XIOS User Guide

Draft

February 18, 2015

Chapter 1

Calendar

1.1 How to define a calendar

XIOS has an embedded calendar module which needs to be configured before you can run your simulation.

Only the calendar type and the time step used by your simulation are mandatory to have a well defined calendar. For example, a minimal calendar definition could be:

- from the XML configuration file:

```
<?xml version="1.0"?>
<simulation>
  <context id="test">
    <calendar type="Gregorian" timestep="1.5h" />
  </context>
</simulation>
```

- from the Fortran interface:

```
! ...
TYPE(xios_context) :: ctx_hdl
! ...
! Context initialization ommited, see the
    ↳ corresponding section of this user manual and
    ↳ of the reference manual
CALL xios_get_handle("test",ctx_hdl)
CALL xios_set_current_context(ctx_hdl)
CALL xios_define_calendar(type="Gregorian", timestep
    ↳ =1.5*xios_hour)
```

The calendar type definition is done once and for all, either from the XML configuration file or the Fortran interface, and cannot be modified. However there is no such restriction regarding the time step which can be defined at a different time than the calendar type and even redefined multiple times.

For example, it is possible to the achieve the same minimal configuration as above by using both the XML configuration file:

```xml
<?xml version="1.0"?>
<simulation>
  <context id="test">
    <calendar type="Gregorian" />
  </context>
</simulation>
```

and the Fortran interface:

```fortran
! ...
TYPE(xios_context) :: ctx_hdl
! ...
! Context initialization ommited, see the corresponding
    ↳ section of this user manual and of the reference
    ↳ manual
CALL xios_get_handle("test",ctx_hdl)
CALL xios_set_current_context(ctx_hdl)
! xios_define_calendar cannot be used here because the
    ↳ type was already defined in the configuration file.
! Ommiting the following line would lead to an error
    ↳ because the timestep would be undefined.
CALL xios_set_timestep(timestep=1.5*xios_hour)
```

The calendar also has two optional date parameters:

- the start date which corresponds to the beginning of the simulation
- the time origin which corresponds to the origin of the time axis.

If they are undefined, those parameters are set by default to "***0000-01-01 00:00:00***". If you are not interested in specific dates, you can ignore those parameters completely. However if you wish to set them, please note that they must not be set before the calendar is defined. Thus the following XML configuration file would be for example invalid:

```xml
<?xml version="1.0"?>
<simulation>
  <context id="test">
    <!-- Invalid because the calendar type cannot be
        ↳ known at that point -->
    <calendar start_date="2011-11-11 13:37:42" />
  </context>
</simulation>
```

while the following configuration file would be valid:

```xml
<?xml version="1.0"?>
```

```
<simulation>
  <context id="test">
    <!-- The order of the arguments does not matter so
        ↳ this is valid -->
    <calendar time_origin="2011-11-11␣13:37:42" type="
        ↳ Gregorian" />
  </context>
</simulation>
```

Of course, it is always possible to define or redefine those parameters from the Fortran interface, directly when defining the calendar:

```
! ...
TYPE(xios_context) :: ctx_hdl
! ...
! Context initialization ommited, see the corresponding
    ↳ section of this user manual and of the reference
    ↳ manual
CALL xios_get_handle("test",ctx_hdl)
CALL xios_set_current_context(ctx_hdl)
CALL xios_define_calendar(type="Gregorian", time_origin=
    ↳ xios_date(1977, 10, 19, 00, 00, 00), start_date=
    ↳ xios_date(2011, 11, 11, 13, 37, 42))
```

or at a later time:

```
! ...
TYPE(xios_context) :: ctx_hdl
! ...
! Context initialization ommited, see the corresponding
    ↳ section of this user manual and of the reference
    ↳ manual
CALL xios_get_handle("test",ctx_hdl)
CALL xios_set_current_context(ctx_hdl)
CALL xios_define_calendar(type="Gregorian")
CALL xios_set_time_origin(time_origin=xios_date(1977, 10,
    ↳  19, 00, 00, 00))
CALL xios_set_start_date(start_date=xios_date(2011, 11,
    ↳ 11, 13, 37, 42))
```

To simplify the use of dates in the XML configuration files, it is possible to partially define a date as long as the omitted parts are the rightmost. In this case the remainder of the date is initialized as in the default date. For example, it would be valid to write: `start_date="1977-10-19"` instead of `start_date="1977-10-19 00:00:00"` or even `time_origin="1789"` instead of `time_origin="1789-01-01 00:00:00"`. Similarly, it is possible to express a date with an optional duration offset in the configuration file by using the `date + duration` notation, with `date` potentially partially defined or even completely omitted. Consequently the following examples are all valid in the XML configuration file:

- `time_origin="2011-11-11 13:37:00 + 42s"`

- `time_origin="2014 + 1y 2d"`
- `start_date="+ 36h"`.

1.2 How to define a user defined calendar

Predefined calendars might not be enough for your needs if you simulate phenomenons on another planet than the Earth. For this reason, XIOS can let you configure a completely user defined calendar by setting the **type** attribute to "***user_ defined***". In that case, the calendar type alone is not sufficient to define the calendar and other parameters should be provided since the duration of a day or a year are not known in advance.

