

# A Graphical User Interface for configuring YAC

Maxim Yastremsky and René Redler

*Max-Planck-Institut für Meteorologie, Hamburg*

Aug 2015

---

We introduce the usage of the graphical user interface to generate a coupling configuration XML file for YAC. We explain the necessary steps to define the coupling between a pair of model components and describe the options we offer to configure a particular interpolation method. In the appendix we provide an example of the XML files which are required for input and explain some elements of the resulting XML file generated by the GUI.

## 1. Starting the GUI

The java source code for the GUI is shipped together with the source code of the coupler library. A precompiled archive (jar) file is available as well to start right away. User who would like to compile a jar file on their own require ant.

Once, ant is available the whole java project is built within the gui directory using

```
ant build
```

and the java library is generated by typing

```
ant create_run_jar
```

In its current version 1.0.4 the GUI requires a XML description of model components as input to allow for the configuration of the coupling. An example of a component XML file is provided in Appendix A. Compared to the coupling XML file (an example is provided in Appendix B) a component XML file has a very simple structure and can easily (and probably much faster) be generated with a simple text editor.

To assist users in constructing the coupling XML file the YAC GUI can be launched via command line

```
java -jar CouplingGui.jar
```

The start window appears as shown in Fig. 1. To create a new coupling from scratch the user needs to load two components represented by XML files by clicking on **Component 1** and **Component 2** buttons below the file status line. After clicking on any of these buttons the standard Open File Dialog will appear. When the user has selected the component XML file and clicked on the **Open** button, the content of the component file will be loaded and displayed in the component panel. Typically, this will consist of a list of transients (see Fig. 2). Likewise, the second component has to be loaded. For each transient the GUI displays the name of the field, the name of the numerical grid on which the field is defined and its collection size. As YAC currently supports a 2-dimensional interpolation in the horizontal (on the sphere) only, the collection size can either be the number of vertical levels of this field or the number of horizontal fields hidden behind this particular field name, sometimes also referred to as bundles.

## 2. New coupling

To start creating a coupling between any two physical fields (transients) of two different components the user needs to click the check box representing the source field. In this case all transients which cannot be coupled with this source transient will remain inactive. The transient is considered as valid for coupling only if it has the same name and collection size. After finding the second possible transient for coupling and the checking of its corresponding checkbox, a red arrow connecting these two transients will be drawn. The direction of this arrow is pointing into the direction of the coupling, from the source to the target. Both directions of coupling are possible: from the left component to the right one and vice versa. Once an interpolation instruction is defined the colour of arrow is turned into green, like shown in Fig. 2 for the heat flux.

“collection size” has the same meaning as in the Fortran/c user interface where

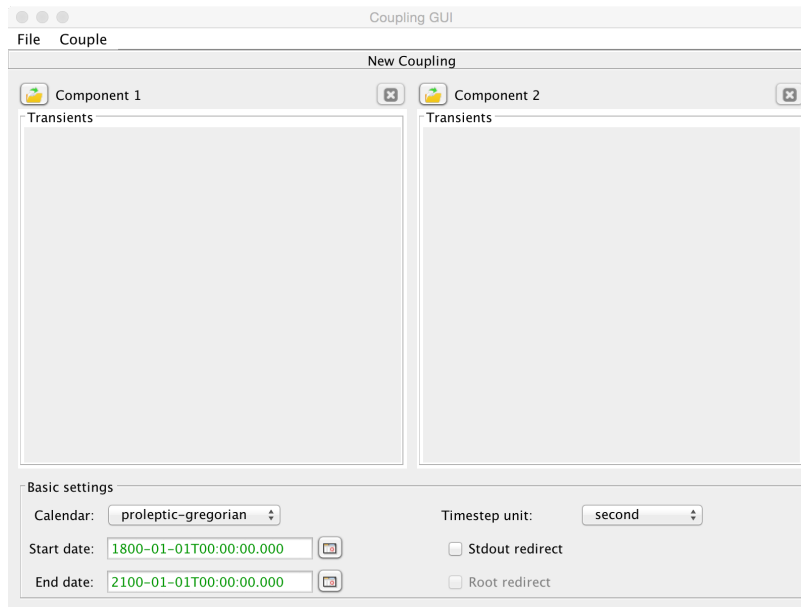


Figure 1: YAC XML configuration GUI start window.

