ELTM 1.1 Documentation

Malo DAVID

Table des matières

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 *Akel*.
Son utilisation n'est pas dissociable du *C++* car il lui faut un contexte ELTM 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 êtres 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(monID2)
```

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* :

```
set monID = (mon texte
             vraiment
             très
             très
             long)
set monID2 = mon autre texte

set monID3 = get(monID2)
monID3 = nouveau texte
monID = get(monID3)
```

2.1.1 La concatenation

L'ELTM 1.1 est désormais capable de concatenation. Le code suivant n'est pas possible en ELTM 1.0 :

```
set monID = test
set concatenation = ceci est un get(monID)
// équivaut à : set concatenation = ceci est un test
```

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 :

```
#include <Akel.h>

int main()
{
    Ak::ELTMcontext eltm;
    eltm.newContext("fichier.eltm");

    return 0;
}
```

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 :

```

/* ==================== CODE ELTM ====================== */
// fichier.eltm

set monID = (mon texte
             vraiment
```

```
7                texte
8                texte
9                long)
10 set monID2 = mon autre texte
11
12 set monID3 = get(monID2)
13 monID3 = nouveau texte
```

```cpp

/* ===================== CODE C++ ===================== */

#include <Akel.h>
#include <iostream>

int main()
{
    Ak::ELTMcontext eltm;
    eltm.newContext("fichier.eltm");

    // Première méthode avec l'objet
    // de la classe ELTMcontext
    std::cout << eltm.getText("monID") << std::endl;

    // Deuxième méthode directement
    // avec la classe ELTMcontext
    std::cout << Ak::ELTMcontext::getText("monID3");

    return 0;
}
```

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 de *Java*. Pour les commentaires simples sur une seule ligne, ils commencent par *//*, pour les commentaires longs, ils commencent par */** et se terminent par ***/* :

```
set monID = (mon texte
             vraiment
             très       // Ceci est un commentaire court
             très
             long)
set monID2 = mon autre texte

/*
    Et ceci est
    un commentaire
    long
*/
```

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 <Akel.h>
5  #include <iostream>
6
7  int main()
8  {
9      Ak::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 fichiers

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'ils 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 :

```
// main.eltm

// Tous les fichiers importés auront accès à mainID
import fichier1.eltm // fichier1 importé avant fichier2
import fichier2.eltm // fichier2 à accès à fichier1

set mainID = texte principal.
```

```
// fichier1.eltm

begin module monModule

    set ID2 = texte du fichier1.eltm

end module
```

```
// fichier2.eltm

set resultat = get(monModule.ID2) /* autorisé car fichier1
                                     a été importé avant
                                     fichier2
                                  */
set resultat2 = get(mainID)
```

```

/* ====================== CODE C++ ====================== */

#include <Akel.h>
#include <iostream>

int main()
{
    Ak::ELTMcontext eltm;
    eltm.newContext("main.eltm");

    std::cout << eltm.getText("resultat") << std::endl;
    std::cout << eltm.getText("resultat2") << std::endl;

    return 0;
}
```

```
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 déclare un identifiant et l'associer avec un texte.

```
set ID = text
```

3.2 get()

get() récupère un texte d'une ID pour la mettre dans une autre.

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

3.3 begin

begin instancie des fonctionnalités de l'ELTM comme les modules.

```
begin module MODULE
```

3.4 end

end définit la fin d'une fonctionnalité de l'ELTM, comme les modules, instanciée avec begin.

```
begin module MODULE
end module
```

3.5 module

module instancie un nouveau module pour associer un nom à des ID instanciées en son sein.

```
begin module MODULE
end module
```

3.6 //

// commence un commentaire sur une seule ligne.

```
// commentaire
```

3.7 /*

/* commence un commentaire sur plusieurs lignes.

```
/* commentaire
    très
    long */
```

3.8 */

*/ termine un commentaire déclaré sur plusieurs lignes.

```
/* commentaire
    très
    long */
```

3.9 import

import rattache un nouveau fichier à l'ensemble de fichiers interprétés par le Contexte.

```
import fichier.eltm
```

4 Références *C++*

4.1 ELTMcontext

ELTMcontext instancie un nouveau contexte ELTM pour interpréter du code ELTM.

```
Ak::ELTMcontext eltm;
```

4.2 newContext

newContext interprète du code ELTM donné.

```
Ak::ELTMcontext eltm;
eltm.newContext("fichier.eltm");
```

4.3 getText()

getText() récupère un texte contenu dans une ID.

```
Ak::ELTMcontext::getText("ID");
// Ou
Ak::ELTMcontext eltm;
eltm.newContext("fichier.eltm");
eltm.getText("ID");
```