

Fmu Based Simulation Framework Manuel du Développeur

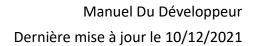






Table des matières

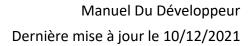
1.	INTRODUCTION	
1.1	Glossaire	5
1.2	Références	5
4.2	Présentation générale	,
1.3	Presentation generale	
2.	LE SYSTEME EXECUTIF	9
2.1	Mode simulation	
2.2	Mode batch nogui	g
2.3	Mode full batch	
2.4	Mode backtrack	10
2.	.4.1 Spécificité avec une configuration time-depend (Flag SimuMpc) :	11
2.5	Mode rejeu	11
3.	CONFIGURATION DE L'APPLICATION	12
3.1	Section simulation	12
3.	.1.1 Performance meter	14
3.2	Section séquences	14
3.3	Section Node	15
3.4	Section models	15
3.	.4.1 Modèle de type manuel	16
3.	.4.2 Modèle de type FMU	16
3.	.4.3 Modèle de type CAO (FbsfEditor)	17
3.5	Section PluginsList pour les plugins qml	17
_		
4.	DEVELOPPEMENT DE MODELES	18
4.1	Module de type manuel	18
4.	.1.1 Gestion des Paramètres globaux du module (config xml)	





4.1.2	Généralisation de la notion de paramètre de la norme FMI	21
4.1.3	Api de connexion aux données publiques	22
4.1.4	Abonnement à la ZE	24
4.1.5	Publication dans la ZE	24
4.1.6	API de contrôle du système exécutif	25
4.1.7	Compilation d'un module manuel	25
4.2 Mo	dule de type Graphique et plugin Qml	26
4.2.1	Gestion des paramètres pour un modèle CAO de FbsfEditor	27
4.2.2	Gestion de la ZE	27
4.2.3	Abonnement à la ZE	27
4.2.4	Publication vers la ZE	28
5. AP	PI DE CONTROLE	29
5.1 Con	ncepts	29
5.2 Met	thodes	29
5.2.1	Gestion d'instance	29
5.2.2	Paramétrage de la simulation	29
5.2.3	Contrôle de simulation	29
5.2.4	Acquisition de l'état de la simulation	30
5.3 Con	npilation	31
6. AN	INEXES	32
6.1 Inst	tallation du Framework	32
6.1.1	Prérequis	32
6.1.2	Procédure d'installation	32
6.1.3	Principes généraux d'organisation	33
6.2 Dév	veloppement de modèles	33
7. GE	STION DU TEMPS SIMULE	36
7.1 Exe	mples d'affichage	37
7.1.1	Default : aucune spécification	
7.1.2	Spécification de date et temps avec formats	37
7.1.3	Spécification d'instant sans format	38
7.1.4	Spécification d'instant avec format cumulatif	38
7.1.5	Cumul en secondes :	39

L3S, 9 Avenue Paul Verlaine 38030 GRENOBLE CEDEX 2— France - Tel: + 33 952567220 - Fax: + 33 957567220 - Web site: http://www.L-35.fr
SARL au capital de 50 000 EUR — Grenoble B753885045 — SIRET: 75388504500017





7.2	Formattage	. 40
7.3	Formattage de date	40
7.4	Formattage de temps	. 40
7.5	Accessibilité	. 41
7.6	Exemples C++	42



1. Introduction

1.1 Glossaire

Application Decomposing Interfere	
Application Programming Interface	
Langages de programmation de haut niveau standardisés	
Logiciel libre de gestion de versions décentralisé	
Interface Homme Machine	
Jeu de Données	
Qt Markup Langage	
Logiciel Simulation Services & Support	
Model View Controller (Patron de conception)	
Analyseur syntaxique	
Fonctionnalité qui permet de parcourir l'historique de l'évolution de	
variables disponibles à l'IHM	
Mode spécifique pour des applicatifs utilisés dans les simulateurs	
d'optimisation. Convertie les vecteurs de la ZE en vecteurs temporels	
définissants les valeurs du passé et futur.	
Permet d'indiquer aux objets QML que le mode TIME	
DEPEND est actif.	
Fmu Based Simulation Framework	
Editor de document CAO (Graphique ou Logic) natif de FBSF	
Functional Mock-up Interface, définit une interface normalisée à utiliser	
dans les simulations.	
Functional Mock-up Unit, module de simulation à la norme FMI	
Langage de structuration de données, utilisé notamment pour la	
configuration d'un simulateur FBSF	
Zone d'Echange pour les variables publiques du simulateur	

1.2 Références

- Norme Fmi (https://www.fmi-standard.org)
- Documentation Qt et Qt Quick Control (https://doc.qt.io/)
- Manuel utilisateur Fbsf du « Plotter »
- Manuel utilisateur Fbsf du « Monitor »
- Manuel utilisateur Fbsf de l'« Application Tool Bar »
- Manuel utilisateur FbsfEditor



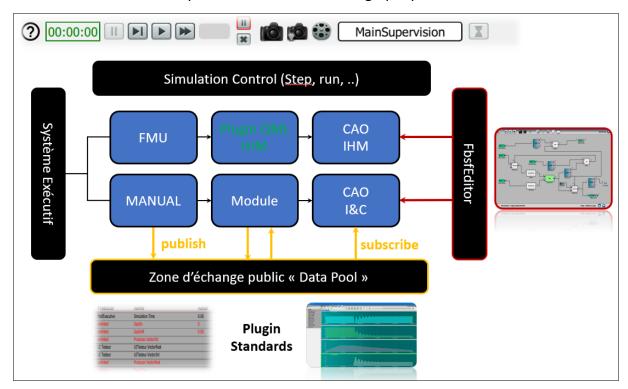
1.3 Présentation générale

FBSF est un environnement multiplateformes / multi-machines d'intégration de fonctions de simulation. Il propose un environnement d'exécution et des outils de contrôle et de supervision des fonctions de simulation.

FBSF est aussi un moteur de « co-simulation » permettant d'intégrer n'importe quel modèle **FMU** de simulation compatible avec la norme FMI 2.0.

Il intègre notamment un système exécutif pour applications modulaires comportant les fonctionnalités suivantes :

- Contrôle du cycle de vie des modules fonctionnels instanciés,
- Interconnexion des données « publiques » produites et consommées entre modules ZE,
 - o API d'abonnement et de publication (publish, subscribe)
- Contrôle du cadencement de modules fonctionnels,
- Communications asynchrones entre interfaces graphiques et modèles.



Les notions de module et interface sont implémentées par les concepts de : Module édité manuellement (développement ou encapsulation de codes scientifiques, interfaces avec applications industrielles temps réel, ...)

Module FMI 2.0 (Functional Mockup Unit),



- Editeur CAO de modèle de contrôle commande,
- Editeur CAO de pages graphiques,
- Plugin d'Interface graphique fondée sur la technologie QML.

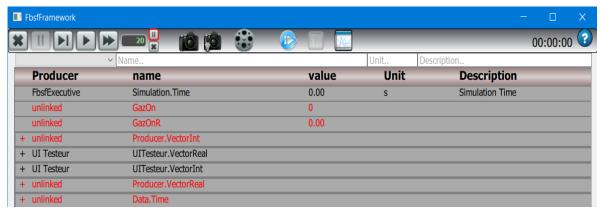
Le concept de Framework désigne l'ensemble des fonctions exécutives et les outils mis à disposition de l'utilisateur.