it is used to describe the number of vertical levels or the number of horizontal fields that are stored with this one transient. The same horizontal interpolation stack is applied to all members in the collection.

### 3. Basic settings

Time settings of coupling can be specified in the Basic settings panel of the main window. The parameter **Calendar** has 3 options: **Proleptic-gregorian**, **360d** and **365d** which activates either the Proleptic-Gregorian calendar<sup>1</sup>, a calendar with an equal length of all months (30 days), or a calendar without any leap years. Specific dates of coupling can be set manually in the fields **Start date** and **End date**. After any user input or other interactions with values of these fields the user will be notified by the colour of the strings whether the provided date is correct or not according to the specified calendar. A green colour of values means that the date is correct for the selected calendar while red means it is not. For the Proleptic-gregorian calendar also a quick date picker is available as an alternative. It helps to select any date with a few clicks. This widget will disappear after selecting the day of the chosen year and month or with the starting of any user interactions in the date edit field.

<sup>1</sup>[http://en.wikipedia.org/wiki/Proleptic\\_Gregorian\\_calendar](http://en.wikipedia.org/wiki/Proleptic_Gregorian_calendar)

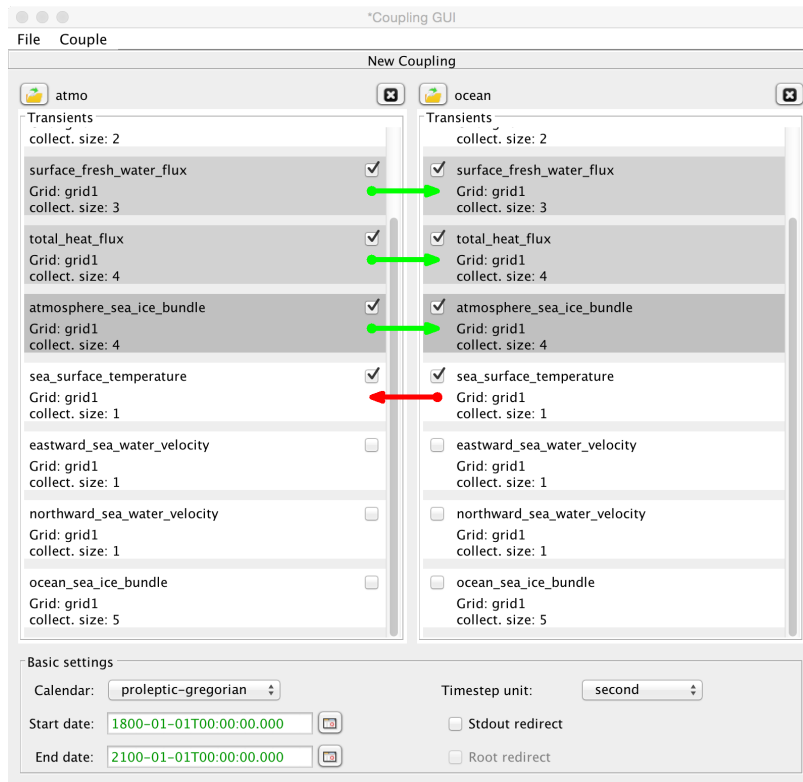


Figure 2: Example of a coupling configuration.

The `Timestep` unit parameter indicates which time units are to be used in the timestep parameter tab described in Sec. 4.1.2.

