REFERENCE DE CONFIGURATION XML

1. Les étapes de création d'un fichier de configuration XML

Cette partie présente étape par étape la manière de créer un fichier de configuration XML pour le serveur des entrées/sorties du projet XMLIOSERVER. Les règles syntaxiques décrites dans les premiers paragraphes restent valides jusqu'à la fin de ce tutoriel.

a) Un fichier de configuration «vide»

iodef.xml

```
1. <?xml version="1.0"?>
2. <simulation/>
```

XML 1: Configuration vide de base

Rappel: Notions du langage XML

- L'abréviation XML signifie e**X**tensible **M**arkup **L**anguage.
- C'est un langage informatique de balisage generique qui sert, dans notre cas, à stocker des données textuelles Unicode.
- Il existe plusieurs versions du langage XML, celle que nous utilisons est la version 1.0 publiée en février 1998 (c'est la plus répandue).
- Le XML est basé sur le modèle d'un arbre, qui comprend plusieurs nœuds de différents types: document, éléments, textes, … dont le rôle sera expliqué au fur et à mesure.
- Le **nœud de type document** est l'élément racine de l'arbre, il est unique et peut éventuellement avoir des enfants.

Pour créer un document XML valide, on commence par écrire l'entête du fichier, appelé *prologue* dans la nome, qui indique la version du langage que nous allons utiliser.
Cette information est obligatoire et sera dans notre cas toujours la même, c'est-à-dire:
<?xml version="1.0"?>

Une autre information facultative peut être ajoutée, il s'agit de l'encodage textuel utilisé dans le fichier. Là encore, nous utiliserons toujours le format *UTF-8* qui permet de gérer tous les caractères Unicode:
<?xml version="1.0" encoding="UTF-8"?>

On indique ensuite le nœud document qui est la racine unique de l'arborescence XML.
Ce nœud a toujours pour nom la chaîne «simulation» dans le fichier de configuration principal, quelque soit le cas de figure dans lequel on se place, et peut ne pas avoir d'enfant (auquel cas aucune sortie n'est paramétrée statiquement).
Son nom est insensible à la casse et ses attributs éventuels ne sont pas analysés par le «parseur XML».

Exemples:

<simulation/> ← valide.
<SimuLatIon/> ← valide, car insensible a la casse.
<simulation id="ma_simulation" /> ← valide mais l'attribut nommé «id» ayant pour valeur "ma_simulation" ne sert a rien ...
<simmulation/> ← invalide, le nœud document doit avoir pour nom «simulation».

Un avertissement est transmis à l'utilisateur mais l'exécution se poursuit.
Dans le cas où le nœud document dispose de nœuds enfants, il s'écrit sous cette forme.
<simulation>....</simulation> ← *valide.*
<SimuLatIon>....</SimuLatIon> ← *valide, car insensible a la casse.*
<SimuLatIon>....</simulation> ← *invalide, car les noms de la balise ouvrante et de la balise fermante sont incompatibles au niveau du traitement XML.*
Une erreur du type «Tag mismatch» est générée au moment de l'exécution signalant également la position du problème. Le traitement s'interrompt inopinément.

Notes :

1. Dans la suite de ce document, on admettra qu'un avertissement n'interrompt pas l'exécution du programme contrairement à la détection d'une erreur, laquelle est susceptible de modifier trop profondément le comportement du code par rapport aux souhaits de l'utilisateur.
2. On n'admettra aussi que les noms des nœuds sont insensibles à la casse, contrairement aux valeurs qui leur sont associées.
   Ex: <context id="id1" /> ↔ <CONTEXT ID="id1" /> ← valides tous les deux.
3. La présence de deux nœuds racines génère une erreur de type «Junk after document element».
4. Pour le moment, l'absence de prologue est tolérée bien que fortement déconseillée pour respecter les standards.
5. Attention à la position des barres obliques au niveau des balises, une erreur du type «Tag mismatch» est générée en cas de problème à ce niveau.
6. La présence de guillemets simples à la place de guillemets doubles est acceptée.
7. Les caractères erronés rencontrés dans le fichier génèrent une erreur «Invalid token».

b) Les contextes de sorties

Rappel (suite): Notions du langage XML

- Les retours a la ligne et la présence d'une indentation particulière n'ont pas de signification dans le langage XML mais permettent une lecture plus aisée.
- Le **nœud de type commentaire** est délimité par <!-- et --> et n'est pas interprété par le programme lors de l'analyse du document.
- Le **nœud de type attribut** est toujours associé à un autre nœud.
Il est compose d'un nom et d'une valeur présentés sous la forme *nom="valeur".*
Pour un nom d'attribut et un nœud donnés, un attribut est toujours unique.
- Le **nœud de type élément** est probablement le plus utilisé dans un document XML.
Il dispose d'un nom qui le qualifie et accepte comme enfants tous les autres types de nœud.

iodef.xml

```
1. <?xml version="1.0"?>
2. <simulation>
3. <!-- Ici, je commente l'ajout des éléments -->
4.   <context id="c1"  calendar_type="Gregorian" start_date="12/07/2005 - 01:30:40">
5.     <!-- context vide -->
6.   <context id="c2"/>
7. </simulation>
```

XML 2: Configuration valide comprenant quelques contextes.

