7. Gestion des paramètres de configuration : QSettings

7.1. Introduction

QSettings est une classe de Qt permettant de stocker et de récupérer des paramètres de configuration persistants de manière simple et efficace. Elle est largement utilisée pour gérer les préférences d'application telles que les réglages utilisateur, les paramètres de fenêtre, les configurations réseau, etc.

7.2. Principales caractéristiques de QSettings

- **Persistance des données** :

  QSettings stocke les données de configuration de manière persistante, c'est-à-dire que les données sont conservées même après la fermeture de l'application.

- **Support Multi-Plateforme** :

  - Sous Windows, les paramètres sont stockés dans le **Registre**.
  - Sous macOS, ils sont enregistrés dans les **plist** (Property List files).
  - Sous Linux, les paramètres sont enregistrés dans des fichiers texte au format INI.

- **Support des Types de Données** :

  QSettings peut gérer divers types de données, notamment des chaînes (**QString**), des nombres, des booléens, **QStringList**, **QSize**, **QPoint**, **QColor**, etc. et même des structures plus complexes comme **QVariant**.

- **Format des données** :

  Les paramètres peuvent être stockés dans deux formats principaux : INI (fichiers de configuration simples) ou Registre (spécifique à Windows).

7.3. Utilisation de QSettings

- **Initialisation** :

  QSettings est initialisée en spécifiant le nom de l'organisation et celui de l'application :

  Initialisation
  ```
  QSettings settings("MyCompany", "MyApp");
  ```

  Sans préciser le format des données, comme dans l'exemple ci-dessus, le constructeur utilise le format Natif, sous Linux le format INI, sous Windows la base de registres et sous macOS un fichier plist le contenu est alors lisible sous un format JSON. Le format peut-être explicitement préciser en utilisant un troisième paramètre pour le constructeur. Il prend entre autre une des valeur suivante :

QSettings::NativeFormat	0	Stocke les paramètres en utilisant le format de stockage le plus approprié pour la plateforme. (valeur par défaut)
QSettings::IniFormat	1	Stocke les paramètres dans des fichiers INI. Les fichiers INI ne font pas de distinction entre les données numériques et les chaînes utilisées pour les encoder, donc les valeurs écrites sous forme de nombres seront lues sous forme de Qstring.
QSettings::Registry32Format	2	Uniquement pour Windows : Accède explicitement au registre système 32 bits à partir d'une application 64 bits exécutée sur Windows 64 bits.
QSettings::Registry64Format	3	Uniquement pour Windows : Accède explicitement au registre système 64 bits à partir d'une application 32 bits exécutée sur Windows 64 bits.

  Qsettings::NativeFormat est à privilégier. D'autres valeurs pour cette énumération existes, se référer à la documentation officiel de Qt.

  La localisation de la configuration, si elle n'est pas précisé dans le deuxième paramètres, est pour Windows dans la **base de registres**, pour Linux dans le répertoire **~/.config**, pour macOS dans le répertoire **~/Library/Preferences/**

- **Écriture des paramètres** :

  Les paramètres sont enregistrés avec des clés sous forme de chaînes et des valeurs associées.

  Écriture des paramètres
  ```
  settings.setValue("window/size", QSize(800, 600));
  settings.setValue("window/position", QPoint(100, 100));
  settings.setValue("user/username", "rbidochon");
  ```

- **Lecture des paramètres** :

  Pour lire les paramètres, on utilise la méthode `value()`, avec la possibilité de spécifier une valeur par défaut si la clé n'existe pas.

  Lecture des paramètres
  ```
  QSize size = settings.value("window/size", QSize(640, 480)).toSize();
  QPoint position = settings.value("window/position", QPoint(50, 50)).toPoint();
  QString username = settings.value("user/username", "defaultUser").toString();
  ```

  Le deuxième paramètre de la méthode **value** est facultatif, il indique une valeur par défaut si la clé n'existe pas.

- **Parcourir les paramètres** :

  QSettings permet également de parcourir les clés existantes pour itérer sur les paramètres enregistrés.

  Parcours des paramètres
  ```
  QStringList keys = settings.allKeys();
  for (const QString &key : keys) {
      qDebug() << key << " = " << settings.value(key).toString();
  }
  ```

- **Suppression des paramètres** :

  Les paramètres peuvent être supprimés avec `remove()` :

  Suppression d'un paramètre
  ```
  settings.remove("user/username");
  ```

7.4. Exemple simple

Écriture des paramètres
```
#include <QSettings>
#include <QDir>
#include <QSize>
#include <QPoint>
#include <QDebug>
int main() {
    QSettings settings("MyCompany", "MyApp");
    // Écrire des paramètres
    settings.setValue("window/size", QSize(1024, 768));
    settings.setValue("window/position", QPoint(200, 150));
    settings.setValue("user/username", "LinuxUser");
    qDebug() << "Settings written successfully!";
    return 0;
}
```

Lecture des paramètres
```
#include <QSettings>
#include <QSettings>
#include <QDir>
#include <QSize>
#include <QPoint>
#include <QDebug>
int main() {
    QSettings settings("MyCompany", "MyApp");
    // Lire des paramètres avec une valeur par défaut si le paramètre n'existe pas
    QSize size = settings.value("window/size", QSize(800, 600)).toSize();
    QPoint position = settings.value("window/position", QPoint(100, 100)).toPoint();
    QString username = settings.value("user/username", "defaultUser").toString();
    qDebug() << "Window size:" << size;
    qDebug() << "Window position:" << position;
    qDebug() << "Username:" << username;
    return 0;
}
```

- **Fichier de configuration au format INI**

Sous Linux, après avoir exécuté le premier programme, un fichier `MyApp.conf` sera créé dans `~/.config/MyCompany/` avec le contenu suivant :

MyApp.conf

```
[user]
username=LinuxUser

[window]
position=@Point(200 150)
size=@Size(1024 768)
```

Remarques

Les chemins spécifiques aux utilisateurs (`~/.config`) permettent de gérer les configurations spécifiques à chaque utilisateur.

