Commented example – Meteogram

Table of total precipitation
3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

bettems@kesch: fieldextra control_file

fieldextra is the fortran program executable
control_file contains the namelist defining the program behaviour
Commented example – Meteogram

Table of total precipitation
3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

```
... HEADER ...

&Process in_file = "/support/input/cosmo-e/000/lfff<DDHH>0000"
 tstart = 3, tstop = 9, tincr = 3,
 out_file = "/support/results/meteogram_precipitation.txt"
 out_type = "METEOG", out_type_fmt = "f71_dh_prec",
 out_type_text1 = "Precipitation rain+snow in the last 3 hours mm : mean over 5 gridpoints"
 locgroup = "nat"
 loclist = "GVE", "DOL", "FRE", "NEU", "CDF", "CHA", "CGI", "PUY", "PAY" /

&Process in_field="RAIN_GSP", hoper = "c5", toper = "delta, 3, hour", toper_mask = "lead_time>3"/
&Process in_field="RAIN_CON", hoper = "c5", toper = "delta, 3, hour", toper_mask = "lead_time>3"/
&Process in_field="SNOW_GSP", hoper = "c5", toper = "delta, 3, hour", toper_mask = "lead_time>3"/
&Process in_field="SNOW_CON", hoper = "c5", toper = "delta, 3, hour", toper_mask = "lead_time>3"/

&Process out_field="TOT_PREC" /
```

Available in ./cookbook/meteogram_precipitation.nl
Commented example – Meteogram

Table of total precipitation
3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

Define input and output characteristics, define domain subset

```plaintext
&Process
  in_file = "./support/input/cosmo-e/000/lfff<DDHH>0000"
  tstart = 3, tstop = 9, tincr = 3,
  out_file = "./support/results/meteogram_precipitation.txt"
  out_type = "METEOG", out_type_fmt = "f71_dh_prec",
  out_type_text1 = "Precipitation rain+snow in the last 3 hours mm : mean over 5 gridpoints"
  locgroup = "nat"
  loclist = "GVE", "DOL", "FRE", "NEU", "CDF", "CHA", "CGI", "PUY", "PAY" /

&Process in_field="RAIN_GSP", hoper = "c5", toper = "delta,3,hour", toper_mask = "lead_time>3"/
&Process in_field="RAIN_CON", hoper = "c5", toper = "delta,3,hour", toper_mask = "lead_time>3"/
&Process in_field="SNOW_GSP", hoper = "c5", toper = "delta,3,hour", toper_mask = "lead_time>3"/
&Process in_field="SNOW_CON", hoper = "c5", toper = "delta,3,hour", toper_mask = "lead_time>3"/

&Process out_field="TOT_PREC" /
```
Commented example – Meteogram

Table of total precipitation
3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

&Process
  in_file = "./support/input/cosmo-e/000/lff<DDHH>0000"
  tstart  = 3,  tstop = 9,  tincr = 3,
  out_file = "./support/results/meteogram_precipitation.txt"
  out_type = "METEOG",  out_type_fmt = "f71_dh_prec",
  out_type_text1 = "Precipitation rain+snow in the last 3 hours mm : mean over 5 gridpoints"
  locgroup = "nat"
  loclist  = "GVE", "DOL", "FRE", "NEU", "CDF", "CHA", "CGI", "PUY", "PAY" /

Define fields to collect

&Process in_field="RAIN_GSP", hoper = "c5", toper = "delta,3,hour", toper_mask = "lead_time>3"/
&Process in_field="RAIN_CON", hoper = "c5", toper = "delta,3,hour", toper_mask = "lead_time>3"/
&Process in_field="SNOW_GSP", hoper = "c5", toper = "delta,3,hour", toper_mask = "lead_time>3"/
&Process in_field="SNOW_CON", hoper = "c5", toper = "delta,3,hour", toper_mask = "lead_time>3"/

&Process out_field="TOTPREC" /
Commented example – Meteogram

Table of total precipitation
3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