Further the user can request the redirection of stdout and stderr by checking `Stdout redirect`. When this is activated, output will be redirected after the application has called `yac_finit` or `yac_cinit`. The output file name gets the name of the respective component followed by the local MPI process ID. When `Stdout redirect` is activated the user has the additional choice to let only the component root processes redirect their output rather than letting each process write into its own file.

## 4. Subpanels

### 4.1. Setting parameters of specific transients coupling

In case the user has coupled two transients, additional parameters have to be provided for the coupling. To open the dialog for setting up coupling parameters (Fig. 3) the user has to click on any of the coupled transients in the component panel area. The coupling parameters dialog will appear after that. This dialog has 3 tabs: Interpolation, Timestep and More.

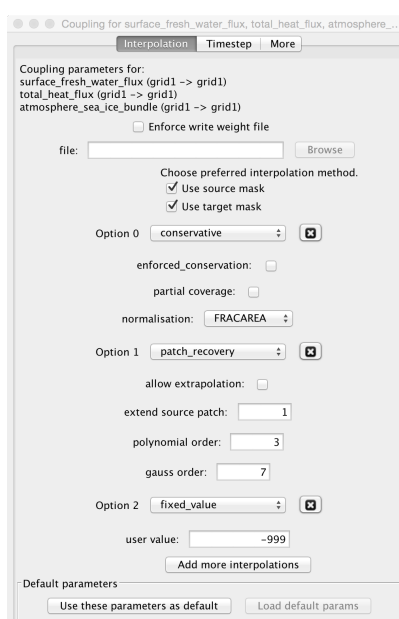


Figure 3: Coupling interpolation parameters window.

#### 4.1.1. Interpolation parameters tab

Interpolation parameters that can be specified for the coupling are the mask handling, i.e. whether the source and/or target masks shall be considered, and the sequence of interpolation methods which will be used for the interpolation. The specification of interpolation methods is organized in a list. The default number of interpolation methods which can be specified is three, but additional method options will appear after clicking on the **Add more interpolations** button. The user has to define at least one interpolation method in order to activate the coupling. Otherwise, no interpolation weights will be calculated and the particular exchange of transients will remain inactive.

Most interpolation methods require more specific parameters. These will be dynamically added next to the particular interpolation method and described in some more detail in Sec. 4.2.

The complete selection can be saved as default setting for defining further transient couples with **Use these parameters as default**. This will then activate **Forget default params**. For a next transient couple definition **Load default params** will apply the currently stored default.

#### 4.1.2. Timestep parameter tab

On the timestep tab (Fig. 4) the user can indicate time parameters of the coupling. In particular we require the user to specify the model time step or in other words, the period with which the respective exchange routines are called by the application. Next we require the user to specify the coupling period for the transient. The units of these parameters are controlled in the main window of the application in the Basic settings panel described earlier in Sec 3.

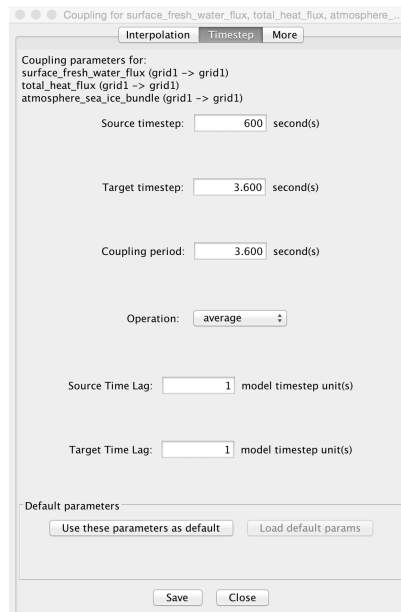


Figure 4: Coupling timestep parameters window.

In addition the **Source** and **Target Time lag** can be specified. The lag is a positive integer number or zero to adjust the internal event trigger clock for the source (put) and target (get) operations according to the timestepping algorithm used in the application. A source time lag of 2 will put forward the internal clock

for the put event by 2 times the source time step.