Les fichiers de configuration sont facilement accessibles et éditables avec un éditeur de texte.

7.5. Pour aller plus loin

7.5.1. Vérification de l'existence d'une clé

La méthode **contains** de **QSettings** est utile pour vérifier si une clé spécifique existe dans les paramètres de configuration. Cela permet d'éviter de lire des valeurs non définies ou de vérifier la présence d'une clé avant de l'utiliser.

QSetting::contains

```
bool contains(const QString &key) const;
```

- **key** : La clé sous forme de chaîne de caractères que vous souhaitez vérifier.
- **Retour** : Renvoie true si la clé existe, sinon false.

Exemple d'utilisation

```
#include <QSettings>
#include <QDir>
#include <QDebug>
int main() {
   QSettings settings("MyCompany", "MyApp");

   // Vérifier si une clé existe avant de l'utiliser
   if (settings.contains("window/size")) {
      QSize size = settings.value("window/size").toSize();
      qDebug() << "Window size existe:" << size;
   } else {
      qDebug() << "Window size non affectée, utilisation de la valeur par défaut.";
      settings.setValue("window/size", QSize(800, 600));
   }
   if (settings.contains("user/username")) {
      QString username = settings.value("user/username").toString();
      qDebug() << "Username :" << username;
   } else {
      qDebug() << "Username non trouvé, utilisation de la valeur par defaut.";
      settings.setValue("user/username", "DefaultUser");
   }
   return 0;
}
```

Avant de lire une valeur, le programme vérifie si la clé `window/size` ou `user/username` existe. Si une clé n'existe pas, une valeur par défaut est enregistrée.

L'utilisation de la méthode **contains** permet d'éviter les erreurs en vérifiant l'existence des paramètres avant de lire des valeurs non définies. Elle facilite l'implémentation d'une logique conditionnelle basée sur la présence de ces paramètres, ce qui est particulièrement utile pour vérifier l'état de la configuration avant de charger des valeurs ou d'afficher des données à l'utilisateur. Cette méthode garantit une gestion des paramètres plus robuste et fiable, surtout lors de l'initialisation ou de la validation des configurations d'une application.

7.5.2. Structuration Hiérarchique des Paramètres

Bien que non indispensable, il est possible de structurer les paramètres sous forme de groupes logiques, ce qui s'avère particulièrement utile pour les configurations complexes. Cette approche permet de réduire les répétitions lorsque plusieurs clés partagent un même préfixe, rendant les fichiers de configuration plus lisibles et mieux organisés en sections.

La structuration hiérarchique est facilitée par les méthodes suivantes :

- **beginGroup(const QString &prefix)** : Définit un préfixe commun pour les clés suivantes.
- **endGroup()** : Met fin au groupe en cours.

7.5.3. Exemple d'utilisation

Voici un exemple complet utilisant **beginGroup**, **endGroup**, et **contains**, qui illustre comment structurer les paramètres hiérarchiquement tout en vérifiant leur existence avant de les lire :

Gestion de Configuration avec Groupes et Vérification des Paramètres

```
#include <QSettings>
#include <QDir>
#include <QSize>
#include <QPoint>
#include <QDebug>
int main() {
    // Initialisation de QSettings avec organisation "MyCompany" et application "MyApp"
    QSettings settings("MyCompany", "MyApp");
    // Écriture de paramètres dans des groupes
    settings.beginGroup("window");
    settings.setValue("size", QSize(1024, 768));
    settings.setValue("position", QPoint(200, 150));
    settings.endGroup();
    settings.beginGroup("user");
    settings.setValue("username", "LinuxUser");
    settings.endGroup();
    // Lecture des paramètres avec vérification préalable
    settings.beginGroup("window");
    if (settings.contains("size")) {
        QSize size = settings.value("size").toSize();
        qDebug() << "Window size:" << size;
    } else {
        qDebug() << "Window size not set.";
    }
    if (settings.contains("position")) {
        QPoint position = settings.value("position").toPoint();
        qDebug() << "Window position:" << position;
    } else {
        qDebug() << "Window position not set.";
    }
    settings.endGroup();
    settings.beginGroup("user");
    if (settings.contains("username")) {
        QString username = settings.value("username").toString();
        qDebug() << "Username:" << username;
    } else {
        qDebug() << "Username not set.";
    }
    settings.endGroup();
    return 0;
}
```

Remarque

QSettings ne fournit **pas de mécanisme de cryptage natif** pour les fichiers de configuration. Pour manipuler des données sensibles, il est nécessaire de chiffrer manuellement ces données avant de les enregistrer.