&Process
  in_file = "./support/input/cosmo-e/000/lfff<DDHH>0000"
  tstart  = 3,  tstop  = 9,  tincr  = 3,
  out_file = "./support/results/meteogram_precipitation.txt"
  out_type = "METEOG",  out_type_fmt = "f71_dh_prec",
  out_type_text1 = "Precipitation rain+snow in the last 3 hours mm : mean over 5 gridpoints"
  locgroup = "nat"
  loclist  = "GVE", "DOL", "FRE", "NEU", "CDF", "CHA", "CGI", "PUY", "PAY" /

&Process in_field="RAIN_GSP", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="RAIN_CON", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="SNOW_GSP", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="SNOW_CON", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/

&Process out_field="TOT_PREC" /

Define operations to apply on collected fields
(large choice of operators available)
Commented example – Meteogram

Table of total precipitation
3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

&Process
in_file = "/support/input/cosmo-e/000/lfff<DDHH>0000"
tstart = 3, tstop = 9, tincr = 3,
out_file = "/support/results/meteogram_precipitation.txt"
out_type = "METEOG", out_type_fmt = "f71_dh_prec",
out_type_text1 = "Precipitation rain+snow in the last 3 hours mm : mean over 5 gridpoints"
locgroup = "nat"
loclist = "GVE", "DOL", "FRE", "NEU", "CDF", "CHA", "CGI", "PUY", "PAY" /

&Process in_field="RAIN_GSP", hoper = "c5", toper = "delta,3,hour",
toper_mask = "lead_time>3"/
&Process in_field="RAIN_CON", hoper = "c5", toper = "delta,3,hour",
toper_mask = "lead_time>3"/
&Process in_field="SNOW_GSP", hoper = "c5", toper = "delta,3,hour",
toper_mask = "lead_time>3"/
&Process in_field="SNOW_CON", hoper = "c5", toper = "delta,3,hour",
toper_mask = "lead_time>3"/

&Process out_field="TOT_PREC" /
Define fields to compute
(refers to a fieldextra procedure, easily extensible)
Commented example – Meteogram

**Table of total precipitation**
3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

- **control_file** contains the namelist defining the program behaviour

  ```
  header
  &RunSpecification
  &GlobalResource
  &GlobalSettings
  &ModelSpecification

  product definition &Process (repeated)
  ```

- **external resources**

  ```
  definition of field names dictionary in &GlobalResource
  definition of locations location_list in &GlobalResource
  ```
Commented example – Meteogram

Table of total precipitation
3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

• program diagnostic and profiling

standard error & output
file fieldextra.diagnostic

controled by the values of verbosity and additional_diagnostic

verbosity = ‘high’ in &RunSpecification
additional_diagnostic = .true. in &RunSpecification
additional_profiling = .true. in &RunSpecification
Design – Input & Output

- **INCORE storage** used to store resources for common operations (see next slide).

- Each **input** is read **once**.
- Storage is allocated **on demand** for each **output**.
- Each input record is evaluated, and dispatched in the corresponding output storage when requested (in memory).

→ **io optimization at the cost of memory usage!**

- When all data for some output have been collected, the corresponding data is written on disk and the memory is released.
- For output supporting append mode, data is processed piecewise after reading each related validation time (**just on time** mode).
Design – Incore storage

- **INCORE** global persistent storage is used to:
  - associate grid points to specified *locations & regions*
  - produce grid point *height information* for some output
  - specify model *base grid* when working with staggered fields, or fields defined on a larger domain
  - specify grid for *re-gridding*
  - *merge* and *compare* different fields
  - provide access to *programmatically derived* constant fields (see below)

- **Programmatically derived constant fields** will be available from INCORE storage when HSURF is present:
  - RLAT, Rlon (geog. latitude, longitude [deg])
  - SWISS_WE / SWISS_SN (swiss coord. [m])
  - BOAGAW_WE / BOAGAW_SN (Gauss-Boaga coord., west sector [m])
  - BOAGAE_WE / BOAGAE_SN (Gauss-Boaga coord., east sector [m])
  - HHL / HFL (COSMO height of model levels [m])
Design – Iterative processing (1)

\[ F_{11}(\Psi) : 1 \text{ to } 1 \text{ field operator} \]
\[ F_{n1}(\Psi) : n \text{ to } 1 \text{ field operator} \]
\[ F_{nm}(\Psi) : n \text{ to } m \text{ field operator} \]
Design – Iterative processing (2)