Like with the interpolation parameters tab default settings can be stored and applied.

### 4.1.3. More coupling parameters

On the last tab (Fig. 5) the user can specify the following parameters: **Debug mode** and its stage, **Enforce write restart** and **Mask value** (optional). Note that these are not yet interpreted by the YAC library.

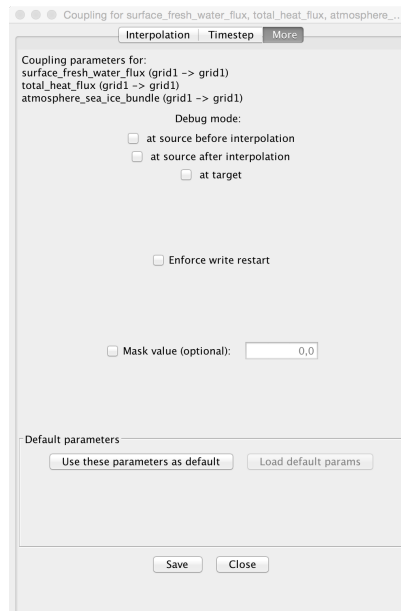


Figure 5: More coupling parameters.

## 4.2. Interpolation specific parameters

### 4.2.1. Average interpolation

*Partial coverage* (Boolean)

By setting this to `true`, partially covered target cells will receive an inter-

polated value.

*Weighted* (Enumeration: Distance weighted or Arithmetic average, default Arithmetic average,)

The values at the vertices or other locations of an element can be combined to a simple arithmetic average or can contribute weighted by their distances to the source points. In the latter case distances between sources and target location are calculated on the sphere. These distances are then normalised to 1. Using the vertices of a quadrilateral element the distance-weighted average is equivalent to a bi-linear interpolation.

#### 4.2.2. N-nearest Neighbor interpolation

*N* (Integer value)

Number of source point values requested for the nearest-neighbour interpolation.

*Weighted* (Enumeration: Distance weighted or Arithmetic average, default Arithmetic average,)

The *n* nearest neighbours can be combined to a simple arithmetic average or can contribute weighted by their distances to the source points. In the latter case distances between sources and target location are calculated on the sphere. These distances are then normalised to 1.

#### 4.2.3. Conservative interpolation

*Enforced conservation* (Boolean, default **false**)

By setting this to **true**, local conservation is achieved by correcting the local weights obtained from the partial area contributions such that they locally add up to 1 as far as numerical precision allows.

*Partial coverage* (Boolean, default **false**)



By setting this to `true`, partially covered source cells will be interpolated.

*Normalization* (Enumeration: FRACAREA or DESTAREA, default DESTAREA)

In case *Partially coverage* is set to `true`, a request for FRACAREA will use the fractional (partially covered) area to normalise the weights. When DESTAREA is selected weights will be normalised with the destination area, the area of the target cell.

#### 4.2.4. Patch recovery interpolation

*Polynomial order* (Integer value, default 1)

The order of the polynomial is selected here. Currently we support full 3d 1st, 2nd and 3rd order polynomials. The polynomial fit is applied to the fix points on the sphere in the cartesian x-y-z coordinate system. For example, the 1st order polynomial thus has the form

$$P_1 = a_0 + a_1x + a_2y + a_3z.$$

*Gauss order* (Integer value, default 1)

This value defines in an abstract way the number of fix points per source cell over which the polynomials are fitted. The number of fix points is set for a triangle. All other elements (number of edges larger than three) are split into triangles according to the number of edges and the requested number of fix points is applied to each resulting triangle.

