

# ELTM 1.0 Documentation

Malo DAVID

## Table des matières

<b>1</b>	<b>Qu'est ce que l'ELTM ?</b>	<b>2</b>
<b>2</b>	<b>Apprendre le langage</b>	<b>2</b>
2.1	Lier un texte à un identifiant . . . . .	2
2.2	Le Contexte ELTM . . . . .	3
2.3	Récupérer mes textes dans mon code <i>C++</i> . . . . .	4
2.4	Les commentaires . . . . .	5
2.5	Les modules . . . . .	5
2.6	Séparer son code ELTM sur plusieurs fichier . . . . .	6
<b>3</b>	<b>Références</b>	<b>8</b>
3.1	Set . . . . .	8
3.2	get() . . . . .	8
3.3	begin . . . . .	8
3.4	end . . . . .	8
3.5	module . . . . .	8
3.6	// . . . . .	8
3.7	/* . . . . .	8
3.8	*/ . . . . .	9
3.9	import . . . . .	9
<b>4</b>	<b>Références <i>C++</i></b>	<b>10</b>
4.1	ELTMcontext . . . . .	10
4.2	newContext . . . . .	10
4.3	getText() . . . . .	10

# 1 Qu'est ce que l'ELTM ?

L'ELTM (Extension Language for Text Management) est un langage d'extension du *C++* permettant de gérer plus facilement les textes de vos projets avec *AtlasEngine*.

Son utilisation n'est pas dissociable du *C++* car il lui faut un contexte instancié pour l'utiliser. Il est extrêmement simple et ne demande que peu de temps pour l'apprendre.

## 2 Apprendre le langage

L'ELTM ne peut manipuler que des chaînes de caractères (strings). Les fichiers ELTM ne possèdent pas d'extension définie (ils peuvent être définis en *.txt*) mais je préconise d'utiliser le *.tm* ou *.eltm* pour savoir que ce sont des fichiers ELTM.

### 2.1 Lier un texte à un identifiant

L'ELTM gère vos textes en les liant à des identifiants. Cela se présente, de la manière la plus simple, comme ceci :

```
1 set monID = mon texte basique tenant sur une seule ligne
```

Ici, le texte *mon texte basique tenant sur une seule ligne* est lié à l'ID *monID*.

Si votre texte est très très long et que c'est compliqué de le faire tenir sur une ligne, il y a moyen de le définir ainsi :

```
1 set monID = (mon texte
2             vraiment
3             très
4             très
5             long)
```

*PS : à noter que l'ELTM ne tient pas compte des indentations, à vrai dire il ne sait même pas ce que c'est.*

Un autre moyen de lier un texte à un identifiant est de récupérer le texte d'un autre ID grâce à la fonction *get()* :

```
1 set monID = (mon texte
2             vraiment
3             très
4             très
5             long)
6 set monID2 = mon autre texte
7
8 set monID3 = get(ID2)
```

Lorsqu'un ID a déjà été lié et que l'on veut changer le texte lié à l'ID nous ne devons pas utiliser *set* :

```
1  set monID = (mon texte
2                  vraiment
3                  très
4                  très
5                  long)
6  set monID2 = mon autre texte
7
8  set monID3 = get(ID2)
9  monID3 = nouveau texte
10 monID = get(monID3)
```

## 2.2 Le Contexte ELTM

L'ELTM est compréhensible par le *C++* grâce au *Contexte* qui est comme un traducteur ou un interpréteur. Il s'instancie ainsi :

```
1  #include <AtlasEngine.h>
2
3  int main()
4  {
5      AE::ELTMcontext eltm;
6      eltm.newContext("fichier.eltm");
7
8      return 0;
9  }
```

Si le *Contexte* ne trouve aucune erreur dans votre code ELTM, vous pouvez désormais utiliser vos textes sans problème.

## 2.3 Récupérer mes textes dans mon code C++

Votre *Contexte* ELTM est désormais instancié, il a interprété votre code sans erreur et tous les voyants sont verts pour que vous utilisiez vos textes, mais comment faire ?

Pas de panique l'ELTM possède une fonction nommée *getText(std :: string ID)*. Elle peut s'utiliser de 2 manières différentes :

```
1
2  /* ===== CODE ELTM ===== */
3  // fichier.eltm
4
5  set monID = (mon texte
6               vraiment
7               très
8               très
9               long)
10 set monID2 = mon autre texte
11
12 set monID3 = get(ID2)
13 monID3 = nouveau texte
```

```
1
2  /* ===== CODE C++ ===== */
3
4  #include <AtlasEngine.h>
5  #include <iostream>
6
7  int main()
8  {
9      AE::ELTMcontext eltm;
10     eltm.newContext("fichier.eltm");
11
12     // Première méthode avec l'objet
13     // de la classe ELTMcontext
14     std::cout << eltm.getText("monID") << std::endl;
15
16     // Deuxième méthode directement
17     // avec la classe ELTMcontext
18     std::cout << AE::ELTMcontext::getText("monID3");
19
20     return 0;
21 }
```

La sortie dans le terminal est :

```
User@User-pc:~$ mon texte vraiment très très long
User@User-pc:~$ nouveau texte
```

## 2.4 Les commentaires

L'ELTM possède un système de commentaires similaires à ceux du *C++* ou du *Java*. Pour les commentaires simples sur une seule ligne, ils commencent par `//`, pour les commentaires longs, ils commencent par `/*` et se terminent par `*/` :

```
1  set monID = (mon texte
2                      vraiment
3                      très      // Ceci est un commentaire court
4                      très
5                      long)
6  set monID2 = mon autre texte
7
8  /*
9      Et ceci est
10     un commentaire
11     long
12  */
```