• For each output, define the set of associated input files
  • data can only be extracted from this set
  • INCORE storage can be part of this set

• **First iteration**: collect all necessary input fields
  • all fields must be unique (condition can be relaxed with out_duplicate_infield)
  • all field must be defined on a compatible grid (cropping & re-gridding are available)
  • fields required in a subsequent iteration must be collected at that stage

• **Next iterations (repeated up to 6 times)**: collect or compute fields
  • only data in previous recipient are available
  • if fields are not available, they must be computed:
    in this case the parent fields must be available in the previous recipient
  • the main parent (defined in dictionary or by the type of required operator) defines the characteristics of the produced field

• Only the last recipient is available for output generation
  • all fields available in the last recipient are written in the output file
First iteration:
Each extracted field may be transformed by one or more operators, in the order regridding (regrid), change meta-information (set_*), merge (merge_with), compare (compare_with), lateral transform (hoper), scale/offset, vertical transform (voper), temporal transform (toper), local transform (poper, poper2, poper3), spatial filter (*_filter), reset identity (new_field_id)

Next iteration(s):
Each extracted field may be transformed by one or more operators, in the order lateral transform (hoper), scale/offset, vertical transform (voper), temporal transform (toper), local transform (poper, poper2, poper3), spatial filter (*_filter), change meta-information (set_*), reset identity (new_field_id)

After last iteration:
A last set of global operations may be applied, in the order n to m operator (out_postproc_module), regridding (out_regrid_target), reset meta-info (set_* ...), filter data (out_filter_*)
This procedure …

- write a temporary GRIB file at the end of the first iteration
- use this temporary file as input for the second iteration
- dependencies are detected and files are processed in the correct order

… can be repeated once!
Design – Namelist: basic

A single namelist file to produce n output

Global settings
  • resources files
  • run specifications
  • default values

Definitions for output_1

Definitions for output_2

Definitions for output_n

Collecting fields for output_2, from input_1

in_file = input_1
  + input characteristic

out_file = output_2
  + output characteristics
  + geographical subset
  + computation mode
  + more (filter, interpolation, dictionary)

selection_mode = interpretation of in_field

in_field = infield_1,
  + associated transformations

in_field = infield_i,
  + associated transformations

*) out_file information must be repeated with each new definition of in_file

Generation of fields for output_2 last iteration

out_field = outfield_1
  + associated transformations

out_field = outfield_j
  + associated transformations
Design – Namelist : selection_mode

Typical usage:

- **PRODUCT GENERATION**
  - **INCLUDE ONLY** (default)
  - **INCLUDE ALL**
  - **EXCLUDE**

- **TRANFORMATION OF INPUT FILE**

- **MERGING 2 INPUT**

**INCLUDE ONLY**

- **Input file**
- **in_field**
- **tmp1_field**
- **out_field**
- **Output file**

**INCLUDE ALL**

- **Input file**
- **in_field**
- **tmp1_field**
- **out_field**
- **Output file**

**EXCLUDE**

- **Input file**
- **in_field**
- **tmp1_field**
- **out_field**
- **Output file**

**COMPULSORY**

**OPTIONAL**
Design – Namelist : time levels (1)

• A generic name may be used to loop over a set of input files
  • typically to process a standard set of model output, characterized by one file per validation date
  • a key is inserted in the input file name (<DDHH>, <DDHHMMSS>, <YYYYMMDDHH:initial_date> …)
  • a list of times is defined explicitly (tlist) or by an implicit loop (tstart, tstop, tincr)
    → the key is replaced in turn by each time, the same extraction pattern is applied on each input.

