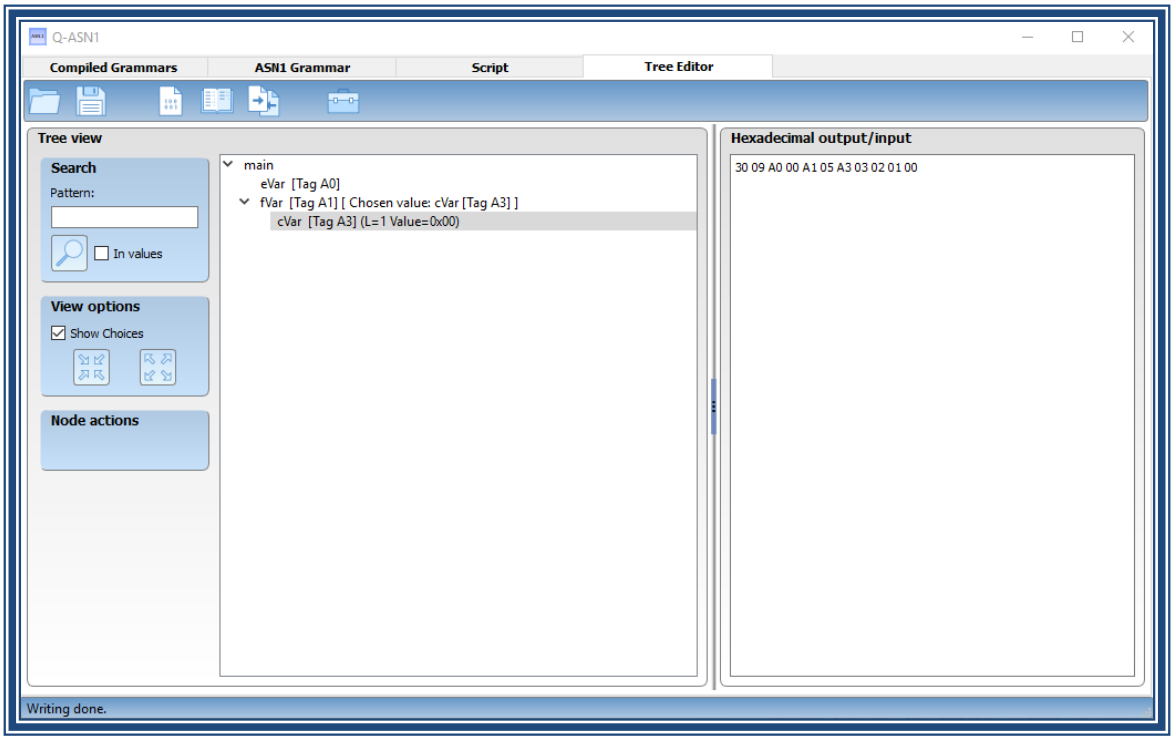
Q-ASN.1: travaillez en toute simplicité avec l'ASN.1 !



Alexandre Szydlowski, 2018

Sommaire

Introduction

L'ASN.1 est un standard défini conjointement par l'organisation internationale de normalisation, la commission électrotechnique internationale et l'union internationale des télécommunications.

ASN.1 (Abstract Syntax Notation One) est un standard international spécifiant une notation destinée à décrire des structures de données. La description en ASN.1 d'une structure de données a pour but d'obtenir une structure pouvant être sérialisée et désérialisée indépendamment d'un encodage lié à un matériel particulier, et sans ambiguïté.

S'il n'y a qu'une seule manière de décrire une structure en ASN.1, il existe en revanche plusieurs encodages pour les données décrites par l'ASN.1, par exemple les encodages BER, DER ou encore CER.

