# TP2 : Production de variables

| | |
|---|---|
| **Objectifs:** | Produire des variables sur différentes unités spatiales, utiliser des boucles de parcours de l'espace |
| **Pré-requis:** | TP1 |
| **Fonctionnalités:** | `DECLARE_PRODUCED_VARIABLE()`<br>`openfluid::core::SpatialUnit`<br>`openfluid::core::DoubleValue`<br>`OPENFLUID_UNITS_ORDERED_LOOP()`<br>`OPENFLUID_GetSimulationDuration()`<br>`OPENFLUID_GetCurrentTimeIndex()`<br>`OPENFLUID_InitializeVariable()`<br>`OPENFLUID_AppendVariable()` |

A partir du simulateur "vide" créé lors du TP1 (`training.signal.prod`), nous allons générer un signal de pluie sous la forme d'une variable produite sur toutes les unités de la classe SU. Ce signal est généré par une sinusoïde adaptée pour produire des valeurs de pluie sur une durée correspondant aux deux premiers tiers de la simulation.

## 1 Code source

Les modifications du code source sont à apporter dans le fichier `SignalSim.cpp`.

### 1.1 Signature

Nous allons tout d'abord déclarer dans la signature que le simulateur va produire une nouvelle variable nommée `water.atm-surf.H.rain` sur la classe d'unités `SU`. Cette déclaration se fait à l'aide de l'instruction `DECLARE_PRODUCED_VARIABLE` (Menu contextuel OpenFLUID/ Signature/ Variables/ Declare produced variable). L'instruction `DECLARE_PRODUCED_VARIABLE` comporte 4 paramètres :

- le nom de la variable produite

- la classe d'unité sur laquelle est produite la variable
- la description de la variable
- l'unité (SI) de la variable

**Note:** Utiliser de préférence le menu contextuel dans OpenFLUID-DevStudio pour rajouter une méthode OpenFLUID afin d'éviter les erreurs de frappe.

Une fois complétée, la signature devrait être similaire à:

```
BEGIN_SIMULATOR_SIGNATURE("training.signal.prod");
  DECLARE_NAME("");
  DECLARE_DESCRIPTION("");

  DECLARE_VERSION("13.05");
  DECLARE_STATUS(openfluid::ware::EXPERIMENTAL);

  DECLARE_DOMAIN("");
  DECLARE_PROCESS("");
  DECLARE_METHOD("");
  DECLARE_AUTHOR("Doe␣J.","doe@foobar.org");

  DECLARE_PRODUCED_VARIABLE("water.atm-surf.H.rain","SU","rainfall␣height␣on␣SU","m");

  //Scheduling
  DECLARE_SCHEDULING_DEFAULT;
END_SIMULATOR_SIGNATURE
```

Après construction/installation du simulateur (en cliquant sur `Build`), il est possible de vérifier si la signature a bien été modifiée en exécutant la commande `openfluid report simulators training.signal.prod` ou `openfluid report simulators` (pour l'ensemble des simulateurs disponibles).

## 1.2 initializeRun()

La méthode `initializeRun()` est appelée en début de simulation et permet d'initialiser les variables produites par le simulateur.

```
openfluid::base::SchedulingRequest initializeRun()
{

  openfluid::core::SpatialUnit* SU;
  OPENFLUID_UNITS_ORDERED_LOOP("SU",SU) // debut de la boucle spatiale
  {
    // Initialisation a 0 de la variable water.atm-surf.H.rain pour chaque SU
    OPENFLUID_InitializeVariable(SU,"water.atm-surf.H.rain",0.0);
  }
  return DefaultDeltaT();
}
```

## 1.3 runStep()

La méthode `runStep()` est appelée à chaque index de temps pour lequel le simulateur est actif. Nous allons y ajouter le code de calcul du signal qui sera produit sur chaque unité de la classe SU au travers de la variable `water.atm-surf.H.rain`. Pour cela, nous allons utiliser une boucle spatiale sur les SU, et produire la variable à l'aide de l'instruction `OPENFLUID_AppendVariable`.

Une boucle spatiale est identifiée au travers de l'instruction `OPENFLUID_UNITS_ORDERED_LOOP`, elle débute et se termine avec des accolades `{}`.