• Time operators may be applied on collected and generated fields
  • all collected time levels are available, and only those

• It is possible to filter the times collected in output
  • another list of times defined by an implicit loop (out_tstart, out_tstop, out_tincr) is used, when available,
    to filter the list of times defined by (tlist) or (tstart, tstop, tincr)
    → the validation dates available in output are those associated with the filtered input list
  • this filter does not influence the set of time levels available for the time operators

Example:
centered T2m mean, 6-hourly output

<table>
<thead>
<tr>
<th>&amp;Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_file=&quot;lfff&lt;DDHH&gt;0000&quot; , in_type=&quot;GRIB&quot;,</td>
</tr>
<tr>
<td>tstart=0, tstop=24, tincr=1,</td>
</tr>
<tr>
<td>out_file=&quot;product&quot; , out_type=&quot;GRIB1&quot;,</td>
</tr>
<tr>
<td>out_tstart=3, out_tincr=6 /</td>
</tr>
<tr>
<td>&amp;Process</td>
</tr>
<tr>
<td>in_field= &quot;T_2m&quot; , toper= &quot;mean,-3,3,1,hour&quot; /</td>
</tr>
</tbody>
</table>

← key in input file name
← implicit time loop
← filter output date, implicit time loop
← time operator
Instead of collecting all validation times in the same output, one file per validation time is created by using a generic name for the output file:

- a key is inserted in the output file name (<DDHH>, <DDHHMMSS>, <YYYMMDDHH:initial_date> …)
- the list of times defined by (tlist) or by (tstart, tstop, tincr) is used to set the key values
- filtering defined by (out_tstart ...) is respected

In this case, the set of input files contributing to each output must be explicitly specified!

→ use tlag (see next slide)

These mechanisms are based on the assumption that any file whose name matches ...
...<key>..., key in {DDHH, DDHHMMSS…}
contains fields valid for the same date, and that the value of <key> represents this date!

Example:

centered T2m mean, 6-hourly output, one output per date

&Process
in_field="lfff<DDHH>0000", in_type="GRIB", tstart=0, tstop=24, tincr=1, out_file="product_<DDHH>”, out_type="GRIB1", out_tstart=3, out_tincr=6, tlag=-3,3,1 /
&Process
in_field="T_2m", toper="mean,-3,3,1,hour" /

← key in output file name
← input/output association (for toper)
Explicit specification of contributing input files

- For each output file, fieldextra constructs the set of contributing input files.
- Data can only be collected from this set.
- The set of contributing input files is not univocally defined when multiple output files are defined within the same &Process group, which is the case when a time key is also used in the name of the output file.
- In this case, a one-to-one correspondence is assumed, meaning that each output has only access to a single input, i.e., a single time level (see below).
- When temporal operators requiring multiple time levels are used, the set of input files contributing to each output must be explicitly specified.
- This is done by using the namelist variable `tlag`; `tlag` defines an interval of contributing input files relative to the currently processed output (and refers to the list of times defined by `tlist` or `tstart`...)

Each output has only access to one time level!

Use `tlag` → Multiple time levels are available!
Design – Computation of new fields (1)

Meteorological operator, activated via the name of the new field [$F_{n1}(\Psi)$]

Example:
- RELHUM_2M from T_2M and TD_2M
  - &Process in_field = “TD_2M“ /
  - &Process in_field = “T_2M“ /
  - &Process out_field = “RELHUM_2M“ /

  ← parent 1
  ← parent 2 (main parent, defined in field dictionary )
  ← new field

Implementation:
- Operators of common interest
  - add new routine in fxtr_operator_generic
  - extend case statement in fxtr_operator_generic:field_compute_generic
- Operators of local interest
  - add new routine in fxtr_operator_specific
  - extend case statement in fxtr_operator_specific:field_compute_specific
Design – Computation of new fields (2)