• Un bandeau de contrôle de la simulation avec accès aux IHM applicatives

Le contrôle de la simulation avec les fonctions suivantes

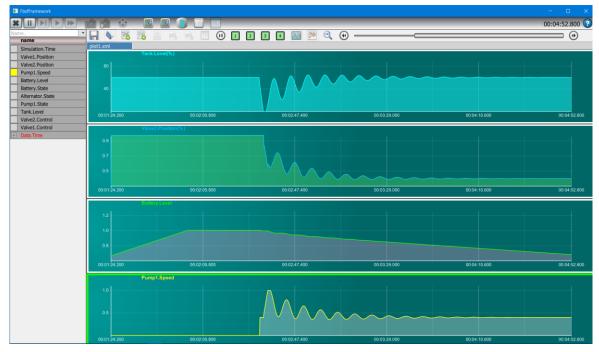
00:00:13 00:00:13	Temps de calcul de la simulation (Grisé quand le simulateur est en mode calcul)	
	Contrôle simulation : pause, pas à pas, marche, accéléré et indicateur d'erreur avec accès au logfile	
20	Spécification du nombre de pas en marche et accéléré Indication de l'action en fin d'exécution : Pause ou arrêt	
(6) (9)	Ecriture et lecture d'état sur/depuis un fichier	
	Enregistrement de l'historique sur un fichier pour le rejouer.	
La visualisation d'IHM de contrôle et de supervisions par sélecteur de vues		
	Affichage du Plotter Fbsf	
Plotter		
	Affichage du Moniteur Fbsf	
MainSupervision Monitor Plotter	Affichage du Moniteur Fbsf Affichage d'IHM de supervision	

Un moniteur de données pour visualisation des données publiques.





• Un afficheur de courbes évolué pour visualisation des profils temporels des données publiques.



Cet afficheur de courbes peut être configuré pour convertir des valeurs dans des unités décrites à travers le fichier « **DisplayUnit.csv** ».

Pour être pris en compte, ce fichier doit être positionné à la racine du projet de la variable d'environnement **APP_HOME** et sa syntaxe est la suivante :

Unit;DisplayUnit;a;b degC; K;1;273.15 Pa;Bar;1.00E-05;0 pa;Bar;1.00E-05;0

Note:

La définition et configuration d'un simulateur FBSF se fait par édition d'un fichier xml qui est passé en argument de l'exécutable FbsfFrameWork.exe.

A titre d'exemple, le cadencement et l'ordre d'exécution des modules sont indiqués à ce niveau.

La syntaxe et les différentes options de ce fichier sont explicitées dans le chapitre 3.



2. LE SYSTEME EXECUTIF

Le rôle du système exécutif est de prendre en charge les modèles et interfaces graphique du point de vue cadencement synchronisé et de l'interconnexion des données produites et consommées de la **ZE**.

L'organisation par séquence permet de :

- Paralléliser l'exécution des modèles par groupes,
- De sous itérer durant un pas de calcul pour un groupe de modèles.

Note:

Dans une séquence de modèles, le flux des données est alimenté en mode « pipe-line », c'est-à-dire que le calcul du modèle précédent est immédiatement disponible pour le modèle suivant.

Pour garantir la reproductibilité des modèles couplés de façon explicite, il est préconisé de les exécuter dans la même séquence.

<u>Mode sous itérations</u>: Une séquence qui sous itère exécute son groupe de modèle N fois dans un pas de calcul (TimeStep). Les données sont également propagées en mode « pipeline » pour la séquence qui sous itère.

2.1 Mode simulation

L'ensemble des modèles est exécuté selon un cadencement prédéfini avec possibilité d'interagir avec le processus au moyen de l'interface graphique.

Arguments en mode simulation": FbsfFramework.exe config.xml

2.2 Mode batch nogui

Ce mode consiste à exécuter la simulation y compris avec des modules de type ModuleLogic présents dans la configuration, sans IHM visible.

Les modèles s'exécutent sans aucune interface graphique et en séquence conformément à la configuration jusqu'à une instruction de fin d'exécution contrôlée par un module applicatif. Pour exécuter un ordre d'arrêt de la simulation, un module manuel dispose d'une méthode de contrôle des états du système exécutif (voir le chapitre 4.1.6 Erreur! Source du renvoi i ntrouvable.).

Arguments en mode batch : FbsfFramework.exe config.xml -no-gui

2.3 Mode full batch

Ce mode a pour objectif de réduire les ressources Qt nécessaire au fonctionnement de l'application. Il permet exécuter la simulation sur la base de module manuels ou Fmu, mais



sans aucun module de type Visual (ModuleLogic, ModuleGraphic) présents dans la configuration.

Pour produire un jeu de binaires minimal pour le mode full batch :

Dans QTCreator: Pour activer le build du mode full batch on ajoute une "build configuration" BATCH dans la page du Build du projet FBSF avec en Additional arguments du Build step "qmake": CONFIG+=BATCH

Build Steps

qmake: qmake.exe FBSF.pro Details ▲		
qmake build configuration:	Release	
Additional arguments:	CONFIG+=BATCH	

Avec un script Build.bat : script de build en ligne de commande

- Avec argument "batch" execution du build full batch
- Sans argument: prompt le choix release ou debug.

Ressources binaires nécessaires pour un mode batch (hors application APP HOME):

- **FbsfBatch.exe** (Executable full batch)
- FbsfBaseModel.dll, FbsfPublicData.dll et FbsfNetwork.dll (librairies FBSF)
- Qt5Core.dll, Qt5Network.dll et Qt5Xml.dll (librairies Qt)
- 7z.dll et 7z.exe (ressource pour les FMU)

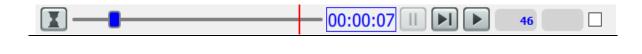
Arguments en mode simulation": FbsfBatch.exe config.xml

Note: Les jeux de librairies binaires full batch ne sont pas compatibles avec les autres modes. Il convient donc de les exporter avec l'exécutable et les ressources dans un dossier dédié à l'exécution d'une simulation en mode full batch.

2.4 Mode backtrack

Seule une IHM de supervision réalisée avec « FbsfEditor Graphic » ou à travers un plugin qml « images à façon » est éligible à ce mode qui permet de parcourir l'historique de l'évolution de variables disponibles à l'IHM depuis le temps t=0 jusqu'au moment de l'activation de ce mode.

L'utilisateur peut se déplacer dans le passé d'une simulation et voir les IHM Graphiques se mettre à jour avec les valeurs du passé. L'utilisateur peut se déplacer dans le passé, pas à pas ou rejouer en continue une période de temps.