## 2.5 Les modules

Les modules sont une partie importante de l'ELTM, ils sont comparables aux *namespaces* du *C++*. Ils permettent d'associer des ID et leur texte avec des noms de modules. Ils s'utilisent ainsi :

```
1
2  /* ===== CODE ELTM ===== */
3
4  set monID = mon texte
5
6  begin module monModule
7
8      set ID2 = autre texte
9      set ID3 = get(monID)
10
11  end module
12
13  monID = get(monModule.ID2)
```

```
1
2  /* ===== CODE C++ ===== */
3
4  #include <AtlasEngine.h>
5  #include <iostream>
6
7  int main()
8  {
9      AE::ELTMcontext eltm;
10     eltm.newContext("fichier.eltm");
11
12     std::cout << eltm.getText("monModule.ID2") << '\n';
13     std::cout << eltm.getText("monModule.ID3") << '\n';
14
15     return 0;
16 }
```

La sortie dans le terminal est :

```
User@User-pc:~$ autre texte
User@User-pc:~$ mon texte
```

Une fois une ID définie dans un module, elle est indissociable de celui-ci et doit obligatoirement être appelée avec son module. Les modules sont très utiles pour classer les textes et les ID dans des catégories, par exemple, pour des textes d'erreurs, nous pouvons faire un module *Erreurs* et placer nos textes dedans.

## 2.6 Séparer son code ELTM sur plusieurs fichier

Comme dans quasiment tous les langages de programmation, l'ELTM peut s'utiliser sur plusieurs fichiers et le *Contexte* est capable de gérer ça. Pour cela, il utilise le mot clé *import*, suivi du chemin et du nom du fichier (avec son extension). Cependant, la gestion des imports de fichiers en ELTM est un petit peu particulière comparée à celle du *Python*, du *C++* ou encore du *BCPL*, pour remonter dans les langages plus anciens.

En ELTM, lorsqu'un fichier est importé quelque part, automatiquement tous les fichiers déjà importés peuvent y avoir accès, même s'il ne l'ont pas importé eux-mêmes. C'est aussi valable pour le fichier principal (celui donné au contexte), du moment qu'un fichier ELTM est importé par un autre, il aura accès aux textes du fichier principal. Il est donc inutile d'importer le fichier principal dans un autre fichier. Par exemple, voici 3 fichiers ELTM pour illustrer mon propos :

```
1 // main.eltm
2
3 // Tous les fichiers importés auront accès à mainID
4 import fichier1.eltm // fichier1 importé avant fichier2
5 import fichier2.eltm // fichier2 à accès à fichier1
6
7 set mainID = texte principal.
```

```
1 // fichier1.eltm
2
3 begin module monModule
4
5     set ID2 = texte du fichier1.eltm
6
7 end module
```

```
1 // fichier2.eltm
2
3 set resultat = get(monModule.ID2) /* autorisé car fichier1
4                                   a été importé avant
5                                   fichier2
6                                   */
7 set resultat2 = get(mainID)
```

```

1
2  /* ===== CODE C++ ===== */
3
4  #include <AtlasEngine.h>
5  #include <iostream>
6
7  int main()
8  {
9      AE::ELTMcontext eltm;
10     eltm.newContext("main.eltm");
11
12     std::cout << eltm.getText("resultat") << std::endl;
13     std::cout << eltm.getText("resultat2") << std::endl;
14
15     return 0;
16 }

```

```

User@User-pc:~$ texte du fichier1.eltm
User@User-pc:~$ texte principal.

```

Je vous conseil donc de faire un fichier spécial contenant tous les imports nécessaires et de l'importer dans le fichier principal au tout début.

*PS : l'ELTM possède une sécurité interne contre les inclusions infinies*

## 3 Références

### 3.1 Set

*set* sert à déclarer un identifiant et l'associer avec un texte.

```
1 set ID = text
```

### 3.2 get()

*get()* sert à récupérer un texte d'une ID pour la mettre dans une autre.

```
1 set ID = text
2 set ID2 = get(ID)
```

### 3.3 begin

*begin* sert à instancier des fonctionnalités de l'ELTM comme les modules.

```
1 begin module MODULE
```

### 3.4 end

*end* sert à définir la fin d'une fonctionnalité de l'ELTM, comme les modules, instanciée avec *begin*.

```
1 begin module MODULE
2 end module
```

### 3.5 module

*module* sert à instancier un nouveau module pour associer un nom à des ID instanciées en son sein.

```
1 begin module MODULE
2 end module
```

### 3.6 //

// sert à commencer un commentaire sur une seule ligne.

```
1 // commentaire
```

### 3.7 /\*

/\* sert à commencer un commentaire sur plusieurs lignes.

```
1 /* commentaire
2 très
3 long */
```



### 3.8 \*/

*\*/* sert à terminer un commentaire déclaré sur plusieurs lignes.

```
1  /* commentaire
2     très
3     long */
```

### 3.9 import

*import* sert à rattacher un nouveau fichier à l'ensemble de fichiers interprétés par le Contexte.

```
1  import fichier.eltm
```

## 4 Références C++

### 4.1 ELMcontext

*ELMcontext* sert à instancier un nouveau contexte ELM pour interpreter du code ELM.

```
1 AE::ELMcontext eltm;
```

### 4.2 newContext

*newContext* sert à interpréter du code ELM donné.

```
1 AE::ELMcontext eltm;  
2 eltm.newContext("fichier.elm");
```

### 4.3 getText()

*getText()* sert à récupérer un texte contenu dans une ID.

```
1 AE::ELMcontext::getText("ID");  
2 // Ou  
3 AE::ELMcontext eltm;  
4 eltm.newContext("fichier.elm");  
5 eltm.getText("ID");
```