**Named operator**, activated by setting “use_operator=…” \([F_{n1}(\Psi)]\)

### Example

- **W_SO_CST from W_SO, FR_LAND, and SOILTYP**
  - \&Process in_field = “W_SO” /
  - \&Process in_field = “FR_LAND” /
  - \&Process in_field = “SOILTYP” /
  - \&Process out_field = “W_SO_CST”,
    - use_operator = “wso_offset_v001”,
    - use_tag = “W_SO,FR_LAND,SOILTYP” /

### Implementation

**Operators of common interest**
- add new routine in fxtr_operator_generic
- extend case statement in fxtr_operator_generic:field_compute_generic
- extend module parameter fxtr_operator_generic:allowed_generic_nm_operator

**Operators of local interest**
- add new routine in fxtr_operator_specific
- extend case statement in fxtr_operator_specific:field_compute_specific
- extend module parameter fxtr_operator_specific:allowed_specific_nm_operator
Design – Computation of new fields (3)

Post-processing operator, activated by setting “out_postproc_module=…” \(F_{nm}(\Psi)\)

![Diagram showing the relationship between parents and children in the computation of new fields.]

- **Parent** 1 → **Child** 1
- **Parent** 2 → **Child** 2
- …
- **Parent** n → **Child** m

Parents defined in pp routine

Final iteration

\[
\text{out}_\text{postproc}_\text{module} = \text{"post-processing routine name"}
\]

Output file

### Example

Transform wind and cloud cover to derive MeteoSwiss forecast matrix input

```
&Process
  in_file = "iff<DDHH>0000",
  tstart = 0, tstop = 24, tincr = 1,
  loggroup = "nat", loclist = "GVE",
  out_file = "forecast_matrix.txt",
  out_type = "FLD_TABLE",
  out_postproc_module = "pp_forecast_matrix" /
...
&Process out_field = "DD10MC_AM" /
...
&Process out_field = "DRSRRG_D" /
```

← operator applies to entire output
← parent 1
...
← parent n

### Implementation

**Operators of common interest**

- add new post-processing routine in fxtr_operator_generic
- extend case statement in fxtr_operator_generic:data_postprocess_generic
- extend module parameter fxtr_operator_generic:allowed_generic_pp_procedure

**Operators of local interest**

- add new post-processing routine in fxtr_operator_specific
- extend case statement in fxtr_operator_specific:data_postprocess_specific
- extend module parameter fxtr_operator_specific:allowed_specific_pp_procedure
Design – Shared memory parallelism (1)

- Shared memory multitasking is available and implemented with OpenMP directives

- **Two levels of parallelism** are implemented and can be simultaneously used
  - parallel production of **output** (outer loop parallelism)
  - parallelization of some of the **algorithms** used during the production of each output (inner loop parallelism)

- Two (exclusive) types of **algorithm parallelization** are available
  - Grid points partitioning, in (i,j) space
  - Parallel computation when the same operator is applied on multiple records within the current iteration, and grid point partitioning is not possible

- No distributed memory parallelism
- No parallel processing of input

See the example./cookbook/multiple_products.nl
Parallel production of output (outer loop parallelism, marked with red below)

1. For each record of each input
2. Once a complete set of records is available

The following operations are applied in parallel (loop over output):

(1) For each record in turn:
   check use of current record by output, process and store record
(2) Once a complete set of records is available:
   iterative processing of fields, format and write output
Algorithm parallelization (inner loop parallelism, marked with below)

Within each processing iteration associated with each output, for each operator in turn (hoper, poper...):

- parallel computation using grid points partitioning in (i,j) space, when no halo required
- or
- parallel computation using fields partitioning, when the same transformation is applied on multiple fields
Code structure
**Modules**