	Bouton d'activation du mode uniquement pour les IHM
-	Slider de contrôle de l'historique du back track
00:00:07	Temps actuel de l'historique
	Symbolise le temps présent en mode time depends
	Contrôle de l'historique : pause, pas à pas, marche
46 1	Nombre de pas, Nombre de FPS et boucle infini pour le run

Note:

Pour être éligible à ce mode, les plugin qml « images à façon » devront déclarer une variable dans leur fichier « main.qml » : property bool backtrackable : true

2.4.1 Spécificité avec une configuration time-depend (Flag SimuMpc) :

Une barre rouge détermine la limite entre le passé et le futur accessible par le slider.

Le choix pris dans la bibliothèque graphique standard de « FbsfEditor » est de se positionner à la dernière valeur du passé en mode run simulation.

2.5 Mode rejeu

Deux cas possibles:

- **Rejeu partiel :** Une partie des modèles est rejoué, une autre partie continue à calculer. Ceci permet de rejouer plusieurs scénarios pour étudier les variations de réponse d'un ensemble de modèles.
- **Rejeu complet** : Le calcul est inactif, le processus de simulation est rejoué intégralement. Un slider permet de reprendre le rejeu en tout point entre début et fin.

Arguments en mode rejeu : FbsfFramework.exe config.xml -r[eplay] file.dat



3. CONFIGURATION DE L'APPLICATION

Le framework FBSF intègre des outils graphiques configurés en tant que plugins standards (moniteur et traceur de courbes). Le ficher de configuration « *plugins.xml* » est intégré aux fichiers d'installation du framework mais il peut être placé dans le répertoire de base de l'application pour le spécialiser.

On peut par exemple supprimer le moniteur ou ajouter des pages de courbes à charger lors de l'initialisation de l'application.

```
<?xml version="1.0" encoding="UTF-8"?>
</tems>
</pluginsList>
</plugin>
</name>Monitor</name>
</publicle>
</plugin>
</pluginsList>
</pluginsList>
```

Une application développée à l'aide de FBSF se lance par l'appel de l'exécutable « FbsfFramework.exe » et d'un fichier de configuration xml en argument dont le nom est libre.

Ce fichier de configuration décrivant l'ensemble de l'application est sous format xml, il comporte plusieurs sections décrites dans les chapitres suivants :

3.1 Section simulation

Définition des paramètres généraux de la simulation.

```
<simulation version="1.0">
       <timestep>0.1</timestep>
       <speedfactor>0.5</speedfactor>
       <simuMpc>false</ simuMpc>
       <perfMeter>false</perfMeter>
       <dataFlowGraph>false</dataFlowGraph>
       <publishparam>true</publishparam>
       <snapshotparam>true</snapshotparam</pre>
       <hidePauseBtn>true</hidePauseBtn>
       <hideStepBtn>true</hideStepBtn
       <hideRunBtn>false</hideRunBtn>
       <hideSpeedBtn>false</hideSpeedBtn>
       <hideStepCrtBtn>false</hideStepCrtBtn>
       <hideSnapControlBtn>false</hideSnapControlBtn>
       <hideNavigationBtn>false</hideNavigationBtn
        <hideBacktrackBtn>false</hideBacktrackBtn>
</simulation>
```

L3S, 9 Avenue Paul Verlaine 38030 GRENOBLE CEDEX 2—France - Tel: + 33 952567220 - Fax: + 33 957567220 - Web site: http://www.L-3S.fr



Les paramètres généraux sont :

• <timestep> Période de cadencement en secondes ([0.5])

• < speedfactor> Facteur d'accélération (<1) et de ralenti (>1) ([1.0])

 <simuMpc> Flag permettant aux objets de la Library « Logic ou Graphic » de savoir si le mode "time depend" est actif ([false])

 <perfMeter> Flag permettant la journalisation et la publication en zone d'échange des temps de calcul ([false]). Voir 3.1.1

• < dataFlowGraph > Flag permettant de générer le flux de données de la zone d'échange dans un fichier « DataFlowGraph.dot » ([false])

• < hidePauseBtn> permet de cacher le bouton associé de la Tool Bar ([false])

<Publishparam> voir chapitre 4.2.1 ([false])
 < snapshotparam > voir chapitre 4.2.1 ([false])

 <initialPlotList> Fichier de définition des fenêtres de courbes à afficher à l'initialisation du simulateur (en relative par rapport à la variable APP_HOME)

<intialimage> Choix du plugin à afficher à l'initialisation du simulateur (valeur « name » du module souhaité ex Monitor, Plotter, ...)

En mode standard, pour un affichage date UTC avec temps UTC initial [optionnel], ajouter :

<timeformat>utc</timeformat> <timestart>2015-03-25T12:00:00Z</timestart>

Pour le format date voir : https://www.w3schools.com/js/js date formats.asp

En mode "time depend" : le mode UTC est activé par défaut.

Note:

Le format du fichier de sortie « DataFlowGraph.dot » est au standard du langage DOT

Chaque donnée échangée dans la ZE est identifiée par les attributs suivants :

Forward: au sein d'un même cycle, la donnée est systématiquement consommée après d'avoir été produite

Backward: au sein d'un même cycle, la donnée est systématiquement consommée avant d'avoir été produite (i.e., la donnée consommée est en réalité celle du cycle précédent, ou la valeur d'initialisation pour le premier cycle)

Ambiguous: la provenance réelle de la donnée est ambiguë ou changeante selon les cycle -> cas des flux entre séquences ou entre sous-séquences, voir plus loin

Duplicate : la donnée a été publié par plusieurs modules, et seul le dernier module l'ayant publié est considéré comme producteur. Les autres arcs seront indiqués et marqués comme « duplicate »



3.1.1 Performance meter

Le fichier PerfMeter.csv est produit dans le working directory et se structure comme suit :

- Les colonnes : phase d'exécution, type d'élément, nom de l'élément, mesure du temps cpu.
- Les 3 sections correspondant aux appels dolnit(), doStep() et doTerminate() : initialisation, récurrence des cycles avec step numéroté et finalisation.
- Chaque section présente les informations structurées selon la hiérarchie d'éléments de la configuration (Executive/Sequence(s)/Module(s)).

	Туре	Name	CpuTime
Init	Executive		0
Init	sequence	Principale	10
	module	Producer	10
step 1	Executive		0
step 1 sequence		Principale	28
	module	Producer	28
step 2	Executive		2
step 2 sequence Principale		19	
	module	Producer	19
Final	Executive		0
Final	sequence	Principale	0
	module	Producer	0

3.2 Section séquences

Définition des processus de cadencement et ordonnancement des modèles < sequences >

Une séquence est caractérisée par :

L3S, 9 Avenue Paul Verlaine 38030 GRENOBLE CEDEX 2- France - Tel: + 33 952567220 - Fax: + 33 957567220 - Web site: http://www.L-3S.fr
SARL au capital de 50 000 EUR - Grenoble B753885045 - SIRET : 75388504500017



- <name> Son nom d'instance
- <period> Facteur de sur/sous itération (optionnel avec défaut=1)
 - "period" > 1 pour un mode « basse fréquence »
 - "period" < 1 pour un mode « haute fréquence »</p>
- Sous-Section < models> : Les types de modèles instanciés (manuel, fmu)

3.3 Section Node

```
<sequences>
<sequence>
<models>
   <model> ...... </model>
   <node>
               <SubSequences>
                       <SubSequence>
                           <name>.....</name>
                           <models>
                              <model> ...... </model>
                              <model> ...... </model>
                              <model> ...... </model>
                            </models>
                       </SubSequence>
                       <SubSequence>
                           <name>.....</name>
                            <models>
                              <model> ...... </model>
                            </models>
                       </SubSequence>
               </SubSequences>
   </node>
   <model> ...... </model>
</models>
</sequence>
</sequences>
```

Note:

Les nœuds « *nodes* » ont un fonctionnement qui se rapproche des séquences, mais se place au même niveau qu'un modèle pour permette des exécutions en parallèle avec des points de synchronisation. On peut de façon illimitée inclure des nœuds dans des nœuds parents.

