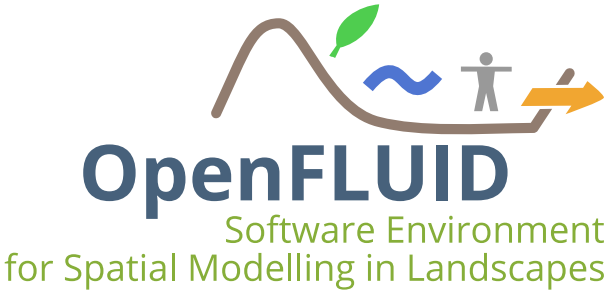


TP3 : Utilisation de variables en entrée

Objectifs:	Utiliser des variables produites par d'autres simulateurs
Pré-requis:	TP1, TP2
Fonctionnalités:	Déclaration de variable requise `OPENFLUID_GetVariable()`

Nous allons créer un simulateur (`training.su.prod`) qui produit des variables de ruissellement et d'infiltration sur les SU à partir du signal de pluie, et en appliquant la méthode SCS.

1 Code source

1.1 Création du simulateur

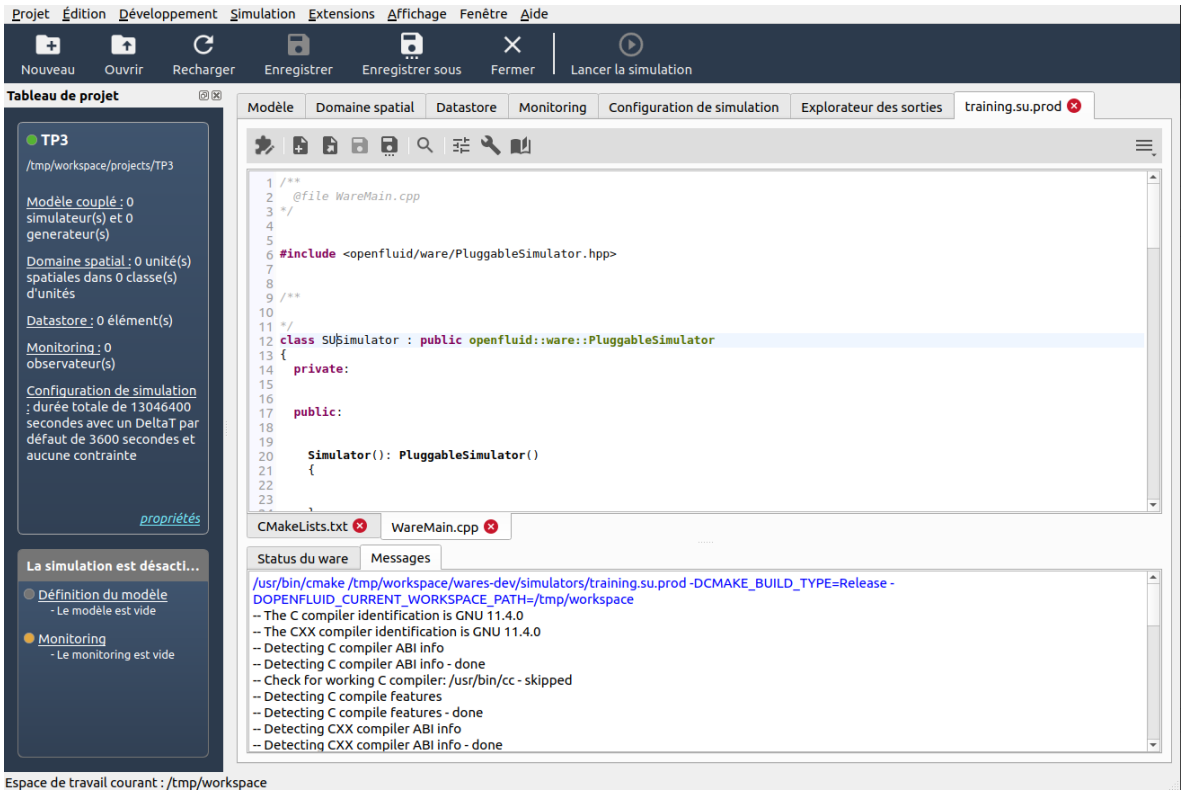
Nous allons créer le simulateur à l'aide de l'environnement de développement intégré à OpenFLUID-Builder.