Gauss Order	Number of fix points in triangle	Number of fix points in polygons
1	1	1 × # edges
2	3	3 × # edges
3	4	4 × # edges
4	6	6 × # edges
5	7	7 × # edges
6	12	12 × # edges
7	13	13 × # edges

*Extend source patch* (Integer value, default 1)

For a particular target cell the original source patch is made up of all source cells which have an overlap with this target cell (exactly those cells that are used for the 1st order conservative remapping). The *Extend source patch* value specifies the number of source cell rings by which the patch shall be extended. Currently we only support 0 or 1, where 0 represents the original patch. By selecting 1 we extend the patch by one more ring of cells around the original patch.

*Allow extrapolation* (Boolean, default **false**)

By default we apply the interpolation only if the target location is located inside any of the cells of the source patch. By setting *Allow extrapolation* to **true** we use the polynomial to extrapolate to the target location.

#### 4.2.5. Fixed value

*User value* (Float/Real value)

A fixed value can be specified by the user which is assigned to target cells or points. If it is selected as first method in the stack all target cells or points will be assigned to this value and all remaining interpolation methods in the stack will have no effect. If selected as 2nd option or later the user value will only be assigned to those target points which could not be considered (interpolated) by any of the previous methods in the stack. The Fixed value interpolation shall thus be set as the final interpolation in the stack. No warning is thrown if this is not the case.

#### 4.2.6. User file interpolation

Files containing the interpolation weights can be generated off-line e.g. using the `cdo`. By selecting the user file interpolation YAC can be forced to read in a file. The user can force YAC to write out the interpolation weights into a file.

## 5. Saving a new XML file

To save the coupling that has been created through the previously described actions in the application the **Save** and **Save as ...** options are available in the File menu. In case the user did not save the coupling before, for both of these menu options firstly the **Saving File** dialog will appear. After a successful saving of the file, the file status string will show the file name of the newly created coupling XML file. Subsequent plain save operations will overwrite the existing file. The GUI does not provide any automated versioning of XML files. This is left to the user with the **Save as ...** option.

## 6. Modification of an existing coupling configuration

The above allows for saving intermediate steps or the final setting. In order to continue the work to create or to modify an existing setting a valid coupling XML file can be loaded using **open** in the File menu. As described in the previous sections all parameters can be modified, and connections between any two fields can be activated or deactivated.

## 7. Multi component coupling

Once a pair of components has been loaded following the description above (and the coupling been defined) additional component (component XML descriptions) can be loaded via the menu entry **Couple add component**.

Alternatively we allow to use the “add component” mechanism to first load all components before starting with any further configuring.

When selecting **select active components** from the **Couple** menu a window similar to what is depicted in Fig. 6 will pop up. In the upper part the GUI lists the available component descriptions while the lower part displays those component pair for which some coupling between transients has already been defined.

A new set of component pairs can now be selected from the list in the upper half. By clicking **Apply** the selected two components will now be displayed similar for

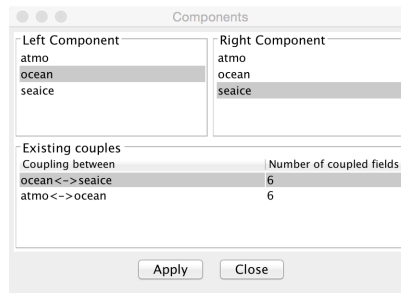


Figure 6: Selection of component pairs.

Fig. 2 and the coupling can be defined. Already (partly) defined component pairs can also be selected from the lower part, by a click on **Apply** this particular pair is now on display for further editing.