Two approaches are possible depending on whether you want that your custom calendar to have months or not: either use the **month_lengths** attribute to define the duration of each months in days or use the **year_length** attribute to define the duration of the year in seconds. In both cases, you have to define **day_length**, the duration of a day in seconds. Those attributes have to be defined at the same time than the calendar type, either from the XML configuration file or the Fortran interface, for example:

```
<?xml version="1.0"?>
<simulation>
  <context id="test">
    <calendar type="user_defined" day_length="86400"
        ↳ month_lengths="(1, 12) [31 28 31 30 31 30 31 31
        ↳ 30 31 30 31]" />
  </context>
</simulation>
```

or

```
! ...
TYPE(xios_context) :: ctx_hdl
! ...
! Context initialization ommited, see the corresponding
    ↳ section of this user manual and of the reference
    ↳ manual
CALL xios_get_handle("test",ctx_hdl)
CALL xios_set_current_context(ctx_hdl)
CALL xios_define_calendar(type="Gregorian", day_length
    ↳ =86400, year_length=31557600)
```

Note that if no months are defined, the format of the dates is modified in the XML configuration file since the month must be omitted. For example, `"2015-71 13:37:42"` would be the correct way to refer to the 71st day of the year 2015 at 13:37:42. If you use the Fortran interface, the month cannot be omitted but you have to make sure to always set it to `1` in that case. For example, use `xios_date(2015, 01, 71, 13, 37, 42)`for `"2015-71 13:37:42"`.

Moreover, it is possible that the duration of the day is greater than the duration of the year on some planets. In this case, it obviously not possible to define months so you have to use the **year_length** attribute. Additionally the day must also be omitted from the dates in the configuration file (for example `"2015 13:37:42"`) and must always be set to `1` when using the Fortran interface (for example `xios_date(2015, 01, 01, 13, 37, 42)`).

If months have been defined, you might want to have leap years to correct the drift between the calendar year and the astronomical year. This can be achieved by using the **leap_year_drift** and **leap_year_month** attributes and optionally the **leap_year_drift_offset** attribute. The idea is to define **leap_year_drift**, the yearly drift between the calendar year and the astronomical year as a fraction of a day. This yearly drift is summed each year to know the current drift and each time the current drift is greater or equal to one day, the year is considered a leap year. In that case, an extra day is added to the month defined by **leap_year_month** and one day is subtracted to the current drift. The initial drift is null by default but it can be fixed by the **leap_year_drift_offset** attribute.

The following configuration file defines a simplified Gregorian calendar using the user calendar feature:

```
<?xml version="1.0"?>
<simulation>
  <context id="test">
    <calendar type="user_defined"
          day_length="86400"
          month_lengths="(1, 12) [31 28 31 30 31 30 31 31
             ↳  30 31 30 31]"
          leap_year_month="2"
          leap_year_drift="0.25"
          leap_year_drift_offset="0.75"
          time_origin="2012-02-29 15:00:00"
          start_date="2012-03-01 15:00:00" />
  </context>
</simulation>
```

As you know, the astronomical year on Earth is approximately a quarter of day longer than the Gregorian calendar year so we have to define the yearly drift as 0.25 day. In case of a leap year, an extra day is added at the end of February which is the second month of the year so **leap_year_month** should be set to 2. We start our time axis in 2012 which was a leap year in the Gregorian calendar. This means there was previously three non-leap years in a row so the current drift was (approximately) 3×0.25 days, hence **leap_year_drift_offset** should be set to 0.75. At the beginning of 2013, the drift would have been 0.75+0.25 = 1 day so 2012 will be a leap year as expected.

1.3 How to use the calendar

The calendar is created immediately after the calendar type has been defined and thus can be used even before the context definition has been closed.

Once the calendar is created, you have to keep it updated so that it is in sync with your simulation. To do that, you have to call the **xios_update_calendar** subroutine for each iteration of your code:

```
! ...
INTEGER :: ts
! ...
DO ts=1,end
  CALL xios_update_calendar(ts)
  ! Do useful stuff
ENDDO
```

The current date is updated to $start\_date + ts \times timestep$ after each call.

Many other calendar operations are available, including:

- accessing various calendar related information like the time step, the time origin, the start date, the duration of a day or a year, the current date, etc.
- doing arithmetic and comparison operations on date:

```
TYPE(xios_date) :: date1, date2
TYPE(xios_duration) :: duration
LOGICAL :: res
! we suppose a calendar is defined
CALL xios_get_current_date(date1)
duration = xios_duration(0, 0, 1, 0, 0, 0, 0, 0) + 12
    ↳ * xios_hour
date2 = date1 + duration + 0.5 * xios_hour
res = date2 > date1
duration = date2 - date1
```

- converting dates to
  - the number of seconds since the time origin, the beginning of the year or the beginning of the day,
  - the number of days since the beginning of the year,
  - the fraction of the day or the year.

For more detailed about the calendar attributes and operations, see the XIOS reference guide.