Quelques liens utiles concernant l'ASN.1 :

- en.wikipedia.org/wiki/Abstract_Syntax_Notation_One
- en.wikipedia.org/wiki/X.690
- oss.com/resources/resources

Q-ASN.1

Q-ASN.1 supporte l'encodage DER, et une partie des types défini par l'ASN.1 (NULL, BOOLEAN, INTEGER, ENUMERATED, REAL, BIT STRING, IA5String, UTF8String, OCTET STRING, UTCTime, OBJECT IDENTIFIER, SEQUENCE, SET, SEQUENCE OF, CHOICE). Il supporte les variables optionnelles, les tags automatiques, implicites et explicites, ainsi que partiellement les contraintes, et les extensions.

J'ai commencé le développement de Q-ASN.1 dans le but de pratiquer le C++. Le point de départ a été l'encodage des données, puis petit à petit, l'interface graphique, puis l'ajout d'un compilateur ASN.1.

Q-ASN1 permet ainsi :

- la sérialisation et désérialisation de données encodées en ASN.1 DER
- la visualisation et l'édition des données sous forme d'un arbre
- la compilation d'un script ASN.1 en C++, et en JavaScript
- la lecture de scripts JavaScript, permettant le chargement dynamique d'une structure de données
- l'intégration dans votre programme par l'appel à aux dlls

Le point fort de Q-ASN.1 réside dans l'utilisation de JavaScript pour charger une grammaire ASN.1. En effet, la plupart des compilateurs ASN.1 permettent de générer du code en langage qui nécessiteront une compilation avant d'être utilisés. Ce n'est pas le cas du JavaScript, qui peut être lu immédiatement grâce au moteur JavaScript intégré dans Qt : QtScript.

En réalité, la partie JavaScript peut rester totalement invisible pour l'utilisateur : à partir d'un script ASN.1, on clique sur un bouton, et on obtient un arbre directement éditable, et sérialisable ! Bien entendu, il est aussi possible de remplir les données de cet arbre par désérialisation.

Le JavaScript offre néanmoins une possibilité supplémentaire : remplir les données de la structure. En effet, toutes les fonctions pour lire et éditer les valeurs sont accessibles, et apportent donc un moyen plus efficace que le remplissage « à la main » dans l'arbre.

Cette possibilité de génération dynamique d'arbre ASN.1 permet de développer une grammaire et faire différents essais, en s'affranchissant de l'étape de compilation, qui peut être longue et fastidieuse.

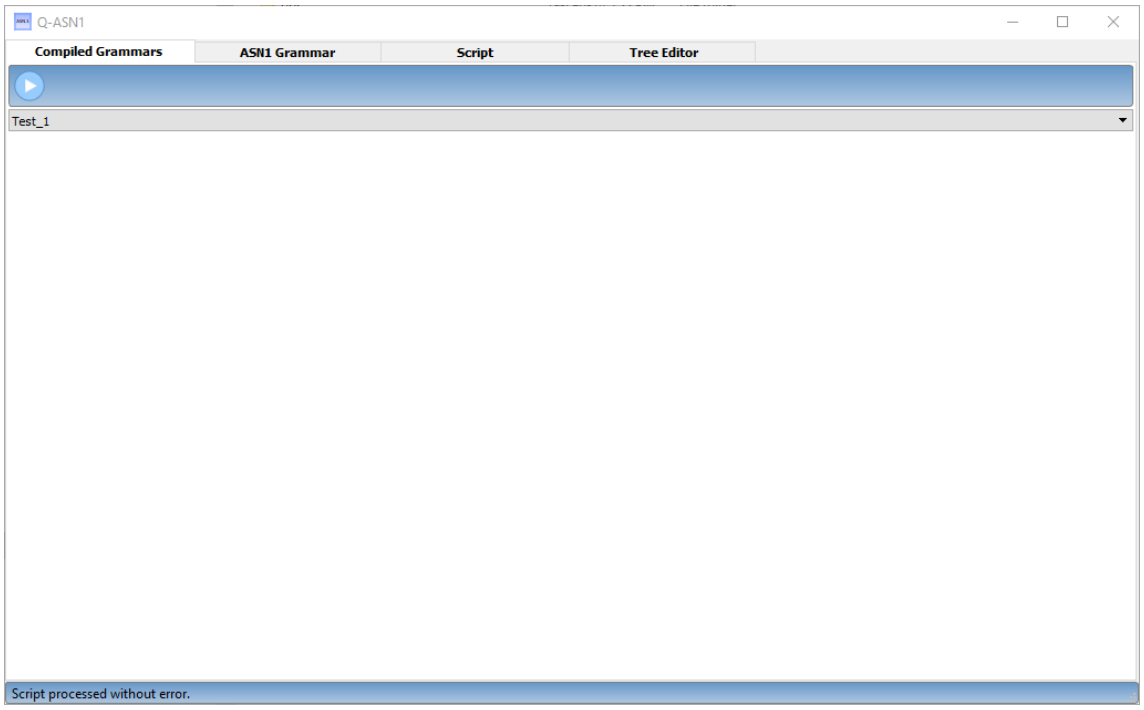
Une fois le script ASN.1 mis au point, il est possible de générer un fichier C++ utilisable dans un programme externe, afin d'améliorer les performances. Il faut alors faire appel aux dlls de Q-ASN.1.