Nous allons tout d'abord créer un nouveau projet OpenFLUID nommé TP3 (dans `<Bureau>/formation/projects/TP3`) dont les données seront basées sur le projet TP2. Lancer OpenFLUID-Builder, créer un nouveau projet nommé TP3 et cocher l'option *Importer des données/ Importer des données d'un projet existant* et choisir le projet *TP2*.

Une fois dans ce nouveau projet, pour créer un nouveau simulateur, aller dans *Développement/Nouveau simulateur*. Dans la fenêtre qui s'ouvre, ouvrir le menu d'édition de signature et paramétrer le simulateur à créer comme suit :

- Id du ware : `training.su.prod`
- Nom de la classe C++ : `SUSimulator`

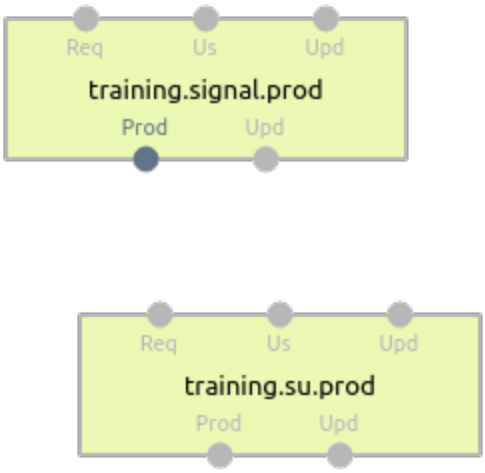
Cliquer sur *OK*, un nouvel onglet est apparu dans OpenFLUID-Builder, cet onglet va nous permettre (tout comme OpenFLUID-DevStudio) de construire et développer le simulateur.



Cliquer sur l'icone *Configurer* puis sur l'icone *Construire* afin de construire et d'installer le simulateur `training.su.prod`.

Aller dans l'onglet *Modèle* et cliquer sur *Ajouter un simulateur*, le simulateur `training.su.prod` apparaît désormais dans la liste des simulateurs disponibles. Sélectionner-le afin de l'ajouter à votre projet et positionner-le après le simulateur `training.signal.prod`.

Pour le moment ce nouveau simulateur ne fait rien et n'est pas en lien avec l'autre simulateur de votre projet. Cela peut notamment être vu en regardant la vue graphique du modèle couplé de votre projet. Cliquer sur l'onglet *Vue graphique*, les deux simulateurs apparaissent mais ne sont pas connectés entre eux.

