with a high dose of fypp.

Field API Introduction

There are two fundamentals derived type in field api:

- The field owner
- The field wrapper

The field owner is designed to handle data entirely with field api

The field wrapper is designed to handle data that needs to be encapsulated with field api but that are declared and initialised outside of field api reach.

Field owner creation example

```
USE FIELD_MODULE
USE FIELD_FACTORY_MODULE
CLASS(FIELD_2RB), POINTER :: FO => NULL()
CALL FIELD_NEW(FW, LBOUNDS=[1,1],UBOUNDS=[N,M], INIT_VALUE=7_JPRB)
! Use it here
CALL FIELD_DELETE(FO)
```

Field wrapper creation example

```
INTEGER :: MY_DATA(:,:)
ALLOCATE(MY_DATA(N,M)

! Lots of compute code here
!
! Now in another subroutine, that received MY_DATA

USE FIELD_MODULE
USE FIELD_FACTORY_MODULE
CLASS(FIELD_2RB), POINTER :: FW => NULL()
CALL FIELD_NEW(FW, DATA=MY_DATA)
! Use it here
CALL FIELD_DELETE(FO)
```

Field API, usage example

```
CLASS(FIELD_3RB) :: F => NULL()
REAL(KIND=JPRB) :: PTR(:,:,:)
CALL FIELD_NEW(F, UBOUNDS=[NPROMA,NFLEVG,NGPBLKS])
! Get a pointer on host's data (cpu) and do some work with it
CALL F%GET_HOST_DATA_RDWR(PTR)
PTR=A+B
! Get a pointer on device's data (gpu) and work with it on the gpu
! Since we have asked for a read-write pointer on the cpu data before, field api will copy the
data to the gpu before handling a pointer to the user.
CALL F%GET_DEVICE_DATA_RDWR(PTR)
!$ACC KERNELS
DO 1,10
!COMPUTE STUFF WITH PTR
ENDDO
!$ACC END KERNELS
```

Field API misc.

- The user must not forget to use the GET_HOST_* and GET_DEVICE_* subroutines before any computations, otherwise field api won’t know if it needs to transfer the data between the CPU and the GPU.
- The user must not forget to call FIELD_DELETE to clean the data on CPU and GPU.
- Debugging helper with module FIELD_INIT_DEBUG_VALUE_MODULE
- Statistics on the number of transfer and transfer time

Field API limitation and future

Only works on GPU with OpenACC and the Nvidia compiler

We are now looking to make it work with OpenMP and the AMD compiler

Refactoring with field API, YOMRADF

State of type TRADF (yomradf module) as of CY50 :

- Defined in arpifs/module/yomradf.F90
- Contains multiple allocatable arrays
- Initialisation code is in arpifs/phys_radi/suecradf.F90
- Deletion code is in arpifs/utility/dealmod.F90
- No Field API

Refactoring with field API, YOMRADF

Usage of type TRADF (yomradf module) as of CY50 :

- TRADF is contained in MODEL_PHYSICS_RADIATION_TYPE, itself contained in MODEL
- Field API encapsulation is declared in RADIATION_VARIABLES (field_variables_mod.fypp), contained in FIELD_VARIABLES
- Field API initialisation is in field_registry_mod.fypp
- Additional pointers to the field_api encapsulation declared and initialized in AUX_RAD_TYPE (ecphys_aux_rad_type_mod.F90)

Refactoring with field API, YOMRADF

After refactoring, in the future CY50T1 :

- TRADF is defined in arpifs/module/yomradf.fypp
- Small additions of fypp to remove some verbosity
- Added Field API to replace allocatable arrays
- Added constructor and destructor method to the TRADF type
- Those methods are still called in suecrad and dealmod but the code is located in yomradf.fypp
- Added Field API helper method (update_view, copy)
- Additional pointers to the Field API encapsulation are still declared and initialised in AUX_RAD_TYPE (ecphys_aux_rad_type_mod.F90)
- **Moved TRADF from MODEL to FIELDS**

Refactoring example – YOMRADF

To look at this work in more details in the IAL github repository: see PR 247 (YOMRADF) and PR 300 (YOMTRC).

Ideally all data arrays where the first dimension is NPROMA (mostly in YDFIELDS, some in YDMODEL) and all data related to direct forecasting would be encapsulated directly with field API.

```
!Before refactoring
TYPE :: TTRC
REAL(KIND=JPRB),ALLOCATABLE:: GRSURF  (:,:)
REAL(KIND=JPRB),ALLOCATABLE:: GDEOTI  (:,:,:)
…
END TYPE
```

```
!After refactoring
TYPE :: TTRC
CLASS(FIELD_2RB), POINTER :: F_GRSURF => NULL()
REAL(KIND=JPRB),  POINTER :: GRSURF(:) => NULL()
CLASS(FIELD_3RB), POINTER :: F_GDEOTI => NULL()
REAL(KIND=JPRB),  POINTER :: GDEOTI(:,:) => NULL()

…
END TYPE
```

Field API - Links

The official repository: https://github.com/ecmwf-ifs/field_api

There is some documentations, but you can also look at the test-cases to see simple usage example:
https://github.com/ecmwf-ifs/field_api/tree/main/tests

If you try field API and notice compilation failure or strange behaviour, please open a bug report in the GitHub repository.

You can ask me questions about field API: judicael.grasset@meteo.fr

If you are interested in helping with the refactoring you can contact Philippe Marguinaud at philippe.marguinaud@meteo.fr