**Main**

*Parse namelist*

*Driver for product generation*

*Driver for field manipulation*

*Transform field*

*Compute new field*

*Support procedures (thermo...)*

*Generate output*

**PROGRAM:**

fieldextra

**MODULES (core functionality):**

fxtr_control,

fxtr_kernel,

fxtr_operator_main,

fxtr_operator_column, fxtr_operator_regrid,

fxtr_operator_generic, fxtr_operator_specific, fxtr_operator_probability,

fxtr_operator_support,

fxtr_write_generic, fxtr_write_obsolete, fxtr_write_specific

**MODULES (program specific support):**

fxtr_definition,

fxtr_resource_dictionary, fxtr_resource_gis, fxtr_resource_stat,

fxtr_storage, fxtr_attribute, fxtr_profiling

**MODULES (generic support):**

support_grib1, support_grib2, support_netcdf, support_blk_table

support_vertical_mesh, support_icon_grid,

support_storage, support_diagnostic,

support_openmp

support_datetime, support_gis, support_math, support_misc

**MODULES (imported from COSMO):**

cosmo_data_parameters

cosmo_meteo_utilities, cosmo_pp_utilities, cosmo_utilities
Main data structure

**TYPE ty_out_store**  (→ see fxtr_definition)

Variables of this type are used as  
*main repository for fields values and meta-information associated with each output.*

- **Field values** are collected in `values(:,:,:)` array, where:
  
  first dim. is for *location* index, second dim. is for *field* index, third dim. is for *validation date* index

  A field in this context is a 2D field on a specific surface (ground, model, pressure…).
  A location is a grid point, but the set of used locations is not necessarily a rectangular domain.
  Note that the field values are stored in a 1-dimensional section of the values array (unlike the COSMO model).

  The number of locations is given by `nbr_gp`, the number of fields by `nbr_field` and the number of validation dates by `nbr_vdate`.

- **The characteristics of a field** are documented in  
  `field_id(:,), field_epsid(:,), field_pdfid(:,), field_level(:,), field_product(:,), field_trange(:,,...)`  
  (field_id(:,)%name is field name, field_id(:,)%tag is user defined tag)

  The **characteristics of a location** are documented in  
  `gp_lat(:,), gp_lon(:,), gp_coord(:,,...)`

  The **validation date** are documented in  
  `validation_date`:

  Other information, **common** to all fields of the considered output, is available in:  
  `ofile_name, grid_hcoord, grid_vcoord, ...`
Main program

0. Initialization sequence, first part.
1. Read parameters defining program behaviour.
2. Initialization sequence, second part.

[input_file_loop: DO] Loop through all input files, in sorted order. This loop is executed twice: first to collect fields for special output (INCORE, INSPECT), then to collect fields for standard output.

3. Generate output file
(just on time mode, all fields collected).
4. Skip or open input file.
   4.1 Skip input when all associated input/output pairs are inactive
   4.2 Select file to process
   4.3 Wait for file
   4.4 Process file
      4.4.1 Detect type of first record, set calling order for API
           GRIB file: DWD lib (GRIB1), ECMWF lib (GRIB2)
           NetCDF file: NETCDF lib
           BLK_TABLE file: internal API
      [loop_over_api: DO] Loop over decoding API
      4.4.2 Skip record if non matching API
      4.4.3 Open file
      4.4.4 Get global header

5. Read and decode input field.
   5.1 Clone cache (input missing)
   5.2 Standard input file.
       5.2.1 Read next record (skip data section, data will be read and decoded on request later on)
       5.2.2 Decode meta-information
       5.2.3 Process meta-information
       5.2.4 Check meta-information
      5.3 Pseudo input file __INCORE__.
      [output_file_loop: DO] Loop through all output files

6. Dispatch input field in output storage.
   6.1 Does the current field contributes to the current output?
   6.2 Unpack or generate field values
   6.3 Re-grid field when required.
   6.4 Dispatch field in output storage.
      [END DO output_file_loop, input_field_loop, loop_over_api]

7. Re-order fields in output storage
      [END DO input_field_loop]

8. Operations requiring access to special storage (INCORE, INSPECT).
9. Generate output files (last call).
10. Clean-up and final diagnostics.
Calling tree: product generation

Fieldextra

1. fxtr_control: process_namelist
2. fxtr_kernel: store_field
3. fxtr_kernel: generate_output
   3. fxtr_kernel: prepare_data
      3. fxtr_operator_main: compute_field
         3. (sc_pcat_*, field_id%name)
         3. fxtr_operator_generic: field_compute_generic
         3. fxtr_operator_specific: field_compute_specific
         3. fxtr_operator_probability: eps-derived_product, neighbourhood_probability

fxtr_write_generic
fxtr_write_specific
Iterative data processing : implementation