3.4 Section models

Dans cette section chaque module à instancier doit être défini par des attributs dépendants de son type.



3.4.1 Modèle de type manuel

Un modèle manuel est caractérisé à minima par :

<module> Son identifiant informatique (radical du nom de la .dll,.so),

<name> Son nom d'instance<type> Son type manual

<param1> Paramètre utile dans le code du module voire chapitre 4.1.1

3.4.2 Modèle de type FMU

L'objectif du standard FMI est de fournir une interface standard pour coupler deux ou plusieurs outils de simulation dans un environnement de cosimulation.

Dans ce cadre, la communication entre les sous-systèmes est réduite à des points de communication discrets, entre lesquels les sous-systèmes sont traités de manière indépendante par leurs solveurs respectifs. Le système exécutif FBSF se comporte en algorithme maître du cadencement et du contrôle des échanges de données entre les sous-systèmes et la synchronisation de tous les solveurs esclaves.

Le modèle FMU est distribué sous forme d'archive et décompressé par le module générique FBSF au chargement de l'application.

Les données d'interface du FMU connectées avec les données publiques sont :

- Les causalités « outputs » publiées produites sous leur identifiant FMU
- Les causalités « inputs » publiées consommées sous leur identifiant FMU

La configuration d'un module FMU est la suivante

• <name> Son nom d'instance



<path> Chemin d'accès relatif au répertoire d'exécution (chemin du fichier .fmu)

• <type> Son type fmu

<starttime> Un temps de départ ([0.0])

<timestep> Un pas de temps de calcul ([timestep de la section générale])

<dumpcsv> Une option dump des variables (true/[false])

<prefixinput> Préfixe input et paramètre fixed avec nom de module (true/[false])

 <prefixoutput> Préfixe output et paramètre tunable avec nom de module (true/[false])

3.4.3 Modèle de type CAO (FbsfEditor)

Un modèle Visual Graphic ou Logic est caractérisé par :

<module> ModuleLogic ou ModuleGraphic

<name> Son nom d'instance<type> Son type visual

<document> Chemin d'accès relatif au document qml

3.5 Section PluginsList pour les plugins qml

Les interfaces graphiques définies comme des plugins < PluginsList>

```
<PluginsList>
    <Plugin>
    <name>IHM1</name>
    <path>ModuleIHM</path>
    </Plugin>
</PluginsList>
```

Un plugin graphique est caractérisé par :

<name> Son nom d'instance

<path> Chemin d'accès au répertoire d'exécution (chemin du fichier main.qml)



4. DEVELOPPEMENT DE MODELES

4.1 Module de type manuel

Un module de type manuel permet à l'utilisateur de réaliser son propre modèle physique ou d'encapsuler un code de calcul tiers dans le formalisme FBSF.

Chaque fonction de simulation, ou module, est implémentée comme une librairie dynamique (dll, .so) chargée dynamiquement avec l'application. La description du séquencement et de l'ordonnancement des modèles est réalisée au moyen d'un fichier de configuration.

Le module est référencé par :

- Son type identifié par le nom (radical) de la librairie dynamique,
- Son nom d'instance dans la configuration.

Il peut être instancié plusieurs fois, à condition de lui donner un nom d'instance unique dans la configuration.

Un module de type manuel est implémenté selon une interface normalisée (**héritage de la classe FBSFBaseModel**) vis-à-vis du système exécutif qui lui permet de contrôler le cycle de vie du code :

- Phase initial: chargement des ressources du module (dolnit())
- Phase calcul: exécution du pas de calcul du module (doStep())
- Phase terminale : libération des ressources (doTerminate())

Un module manuel (ex : <u>ModuleA</u>) est une classe C++ déclarée et définie comme suit par son .h et son .cpp :

```
#ifndef ModuleA H
#define ModuleA_H
#include <QtCore/qqlobal.h>
#if defined(ModuleA_LIBRARY)
# define ModuleA SHARED EXPORT Q DECL EXPORT
# define ModuleA SHARED EXPORT Q DECL IMPORT
#endif
#include "FbsfBaseModel.h"
class ModuleA_SHARED_EXPORT ModuleA
: public FBSFBaseModel
public:
         ModuleA();
                   doInit();
         Int
                   doTerminate();
         QMap<QString, ParamProperties> getParamList(){return mListParam;};
private :
          QMap<QString, ParamProperties> mListParam;
extern "C" Q_DECL_EXPORT FBSFBaseModel* factory()
```



```
return new ModuleA();
}
#endif // ModuleA_H
```

Exemple de fichier de déclaration ModuleA.h

```
#include "ModuleA.h"
// Constructor
ModuleA::ModuleA()
: FBSFBaseModel()
 member(0)
// Initialization step
int ModuleA::doInit()
          // do initialization
          return 1;
// Computation step
int ModuleA::doStep()
          // do step computation
          return 1;
// Finalization step
int ModuleA::doTerminate()
          // do finalization
          return 1:
```

Exemple de fichier d'implémentation ModuleA.cpp

4.1.1 Gestion des Paramètres globaux du module (config xml)

Il est fortement conseillé d'utiliser la classe définie dans « *ParamProperties.h* » livrée par le framework au niveau de « *FbsfBaseModel* » :

```
enum class Param_quality: int
  cMandatory,
  cOptional
enum class Param_type: int
  cStr,
  cDbl,
  cInt.
  cBool,
  cPath,
  cDateAndTime,
  cDateAndTimeUTC,
  cEnumString,
  cEnumInt
class ParamProperties
public:
  ParamProperties()= default;
```



```
ParamProperties(const Param_quality aP_qual,
        const Param_type aP_type,
        const QString & aDescription,
        const QString & aUnit,
        const QVariant & aHint,
        const QVector<QVariant> & aEnumEntries = QVector<QVariant>(),
        const QVariant & aDefault = QVariant(),
        const QVariant & aMin_strict = QVariant(),
        const QVariant & aMax_strict = QVariant(),
        const QVariant & aMin_warn = QVariant(),
        const QVariant & aMax_warn = QVariant()
 mP_qual(aP_qual),
 mP_type(aP_type),
 mDescription(aDescription),
 mUnit(aUnit),
 mHint(aHint),
 mEnumEntries(aEnumEntries),
 mDefault(aDefault),
 mMin_strict(aMin_strict),
 mMax_strict(aMax_strict),
 mMin_warn(aMin_warn),
  mMax_warn(aMax_warn)
Param_quality mP_qual;
Param_type mP_type;
/// A comment to describe the field
QString mDescription;
/// A unit for numeric types
QString mUnit;
/// Hint value (to guide user)
QVariant
          mHint;
/// List of possible entries for enumarated types
QVector<QVariant> mEnumEntries;
/// Default value in case of optionnal parameter
QVariant
           mDefault;
/// Bounding variants for ordered types
QVariant mMin_strict, mMax_strict;
QVariant mMin_warn, mMax_warn;
```