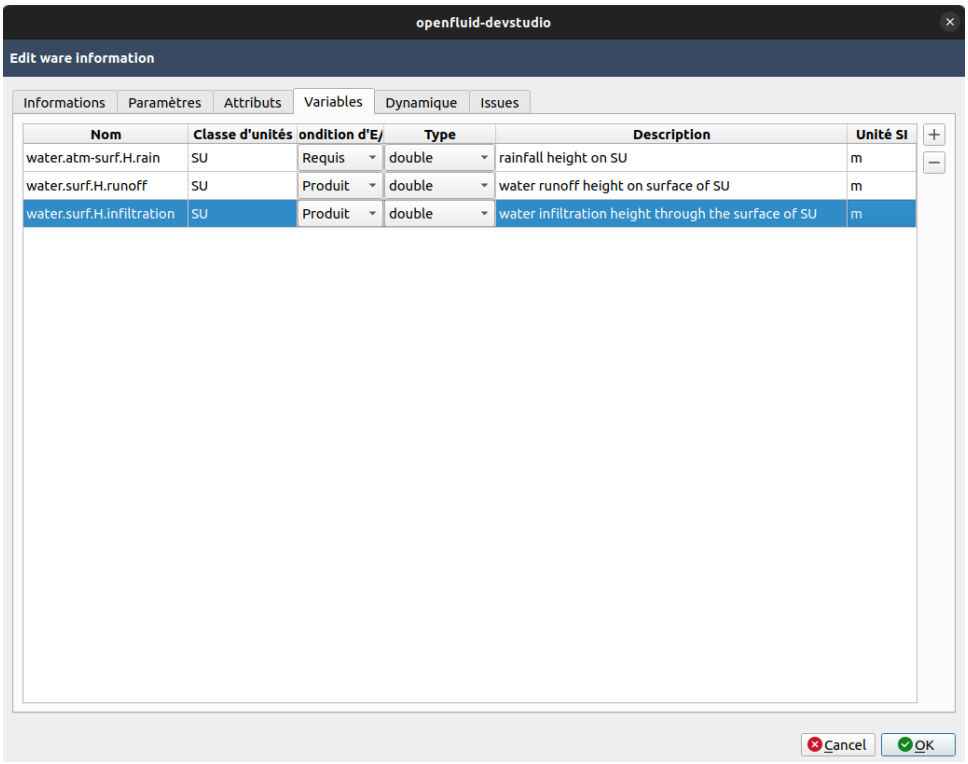


1.2 Signature

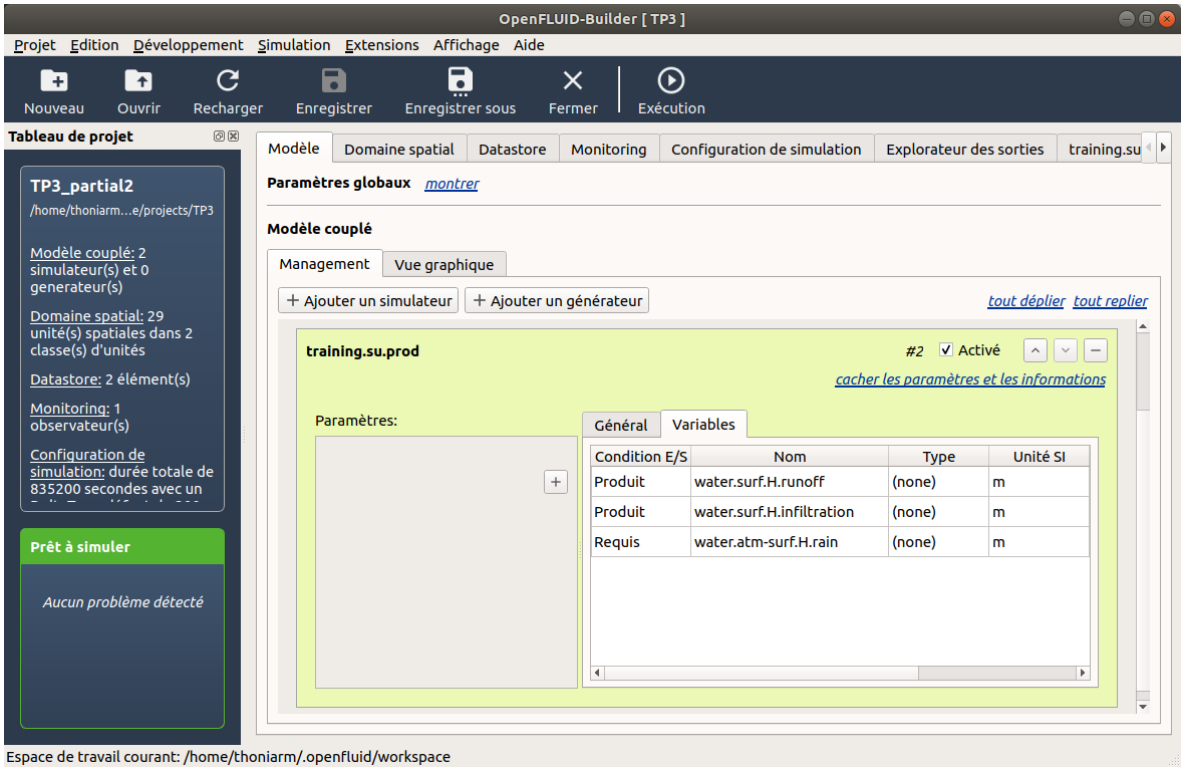
Retourner dans l'onglet `training.su.prod`, et dans `WareMain.cpp`. Mettre à jour la description ainsi que l'auteur du simulateur. Ajouter également un pas de temps par défaut. (Cocher *La planification utilise le DeltaT par défaut* dans l'onglet *Dynamique*)