Le générateur du signal proposé pour cet exercice comporte deux phases:

- Une fonction basée sur un cosinus entre la moitié et les trois-quarts de la simulation,
- une valeur égale à 0.0 en dehors de cette période.

Soit $P_i$ la hauteur de pluie sur le pas de temps courant (m),
$P_{total}$ la hauteur totale de pluie au cours de la simulation (m),
$t_i$ l'index de temps courant,
$t_{debut}$ l'index de temps de début du signal de pluie, $t_{fin}$ l'index de temps de fin du signal de pluie,
$\Delta t$ la durée du pas de temps courant (constant dans ce cas),
$t_{total}$ la durée de la simulation

$$t_{debut} = \frac{1}{2} \cdot t_{total},\ t_{fin} = \frac{3}{4} \cdot t_{total}$$

$$\underline{\text{si}}\ t_{debut} \leq t_i < t_{fin}\ \underline{\text{alors}}\ P_i = P_{total} \cdot \frac{\Delta t}{t_{fin} - t_{debut}} \cdot (1 - \cos \frac{2.\pi.(t_i - t_{debut})}{t_{fin} - t_{debut}})$$

$$\underline{\text{sinon}}\ P_i = 0.0$$

Si vous le souhaitez, vous pouvez intégrer votre propre équation de génération du signal.
Une fois complétée, la méthode runStep() devrait être similaire à:

```
openfluid::base::SchedulingRequest runStep()
{
  openfluid::core::SpatialUnit* pSU;  // pointeur sur la SU courante
  openfluid::core::DoubleValue RainValue;  // valeur du signal a calculer

  const double TotalRainValue = 0.06;  // hauteur totale de pluie (m) durant l'evenement
  openfluid::core::TimeIndex_t FirstRainIndex = OPENFLUID_GetSimulationDuration()*0.5;
  openfluid::core::TimeIndex_t LastRainIndex = OPENFLUID_GetSimulationDuration()*0.75;
  openfluid::core::TimeIndex_t CurrentIndex = OPENFLUID_GetCurrentTimeIndex();

  // calcul de la valeur du signal
  RainValue = 0.0;
  if (CurrentIndex >= FirstRainIndex && CurrentIndex < LastRainIndex)
  {
    RainValue = TotalRainValue*
      (double(OPENFLUID_GetDefaultDeltaT())/double(LastRainIndex-FirstRainIndex))*
      (1-std::cos(double(2*(CurrentIndex-FirstRainIndex)*3.1415926)
```

```
                    /double(LastRainIndex-FirstRainIndex)));
  }

  OPENFLUID_UNITS_ORDERED_LOOP("SU",pSU)  // debut de la boucle spatiale
  {
    // production de la valeur du signal
    OPENFLUID_AppendVariable(pSU,"water.atm-surf.H.rain",RainValue);
  }   // fin de la boucle spatiale

  return DefaultDeltaT();
}
```

# 2 Simulation

Pour la simulation, nous allons utiliser le jeu de données "Bassin versant TP" situé dans le dossier `<Bureau>/formation/datasets/TP1-7`. Le jeu de données comporte 15 unités de classe SU, 14 unités de classe RS.

## 2.1 Création du projet

Lancer OpenFLUID-Builder, cliquer sur *Créer un projet*, vérifier que le chemin du répertoire de travail pointe bien sur
`<Bureau>/formation/projects`, nommer votre projet `TP2` et cliquer sur OK.

La première phase consiste à importer les données spatiales qui vont créer le domaine : nous utiliserons les couches *subroujan_rs_wgs84.shp* et *subroujan_su_wgs84.shp* contenues dans le dossier `<Bureau>/formation/datasets/TP1-7`. Importer ces données à l'aide de l'extension `Import de données spatiales (OGR/GDAL)` et du bouton *Ajouter une source fichier*.

Dans l'onglet `Structure spatiale`, indiquer pour la couche *subroujan_rs_wgs84.shp*, la classe d'unité à créer de type *RS*, laisser les autres champs par défaut. Dans l'onglet `Attributs spatiaux`, cocher les colonnes suivantes à importer : *width, slope, height* et *length*. Dans l'onglet `Fichiers et Datastore`, cocher les deux options et indiquer pour la première option le nom du fichier *roujan_rs.shp* et pour la deuxième option l'ID *RS*.