Il est possible de déclarer un « Prototype » que les paramètres (déclaré dans le fichier xml de la simulation) devront suivre. Pour ce faire il faut ajouter une variable mListParam du type Qmap<QString, ParamProperties> (dans le .h du module) de cette manière :

```
QMap<QString, ParamProperties> mListParam =
{
    {"document", //le nom du paramètre (la key pour la Qmap)
    ParamProperties{
```



```
Param quality::cOptional, //le paramètre est il obligatoire ou non
Param_type::cStr, // le type du paramètre (string, int etc...)
"Path to the Document", // descrition du paramètre
"", // unité du paramètre (ex:cm, $, ...)
"ModuleTank/GraphicTank.qml", // un exemple de valeur possible
{},//list pour les enum type (optionnel et inutile ici)
"ModuleMetz/GraphicMetz.qml"} // valeur par default (pour les paramètres cOptional)
},
{"testint", // Autre exemple (du type int)
ParamProperties{Param_quality::cMandatory, Param_type::cInt, "un int", "", 12, // un exemple de valeur
possible
{},"",
1, 20, // Limite basse et haute de la valeur (erreur critique) (optionnel)
5, 15 // Limite basse et haute de la valeur (affiche un warning) (optionnel)
{"testenumstring", // Autre exemple (du type EnumString)
ParamProperties{Param_quality::cMandatory, Param_type::cEnumString, "enum string", "", "", {"OUI",
"NON"}}}
}; // fin de la variable
```

Un code manuel est capable de récupérer des paramètres de configurations déclarés dans la section globale du fichier de configuration avec le code suivant :

```
MTimeStep=AppConfig()["timestep"].toFloat();
```

et ceux déclarés dans sa section < model > du fichier de configuration avec le code suivant :

Int Param1=config()["param1"].toInt();

4.1.2 Généralisation de la notion de paramètre de la norme FMI

La gestion des paramètres de la norme FMI établit la classification identifiée par le mot clé *variability* :

- Si fixed: le paramètre est constant, il n'est pas modifiable en cours de calcul. Sa valeur est déterminée et fixée lors de la conception du modèle. Un paramètre de type fixed est déclaré comme un exporteur.
- Si tunable: le paramètre est modifiable avant le lancement du calcul et en cours de calcul entre chaque pas de temps. Un paramètre de type tunable est déclaré comme un importeur.

Une classe intermédiaire "FBSFBaseModelFMI" implémente le parsing des fichiers de type « *modelDescription.xml* » et déclare les paramètres "*fixed*" et "*tunable*" du module manuel.

La classe FBSFBaseModel implémente une API pour déclarer ces paramètres (*fixed* et *tunable*).



exemple ModuleFMI.h:

```
class ModuleFMI_SHARED_EXPORT ModuleFMI
: public FBSFBaseModeeFMI
example ModuleFMI.cpp:
int ModuleFMI::doInit()
{
        QString file=config ()["description"];
        return parseModelDescription(file);
}
int ModuleFMI::doStep()
{
        FmiVariable t1=parameter("Tunable.k");
        qDebug () << t1.name() << t1.value();
        FmiVariable c1=parameter("Constant.k");
        qDebug () << c1.name() << c1.value();*
        return FBSF_OK;
}
```

La configuration du module spécifie le nom du fichier de description :

4.1.3 Api de connexion aux données publiques

Une donnée publique de la **ZE** est identifiée par un nom, une adresse locale, une unité et un commentaire.

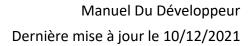
Les données sont connectées par identité de nom. Pour connecter une donnée produite avec une donnée consommée, il faut publier la donnée produite sous le même nom que la donnée consommée déclarée par abonnement.

Les données publiques sont des scalaires ou vecteurs 1D de type informatique *integer* et *float* (32 bits).

Les vecteurs sont :

- Des vecteurs de données dont chaque élément se comporte comme un scalaire vis-àvis du temps.
- Des vecteurs « time-depend » (applications d'optimisation). Ce type de vecteur permet de déclarer des séquences de valeurs temporelles avec une notion de passé et futur. La notion de décalage par pas « timeshift » est paramétrable ainsi que le pivot passé/futur (index).

Ce mode « *time-depend* » est activé par la publication d'un vecteur temps nommé «*Data.Time*» et la présence de la variable SimuMpc dans la section globale de la





configuration.



4.1.4 Abonnement à la ZE

Scalar int	subscribe (QString name, int *address,QString unit, QString comment);
Scalar float	subscribe (QString name, float *address,QString unit, QString comment);
Data vector int	subscribe (QString name, QVector <int>* vector, QString unit, QString comment);</int>
Data vector float	Subscribe (QString name, QVector <float>* vector, QString unit, QString comment);</float>
Time depend vector int	<pre>subscribe (QString name, QVector<int>* vector, QString unit, QString comment int timeShift, int index);</int></pre>
Time dépend vector float	<pre>subscribe (QString name, QVector<float>* vector, QString unit, QString comment int timeShift, int index);</float></pre>

Note:

Un vecteur déclaré en abonnement n'a pas de taille, il sera retaillé automatiquement selon la taille du vecteur publié.

4.1.5 Publication dans la ZE

Scalar int	<pre>publish (QString name, int *address,QString unit, QString comment);</pre>
Scalar float	<pre>publish (QString name, float *address,QString unit, QString comment);</pre>
Data vector int	<pre>publish (QString name, QVector<int>* vector, QString unit, QString comment);</int></pre>
Data vector float	<pre>publish (QString name, QVector<float>* vector, QString unit, QString comment);</float></pre>
Time depend vector int	<pre>publish (QString name, QVector<int>* vector, QString unit, QString comment int timeShift, int index);</int></pre>
Time dépend vector float	<pre>publish (QString name, QVector<float>* vector, QString unit, QString comment int timeShift, int index);</float></pre>



4.1.6 API de contrôle du système exécutif

La méthode suivante permet le contrôle par programmation des états et modes du système exécutif :

ExecutiveControl (« commande » [, «arg1»[, «arg2»]])

Avec l'argument « **commande** » de type **Qstring** pouvant prendre les valeurs présentées dans le tableau. L'argument optionnel est de type **Qstring**.