1. In main procedure
   + horizontal re-gridding (*in_regrid_target*)

2. In fxtr_kernel:store_field
   + modification of field meta-information
     (*set_units*, *set_reference_date* ...)
   + merge with another field (*merge_with*)
   + compare with another field (*compare_with*)
   + horizontal operator (*hoper*) and horizontal reduction of field
     (in advance of B2. when possible )

3. In fxtr_kernel:prepare_data
   A. Processing of constant fields with respect to the time dimension
   B. Iterative processing of fields
      B1.1 build extended information about generated fields
         B1.1.1 look for field in previous set
         B1.1.2 look for main parent in previous set
         B1.1.3 derive relationship between child and parent
         B1.1.4 build full list of fields to generate/extract
      B1.2 calculate new fields or copy fields from previous iteration
      B2. horizontal operator (*hoper*) and horizontal reduction of field
      B3. linear transformation (*scale*, *offset*)
      B4. transformation in a column (*voper*)
      B5. apply time operator (*toper*)
      B6. apply point operator (*poper*)
      B7. apply spatial filter (*in_filter* ...)
      B8. reset field identity (*new_field_id*)
      B9. purge data from dates with inhibited print out
   C. Non iterative field transformations
      C1. post-processing operator (*out_postproc_module*)
      C2. re-gridding operator (*out_regrid_target*) or project on user specified domain (*slice* ...)
      C3. programatic setting of some local meta-information
      C4. reset field meta-information with user specified values
          (*set_units*, *set_reference_date* ...)
      C5. Check consistency of meta-information
   D. Prepare data for print out
      D1. filter out constant fields when requested
      D2. purge data from dates with inhibited print out
      D3. derive information common to all fields
      D4. update additional elements of data storage

TYPE(ty_out_store) ::

Fieldextra 13.0.0
User specific output format
Module fxtr_write_specific

SUBROUTINE write_specific
• ! 0.2 Decision about format

SUBROUTINE write_test
• ! ... List of expected fields
• ! 0.1 Initialize variables
• ! 0.2 Check usage of explicit locations
• ! 0.3 Check that data_pout content is supported by output type
• ! 0.4 Open output file
• ! 1. Look for data
• ! 2. Write header
• ! 3. Write data

Try to understand all details of the write_test routine, clone this routine to produce your preferred format (TEST2)!
Some typical applications
Some typical applications
Pre-processing

- **Merge** surface temperature from IFS over sea and from COSMO over land to produce a single field suited for the assimilation cycle
- **Interpolate** Swiss radar composite from kilometric grid onto the COSMO-2 grid for feeding the latent heat nudging process
- **Rebuild EPS members** by adding EPS perturbation to a more recent determinist forecast
Some typical applications
Post-processing

- **Meteograms** (as text) at specified locations
- **Cropping** and **regridding** for user specific grids
- **Data thinning** of model output for verification purposes
- Computation of **geostrophic** wind and related quantities
- **Interpolation** of wind field on specified theta and PV surfaces
- **Split** file with multiple EPS members or validation dates in pieces
- **Fill holes** in a 2-dimensional field not defined everywhere (e.g. HZEROCL)
- Mix multiple model output in a single XML file for **seamless forecast**
- **Convert** GRIB1 to GRIB2, incl. the specification of generalized height based vertical coord.
- **Kalman** correction at selected locations
- **MOS** based estimation of fields (including fields not part of model output!)
  - *Fieldextra expects the coefficients of the statistical model as external resource*
  - *Statistical filter computation is done outside of fieldextra!*

- Generation of **EPS products**
- Generate **lagged ensemble** from COSMO-2 rapid update cycle
- **Clone** missing member in COSMO-E output by changing member id

- **Real time production**: wait for model output, produce partial output every dH hours
Some typical applications