Procéder de même pour la couche *subroujan_su_wgs84.shp*, qui sera la classe de *SU*, avec un import des champs *slope, flowdist* et *area*.

**Note:** Pour plus d'informations concernant cette extension, se reporter au tutoriel *Import de données spatial avec l'extension Spatial data import (OGR/GDAL)*

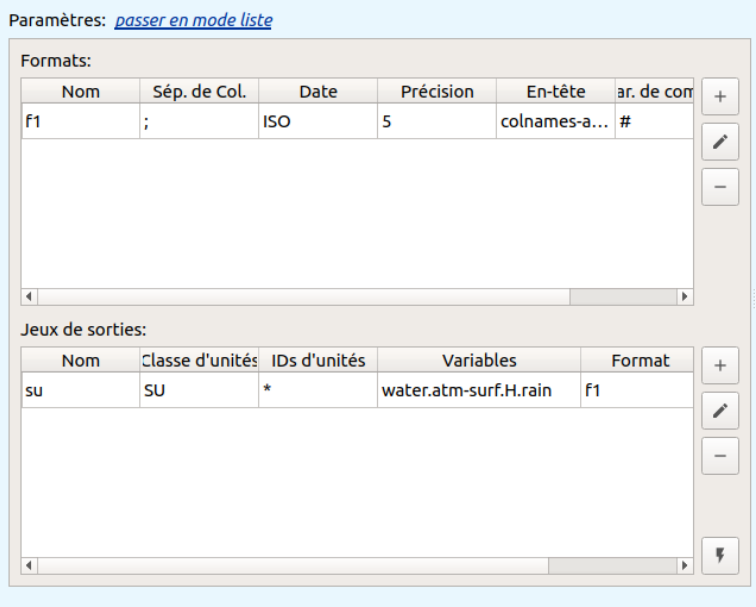
Une fois le domaine créé, configurer la simulation pour que celle-ci se déroule sur une période du 1er mars 2013 à 6h00 au 10 mars 2013 à 22h00 avec un pas de temps d'échange de 300 secondes.

Il faut maintenant ajouter notre simulateur dans le modèle. Pour cela, dans l'onglet *Modèle*, cliquer sur *Ajouter un simulateur* pour ajouter le simulateur que l'on vient de créer (`training.signal.prod`)

Ensuite, ajouter un observateur, via l'onglet *Monitoring* en cliquant sur *Ajouter un observateur*. Une des fonctions types d'un observateur est d'extraire des valeurs de variables au cours de la simulation pour les sauvegarder dans un format donné.

Ajouter un observateur de type *export.vars.file.csv*.

Paramétrer l'observateur pour sauvegarder la variable *water.atm-surf.H.rain* pour toutes les SU : Pour paramétrer cet observateur en mode "assistant",cliquer sur le bouton "+" de la partie *formats*. Définir un nom pour ce format, ici "f1" dans l'exemple, puis ajuster les paramètres si nécessaire. Ensuite ajouter un *jeu de sortie* en cliquant sur le bouton "+" de cette partie. Il faut alors le nommer, choisir le format "f1" défini juste avant puis modifier les paramètres pour récupérer les informations des SU.




**Note:** Pour paramétrer cet observateur, il faut définir un format pour la date, le caractère de séparation de colonne, et le caractère de commentaire de ligne (qui permet d'ignorer la ligne lors d'un post-traitement). Se reporter au tutoriel sur l'utilisation des observateurs pour des compléments d'information.

## 2.2 Exécution de la simulation

## 2.3 . . . avec l'interface OpenFLUID-Builder

**Note:** Il est conseillé d'enregistrer votre projet avant chaque lancement de simulation. Vous pouvez activer l'enregistrement automatique des projets depuis les *Préférences*, rubrique *Interface*

Lancer la simulation en cliquant sur le bouton *Exécution* de la barre d'outils.

Si tout s'est bien passé, les résultats de la simulation sont accessibles via le navigateur de projet, dans l'onglet *Explorateur des sorties*,
Double-cliquer sur le fichier .csv correspondant à l'unité spatiale pour laquelle vous voulez visualiser les résultats.

