|  |
| --- |
|  |
| Syntaxi |
| Extracteur de documentation |
|  |
| **Thomas Auguey** |
| **22/10/2015** |

|  |
| --- |
| Système de documentation. |

SOMMAIRE

[Présentation 3](#_Toc434866863)

[Motivations 3](#_Toc434866864)

[Cahier des charges 3](#_Toc434866865)

[Analyse 4](#_Toc434866866)

[Eléments 4](#_Toc434866867)

[Acteurs 4](#_Toc434866868)

[Cas d’utilisations 5](#_Toc434866869)

[Définition du système 6](#_Toc434866870)

[Base de données 6](#_Toc434866871)

[Interface de gestion 6](#_Toc434866872)

[Réalisation 7](#_Toc434866873)

[Base de données 8](#_Toc434866874)

[Librairie Model de données 12](#_Toc434866875)

[Application Console 15](#_Toc434866876)

[Application IHM 16](#_Toc434866877)

[Redistribuable 18](#_Toc434866878)

[Arborescence 18](#_Toc434866879)

# Présentation

Syntaxi est outil d’extraction de documentation. Le texte est récoltée selon une syntaxe définit par l’utilisateur puis stockée dans des objets de données.

L’intérêt du système repose sur la possibilité pour l’utilisateur de créer ces règles d’extraction (particulièrement pratique pour les projets existants). Les règles d’extractions sont basées sur les expressions-régulières (bien connue des programmeurs en informatique). Syntaxie est principalement adapté aux documentations techniques et aux API.

## Motivations

Tirer parti de la flexibilité des expressions régulières permet au programmeur d’adapté sa documentation à son projet et non le contraire. Les principaux logiciels sur le marché force le programmeur à adapter un style de rédaction de sa documentation, (ex : Doxygen), Syntaxi prend la problématique dans le sens inverse en proposant un outil s’adaptant au code existant en se basant sur des expressions régulières pour extraire le texte intéressant.

## Cahier des charges

Le système devra proposer un modèle de données permettant de stocker des objets de contenu texte, les objets seront typés pour permettre leurs tris lors de la mise en forme. Chaque objet sera associé à une expression régulière permettant son extraction depuis un fichier texte. Les objets ainsi créés seront stockés dans un fichier de «projet ».

Le système ne définit pas de mise en forme des objets (classe de données, page de description, fonction, élément de template, …) c’est l’utilisateur final qui devra typer ses expressions dans le but de les identifier et de les mettre en page dans une application tierce.

Pour assister un déploiement rapide une collection de projets types (C++, C#, Java, …) accompagné d’applications clientes serviront de base à la création d’un nouveau projet.

L’application devra permettre de mettre à jour des projets existant et d’exporter les données dans une base SQL distante.

L’application sera proposée sous forme de ligne de commande (pour l’automation des mises à jour) et d’IHM pour une meilleure expérience utilisateur.

# Analyse

## Eléments

### Projet

Le projet regroupe l’ensemble des données persistantes qui seront sauvegardées dans un fichier de données.

### Objet de contenu

Contient le texte scanné par un objet de syntaxe.

### Objet de syntaxe

Contient l’expression régulière appliquée aux fichiers textes pour détecter et extraire les objets de contenu.

### Paramètre de recherche

Définit l’emplacement et les options de recherches des fichiers à scanner.

### Configuration

La configuration regroupe les préférences utilisateur associé au système.

## Acteurs

### Utilisateur

Personne physique utilisateur final de l’application

### Système

Ordinateur exploitant l’application de gestion

## Cas d’utilisations

* Utilisateur :
  + Gérer les syntaxes d’objets
  + Gérer les dossiers de recherche
  + Gérer les sources de données
  + Scanner le contenu (scan puis importe les objets de contenu depuis les dossiers de recherches)
  + Exporter le contenu (exporte les objets de contenu dans une source données)
  + Importer le contenu (importe les objets de contenu depuis une source données)

# Définition du système

## Base de données

Une base de données compatible ODBC permettra de stocker et partager les données entre les clients du système.

## Interface de gestion

Une interface HIM et en ligne de commandes permettra la gestion des données du projet.

# Réalisation

Les programmes sont implémentés en langage C# et repose sur la plateforme de développement .Net de Microsoft avec l’environnement de développement Visual Studio 2013. Le model de données est commun aux 2 programmes et est développé dans une librairie externe.

La base de données est développé dans le langage SQL Server est pourra être porté dans d’autre langages plus tard.