Commande	Arguments	Description
stop	SANS	Arrêt de la simulation avec écriture du fichier résultat
step	SANS	Exécution d'un cycle
pause	SANS	Pause de la simulation
run	N pas pause ou stop	Exécution des cycles de la simulation
speed	N pas pause ou stop	Exécution en mode accéléré
save	filename	Enregistre l'état de la simulation dans <filename></filename>
restore	filename	Restitue l'état de la simulation depuis <filename></filename>
record	filename	Enregistre l'historique de la simulation dans <filename></filename>

4.1.7 Compilation d'un module manuel

La compilation du module nécessite l'utilisation de QtCreator avec un projet configuré de la sorte :

QT -= gui
TARGET = ModuleA
TEMPLATE = lib

 $win 32-msvc^* \qquad \{ LIBS \quad += \$\$(FBSF_HOME)/lib/FbsfBaseModel.lib\} \\ win 32-g++ \qquad \{ LIBS \quad += \$\$(FBSF_HOME)/lib/FbsfBaseModel.a\}$

INCLUDEPATH += \$\$(FBSF_HOME)/FbsfBaseModel

DEFINES += ModuleA_LIBRARY

DESTDIR = \$\$(APP_HOME)/lib

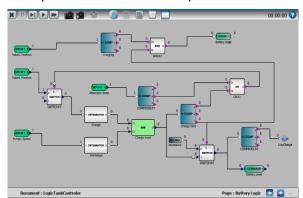
Pour plus de détails voire chapitre 6.2.



4.2 Module de type Graphique et plugin Qml

Le framework propose un éditeur « *FbsfEditor* » pour réaliser les pages de Logiques et Graphiques, voir le manuel d'utilisation de l'éditeur « *FbsfEditor* ».

La copie d'écran ci-dessous présente des modèles de Logique et Graphique.





Pour les plugins d'interface graphique dits « Images à façon », ils sont développés en QML et se conforment à une structure normalisée.

Un fichier main.qml dans le dossier du plugin constitue le point d'entrée.

Il est implémenté comme le montre l'exemple suivant pour une IHM définie dans **monIHM.qml**. Pour un exemple « d'images à façon » voire l'application « **DEMO-HOUSE** ».



4.2.1 Gestion des paramètres pour un modèle CAO de FbsfEditor

Un composant de modèle CAO déclare "parameters" pour tunable et "constant" pour fixed dans son code QML.

L'inspecteur d'objet présente les deux types avec le type constant en « readonly ».

```
property var parameters: {"p1" : 1, "p2" : true }
property var constant: {"k1" : 111 ,"k2" : 222 }
```

Note: Les paramètres et constantes de type "string" ne sont pas concernés.

Options de la configuration :

<publishparam>: Les paramètres «tunable», des modules CAO Logic, FbsfBaseModelFMI et FMU, apparaissent dans la liste du moniteur avec un code couleur. Ils sont modifiables depuis l'interface comme le sont les importeurs non résolus.

<snapshotparam>: Les paramètres «tunable » sont sauvegardés dans les clichés. Le format QML texte des documents CAO permet de modifier la valeur initiale d'un paramètre. Les mots clé nodeParams et nodeConstants identifient les listes de paramètres entre accolades selon la syntaxe : 'nom': valeur, 'nom': valeur,

4.2.2 Gestion de la ZE

Une donnée est déclarée par son nom et son type. Le nom est constitué du tag1 et optionnellement du tag2 qui produisent respectivement les identifiants publiques : «tag1» et «tag1:tag2»

4.2.3 Abonnement à la ZE

Connecter un scalaire déclaré comme une property

```
Property int myIntData:0

SubscribeInt{tag1: "MyInt";tag2: "Data"; onValueChanged: { myIntData = value; } }

Property real myRealData:0.0

SubscribeReal{tag1: "MyReal";tag2: "Data"; onValueChanged: { myRealData = value; } }
```

Connecter un vecteur de données déclaré comme une **property** :



```
SubscribeVectorReal{ tag1: "MyRealVector" onDataChanged: { realVector=data; txt1.text=tag1 + " : "+ realVector [0]} } //1er élément
```

En mode « time-depend »:

```
MyVector.data[idx]// élément du vecteur pas de temps courant (passé+futur)MyVector.timeshift// Le facteur de décalage temporelMyVector.timeindex// Index du temps courant dans le vecteur data[]MyVector.history[idx]// Vecteur historique : lire la valeur historique d'index idxMyVector.history.length// Npas * Timeshift valeurs
```

4.2.4 Publication vers la ZE

Une donnée publiée depuis le plugin graphique est généralement transmise de façon asynchrone.

Connecter un scalaire déclaré comme une property

Connecter un vecteur de données déclaré comme une property :

La bibliothèque d'objets disponibles avec « *FbsfEditor* » est codée en QML pour les documents Logic et Graphic, elle obéit aux règles précédentes.



5. API DE CONTROLE

5.1 Concepts

L'API est basée sur la norme Fmi 2.0 L'API est codée en C++

5.2 Methodes

5.2.1 Gestion d'instance

fmi2Component fmi2Instanciate(int argc, char **argv);

Créée une instance de du fmiComponent nécessaire pour tous les autres appels à L'API.

fmi2Status fmi2FreeInstance(fmi2Component ptr);

Libère la mémoire allouée (ptr et buffers internes).

5.2.2 Paramétrage de la simulation

fmi2Status fmi2SetString(fmi2Component ptr, QString str);

Définis le nom du fichier de configuration pour la simulation.

fmi2Status fmi2EnterInitialisationMode(fmi2Component ptr);

Met l'app en mode Initialisation

Charge la configuration indiquée via fmiSetString.

fmi2Status fmi2ExitInitialisationMode(fmi2Component ptr);

Sort l'app du mode initialisation et lance l'app fbsf.

5.2.3 Contrôle de simulation

fmi2Status fmi2DoStep(fmi2Component ptr);

Lance un pas de calcul asynchrone.

fmi2Status fmi2CancelStep(fmi2Component ptr);

Non codé



fmi2Status fmi2Terminate(fmi2Component ptr);

Informe l'app Fbsf que la simulation est terminée et qu'elle doit quitter.

5.2.4 Acquisition de l'état de la simulation

Les fonctions d'état prennent 3 arguments :

- Le pointeur sur le fmi2Component
- Un fmi2StatusKind qui indique quel état est demandé
- Un pointeur sur une variable qui contiendra l'état

La fonction appelée doit correspondre au type de retour associé au fmi2StatusKind sinon la fonction n'a aucun effet (fmi2PendingStatus doit être appelé avec la fonction fmi2GetString Status car fmi2PendingStatus retourne un état de type fmi2String)

fmi2Status fmi2GetStatus(fmi2Component ptr, const fmi2StatusKind s, fmi2Status *value);

Requiert un état de type fmi2Status (compatible avec fmi2StatusKind::fmi2DoStepStatus)

fmi2Status fmi2GetRealStatus(fmi2Component ptr, const fmi2StatusKind s, fmi2Real *value);

Requiert un état de type fmi2Real (compatible avec fmi2StatusKind::fmi2LastSuccessfulTime)

fmi2Status fmi2GetIntegerStatus(fmi2Component ptr, const fmi2StatusKind s, fmi2Integer *value);

Requiert un état de type fmi2Integer (compatible avec aucun fmi2StatusKind)

fmi2Status fmi2GetBooleanStatus(fmi2Component ptr, const fmi2StatusKind s, fmi2Boolean *value);

Requiert un état de type fmi2Boolean (compatible avec fmi2StatusKind::fmi2Terminated)

fmi2Status fmi2GetStringStatus(fmi2Component ptr, const fmi2StatusKind s, fmi2String *value);

Requiert un état de type fmi2String (compatible avec fmi2StatusKind::fmi2PendingStatus)
Attention : en accord avec la norme fmi2, il est de la responsabilité de l'utilisateur de copier le contenu de la variable 'value' après retour de la fonction pour toute utilisation future.
La zone mémoire pointée est un buffer interne à l'api qui peut être réutilisé et écrasé par n'importe que futur appel à l'api et n'est donc pas sûr d'utilisation.