Le nœud racine peut disposer d'un nombre quelconque d'éléments enfants nommés «context». Les contextes permettent de regrouper globalement des conditions de génération de fichier en

fonction de situations rencontrées par l'utilisateur, comme la différenciation des sorties lors du passage d'un code de calcul à un autre (pour modèle atmosphérique, modèle océanique, … par exemple). Chaque contexte doit être identifié par l'ajout d'un attribut «id» pour être traité.

Quelques cas de figures:

<context id="mon_id" /> ← valide.
<conttext id="mon_id">...</conttext> ← incorrect mais valide. Si le nom de l'élément n'est pas «context», son contenu n'est toutefois pas ignoré car le nœud racine «simulation» ne peut contenir que des nœuds «context». Un avertissement est transmis pour signaler le problème syntaxique.
<context id="mon_id" name="mon_nom" /> ← valide même si l'attribut «name» n'est pas necessaire, un avertissement est transmis a l'utilisateur.
<context id="mon_id" id="mon_autre_id" /> ← invalide, a cause de la non-unicité de l'attribut «id» sur le nœud, une erreur de type «Duplicate attribute» est affichée par le programme.
<context>...</context> ← valide, mais génère un avertissement, l'absence de l'identifiant empêche la récupération du contexte pour le traitement des sorties qui lui sont associées.

A propos des identifiants … et de leurs valeurs

Un identifiant doit être compose de caractères alphanumériques ([A-Z][a-z][0-9]) et de tirets bas (_) sans quoi l'identification échoue et un avertissement est renvoyé.
Il est conseiller d'identifier systématiquement les éléments de groupe ainsi que les éléments finaux sauf dans le cas des références a des champs lors de la définition de fichiers (voir la suite). L'absence d'identification entraîne l'impossibilité d'accéder dynamiquement à certains objets du programme par leurs identifiants et peut donc provoquer un comportement inattendu au niveau du traitement des sorties.

Les attributs de contextes

Il est possible de définir pour chacun des contextes un calendriers qui leur est associé.
Un calendrier est initialisé par son type (attribut *calendar_type*) et une date initiale valide (attribut *start_date*). Le format de la date doit se présenter sous la forme
« JJ/MM/AAAA - hh-mm-ss»

Exemples :

<context id="monid1" calendar_type="Gregorian" start_date="12/07/2005 – 02:20:30"> ← valide
<context id="monid2" calendar_type="Julian" start_date="03/07/2008 – 21:33:44"> ← valide
<context id="monid2" calendar_type="noleap" start_date="29/02/2008 – 21:33:44"> ← invalide car dans le calendrier «no leap», les mois de février ne comptent que 28 jours.
<context id="monid2" calendar_type="Julian" start_date="03|07|2008 – 21:33:44"> ← invalide car le format de la date n'est pas correct.

Note :
Dans le code client en langage Fortran, il est possible de définir le date initiale dans un contexte à partir d'une structure spécifique plutôt qu'une chaîne de caractères.

Liste des attributs de l'élément context				
Nom de l'attribut	**Type de donnée**	**Héritage**	**Valeur par défaut**	**Description de l'attribut**
id	**StdString**	**/**	**/**	Identifiant de l'élément dans l'arborescence
calendar_type	**StdString**	**/**	**/**	Type de calendrier utilisé
start_date	**StdString**	**/**	**/**	Date initiale dans le calendrier

c) Les groupes de définitions

iodef.xml

```xml
<?xml version="1.0"?>
<simulation>
<!-- Définitions des différents contextes d'E/S -->
  <context id="c1">
  <!-- Ajout des groupes de definitions -->
    <field_definition/>
    <file_definition/>
    <axis_definition/>
    <grid_definition/>
    <domain_definition/>
    <variable_definition/>
  </context>
  <context id="c2"/>
</simulation>
```

XML 3: Configuration valide avec groupes de définitions.

Les groupes de définitions sont utilisés pour compartimenter les éléments dont il portent le nom dans l'arborescence d'objets de la bibliothèque XMLIOSERVER.

Si l'on exclut les nœuds contextuels, il existe pour le moment 6 types d'éléments différents dont voici la liste avec une d'une brève explication du rôle de chacun :

- Les éléments de type «field» qui permettent de lister les champs disponibles à la sortie.
- Les éléments de type «grid» qui définissent les grilles associées aux champs.
- Les éléments de type «axis» qui déterminent les propriétés globales des axes verticaux.
- Les éléments de type «domain» qui déterminent les composantes globales et locales des domaines horizontaux qui différent suivant la parallélisation des codes de calculs.
- Les éléments de type «file» qui contiennent les informations sur les fichiers à sortir.
- Les éléments de type «variable» qui répertorient les variables créées dynamiquement.

Le comportement de ces types d'élément est expliqué de manière plus détaillée dans les paragraphes suivants.

