

# **TP1 : Création d'une fonction de simulation**

<b>Objectifs</b> :	Créer une fonction de simulation et de sa signature, construire et compiler pour
	une utilisation avec le moteur
Pré-requis:	TP0

### 1 Créer une fonction de simulation

Il existe 3 façons de créer le code source d'une une fonction de simulation :

- Ecrire le code source "à la main", à partir d'un code vierge : long, fastidieux, source d'erreurs, …
- Utiliser le buddy newfunc d'openfluid-engine : pratique, basique, ...
- utiliser le plug-in OpenFLUID pour Eclipse : facile, assisté, intégré à l'environnement de développement, ...

Nous allons utiliser cette 3ème possibilité pour cet exercice.

### 1.1 Lancement d'Eclipse

Au lancement d'Eclipse, veuillez choisir le chemin du workspace que vous souhaitez utiliser. Il est conseillé d'utiliser celui proposé par défaut (sous Linux : /home/nomutilisateur/workspace). Le workspace est le répertoire qui contiendra l'ensemble des projets de fonctions de simulations développées dans cet exercice ainsi que les suivants.

### 1.2 Création du projet C++

Pour créer un nouveau projet C++, aller dans le menu File > New > Project... Choisir C++ Project et cliquer sur Next

•	New Project	\$	⊡ ×
Select a wizard			
Create a new C++ project			
wizards:			
type hiter text			
👂 🗁 General			
▽ 🗁 C/C++			
🔂 C Project			
🔂 C++ Project			
👂 🗁 CVS			
👂 🗁 Java			
👂 🗁 JavaScript			
👂 🗁 PHP			
👂 🗁 Plug-in Development			
👂 🗁 SVN			
Texlipse			
👂 🗁 Web			
Examples			
	Back Next >	Cancel	

Nommer le projet formation.signal.prod dans le champ *Project name*, choisir *Makefile project / Empty project* comme *Project type*. La rubrique *Toolchains* est à positionner sur *Other Toolchain*. Cliquer sur *Finish* 

C++ Proj	ect 🗆 🗙
C++ Project	
Create C++ project of selected type	
Project name: MonProjet	
Use default location	
Location: /home/fabrejc/Labo/OpenFLUID	/Workspace/MonProjet Browse
Project type: T	oolchains:
Z C Executable	Other Toolchain
Empty Project	Linux GCC
Hello World C++ Project	
Shared Library	
Static Library	
▽ 🗁 Makefile project	
🗢 Empty Project	
Hello World C++ Project	
Show project types and toolchains only	y if they are supported on the platform
(?) < Back No	ext > Cancel <u>F</u> inish

### 1.3 Création du squelette de la fonction de simulation

Nous allons maintenant utiliser le plug-in OpenFLUID pour Eclipse afin de générer le code source "vide" d'une fonction de simulation.

Faire un clic-droit sur le projet que vous venez de créer et choisir *New > Other....* Dans la fenêtre qui s'ouvre, choisir *OpenFLUID > Simulation function wizard* et cliquer sur *Next*.

Sew New	= ×
Select a wizard	
<u>W</u> izards:	
type filter text	
😂 Folder	
👂 🗁 General	
▷ > C/C++	=
👂 🗁 CVS	
🕨 🗁 Java	
🕨 🗁 JavaScript	
▽ 🗁 OpenFLUID	
🔊 Simulation function wizard	
D 🗁 PHP	
N 🗠 Dhua ia Danuslaamaant	
(?) < Back Next > Cancel	

La fenêtre suivante est la 1ère étape de la création de la fonction de simulation. Cette première étape consiste à paramétrer ce qui va être créé.

Choisir SignalFunc.cpp comme nom de fichier (*Source file name*), SignalFunction comme nom de classe C++ qui contiendra la fonction (*Class name*). *Sources output* doit être positionné à single .cpp file, Create build system doit être coché, et *Build system* doit être positionné sur *CMake 2.6+*.

Cliquer sur Next une fois cette étape complétée.

•		
OpenFLUID simulation function		
Sources files and build system		
Container: //formation.signal.prod	Browse	
<b>A</b>		
Sources		
Source file name (.cpp): SignalFunc.cpp		
Sources output: single .cpp file   ≎		
Class name: SignalEunction		
✓ Create build system file(s)		
Build system		
Build system: CMake 2.6+		
Install directory: \$(HOME)/.openfluid/engine/functions		
Rack Next > Cancel		

La deuxième étape consiste à renseigner les informations sur la fonction de simulation, et notamment son identifiant (*ID*) par lequel elle va être connue. Taper formation.signal.prod comme ID de fonction. Les autres champs sont facultatifs, vous pouvez y renseigner votre nom et email. Cliquer sur *Next* une fois cette étape complétée.

•	E *
OpenFLUID simulat	tion function
Function ID and meta	a-information
Function ID:	formation.signal.prod
Function Name:	
Function Domain:	
Description:	3
Author(s):	Chuck Norris
Author(s)'s email(s):	norris@gmail.com
?	< Back Next > Cancel Finish

La troisième et dernière étape consiste à renseigner les informations sur les variables/paramètres/données d'entrées/évènements qui sont produits/utilisés/mis à jour par la fonction de simulation. Nous n'utiliseraons pas cette troisième étape, vous pouvez donc cliquer sur *Finish*.

