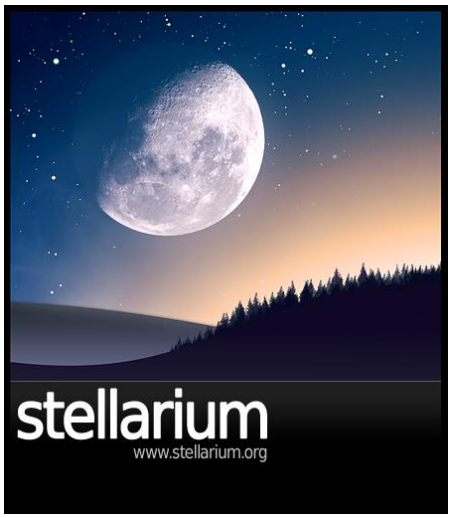
Tutoriel de développement d'un plugin pour Stellarium



Thibaud Le Doledec
Clément Ailloud
12/02/2019

Thibaud Le Doledec
Clément Ailloud

Table des matières

I. Présentation

Ce tutoriel a pour but d'expliquer comment développer un plugin pour Stellarium. J'expliquerai la manière de procéder depuis un environnement Linux vierge (distribution Ubuntu), de l'installation de l'IDE à l'implémentation du plugin dans Stellarium.

II. Initialisation du projet

1. Installation de Qt Creator

Premièrement, nous allons voir l'installation de l'IDE Qt Creator qui permet l'ouverture, l'édition, l'exécution ainsi que le lancement du logiciel Stellarium.

Pour télécharger Qt Creator, il faut se rendre sur la page de téléchargement du site de Qt, puis choisir une licence (la « open source » convient très bien) et enfin suivre les instructions permettant à la fois l'installation de Qt Creator ainsi que de l'environnement Qt (la dernière version est conseillée).

2. Installation de CMake

Il faut ensuite installer l'utilitaire CMake permettant la compilation de Stellarium ainsi que du plugin que nous développerons. Pour ce faire il faut lancer un terminal puis taper la commande suivante :

```
~$ sudo apt-get install cmake
```

Puis pour tester son installation taper la commande suivante :

```
~$ cmake –version
```

3. Téléchargement des sources

Vient ensuite l'initialisation de l'environnement de travail et le téléchargement des sources de Stellarium, les commandes suivantes permettent la création du répertoire de travail et le clonage du dépôt Git :

```
~$ mkdir DEV
~$ cd DEV
~/DEV$ git clone https://github.com/Stellarium/stellarium.git
```

III. Création d'un plugin pour Stellarium

1. Plugin exemple

Afin de créer un plugin pour Stellarium il existe plusieurs plugin exemple :

- HelloStelModule : Le plugin HelloStelModule est le plugin le plus minimaliste de Stellarium et est destiné à fournir l'implémentation minimum d'un plugin aux personnes souhaitant en développer un. Ce plugin affiche « Hello World ! » dans la fenêtre de Stellarium, il est donc aussi un exemple de comment dessiner dans l'application.
- CompassMark : Le plugin CompassMark est un plugin exemple ajoutant un bouton à l'interface et permettant l'affichage de lignes de manière plus avancé que le précédent. Ce plugin est un bon exemple de l'utilisation des fichiers «.qrc » (Qt Resource File)
- PointerCoordinates : Le plugin PointerCoordinates est un plugin exemple ajoutant une boite de dialogue dans Stellarium, il est donc un exemple de l'intégration de fenêtre dans l'interface graphique, il permet également de montrer comment récupérer les informations sur un objet de Stellarium (un astre).

2. Intégration d'un plugin dans Stellarium

Prenons comme exemple un plugin MyPlugin qui comprend une classe composée de MyPlugin.h et MyPlugin.cpp :

1) Afin de faire de cette classe un plugin pour Stellarium, elle doit hériter de la classe StelModule qui définit le comportement d'un plugin. La classe MyPlugin doit alors ré-implémenter 5 méthodes de la classe :
   - La methode : void *init*(); Elle permet la création des éléments graphique du plugin.
   - La methode : void *update*(double); Elle permet l'actualisation des élément garphique du plugin.
   - La methode : void *draw*(StelCore* core); Elle permet l'affichage de certains des éléments graphique du plugin.
   - La methode : double *getCallOrder*(StelModuleActionName actionName) const;
   - La methode : bool *configureGui*(bool show); Elle permet l'affichage des boites de dialogue et des fenêtres du plugin.
2) Pour que cette classe soit chargée par le logiciel Stellarium elle doit intégrer une seconde classe que l'on appelera « MyPluginStelPluginInterface » et qui hérite de QObject et StelPluginInterface. Cette classe doit ré-implémenter 3 méthodes :
   - La méthode : StelModule* *getStelModule*() const; Elle permet de retourner une instance de notre objet MyPlugin.
   - La méthode : StelPluginInfo *getPluginInfo*() const; Elle permet de retourner les informations sur ce plugin.
   - Et la méthode : QObjectList *getExtensionList*() const { return QObjectList(); }