Note :
Il n'est pas nécessaire d'intégrer la balise de définition pour un type d'élément si celle-ci ne contient aucun enfant.

d) Les grilles et groupes de grilles

iodef.xml

```xml
<?xml version="1.0"?>
<simulation>
<!-- Définitions des différents contextes d'E/S -->
  <context id="c1">
    <!-- Ajout des groupes de définitions -->
    <grid_definition>
    <!-- Définitions des grilles et groupes de grilles -->
      <grid_group id="ggr1" axis_ref="ax3" >
        <grid id="gr1" description="desc. Gr1" domain_ref="dom4" />
        <grid id="gr2" name="nom gr2" domain_ref="dom3" />
      </grid_group>
      <grid id="gr4"  domain_ref="dom1"'/>
    </grid_definition>
  </context>
</simulation>
```

XML 4: *Configuration valide incluant la définition des grilles.*

Une grille en 3 dimensions est l'union d'un domaine plan horizontal et d'un axe vertical. L'absence d'axe vertical permet de définir une grille en 2 dimensions tandis que l'absence d'un domaine horizontal définit une grille unidimensionnelle.

Les attributs de grilles

Les attributs *domain_ref* et *axis_ref* indiquent quel axe et quel domaine composent la grille. Ce sont en fait les identifiants de ces objets dans l'arborescence d'objets de la bibliothèque.

Quelques cas de figures:

<grid id="gr3" name="nom gr3" domain_ref="dom2" axis_ref="ax2" /> ← valide (3D)
<grid id="gr3" name="nom gr3" domain_ref="dom2"/> ← valide (2D)
<grid id="gr3" name="nom gr3" axis_ref="ax2" /> ← valide (1D) mais inutilisée pour le moment
<grid id="gr3" name="nom gr3" /> ← invalide car ni axe, ni domaine ne sont référencés.

Liste des attributs de l'élément grid

Nom de l'attribut	Type de donnée	Héritage	Valeur par défaut	Description de l'attribut
id	**StdString**	/	/	Identifiant de l'élément dans l'arborescence
name	**StdString**	**desc**	/	Nom de l'élément (facultatif)
description	**StdString**	**desc**	/	Description de l'élément (facultatif)
domain_ref	**StdString**	**desc**	/	Référence sur élément domaine (facultatif en 1D)
axis_ref	**StdString**	**desc**	/	Référence sur élément axis (facultatif en 2D)

e) Les domaines et groupes de domaines

iodef.xml

```
1.  <?xml version="1.0"?>
2.  <simulation>
3.     <context id="mon_context_zoom">
4.        <!-- Définitions des domaines et groupes de domaines -->
5.        <domain_definition>
6.           <domain_group  ni="21" nj="10" data_dim="2" ...>
7.              <domain id="domaine_simple"/>
8.              <domain id="domaine_zoom"
9.                          zoom_ibegin="7"  zoom_jbegin="3"
10.                         zoom_ni="5"        zoom_nj="5"/>
11.          </domain_group>
12.       </domain_definition>
13. <!-- .... -->
14.             <field id="champ_zoom" domain_ref="domaine_zoom"/>
15.             <field id="champ_simple" domain_ref="domaine_simple"/>
16. <!-- .... -->
17.    </context>
18. </simulation>
```

XML 5: Configuration valide incluant la définition des zoom et de domaine.

Les domaines sont des surfaces 2D permettant d'introduire des informations sur les valeurs de longitude et latitude mais aussi sur la répartition des données localement sur chacun des clients.

Les attributs globaux de domaines

1. *ni_glo* et *nj_glo* déterminent la taille du domaine global;
2. *data_dim* indique la dimension des données transmises par le code client;
3. *zoom_ni* et *zoom_ibegin*, respectivement *zoom_nj* et *zoom_jbegin*, permettent de limiter la zone de sortie des données à une portion du domaine.

Les attributs locaux de domaines

1. *ni* et *nj* déterminent la taille du domaine local;
2. *ibegin* et *iend*, respectivement *jbegin* et *jend*, indiquent où commence et se termine le domaine local défini au niveau d'un client ou d'un serveur;
3. *data_ni* et *data_ibegin*, respectivement *data_nj* et *data_jbegin*, délimitent la zone de validité des données sur un domaine local (notamment pour exclure les recouvrements);
4. *mask* est un tableau 2D, ayant le même profil que le domaine, qui spécifie les valeurs masquées des données reçues des clients;
5. *lonvalue* et *latvalue* sont les valeurs de longitude et latitude en repère rectilinéaire ou curvilinéaire;
6. *data_n_index* est la taille des tableaux de données issues du code;
7. *data_i_index* et *data_j_index* sont les tableaux d'indexation des données sur le domaine.

* *data_nj*, *data_jbegin* et *data_j_index* ne sont jamais définis si la dimension des données *data_dim* vaut 1.

La définition des zoom

Pour configurer un zoom, on place l'intégralité des attributs globaux et locaux dans un groupe de domaines auquel on ajoute la définition du domaine entier puis celle des différents zooms.