5.3 Compilation

L'api est compilée dans le même dossier que fbsfFramework (aka : lib/release | debug | batch) Voici un exemple de .pro permettant de compiler une app c++ avec l'api

TEMPLATE=app

LIBS += -L\$\$(FBSF_HOME)/lib/release -IFbsfApi

INCLUDEPATH += \$\$(FBSF_HOME)/FbsfFramework/FbsfApi

SOURCES += testapi.cpp

HEADERS += testapi.h



6. ANNEXES

6.1 Installation du Framework

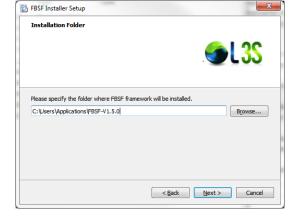
6.1.1 Prérequis

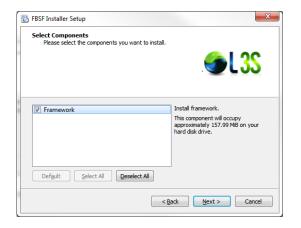
os	Windows 10 et +, Linux
Qt	5.15 et +
Compilateur	MSVC 2015 et + ou GCC 4.2 et +
Fichier de licences	Contacter L3S

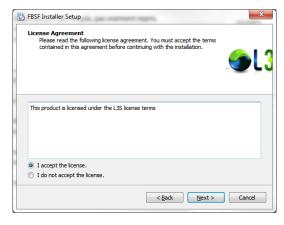
6.1.2 Procédure d'installation

Le Framework est installé sur un espace disque local et son contenu est déposé sous forme de binaires en suivant les instructions de l'installeur.









Pour exécuter une application sous FBSF, il faut disposer d'un fichier de licence «FbsfFramework.lic» déposé à la racine de l'installation FBSF.



6.1.3 Principes généraux d'organisation

Les espace Framework et Application sont définis par les variables d'environnement :

FBSF_HOME: racine installation du Framework,

APP_HOME: racine installation de l'applicatif (plugins, binaires).

Répertoire d'exécution : production/consommation des données runtime (config xml, log, data files, rejeu, clichés, ...).

6.2 Développement de modèles

Pour développer des modèles, il est nécessaire d'installer la version QT préconisée avec le QtCreator.

IDE pour développement	QtCreator	Version 5.15 et +
Compilateur	Visual C++	MSDEV 2015 et +
	GCC	4.2 et +

Pour positionner l'environnement FBSF dans QtCreator, il faut définir un script projet qui définit les chemins d'accès à QT et QTCreator.

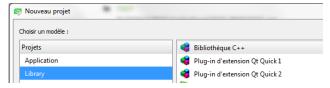
Exemple de fichier production.bat pour Windows :

set QTDIR	= <path qt="" to=""></path>	
set QTPATH	=%QTDIR%\5.7\msvc2013_64\bin;%QTDIR%\Tools\QtCreator\bin	
set PATH	=%~dp0lib;%QTPATH%;%PATH%	
call	<path fbsf="" to="">\fbsfenv.bat</path>	
set APP_HOME	=%~dp0	
start	qtcreator.exe	
<mark>exit</mark>		

Les développements des modules applicatifs sont réalisés avec l'environnement projet de QtCreator. Ce chapitre présente les étapes de configuration du projet sous QtCreator.

Pour créer un module manuel applicatif, Il faut :

Créer un nouveau projet QtCreator de type librairie C++



- Indiquer le nom du type du module (ex : ModuleA)
- Choisir le kit de production : par ex. Desktop QT 5.15 MSVC 2015 64 bits

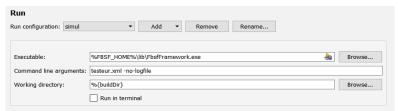




Il important de configurer le projet sans "shadow build",



• Définir la configuration d'exécution référencer l'exécutable FbsfFramework livré, les arguments de la ligne de commande et le répertoire d'exécution



• Compléter le fichier .pro qui doit définir à minima les variables suivantes :

```
QT
                   -= gui
TARGET
                   = ModuleA
TEMPLATE
                   = lib
win32-msvc*
                   { LIBS += $$(FBSF_HOME)/lib/FbsfBaseModel.lib}
                   {LIBS += $$(FBSF HOME)/lib/FbsfBaseModel.a}
win32-q++
INCLUDEPATH
                   += $$(FBSF HOME)/FbsfBaseModel
DEFINES
                   += ModuleA_LIBRARY
DESTDIR
                   = $$(APP HOME)/lib
```

Pour exécuter l'application sous « **QtCreator** » utiliser le bouton de lancement. Environnement d'exécution hors « **QtCreator** » afin de séparer les environnements de développements, il est recommandé de produire un script lié au projet applicatif qui sera exécuté pour lancer l'environnement.

Le script de la partie projet exécute le script « FbsfEnv.bat » de FBSF et positionne l'environnement propre au projet puis positionne les variables propres au projet :

- Ajout du chemin des binaires (.dll ou .so) dans le PATH ou LD LIBRARY PATH
- Ajout de la racine du code applicatif projet dans la variable APP_HOME

Les fichiers log et fichiers produits par l'application sont produits depuis le dossier



d'exécution.

Exemple d'un fichier runtime.bat dans un environnement OS Windows :

set QTDIR	= <path qt="" to=""></path>
set QTPATH	= %QTDIR%\5.7\msvc2013_64\bin;%QTDIR%\Tools\QtCreator\bin
set PATH	= %~dp0lib;%QTPATH%;%PATH%
call	<fbsf rootpath="">\FbsfEnv.bat</fbsf>
set APP_HOME	= %~dp0
if not "%1"	== "" start FbsfFramework.exe %1

Le fichier FbsfEnv.bat du Framework FBSF, **intégré à l'installation**, positionne l'environnement minimal nécessaire au système exécutif.

set FBSF_HOME	=%~dp0
set PATH	=%FBSF_HOME%\lib;%FBSF_HOME%\FbsfFramework\fbsfplugins

Il est ainsi possible de créer un raccourci sur le bureau pour lancer l'exécution de l'application.

L'argument fourni au fichier .bat est le nom du fichier xml de configuration de l'application.

<chemin de l'application>\runtime.bat config.xml

Arguments en mode simulation": FbsfFramework.exe config.xml

Arguments en mode rejeu : FbsfFramework.exe config.xml -r[eplay] file.dat

Arguments en mode batch : FbsfFramework.exe config.xml -b[atch]



7. GESTION DU TEMPS SIMULE

La fonction de gestion du temps simulé permet de spécifier une date et/ou temps de départ ainsi que le format de date et/ou temps pour l'affichage.

La specification s'appuie sur les notions distinctes de Date et de Time. Dans la configuration, il est possible de définir optionnellement une Date et/ou Time de départ au **format ISO 8601** avec optionnellement un format Date et/ou format Time pour l'affichage.