Si tout s'est bien passé, vous devriez obtenir 3 fichiers dans le Project Explorer d'Eclipse :

- SignalFunc.cpp : fichier source de la fonction
- CMakeLists.txt : fichier du système de construction de la fonction (à priori, à ne pas modifier)
- CMake.in.config : fichier de configuration de la construction de la fonction

ainsi qu'un sous-répertoire \_build qui contiendra les résultats de compilation/construction de la fonction.

Project Explorer 🛿	E 🕏	<b>₽ ₽</b>
▽ 📂 formation.signal.prod		
Includes		
👝 _build		
SignalFunc.cpp		
📄 CMake.in.config		
🛕 CMakeLists.txt		

#### **1.4** Finaliser le projet C++

Avant de construire la fonction de simulation créée, il est nécessaire de paramétrer plus finement le projet Eclipse. Pour cela, nous allons nous rendre dans les propriétés du projet, en faisant un clic-droit sur le nom du projet, puis en cliquant sur *Properties*.

Tout d'abord, indiquer dans le champ *Build directory* que la construction de la fonction doit se faire dans le sous-répertoire \_build du projet.

\$	Properties for formation.signal.prod
type filter text	C/C++ Build 🗇 🗘 👻 👻
Resource Builders D C/C++ Build	Configuration: Default [ Active ]
<ul> <li>C/C++ General</li> <li>Project References</li> <li>Run/Debug Settings</li> <li>Server</li> <li>Task Repository</li> </ul>	Builder Settings  Behaviour Builder Builder External builder
Task Tags Validation WikiText	Image: Weight of the second
	Generate Makefiles automatically     Expand Env. Variable Refs in Makefile     Build location     Build directory: \${workspace_loc:/formation.signal.prod/_build}}
	Workspace) File system Variables
?	Hestore Derauits Apply

Ensuite passer sur l'onglet *Behaviour*, mettre la valeur install dans le champ *Build (Incremental build)* afin que la fonction de simulation soit automatiquement placée dans un répertoire connu du moteur de calcul openfluid-engine que nous utiliserons par la suite.

<b>\$</b>	
type filter text	C/C++ Build ⇔ ⇔
Resource       Builders       > C/C++ Build       > C/C++ General       Project References       Run/Debug Settings       Server       > Task Repository       Task Tags       > Validation       WikiText	Configuration:       Default [ Active ]       Image Configurations,         Builder Settings       Image Behaviour         Build settings       Image Build Settings         Image Stop on first build error       Image Description
	O Use optimal jobs number
	Note: See Workbench automatic build preference         Ø Build (incremental build)         Install         Ø Clean         Variables
?	Restore Defaults     Apply       OK     Cancel

## 2 Construire et installer la fonction de simulation

En premier lieu, il est nécessaire d'initialiser le système de construction. Pour cela, nous allons utiliser la ligne de commande du système d'exploitation. Les instructions sui suivent s'appliquent à un système Linux, mais peuvent très facilement être reproduites sous un autre système (Win32, MacOSX, ...)

Après avoir ouvert une console ligne de commande (Terminal), se rendre dans le sous-répertoire \_build du projet via la commande cd /home/nomutilisateur/workspace/formation.signal.prod/\_build Une fois positionné dans ce sous-répertoire \_build, exécuter la commande cmake ... Cette commande effectue un ensemble de vérifications et met en place le système de construction. Elle n'est nécessaire qu'à la première construction du projet, ou lors de la modification du fichier CMakeLists.txt ou CMake.in.config, ce qui ne devrait pas être nécessaire au cours du présent TP.

fabrejc@lisah-crampling: ~/workspace/formation.signal.prod/_build	
Fichier Édition Affichage Terminal Aide	
<pre>fabrejc@lisah-crampling:~\$ cd /home/fabrejc/workspace/formation.signal.prod/_build fabrejc@lisah-crampling:~/workspace/formation.signal.prod/_build\$ cmake The CXX compiler identification is GNU Check for working CXX compiler: /usr/bin/c++ Check for working CXX compiler: /usr/bin/c++ Detecting CXX compiler ABI info Detecting CXX compiler ABI info Detecting CXX compiler ABI info - done checking for medulo identified</pre>	
- found ofelib, version 1.4.2 Configuring done	=
Generating done Build files have been written to: /home/fabrejc/workspace/formation.signal.prod/_build fabrejc@lisah-crampling:~/workspace/formation.signal.prod/_build\$	
▶	~

La construction/installation peut se faire de deux manières :

- Soit en utilisant la ligne de commande en exécutant la commande make install depuis le sous-répertoire \_build
- Soit depuis Eclipse, dans le menu Project > Build Project. Cette 2ème méthode est à privilégier.

Cette construction/installation est à exécuter à chaque fois que le code source de la fonction de simulation est modifié.

Pour vérifier que la fonction a été correctement construite et installée, exécuter la commande :

openfluid-engine -f.

La fonction devrait alors apparaître dans la liste.