More complex products

• Generate a soil type dependent field offset and apply it to correct W_SO
• Find 3D location of points where some conditions are fulfilled (e.g. over-saturation over ice and temperature above -20C)
• Compute spatially upscaled EPS probability
• Create a bitmap for the condition ‘probability of 6h sum of total precipitation exceeding 25mm is larger than 0’
• **Warn product** : compute region based quantiles of some fields under side conditions (e.g. 50% quantile of wind gust for all points below 800m where T_2m below 0C)
• **Freezing rain** : compute the integral of the temperature between the two lowest 0C isotherm in case of an inversion over a cold air pool
• **CAT for aviation** : compute indicators, find the height-surface of maximum CAT, compute the CAT category (low, medium, high) on this surface
• **FABEC product** : compute air density on a set of pressure and height above ground levels, interpolated on a geographical lat/lon grid, in GRIB 2
• **Monitoring** of model output : field values statistics, when values are outside of pre-defined validity range
Access, installation and usage
Access

- **Licenced software**
  - free to all COSMO members.
  - free licences for the R&D community, but without support.

- **Access**
  - [Master code repository on GitHub](https://github.com/MeteoSwiss-APN/fieldextra) (private repository)
  - [Package on COSMO web site](http://www.cosmo-model.org/content/support/software/default.htm)
  - **Full installation at ECMWF on cca** (UNIX group cfxtra) /perm/ms/ch/ch7/projects/fieldextra
  - **Full installation at CSCS on kesch** (UNIX group s83) /users/tsm/project_escha/fieldextra
Access

Package on COSMO site

- **Tar file on COSMO web site, password protected**
  
  http://www.cosmo-model.org/content/support/software/default.htm

  - Source code for libraries (incl. config. script)
  - Source code for fieldextra
  - All necessary Makefiles (for gfortran, ifortran …)
  - All necessary resources (dictionary, location list …)
  - Documentation (admin, compatibility, documentation)
  - Cookbook (used to validate installation)
  - Tools (including fx tools)

*Fieldextra is only validated against the libraries and the associated resources included in the distribution package!*
Installation

• **Follow steps in ./admin/INSTALLATION**
  • How to install, compile and test the code
  • Three cases are considered:
    • *Using GitHub* *(requires access to private repository)*
    • *Using package* from COSMO site
    • *Using another existing installation*

• **New and modified features are documented in ./admin/HISTORY**

• **Backward compatibility issues are documented in ./compatibility files**
Usage

- ./cookbook
  - Commented examples, the best way for learning how to use fieldextra
- ./documentation/FAQ
  - Frequently asked questions
- ./documentation/README.user (and README.user.locale)
  - Comprehensive description of functionalities
- ./documentation/README.grib_issues
  - Summary of issues with GRIB 1 / 2 standards and with GRIB API
- ./admin/GRIB_POLICY
  - GRIB 2 common COSMO policy
Things never work as planned ...

- **Problem by installation?**
  - carefully read and follow `./admin/INSTALLATION`
  - look at `./documentation/FAQ`

- **Namelist not working with newer release?**
  - consider documents in `./compatibility`

- **Problem by usage?**
  - set **verbosity** to *high* (or *debug*) and **additional_diagnostic** to *true*
  - look at `./documentation/FAQ`

- **Do not know how to write the namelist for some application?**
  - get inspired by the **cookbook** examples

- **Get community support at** [fieldextra@cosmo-model.org](mailto:fieldextra@cosmo-model.org)
Roadmap
What shall I expect next?

*Release x.x*

See

https://github.com/MeteoSwiss-APN/fieldextra/milestones
Possible major developments

See

https://github.com/MeteoSwiss-APN/fieldextra-wiki/wiki/Planning

E.g.

• Full support of **ICON grid** (computations on the native grid)
• Support **parallel input**
• Support distributed memory parallelism (**MPI**) 
• Use of **GPU** to accelerate parts of the code
Final discussion
Final discussion

Topics …
Thank you for your attention!