Note: A chaque modification du code source du simulateur, ne pas oublier de sauvegarder les modifications puis de compiler le code source à l'aide de l'icone *Construire*.

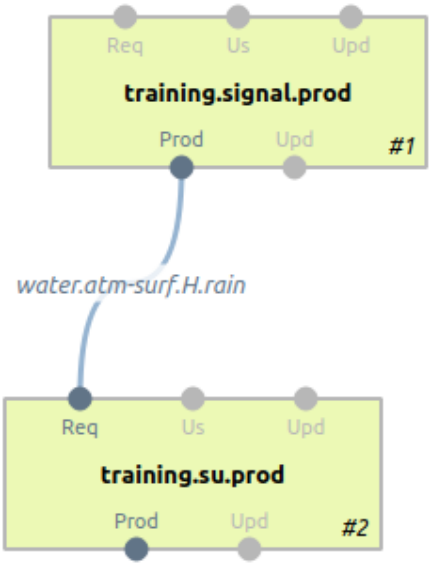
Ce simulateur génèrera des valeurs de ruissellement et d'infiltration à partir d'un signal de pluie produite par un autre simulateur. Nous allons donc déclarer la prise en compte de ce signal de pluie (variable `water.atm-surf.H.rain`) et la production des deux variables (`water.surf.H.runoff` pour le ruissellement et `water.surf.H.infiltration` pour l'infiltration).
La prise en compte d'une variable produite par un autre simulateur sera définie par la déclaration d'une variable requise dans la signature. Une fois complétée, la signature devrait être similaire à :



Aller dans l'onglet *Modèle*, *Management*, dans le simulateur `training.su.prod`, cliquer sur *montrer les paramètres et les informations*. Dans l'onglet *Variables*, sont apparues les différentes variables que va requérir ou produire le simulateur.



Aller dans l'onglet *Vue graphique*, les deux simulateurs du modèle couplé sont désormais couplés par l'échange de la variable `water.atm-surf.H.rain` produite par le simulateur `training.signal.prod` et requis par le simulateur `training.su.prod`.



Note: Les modifications du code source du simulateur (une fois sauvegardées et compilées) sont automatiquement prises en compte par OpenFLUID.

1.3 initializeRun()

Dans la méthode initializeRun() du simulateur `training.su.prod`, nous allons initialiser les variables produites à la valeur 0.
Une fois complétée, la méthode initializeRun() devrait être similaire à:

```
openfluid::base::SchedulingRequest initializeRun()
{
  openfluid::core::SpatialUnit* pSU;
  OPENFLUID_UNITS_ORDERED_LOOP("SU",pSU)
  {
    OPENFLUID_InitializeVariable(pSU,"water.surf.H.runoff",0.0);
    OPENFLUID_InitializeVariable(pSU,"water.surf.H.infiltration",0.0);
  }

  return DefaultDeltaT();
}
```

1.4 runStep()

Dans la méthode runStep(), nous allons calculer le partage ruissellement-infiltration à partir du signal de pluie en utilisant la méthode SCS(USDA, TR-55[1]).