Par exemple, dans le document XML 5, les attributs globaux et locaux du domaine «domaine_simple» sont placés dans le groupe de domaines supérieur. Les deux éléments «domaine_simple» et «domaine_zoom» héritent tous deux de ces attributs mais le second définit en plus les données de zoom.

Un même groupe de domaine devra donc contenir le domaine complet et les zooms sur ce même domaine pour que le mécanisme d'héritages puissent définir ces derniers entièrement.

Liste des attributs de l'élément domain

Nom de l'attribut	Type de donnée	Héritage	Valeur par défaut	Description de l'attribut
id	**StdString**	**/**	**/**	Identifiant de l'élément dans l'arborescence
standard_name	**StdString**	**desc**	**/**	Nom court (facultatif)
long_name	**StdString**	**desc**	**/**	Nom détaillé (facultatif)
n(X)_glo	**int**	**desc**	**(Valeur calculée en fonction des deux autres)**	Taille du domaine global
(X)begin	**int**	**desc**		Début du domaine local
(X)end	**int**	**desc**		Fin du domaine local
n(X)	**int**	**desc**	**/**	Taille du domaine loca
mask	**bool[2]**	**desc**	**plein**	Masque du domaine local (facultatif)
data_dim	**int**	**desc**	**/**	Dimension des données traitées
data_n(X)	**int**	**desc**	**(calculé)**	Indicatif de début des données traitées (fac.)
data_(X)begin	**int**	**desc**	**0**	Indicatif de début des données traitées (fac.)
zoom_n(X)	**int**	**desc**	**n(X)**	Taille du zoom global (facultatif)
zoom_(X)begin	**int**	**desc**	**1**	Début du zoom global (facultatif)
data_n_index	**int**	**desc**	**(calculé)**	Taille de l'indexation (facultatif)
data_(X)_index	**int[1]**	**desc**	**(calculé)**	Tableau d'indexation (facultatif)
lonvalue	**double[1]**	**desc**	**/**	Valeurs de longitude
latvalue	**double[1]**	**desc**	**/**	Valeurs de latitude

* Non documentée: domain_group_ref, zoom_ni_loc, zoom_nj_loc, zoom_ibegin_loc, zoom_jbegin_loc

Quelques cas de figures:

<domain id="mon_domaine" data_dim="3" /> ← invalide car data_dim vaut 1 ou 2.
<domain id="mon_domaine" ni_glo="0" nj_glo="17" /> ← invalide car les valeurs de ni_glo et nj_glo doivent être supérieures ou églales à 0 pour êtres traitées.
<domain id="mon_domaine" ni="-1" nj="17" /> ← invalide car les valeurs de ni et nj doivent être supérieures ou églales à 0 pour êtres traitées.
<domain id="mon_domaine" ibegin="0" ni="36"/> ← invalide car ibegin, jbegin, zoom_ibegin et zoom_jbegin doivent être supérieures ou égales à 1.
<domain id="mon_domaine" ibegin="1" iend=" 20" ni="21"/> ← invalide car les valeurs d'attribut ne respectent pas l'égalité suivante :
n(X) = (X)end - (X)begin + 1

<domain id="mon_domaine" data_n_index="0" /> ←incorrect car une taille de table d'indexation nulle indique l'absence de données.
<domain id="mon_domaine" data_dim="1" data_jbegin="2" data_nj="13"/> ← incorrect car les attributs data_jbegin et data_nj ne sont pas nécessaires dans le cas où la dimension des données vaut 1.
<domain id="mon_domaine" zoom_ibegin="10" zoom_ni="17" ni_glo="15"/> ← invalide car le zoom n'entre pas dans le domaine global.

f) Les axes et groupes d'axes

iodef.xml

```
1. <?xml version="1.0"?>
2. <simulation>
3.     <context id="mon_context">
4.         <axis_definition>
5.             <axis_group>
6.                 <axis id="mon_axe" unit="m" size="7" zvalue="1,2,3,4,5,6,7" />
7.             </axis_group>
8.         </axis_definition>
9.     </context>
10. </simulation>
```

XML 6: Configuration valide incluant la définition d'axes.

Les axes sont des éléments très simples utilisés pour définir les axes verticaux sur des grilles en 3 dimensions en complément de domaines horizontaux et possiblement pour définir des grilles unidimensionnelles.

Les attributs d'axes

La définition d'un axe nécessite la connaissance du tableau de ses composantes (*zvalue*) et de la taille de ce tableau (*size*). L'information sur l'unité est facultative mais fortement recommandée.

A propos des attributs de type tableau

Le document XML 6 décrit l'une des manières de définir un tableau de dimension 1 pour l'attribut *zvalue* de l'axe nommé «mon_axe».

Il est toutefois recommandé de définir directement les tableaux dans le code client en Fortran.