## A. Component XML description

In order to generate a coupling XML file the GUI requires a XML description of two model components which are going to be coupled as input. As can be seen in Fig. A1 the structure of such a component XML file is rather simple and can easily be generated with a simple text editor.

```
[<?xml version="1.0" encoding="UTF-8"?>
<component
  xmlns="http://www.w3schools.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3schools.com component.xsd">
  <id>2</id>
  <name>ICON-ocean</name>
  <model>ICON</model>
  <simulated>ocean</simulated>

  <transient_grid_refs>
    <transient_grid_ref id="1" transient_ref="1" grid_ref="1" collection_size="2" />
    <transient_grid_ref id="2" transient_ref="2" grid_ref="1" collection_size="2" />
    <transient_grid_ref id="3" transient_ref="3" grid_ref="1" collection_size="1" />
    <transient_grid_ref id="4" transient_ref="4" grid_ref="1" collection_size="4" />
    <transient_grid_ref id="5" transient_ref="5" grid_ref="1" collection_size="4" />
    <transient_grid_ref id="6" transient_ref="6" grid_ref="1" collection_size="2" />
  </transient_grid_refs>

  <transients>
    <transient id="1" transient_standard_name="surface_downward_eastward_stress"/>
    <transient id="2" transient_standard_name="surface_downward_northward_stress"/>
    <transient id="3" transient_standard_name="sea_surface_temperature"/>
    <transient id="4" transient_standard_name="water_flux"/>
    <transient id="5" transient_standard_name="heat_flux"/>
    <transient id="6" transient_standard_name="albedo"/>
  </transients>

  <grids>
    <grid id="1" alias_name="R2B04_no_land"/>
  </grids>
</component>
```

Figure A1: Example of a component XML file

The `id` has to be selected such that it is unique among the component XML descriptions, likewise the `name` of the component shall be unique.

A list of `transients` links standard names with a unique local identifier. For each component these IDs can run from 1 to N.

The list of `transient_grid_refs` provides additional information for the transients. The IDs of the `transient_grid_refs` shall again run from 1 to N. These references are later used in the coupling XML description to access the transient information. The number of the `transient_ref` refers to the `transient` ID explained above. A `transient_ref` of 3 refers to `transient` ID 3, and thus to `heat_flux`. Likewise, the `grid_ref` of 1 refers to the `grid` ID 1 defined below. In this case we have only defined one `grid`, thus the `grid_ref` is set to 1 for all transients in the list of `transient_grid_refs`. Last but not least the `collection_size` (size of bundle or number of vertical levels) has to be provided for each transient.