Soit R le ruissellement, P la pluie, S le coefficient de rétention

$$\text{Si } P \leq (0.2 \cdot S) \text{ alors } R = 0.0$$

$$\text{Si } P > (0.2 \cdot S) \text{ alors } R = \frac{(P-0.2 \cdot S)^2}{(P+0.8 \cdot S)}$$

Pour les conditions de simulation de ce TP ($\Delta t = 300s$, $t_{total} = 9j16h$),
$0.000002 \leq S \leq 0.0008$

Note: Pour simplifier le TP, cette méthode de calcul ne prends pas en compte l'état initial du sol à chaque calcul. Il est possible d'améliorer le calcul sur ce point en introduisant un cumul de pluie. Cette amélioration ne sera pas traitée au cours de ce TP

Nous allons utiliser une boucle spatiale sur les SU, récupérer le signal de pluie au travers de l'instruction `OPENFLUID_GetVariable`, et produire le ruissellement et l'infiltration calculés à l'aide de deux instructions `OPENFLUID_AppendVariable`. Pour cet exercice nous fixerons la valeur de S à une valeur arbitraire choisie dans sa plage de validité.

Une fois complétée, la méthode runStep() devrait être similaire à:

```
openfluid::base::SchedulingRequest runStep()
{
  openfluid::core::SpatialUnit* pSU;
  openfluid::core::DoubleValue RainValue;
  openfluid::core::DoubleValue RunoffValue;
  openfluid::core::DoubleValue InfiltrationValue;
  double S;

  OPENFLUID_UNITS_ORDERED_LOOP("SU",pSU)
  {
    S = 0.00005;  // valeur du coefficient de retantion fixe à 0.00005
    // recuperation de la valeur du signal de pluie
    OPENFLUID_GetVariable(pSU,"water.atm-surf.H.rain",RainValue);

    // calcul du ruissellement selon la methode SCS
    RunoffValue = 0.0;
    if (RainValue > (0.2*S))
    {
      RunoffValue = std::pow(RainValue-(0.2*S),2)/(RainValue+(0.8*S));
    }

    // calcul de l'infiltration par deduction
```

[1] Technical Release 55: Urban Hydrology for Small Watersheds. USDA (U.S. Department of Agriculture). 1986. http://www.nrcs.usda.gov/Internet/FSE_DOCUMENTS/stelprdb1044171.pdf

```
    InfiltrationValue = RainValue-RunoffValue;

    // production de l'infiltration et du ruissellement
    OPENFLUID_AppendVariable(pSU,"water.surf.H.infiltration",
       InfiltrationValue);
    OPENFLUID_AppendVariable(pSU,"water.surf.H.runoff",RunoffValue);
  }

  return DefaultDeltaT();
}
```

2 Simulation

2.1 ... avec l'interface OpenFLUID-Builder

Lancer la simulation en cliquant sur l'icone *Exécution*. Et visualiser les résultats de la simulation dans le fichier `pw_outputs.pdf`.

2.2 ... en ligne de commande

Une fois complété, le fichier `model.fluidx` devrait être structuré comme suit:

```
<?xml version="1.0" encoding="UTF-8"?>
<openfluid format="fluidx 4">
  <model>
    <simulator ID="training.signal.prod" enabled="1">
    </simulator>
    <simulator ID="training.su.prod"  enabled="1">
    </simulator>
  </model>
</openfluid>
```

La commande à exécuter est donc :

```
openfluid run <Bureau>/formation/projects/TP3/ -c
```

(à taper sur une seule ligne)

Si tout s'est bien passé, les résultats de la simulation sont accessibles dans `<Bureau>/formation/projects/TP3/OUT`.