Liste des attributs de l'élément axis				
Nom de l'attribut	**Type de donnée**	**Héritage**	**Valeur par défaut**	**Description de l'attribut**
id	**StdString**	**/**	**/**	Identifiant de l'élément dans l'arborescence
standard_name	**StdString**	**desc**	**/**	Nom court (facultatif)
long_name	**StdString**	**desc**	**/**	Nom détaillé (facultatif)
unit	**StdString**	**desc**	**/**	Unité utilisée (facultatif)
zvalue	**double[1]**	**desc**	**/**	Valeurs sur l'axe
size	**int**	**desc**	**/**	Taille de l'axe

g) Les champs et groupes de champs

iodef.xml

```
1.  <?xml version="1.0"?>
2.  <simulation>
3.      <context id="mon_context">
4.          <field_definition>
5.              <field_group  enabled=".FALSE."  level="1">
6.                  <field id="mon_champ" enabled=".TRUE."
7.                          grid_ref="ma_grille" prec="8"/>
8.              </field_group>
9.          </field_definition>
10.         <file_definition>
11.             <file  id="mon_fichier" name="data/exemple.nc">
12.                 <field  field_ref="mon_champ"/>
13.             </file>
14.         </file_definition>
15.     </context>
16. </simulation>
```

XML 7: Configuration valide incluant la définition des champs.

Les objets de type «field» indiquent les champs disponibles dans un contexte de simulation et la manière dont ils doivent être traités et sortis. Ils peuvent être contenus dans leur groupe de définition, comme tous les autres éléments, où directement dans une balise «file» pour déterminer quelles sont les données qu'un fichier doit contenir.

Note:

De manière générale, on préférera définir les attributs des champs dans des éléments contenus dans leur groupe de définition et créer des références à partir d'autres champs regroupés dans les objets de type «file».

Les attributs de champs

Pour déterminer quelle grille associer à un champ, il est possible d'y faire référence directement à l'aide de l'attribut *grid_ref* ou de spécifier un domaine et/ou une grille, via les attributs *axis_ref*

et *grid_ref*, qui vont permettre de créer dynamiquement une nouvelle grille.
Les attributs *freq_op* et *operation* concerne le traitement d'un champ avant sa sortie. Le premier indique la fréquence à laquelle l'opération est appliquée sur les données tandis que la seconde est l'identifiant de l'opération qui doit être déjà implémentée dans le code source de la bibliothèque XMLIOSERVER.
La valeur de *prec* indique la précision des données réelles en écriture, soit 8 en double précision ou 4 en simple précision.
Enfin, *default_value* est la valeur par défaut retenue pour le champ.

Liste des opérations sur les champs			
Nom de la fonction	**Type**	**Identifiant**	**Description de la fonction**
Instant	/	**inst**	Aucun traitement, le champ est sorti tel qu'il est reçu du code de calcul.
Average	**Temporelle**	**ave**	Moyenne temporelle commune
Once	**Temporelle**	**once**	Le champ est écrit une seule fois

Liste des attributs de l'élément field				
Nom de l'attribut	**Type de donnée**	**Héritage**	**Valeur par défaut**	**Description de l'attribut**
id	**StdString**	**/**	**/**	Identifiant de l'élément dans l'arborescence
standard_name	**StdString**	**desc/ref**	**/**	Nom court (facultatif)
long_name	**StdString**	**desc/ref**	**/**	Nom détaillé (facultatif)
unit	**StdString**	**desc/ref**	**/**	Unité utilisée
operation	**StdString**	**desc/ref**	**inst**	Opération à effectuer sur le champs
freq_op	**StdString**	**desc/ref**	**/**	Fréquence d'opération
level	**int**	**desc/ref**	**1**	Niveau de sortie pour ce champ (recommandé)
prec	**int**	**desc/ref**	**4**	Précision de sortie(recommandé)
enabled	**bool**	**desc/ref**	**true**	Autorisation d'écriture(recommandé)
domain_ref	**StdString**	**desc/ref**	**/**	Référence sur un domaine(facultatif)
axis_ref	**StdString**	**desc/ref**	**/**	Référence sur un axe (facultatif)
grid_ref	**StdString**	**desc/ref**	**/**	Référence sur un grille
field_ref	**StdString**	**desc/ref**	**/**	Référence sur un champ pour héritage
default_value	**double**	**desc/ref**	**/**	Valeur par défaut des données

A propos des attributs booléens

Les attributs booléens sont définis dans le document XML suivant la syntaxe Fortran c'est-à-dire .TRUE. pour vrai et .FALSE. pour faux.

Quelques cas de figures:

<field id="mon_champ" enabled="false"/> ← *invalide car la valeur de l'attribut enabled ne correspond pas à celle d'un booléen.*
<field id="mon_champ" grid_ref="ma_grille" prec='6"/> ← *invalide car la précision doit valoir 4 ou 8, respectivement pour la simple et la double précision.*
<field id="mon_champ" grid_ref="ma_grille" domain_ref="mon_domaine" /> ← *incorrect mais valide car on associe deux grilles au champ (dont l'un est abstraite). Cependant, la référence sur la grille prévaut.*
<field id="mon_champ" operation="once" freq_op="1j" /> ← *incorrect mais valide car l'opération ne nécessite pas d'indiquer une fréquence d'opération. Le premier champ reçu est écrit.*
<field id="mon_champ" operation="ave" freq_op="8600" /> ← *invalide car le format de fréquence est le suivant:*
JJ d MO mo AA y HH h MI mi SS s
(Avec JJ le nombre de jours, MO le nombre de mois, AA le nombre d'années,
HH le nombre d'heures, MI le nombre de minutes, SS le nombre de secondes.)