## B. Coupling XML dependencies

A coupling XML file as it was produced by the GUI is shown in Fig. B1.

```

[<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<coupling xmlns="http://www.w3schools.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3schools.com coupling.xsd">
  <redirect redirect_of_root="true" redirect_stdout="true"/>
  <components>
    <component id="1">
      <name>atmo</name>
      <model>ICON</model>
      <simulated>atmosphere</simulated>
      <transient_grid_refs>
        <transient_grid_ref collection_size="2" grid_ref="1" id="1" transient_ref="1"/>
        <transient_grid_ref collection_size="2" grid_ref="1" id="2" transient_ref="2"/>
        <transient_grid_ref collection_size="3" grid_ref="1" id="3" transient_ref="3"/>
        <transient_grid_ref collection_size="4" grid_ref="1" id="4" transient_ref="4"/>
        <transient_grid_ref collection_size="4" grid_ref="1" id="5" transient_ref="5"/>
        <transient_grid_ref collection_size="1" grid_ref="1" id="6" transient_ref="6"/>
      </transient_grid_refs>
    </component>
    <component id="2">
      <name>ocean</name>
      <model>ICON</model>
      <simulated>ocean</simulated>
      <transient_grid_refs>
        <transient_grid_ref collection_size="2" grid_ref="1" id="1" transient_ref="1"/>
        <transient_grid_ref collection_size="2" grid_ref="1" id="2" transient_ref="2"/>
        <transient_grid_ref collection_size="3" grid_ref="1" id="3" transient_ref="3"/>
        <transient_grid_ref collection_size="4" grid_ref="1" id="4" transient_ref="4"/>
        <transient_grid_ref collection_size="4" grid_ref="1" id="5" transient_ref="5"/>
        <transient_grid_ref collection_size="1" grid_ref="1" id="6" transient_ref="6"/>
      </transient_grid_refs>
    </component>
  </components>
  <transients>
    <transient id="1" transient_standard_name="surface_downward_eastward_stress"/>
    <transient id="2" transient_standard_name="surface_downward_northward_stress"/>
    <transient id="3" transient_standard_name="surface_fresh_water_flux"/>
    <transient id="4" transient_standard_name="total_heat_flux"/>
    <transient id="5" transient_standard_name="atmosphere_sea_ice_bundle"/>
    <transient id="6" transient_standard_name="sea_surface_temperature"/>
  </transients>
  <grids>
    <grid alias_name="grid1" id="1"/>
  </grids>
  <dates>
    <start_date>+1800-01-01T00:00:00.000</start_date>
    <end_date>+2100-01-01T00:00:00.000</end_date>
    <calendar>proleptic-gregorian</calendar>
  </dates>
  <timestep_unit>second</timestep_unit>
  <couples>
    <couple>
      <component1 component_id="1"/>
      <component2 component_id="2"/>
      <transient_couple transient_id="1">
        <source component_ref="1" transient_grid_ref="1"/>
        <target transient_grid_ref="1"/>
        <timestep>
          <source>10</source>
          <target>10</target>
          <coupling_period operation="accumulate">60</coupling_period>
          <source_timelag>0</source_timelag>
          <target_timelag>0</target_timelag>
        </timestep>
        <interpolation_requirements use_source_mask="true" use_target_mask="true">
          <interpolation_method="n-nearest_neighbor" n="1" weighted="DISTANCE_WEIGHTED"/>
        </interpolation_requirements>
        <debug_mode at_source_after_interpolation="false" at_source_before_interpolation="false" at_target="false"/>
        <enforce_write_restart>false</enforce_write_restart>
        <enforce_write_weight_file filename="">false</enforce_write_weight_file>
      </transient_couple>
    </couple>
  </couples>
</coupling>

```

Figure B1: Example of a coupling XML file

For each component we have a list of `N` `transient_grid_refs` with consecutive IDs from 1 to `N`, where each `transient_grid_ref` ID may have a different `collection_size`, `grid_ref` and `transient_ref` (Fig. B2).

```

<transient_grid_refs>
  <transient_grid_ref collection_size="2" grid_ref="1" id="1" transient_ref="1"/>
  <transient_grid_ref collection_size="2" grid_ref="1" id="2" transient_ref="2"/>
  <transient_grid_ref collection_size="1" grid_ref="1" id="3" transient_ref="3"/>
  <transient_grid_ref collection_size="4" grid_ref="1" id="4" transient_ref="4"/>
  <transient_grid_ref collection_size="4" grid_ref="1" id="5" transient_ref="5"/>
  <transient_grid_ref collection_size="2" grid_ref="1" id="6" transient_ref="6"/>
</transient_grid_refs>

```

Figure B2: List of transient grid reference IDs

These lists are copied from the component XML descriptions. The `grid_ref` and `transient_ref` ids get updated (see Appendix A). Typically we will have at least two entries in the `grid` element (Fig. B3), thus `grid_ref` becomes either 1 or 2 in this case. In the same way the `transient_ref` ids point to the appropriate entries in the transient list.

```

<grids>
  <grid alias_name="R2B04_no_land" id="1"/>
  <grid alias_name="R2B04" id="2"/>
</grids>

```

Figure B3: List of grids

In the couple (Fig. B4) the two components are listed with its component ID. Next a list of transient couples is provided which defines the source component identified by its component ID and its respective transient grid reference ID and the transient grid reference ID of the target.

```

<couples>
  <couple>
    <component1 component_id="2"/>
    <component2 component_id="1"/>
    <transient_couple transient_id="5">
      <source component_ref="2" transient_grid_ref="5"/>
      <target transient_grid_ref="5"/>
    </transient_couple>
  </couple>
</couples>

```

Figure B4: Coupling dependencies