Utilisation de Q-ASN.1

Avant tout, il faut exécuter ASN1_Qt_gui.exe. L'interface principale se compose de 4 onglets :

- la sélection des grammaires compilées. Celles par défaut n'auront que peu d'intérêt pour les utilisateurs, mais de nouvelles pourront être ajoutées (l'outil devra être recompilé cependant)
- l'éditeur d'ASN.1. L'arbre à gauche liste les mots clefs supportés avec un description. Il est possible de « glisser-déposer » de l'arbre vers l'éditeur afin de composer une nouvelle grammaire facilement.
- l'éditeur JavaScript, qui fonctionne de la même manière que l'éditeur ASN.1.
- l'arbre ASN.1, qui permet l'édition noeud par noeud, la sérialisation, et la désérialisation. L'onglet propose également des outils de conversion « type ASN.1 » vers Hexadécimal et vice versa.

Les grammaires compilées



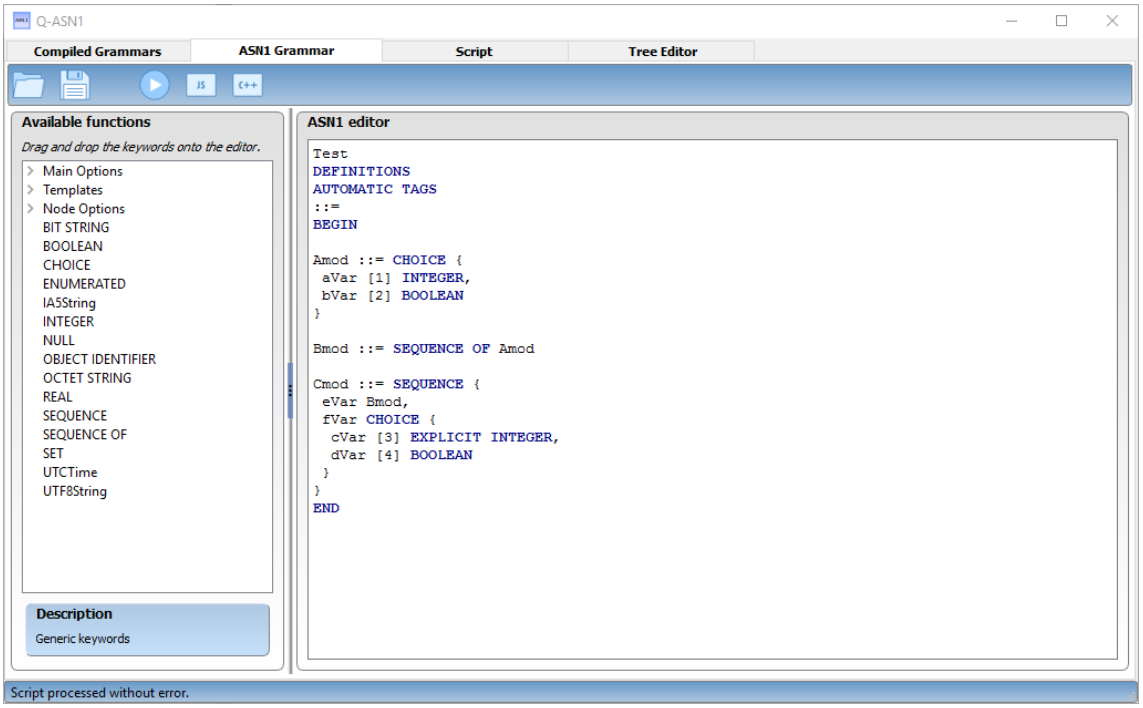
L'interface de selection des grammaires compilées.