Toutes les composantes d'une durée sont optionnelles.

h) Les fichiers et groupes de fichiers

iodef.xml

```
1.  <?xml version="1.0"?>
2.  <simulation>
3.      <context id="mon_context">
4.          <field_definition>
5.              <field_group  enabled=''.FALSE."  level="1">
6.                  <field id="mon_champ" enabled=".TRUE."/>
7.              </field_group>
8.          </field_definition>
9.          <file_definition>
10.             <file  id="mon_fichier"' name="data/exemple.nc" output_freq="1j">
11.                 <field  field_ref="mon_champ"/>
12.             </file>
13.         </file_definition>
14.     </context>
15. </simulation>
```

XML 8: Configuration valide incluant la définition des fichiers.

Les éléments «file» listent les fichiers de sortie à générer ainsi que les champs de données qu'ils sont susceptibles de contenir. C'est un nœud de groupe au comportement particulier puisqu'il peut inclure des objets de type «field» et «field_group».

Les attributs de fichiers

Les attributs les plus importants pour la définition des champs sont le nom du fichier à créer sur le disque (*name*) et la fréquence de sortie des données (*output_freq*) qui doit respecter le format: JJ d MO mo AA y HH h MI mi SS s
(Avec JJ le nombre de jours, MO le nombre de mois, AA le nombre d'années,
HH le nombre d'heures, MI le nombre de minutes, SS le nombre de secondes.)
L'attribut output_level indique la valeur au delà de laquelle les champs ne sont plus sortis en fonction de leur level.

Liste des attributs de l'élément file				
Nom de l'attribut	**Type de donnée**	**Héritage**	**Valeur par défaut**	**Description de l'attribut**
id	**StdString**	**/**	**/**	Identifiant de l'élément dans l'arborescence
name	**StdString**	**desc**	**/**	Nom de l'élément
description	**StdString**	**desc**	**/**	Description de l'élément (facultatif)
output_freq	**StdString**	**desc**	**/**	Fréquence de sortie
output_level	**StdString**	**desc**	**3**	Niveau de sortie maximum (recommandé)

2. Définition de variables

iodef.xml

```
1.  <?xml version="1.0"?>
2.  <simulation>
3.    <context id="mon_context">
4.      <variable_definition>
5.
6.        <!-- Première méthode de définition de variables -->
7.        <variable_group  id="mes_variables1">
8.          un_entier          = 17
9.          un_reel            = 8.5
10.         une_chaine         = "mon code de calcul"
11.         #un_tableau_1d = [ 1., 1.2, 17.5 ]
12.         un_tableau_2d   = [[ 1., 1.2, 17.5 ] ,[ 1., 1.2, 17.5 ]]
13.       </variable_group>
14.       <!-- Seconde méthode de définition de variables (typage) -->
15.       <variable_group  id="mes_variables2">
16.           <variable id="une_chaine" type="string">
17.             "ceci est une variable typée"
18.           </variable>
19.       </variable_group>
20.       <!-- Troisième méthode de définition de variables (typage) -->
21.       <variable_group  id="mes_variables3" type="string">
22.         ma_chaine1  = 17
23.         ma_chaine2  = 1.6
24.         ma_chaine3 = "valeur de chaîne"
25.       </variable_group>
26.       <!-- Quatrième méthode de définition de variables -->
27.       </variable_group include="test/mes_definition_de_vars.xml"/>
28.
29.     </variable_definition>
30.   </context>
31. </simulation>
```

XML 9: Définition dynamique de variables

Les éléments de type «variable» permettent de créer des variables dynamiquement à partir d'informations inclues dans le document XML de définition de l'utilisateur.
Ces variables sont compartimentées dans le contexte auquel elles appartiennent et classées dans des groupes que l'on peut assimiler à des répertoires, ce qui signifie que deux variables qui partagent le même espace de nom ne peuvent avoir le même identifiant.

Par exemple, dans le document XML 6, la variable «*un_entier*» appartient au groupe «*mes_variables1*» du contexte «*mon_context*».
Elle est donc accessible via son chemin, c'est-à-dire «*/mes_variables1/un_entier*», ou directement à l'aide de son identifiant si l'utilisateur a sélectionné le répertoire adéquat par l'appel de la fonction **change_directory** (voir la documentation développeur).

Il existe trois méthodes de définition des variables :

- La première d'entre elles consiste à utiliser la balise de groupe pour y inclure les informations sur les variables les unes à la suite des autres suivant le format:
  variableid = valeur <newline>
- La seconde permet de typer les variables à l'aide de l'attribut «type» de l'élément «variable». Le format requis est alors le suivant:
  <variable id="variableid" type="variabletype"> valeur </variable>
- La troisième méthode ressemble à la première mais le groupe définit le type de toutes les variables qu'il inclut donc celles-ci ne pourront être traitées qu'avec ce type.
- La dernière méthode nécessite l'inclusion d'un fichier externe qui contient les définitions.