3) Pour que Stellarium compile statiquement notre plugin il faut rajouter dans le CmakeLists.txt principale du projet à la ligne 322 : « ADD_PLUGIN(MyPlugin 1) »

4) Et enfin, pour que le logiciel Stellarium charge notre plugin lors de son démarrage il faut rajouter dans le fichier StelApp.cpp à la ligne 93 :

```
#ifdef USE_STATIC_PLUGIN_MYPLUGIN
      Q_IMPORT_PLUGIN(MyPluginStelPluginInterface)
#endif
```

IV. Le plugin Autoscope

1. Description du plugin

Le plugin Autoscope est destinée à pouvoir piloter le télescope de notre projet. Il à doit permettre de tracker des astres et de récupérer ses coordonnées d'azimut et d'altitude afin de pouvoir positionner le télescope, mais il doit également permettre de prendre des photos (grâce à la caméra du télescope) et de les récupérer.

2. Description des classes

Le plugin Autoscope est composée de sept classes :

Classe principale :

- Autoscope.hpp et .cpp : cette classe est la classe principale du projet elle permet la création des autres fenêtres ainsi que des boutons de l'interface graphique. Cette classe permet également la manipulation des objets Stellarium afin de récupérer leurs coordonnées.

Interfaces utilisateur (dossier gui) :

- AutoscopeFtpDialog.hpp et .cpp : Cette classe permet la création d'une fenêtre de dialogue ftp permettant la récupération des photos sur le télescope.
- AutoscopePictureWindowForm.hpp et .cpp : Cette classe permet la création d'une fenêtre destinée à l'affichage des photos récupérées sur le télescope.
- AutoscopeWindowForm.hpp et .cpp : Cette classe permet la création de la fenêtre principale du plugin. Cette fenêtre sert à rechercher et/ou sélectionner un astre mais également l'activation et le paramétrage des autres fenêtres.

Networking (dossier network) :

- CommandParser.hpp et .cpp : Cette classe permet la construction de chaque commande envoyée par le plugin au télescope.
- tcp_client.hpp et .cpp : Cette classe permet la création d'un client TCP utilisé pour envoyer les commandes au télescope.
- tcp_server.hpp et .cpp : Cette classe permet la création d'un serveur TCP, mais elle n'est pas utilisée par le plugin.

V. Utilisation de Stellarium

Dans ce plugin nous avons utilisé des ressources existantes dans Stellarium pour tracker des astres. Nous allons donc expliquer la démarche permettant d'utiliser des ressources pré-existante de Stellarium.

Afin de trouver comment traker un astre, nous nous somme servi d'un plugin similaire : le plugin PointerCoordinates. En effet ce plugin permet d'afficher toutes les informations sur un astre or le plugin Autoscope n'a besoin que des informations d'azimut et d'altitude. Nous avons donc regardé comment ce plugin récupérait ces informations puis nous l'avons imité.

Tous les astres de Stellarium sont des instances d'objets de type « StelObjectP », en utilisant deux gestionnaire, à savoir StelObjectMgr et StelMovementMgr, nous pouvons manipuler ce type d'objet. En d'autres termes, l'instance de StelObjectMgr nous permet de rechercher et de sélectionner un objet grâce à son nom ainsi que d'en récupérer une instance, et l'instance de StelMovementMgr permet de déplacer la vue vers l'objet en question.

En conclusion, utiliser les plugins exemple permet de découvrir l'étendu des possibilités qu'offre Stellarium. Cependant cela ne s'arrête pas aux plugins exemple car le fait que Stellarium soit un logiciel opensource permet à qui en a envie de pouvoir le modifier de façon plus profonde.

VI. Annexes

1. Annexe 1 : Description du contenu du dossier Autoscope

Contenu du dossier Autoscope :

- Doc : Ce dossier contient la documentation du plugin, pour ouvrir la documentation il suffit d'ouvrir le ficher « index.html ».
- Resources : Ce dossier contient les images utilisées par le plugin.
- Src : Ce dossier contient toutes les sources du projet.

2. Annexe 2 : Lien vers le github du projet

Projet Autoscope