Nous allons ajouter un nouvel observateur qui permet d'enregistrer les sorties dans des graphiques au format pdf. Cet observateur étant un peu plus long à paramétrer, nous avons déjà préparé sa configuration.
Enregistrer et fermer votre projet, ajouter le fichier `monitoring_gnuplot.fluidx` à votre jeu de données d'entrée. Ce fichier se trouve dans `<Bureau>/formation/datasets/TP1-7`
et doit être copié dans `<Bureau>/formation/projects/TP2/IN`.
Ce fichier contient un observateur GNUplot pré-paramétré pour les simulations de ce TP et des suivants.

Ouvrir de nouveau votre projet et relancer les simulations. Ouvrir le fichier pdf généré depuis l'onglet des sorties d'OpenFLUID-Builder.

Enregistrer votre projet, et fermer le projet.

## 2.4 ... en ligne de commande

**Note:** L'utilisation de la ligne de commande est donnée ici à titre indicatif, il n'est pas nécessaire d'effectuer ces actions pour l'enchaînement des TP.

Pour lancer la simulation à partir du projet créé précédemment, nous allons utiliser la commande `openfluid run` en précisant le chemin du projet.

- chemin du projet : `<Bureau>/formation/projects/TP2`
- jeu de donnée en entrée du projet: `<Bureau>/formation/projects/TP2/IN`
- résultats : `<Bureau>/formation/projects/TP2/OUT`

La commande à exécuter est donc :

```
openfluid run <Bureau>/formation/projects/TP2/ -c
```

Le paramètre `-c` permet de vider le dossier résultats de fichiers précédents avant de lancer la simulation.

**Attention:** Sous Windows, utiliser le symbole slash "/" comme sous linux pour l'écriture des chemins entrés à la fin de la commande `openfluid run`, et non le backslash "\" habituel des chemins Windows.

Si tout s'est bien passé, les résultats de la simulation sont accessibles dans `<Bureau>/formation/projects/TP2/OUT`.

# 3 Annexe: Formats de dates

| Format | Description |
|---|---|
| %a | locale's abbreviated weekday name. |
| %A | locale's full weekday name. |
| %b | locale's abbreviated month name. |
| %B | locale's full month name. |
| %c | locale's appropriate date and time representation. |
| %C | century number (the year divided by 100 and truncated to an integer) as a decimal number [00-99]. |
| %d | day of the month as a decimal number [01,31]. |
| %D | same as %m/%d/%y. |
| %e | day of the month as a decimal number [1,31]; a single digit is preceded by a space. |
| %h | same as %b. |
| %H | hour (24-hour clock) as a decimal number [00,23]. |
| %I | hour (12-hour clock) as a decimal number [01,12]. |
| %j | day of the year as a decimal number [001,366]. |
| %m | month as a decimal number [01,12]. |
| %M | minute as a decimal number [00,59]. |
| %n | is replaced by a newline character. |
| %p | locale's equivalent of either a.m. or p.m. |
| %r | time in a.m. and p.m. notation; in the POSIX locale this is equivalent to %I:%M:%S %p. |
| %R | time in 24 hour notation (%H:%M). |
| %S | second as a decimal number [00,61]. |
| %t | is replaced by a tab character. |
| %T | time (%H:%M:%S). |
| %u | weekday as a decimal number [1,7], with 1 representing Monday. |
| %U | week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. |
| %V | week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. |
| %w | weekday as a decimal number [0,6], with 0 representing Sunday. |
| %W | week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0. |
| %x | locale's appropriate date representation. |
| %X | locale's appropriate time representation. |
| %y | year without century as a decimal number [00,99]. |
| %Y | year with century as a decimal number. |
| %Z | timezone name or abbreviation, or by no bytes if no timezone information exists. |
| %% | character %. |

Exemples:

- `%Y-%m-%dT%H:%M:%S` ⇔ `2008-01-01T11:13:00`

- %Y-%m-%d %H:%M:%S ⇔ 2008-01-01 11:13:00
- %Y%m%d%H%M%S ⇔ 20080101111300