Cet onglet présente de l'intérêt si vous y avez ajouté vos grammaire ASN.1. Pour ce faire, il vous faudra recompiler l'outil en y intégrant quelques changements. La liste des grammaires compilées se trouve dans le fichier source «ASN1_Qt_gui/UI_GrammarComp.cpp », fonction MakeCompGrammarPane(), ajoutez le nom de votre grammaire à la liste du Combo Box.

Ensuite, dans le fichier «ASN1_Qt_gui/UI_Main.cpp », la fonction CompToObj() devra être étendue avec un appel à la fonction retournant votre structure ASN.1.

Sur l'interface graphique, cliquer sur la flèche vous envoi directement à l'éditeur ASN.1 (quatrième onglet), qui contient maintenant la structure sélectionnée.

L'éditeur ASN.1

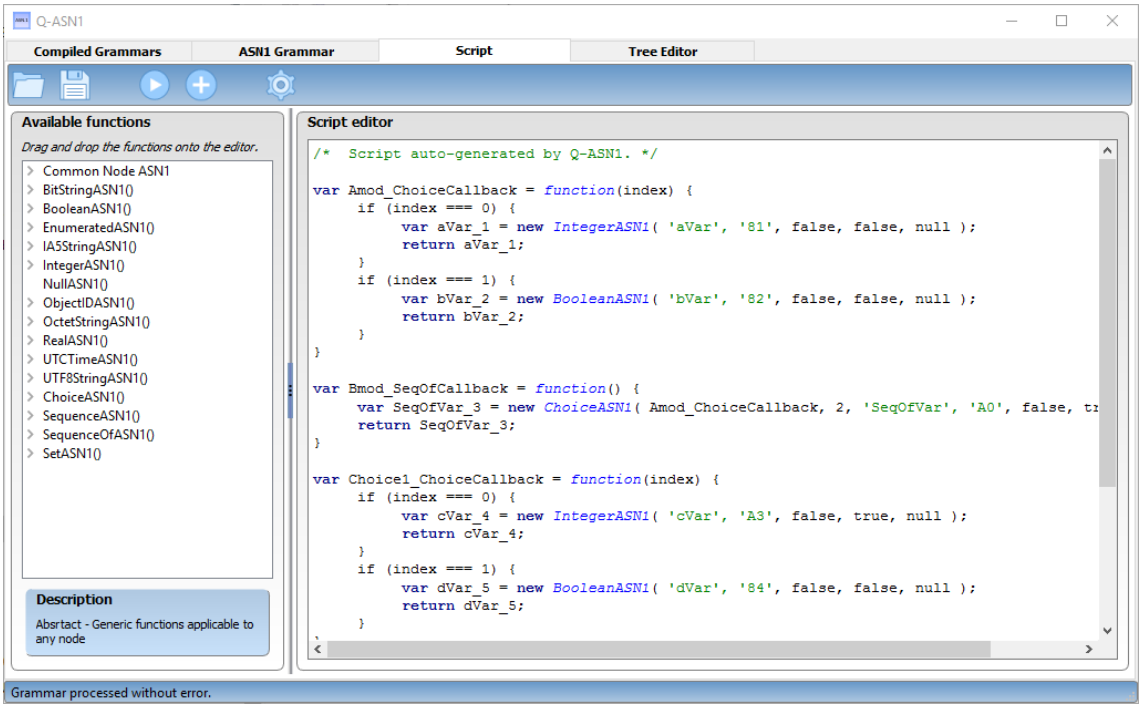


L'interface d'édition de grammaire ASN.1.

Celui-ci permet de composer sa structure ASN.1 facilement. Il est nécessaire de connaître un minimum le standard ASN.1 avant de commencer. La structure écrite, plusieurs possibilités s' offrent à vous:

- charger/sauvegarder, comme leur noms l'indiquent, permettent de sauver votre travail dans un fichier externe. Q-ASN.1 ne sauvegardera rien lui-même.
- convertir en JavaScript. Si la grammaire est correctement lue, il est résultera un script JavaScript qui s'ouvrira directement dans le troisième onglet. S'il y a des erreurs, l'utilisateur sera notifié et renvoyé au script ASN.1 pour correction.
- convertir en C++. Le mécanisme de vérification de la grammaire est le même que pour JavaScript. Si tout est correct, Q-ASN.1 proposera de sauvegarder un fichier C++ réutilisable par ASN1_lib.dll ou par Q-ASN.1.
- la flèche envoi directement l'utilisateur à l'éditeur ASN.1 (quatrième onglet), sans passer par l'étape JavaScript, si la grammaire ne contient pas d'erreur. Derrière la scène, l'outil utilise tout de même JavaScript, mais il peut être pratique de sauter cette étape.

L'éditeur JavaScript



L'interface d'édition de script JavaScript.