Notes :

Étant donnée l'utilisation qui est faite de ce type d'élément, il est impossible modifier la branche de définition des variables dans l'arborescence d'objets de la bibliothèque depuis le code client.

3. Les héritages d'attributs

iodef.xml

```
1.  <?xml version="1.0"?>
2.  <simulation>
3.      <context id="mon_context">
4.          <field_definition>
5.              <field_group  enabled=''.FALSE."  level="1">
6.                  <field id="mon_champ" enabled=".TRUE."/>
7.              </field_group>
8.          </field_definition>
9.          <file_definition>
10.             <file  id="mon_fichier"' name="data/exemple.nc">
11.                 <field  field_ref="mon_champ"/>
12.             </file>
13.         </file_definition>
14.     </context>
15. </simulation>
```

XML 10: Exemple héritage (fichier de base)

La bibliothèque XMLIOSERVER dispose d'un mécanisme d'héritage interne qui permet de lier divers éléments de l'arborescence d'objets les uns aux autres. Cet héritage entraîne la définition des attributs d'un objet à partir de ceux d'un objet parent si ces attributs n'ont pas encore été complétés initialement ou dynamiquement. Un attribut n'est donc jamais écrasé lors de la procédure d'héritage.

Les deux sous-parties suivantes vont permettre d'expliquer la manière dont fonctionnent les héritages en se basant sur l'exemple le document XML 10.

Notes :

1. L'attribut «id» n'est jamais hérité, quelque soit l'objet auquel cet attribut appartient.
2. L'attribut de groupe «include» et l'attribut de groupe de champs «group_ref» ont un comportement particulier qui ne sera pas traité dans cette documentation.

a) Héritages descendants ou de groupe

iodef.xml

```
1. <field_definition>
2.     <field_group  enabled=".FALSE."  level="1">
3.         <field id="mon_champ" enabled=".TRUE."  level="1"/>
4.     </field_group>
5. </field_definition>
```

XML 11: Exemple héritage descendant

Les héritages descendants permettent la transmissions des attributs d'un groupe aux enfants qu'il contient s'il sont du même genre. Ces enfants peuvent aussi être des groupes. Par exemple, dans le document XML 11, le champ nommé «mon_champ» hérite de son groupe parent la valeur de l'attribut *level* car celui-ci n'était pas définit précédemment. A l'opposé, l'attribut *enabled* ayant déjà une valeur, il n'est pas écrasé.

b) Héritages par référence

(Seuls les éléments de type field implémentent cette fonctionnalité pour le moment)

iodef.xml

```
1. <file_definition>
2.     <file  id="mon_fichier"' name="data/exemple.nc">
3.         <field  field_ref="mon_champ"  enabled=".TRUE."  level="1"/>
4.     </file>
5. </file_definition>
```

XML 12: Exemple héritage par référence

Le fonctionnement des héritages par référence est similaire dans son principe aux héritages de groupe. Toutefois, l'objet de base pour la complétion des attributs est dorénavant indiqué par la valeur de «field_ref».

Dans le document XML 12, le fichier «mon_fichier» contient un champ qui hérite de son champ de référence nommé «mon_champ» la valeur des attributs *enabled* et *level*.

La résolution des héritages descendants précéde la résolution des héritages par références.

4. Exemple d'un fichier de configuration simple

iodef.xml

```
<?xml version="1.0" ?>
<simulation>
  <context id="context1" calendar_type="Gregorian" start_date="12/07/2005 -
01:30:40">

    <!--  Définition des champs du premier context -->
    <field_definition enabled=".TRUE." prec="8" level="1" unit="SI">
      <field id="champ1" name="premier_champ" standard_name="premier champ"
           long_name="mon premier champ" grid_ref="grille1" freq_op="2h" />
      <field id="champ2" name="second_champ" standard_name="second champ"
            long_name="mon second champ" grid_ref="grille2" freq_op="1h" />
    </field_definition>

    <!--  Définition des grilles du premier context -->
    <grid_definition>
      <grid id="grille1" name="premiere_grille"
   description="ma premiere grille" domain_ref="domaine1" axis_ref="axe1"/>
      <!--  grid id="grille1" name="premiere_grille"
   description="ma premiere grille" domain_ref="domaine1"/ -->
      <grid id="grille2" name="seconde_grille"
    description="ma seconde grille" domain_ref="domaine2" axis_ref="axe2"/>
    </grid_definition>

    <!-- Définition des domaines du premier context  -->
    <domain_definition>
      <domain id="domaine1" name="premier_domaine" long_name="mon premier
domaine" standard_name="premier domaine" data_dim="2" />
      <domain id="domaine2" name="second_domaine"
          standard_name="second domaine" long_name="mon second domaine"
          ni_glo="100" ni="50" ibegin="25" nj_glo="100" nj="50" jbegin="25"
          data_dim="1" lonvalue="25(50)74" latvalue="25(50)74" />
    </domain_definition>
    <!--  Définition des axes du premier context  -->
    <axis_definition unit="km">
      <axis id="axe1" name="premier_axe" standard_name="premier axe"
                 long_name="mon premier axe" size="20" zvalue="1(20)20" />
      <axis id="axe2" name="second_axe" standard_name="second axe"
                 long_name="mon second axe" size="10" zvalue="1(10)10" />
    </axis_definition>
    <!--  Définition des fichiers du premier context  -->

    <file_definition enabled=".TRUE." output_freq="12h">
      <file id="fichier1" name="data/premier_fichier.nc"
                        description="Mon premier fichier">
        <field_group group_ref="field_definition" />
      </file>
    </file_definition>
  </context>
</simulation>
```