Exemple:

<startDate>2015-05-31</startDate>
<dateFormat>dd-MM-yyyy</dateFormat>
<startTime>08:00:00</startTime>
<timeFormat>hh:mm:ss </timeFormat>

Toutes ces définitions sont optionnelles et conduisent à une combinatoire présentée dans le tableau suivant.

En absence de définition de format, les formats suivants sont utilisés par défaut :

Default date format : dd/MM/yyyyDefault Time format : hh:mm:ss[.zzz] *

En absence de définition de date et/ou time, si un format date et/ou time est précisé, la date et/ou time courant est utilisée.

	Mode standard		
startTime defined		startTime undefined	
startDate defined	Date + Time	Date + currentTime	
	In the set Time and time of a second	If a favorat an action is	
startDate	Instant Time as time format	If no format specified :	
undefined	or cumulative format as :	Mode standard:	
	hhh[:mm[:ss[.zzz]]]	elapsed time as [D] HH:mm:ss[.zzz] *	
	mmm[:ss[.zzz]]	Mode MPC	
	sss.[zzz]	currentDate ()+currentTime ()	

^{*} si non spécifié, [.zzz] est déterminé par le test : if(mTimeStepMS%1000 != 0)



7.1 Exemples d'affichage

7.1.1 Default : aucune spécification

Mode standard : elapsed time

Timestep	initial	Step 1	Step 2
300	00:00:00	00:05:00	00:10:00
Timestep	initial	Step 1	Step 2
0.5	00:00:00.000	00:00:00.500	00:00:01.000

Mode SimuMPC: current date and time

Timestep/Timeshift	initial	Step 1	Step 2
300/4	dd/MM/yyyy	21/12/2021	21/12/2021
	HH:mm:ss	14:00:59	14:20:59

7.1.2 Spécification de date et temps avec formats

<StartDate>2015-05-31</StartDate>

<DateFormat>ddd dd.MMM.yyyy

<StartTime>08:30:00</StartTime>

<TimeFormat>HH.mm.ss</TimeFormat>

Timestep	initial	Step 1	Step 2
300	Sun 31.May.2015	Sun 31.May.2015	Sun 31.May.2015
	08.30.00	08.35.00	08.40.00



- <StartDate>2015-05-31</StartDate>
- <DateFormat>dd-MMM-yyyy</DateFormat>
- <StartTime>08:30:00</StartTime>
- <TimeFormat>HH.mm.ss</TimeFormat>

Timestep	initial	Step 1	Step 2
0.5	31-May-2015	31-May-2015	31-May-2015
	08.30.00.000	08.30.00.500	08.30.01.000

7.1.3 Spécification d'instant sans format

<StartTime>08:30:00</StartTime>

Timestep	initial	Step 1	Step 2
300	08:30:00	08:35:00	08:40:00

7.1.4 Spécification d'instant avec format cumulatif

Cumul en heures:

<TimeFormat>hhh:mm:ss</TimeFormat>

<StartTime>01:00:00</StartTime>

Timestep	initial	Step 1	Step 2
300	1 h 00 mn 00 s	1 h 05 mn 00 s	1 h 10 mn 00 s



Cumul en minutes:

<TimeFormat>mmm:ss</TimeFormat>

<StartTime>00:30:00</StartTime>

Timestep	initial	Step 1	Step 2
1	30 mn 00 s	30 mn 01 s	30 mn 02 s

7.1.5 Cumul en secondes :

<TimeFormat>sss</TimeFormat>

<StartTime>00:30:00</StartTime>

Timeste	initial	Step 1	Step 2
0.5	1800.000	1800.500	1801.000



7.2 Formattage

7.3 Formattage de date

Expression	Output	
d	The day as a number without a leading zero (1 to 31)	
dd	The day as a number with a leading zero (01 to 31)	
ddd	The abbreviated localized day name (e.g. 'Mon' to 'Sun'). Uses QDate::shortDayName.	
dddd	The long localized day name (e.g. 'Monday' to 'Sunday'). Uses QDate::longDayName.	
M	The month as a number without a leading zero (1 to 12)	
ММ	The month as a number with a leading zero (01 to 12)	
МММ	The abbreviated localized month name (e.g. 'Jan' to 'Dec'). Uses QDate::shortMonthName.	
ММММ	The long localized month name (e.g. 'January' to 'December'). Uses QDate::longMonthName.	
уу	The year as two digit number (00 to 99)	
уууу	The year as four digit number. If the year is negative, a minus sign is prepended in addition.	

7.4 Formattage de temps

Expression	Output	
h	The hour without a leading zero (0 to 23 or 1 to 12 if AM/PM display)	
hh	The hour with a leading zero (00 to 23 or 01 to 12 if AM/PM display)	
Н	The hour without a leading zero (0 to 23, even with AM/PM display)	
НН	The hour with a leading zero (00 to 23, even with AM/PM display)	
m	The minute without a leading zero (0 to 59)	
mm	The minute with a leading zero (00 to 59)	



s	The whole second, without any leading zero (0 to 59)	
SS	The whole second, with a leading zero where applicable (00 to 59)	
Z	The fractional part of the second, to go after a decimal point, without trailing zeroes (0 to 999). Thus "s.z" reports the seconds to full available (millisecond) precision without trailing zeroes.	
ZZZ	The fractional part of the second, to millisecond precision, including trailing zeroes where applicable (000 to 999).	

7.5 Accessibilité

	API C++ FbsfBaseModel.h	QML property Node.qml		
Temps écoulé depuis 0.	quint64 simulationTime()	real simulationTime		
Nombre de pas exécutés.	quint64 stepCount()			
Pas de temps en millisecondes.	quint64 timeStepMS()	real timeStep		
Résolution millisecondes (true/false)	bool timeUnitMS()			
	Mode Standard			
from[M]SecsSinceEpoch des données du pas de temps courant pour un temps scalaire.	quint64 dataTimeStamp ()			
Mode MPC				
from[M]SecsSinceEpoch des données du pas de temps indexé pour un vecteur temps.	quint64 dataTimeStamp(int index)			

L3S, 9 Avenue Paul Verlaine 38030 GRENOBLE CEDEX 2— France - Tel: + 33 952567220 - Fax: + 33 957567220 - Web site: http://www.L-35.fr
SARL au capital de 50 000 EUR — Grenoble B753885045 — SIRET: 75388504500017



Dimension past du mode MPC.	quint64 pastSize()	int pastSize
Dimension futur du mode MPC.	quint64 futurSize()	int futurSize
Dimension timeshift du mode MPC.	quint64 timeShift()	int timeShift

7.6 Exemples C++

Obtenir le timestamp en millisecondes quelque soit la résolution du temps.

Mode standard:

qint64 timestampMS=dataTimeStamp()*(timeUnitMS()?1:1000);

Mode MPC

qint64 timestampMS=dataTimeStamp(pastSize()-1)*(timeUnitMS()?1:1000);

Affichage de la date et du temps :

QDateTime::fromMSecsSinceEpoch(timestampMS).toString("dd/MM/yyyy hh:mm:ss");