L'affichage du code JavaScript peut paraître peu utile au premier abord, mais celui-ci possède deux avantages :

- il est possible de le debugger
- il permet de remplir les données de la structure créée. Le langage ASN.1 possède bien une manière de le faire, mais l'éditeur ASN.1 de cet outil ne supporte pas ces fonctionnalités.

Le panneau de gauche résume les fonctions supportées, avec une description. Il est en effet possible d'écrire directement sa grammaire ASN.1 en JavaScript, plutôt qu'en ASN.1. Cela permet d'être plus proche du code C++, et de debugger, bien que cela ne soit pas très intéressant si on s'en tient à ça : le compilateur ASN.1 génère le script lui-même, et si le script est généré, c'est qu'il n'y a (en théorie) pas d'erreur.

Il est à noter que les propriétés ASN.1 de l'objet structure ne peuvent être modifiées après sa génération (par exemple le tag, l'ordre des objets dans un SEQUENCE). C'est logique puisque la structure est statique. Seules les données changent. En d'autres termes : Un noeud INTEGER ne pourra pas être changé en noeud REAL (il faut régénérer la grammaire si c'est le souhait de l'utilisateur), mais sa valeur pour effectivement passer d'un entier à un autre.

Là ou JavaScript devient donc intéressant, c'est pour remplir ces données. Ce travail fastidieux peut être grandement simplifié par un script.

Lorsque la grammaire est générée, il en résulte un seul objet, la racine de la structure, à partir de laquelle il est possible d'appeler les fonctions « set ».

Les valeurs peuvent ainsi être sauvegardées dans le fichier JavaScript, et réutilisées. L'utilisateur a également la possibilité d'utiliser les fonctionnalités du JavaScript, comme les boucles, conditions, ou même d'y inclure son propre code.

Un point essentiel ici, il ne faut pas oublier d'appeler la fonction registerGrammar() sur le noeud racine. En effet, celui-ci sera le noeud affiché à la racine d'arbre dans l'onglet suivant !