5. Vue partielle de l'arborescence

Le schéma ci-dessous présente une partie de l'arborescence construite à l'aide les fichiers de configuration XML et du code client en Fortran.

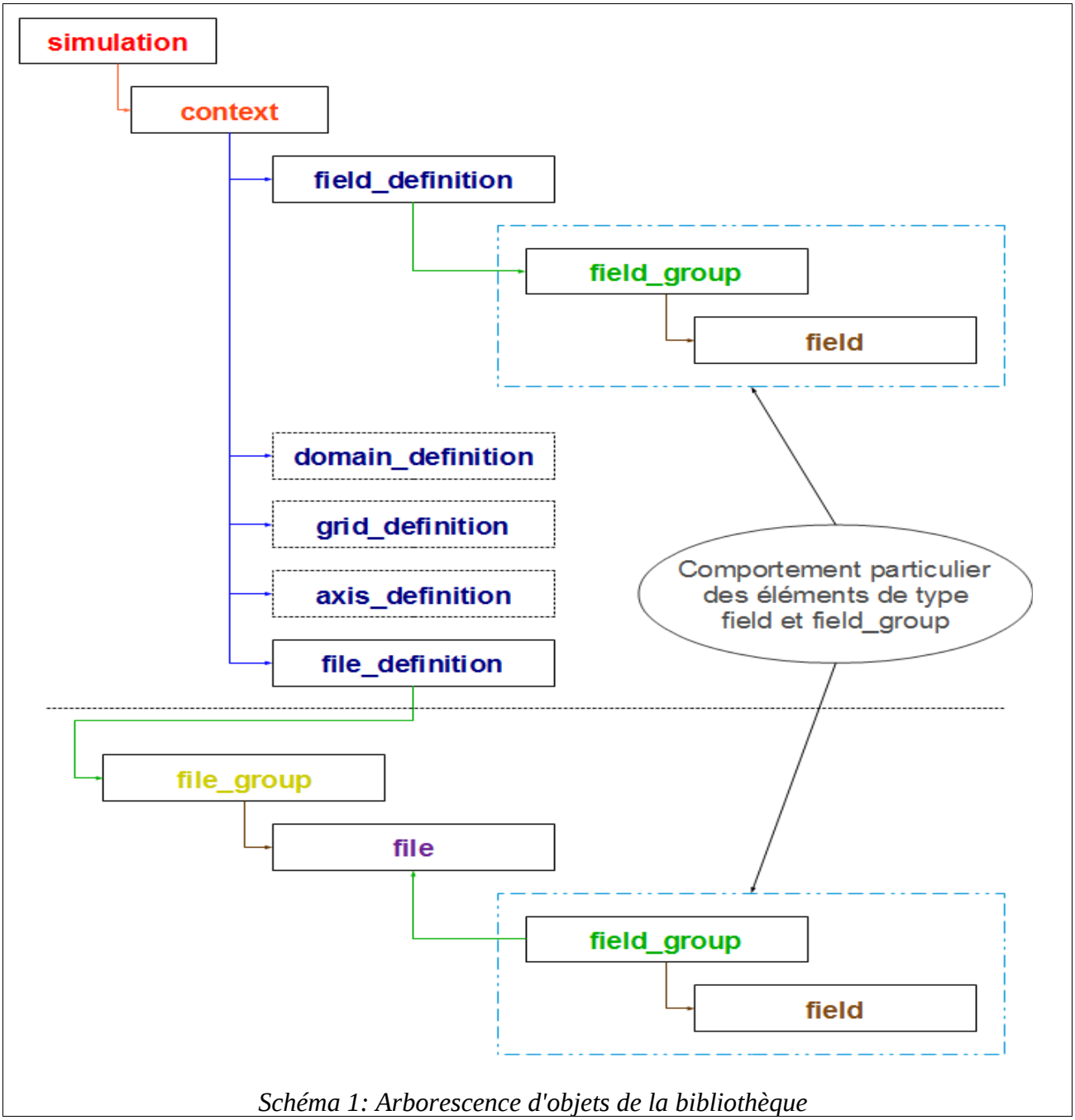

Schéma 1: Arborescence d'objets de la bibliothèque