## Base de données

### Sources

Le code sources du projet permet d’initialiser la base de données dans le SGBD.

|  |  |
| --- | --- |
| BDD/ | Dossier de base du projet de base de données |
| [Provider]/ | Dossier contenant le code source associé au SGBD. [Provider] est remplacé par le nom du fournisseur d’accès (ex : SqlServer) |
| 0.Create.sql | Création de la base de données et utilisateurs |
| 1.Tables.sql | Création des tables et clés primaires et étrangères |
| 3.Functions.sql | Création des procédures stockées |
| 5.Views.sql | Création des vues |
| 6.JeuEssai.sql | Insertion du jeu d’essai pour test |

### Modèle physique de données



### Tables

Définition des tables SQL

|  |  |  |
| --- | --- | --- |
| **Employe**  Employé | | |
| Code | Data Type | Comment |
| Employe\_Id | int | Identificateur |
| Societe\_Id | int | Société |
| Nom | varchar(128) | Nom |
| Prenom | varchar(250) | Prénom |
| Index | Fields | Comment |
| PK\_EMPLOYE | Employe\_Id | Clé primaire |
| FK\_EMPLOYE\_R1\_SOCIETE | Societe(Societe\_Id) | Clé étrangère de la société |
| Trigger | Condition | Comment |
| - | - | - |

### Procédures stockées

Définition des procédures stockées SQL

## Librairie Model de données

Le programme de données est implémenté dans une librairie .

### Sources

Le code sources du projet permet d’initialiser la base de données dans le SGBD.

|  |  |
| --- | --- |
| App/ | Dossier de base du projet de base de données |
| AppModelLib/ | Projet de la libraire du model de données |

### Diagramme de classes



### Entités

Définition des classes d’entités

|  |  |  |
| --- | --- | --- |
| **EditorStates**  Représente les données d’état de l’interface IHM | | |
| Code | Data Type | Comment |
| Employe\_Id | int | Identificateur |
| Nom | string | Nom |
| Prenom | string | Prénom |
| Relation | Data Type | Comment |
| Societe | Societe | Société |

|  |  |  |
| --- | --- | --- |
| **EditorSampleCode**  Extrait de code source utilisé comme exemple pour tester un objet de syntaxe | | |
| Code | Data Type | Comment |
| Employe\_Id | int | Identificateur |
| Nom | string | Nom |
| Prenom | string | Prénom |
| Relation | Data Type | Comment |
| Societe | Societe | Société |

## Application Console

L’application console reprend le model de données et applique les arguments de l’application au projet.

### Sources

|  |  |
| --- | --- |
| App/ | Dossier de base du projet de base de données |
| ConsoleApp/ | Projet de l’application console |

### Arguments de la ligne de commande

|  |  |
| --- | --- |
| -a [action] | Action à réaliser, peut prendre l’une des valeurs suivantes :   * **init** : Réinitialise le projet * **add\_cpp\_syntax**: Ajoute des objets de syntaxes standard C++ * **import\_syntax**: Importe des objets de syntaxes depuis une arborescence de fichiers * **add**: Ajoute des paramètres de recherche * **scan**: Scan les données * **toxml**: Exporte les données dans un fichier XML * **print** : Affiche les données |
| -p [inputProjectFile] | Spécifie l’emplacement du fichier de projet |
| -i [inputDir] | Spécifie l’emplacement des fichiers à scanner |
| -o [outputFile] | Spécifie l’emplacement des fichiers à exporter |
| -f [inputFilter] | Spécifie le filtre à appliquer à la recherche de fichiers. Vous pouvez utiliser l’étoile (\*) comme caractère d’échapement par exemple « \*.cpp » pour rechercher tous les fichiers C++. |
| -d [defDir] | Spécifie l’emplacement des définitions d’objets |
| -v [version] | Spécifie la version du projet |
| -t [title] | Spécifie le titre du projet |
| -r [recursive] | Spécifie que la recherche est récursive |
| -s [syntaxDir] | Spécifie l’emplacement des fichiers de syntaxe (utilisé avec l’action **import\_syntaxe**) |
| -g [groupName] | Spécifie le nom du groupe |

## Application IHM

L’interface de gestion reprend les

### Sources

|  |  |
| --- | --- |
| App/ | Dossier de base du projet de base de données |
| Editor [WPF]/ | Projet de l’application IHM |

### Diagramme de classes



### Entités

Définition des classes d’entités

|  |  |  |
| --- | --- | --- |
| **EditorStates**  Représente les données d’état de l’interface IHM | | |
| Code | Data Type | Comment |
| Employe\_Id | int | Identificateur |
| Nom | string | Nom |
| Prenom | string | Prénom |
| Relation | Data Type | Comment |
| Societe | Societe | Société |

|  |  |  |
| --- | --- | --- |
| **EditorSampleCode**  Extrait de code source utilisé comme exemple pour tester un objet de syntaxe | | |
| Code | Data Type | Comment |
| Employe\_Id | int | Identificateur |
| Nom | string | Nom |
| Prenom | string | Prénom |
| Relation | Data Type | Comment |
| Societe | Societe | Société |

# Redistribuable

Le code sources du projet permet d’initialiser la base de données dans le SGBD.

## Arborescence

|  |  |
| --- | --- |
| App/ | Dossier de base du projet de base de données |
| AppModelLib/ | Projet de la libraire du model de données |
| ConsoleApp/ | Projet de l’application console |
| Editor [WPF]/ | Projet de l’application IHM |