Le debugger JavaScript de Qt peut être lié au besoin avant l'exécution des scripts, en cliquant sur l'option sous la petite roue dans la barre d'action.

Deux boutons d'exécution existent :

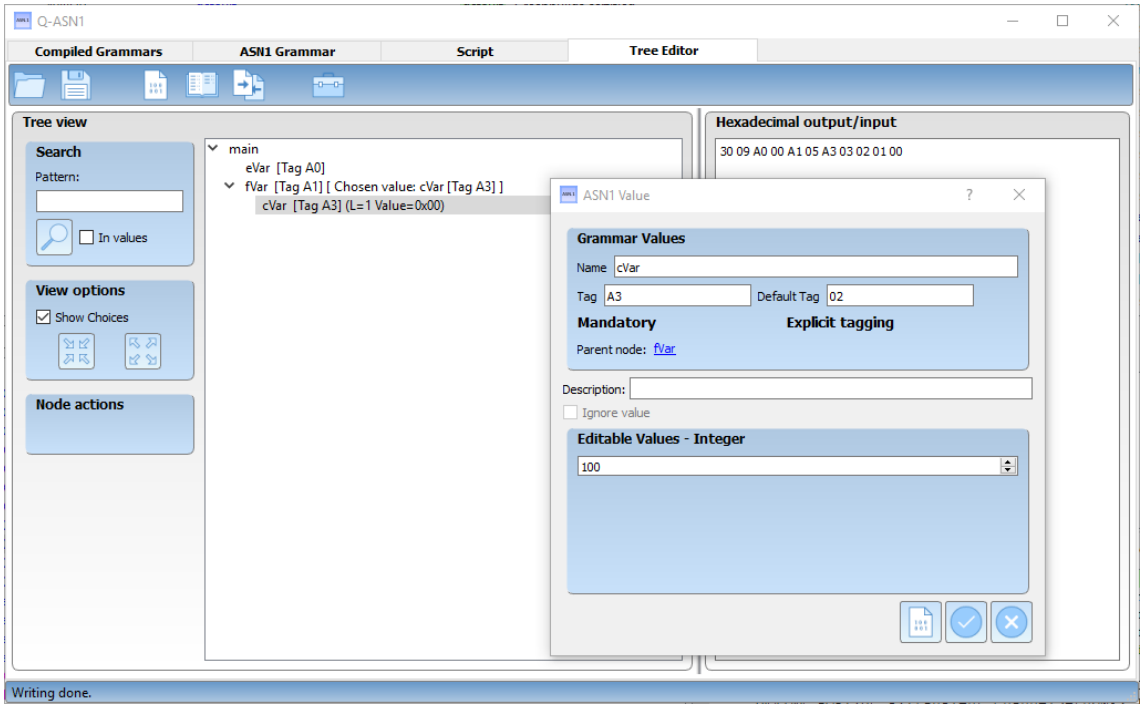
- la « flèche », qui instancie un nouveau moteur JavaScript, et donc efface toutes les données des scripts précédents
- le « plus », qui exécute un script additionnel, en conservant les données des scripts précédents.

Cela permet, par exemple, d'exécuter un script pour générer la grammaire, puis d'exécuter des scripts additionnels pour remplir des données sur la structure.

Pour un moteur JavaScript, il ne peut y avoir qu'un noeud racine, et donc une seule structure. Celle-ci peut en revanche être réutilisée pour lui appliquer de nouvelles données.

Après l'exécution d'un script, s'il n'y a pas eu d'erreur, l'utilisateur est dirigé vers le quatrième onglet.

L'éditeur de données ASN.1 sous forme d'arbre



Fenêtre d'édition pour un noeud INTEGER.

Cet éditeur sous forme d'arbre est bien un éditeur de données. Il n'est en aucun cas possible de changer la structure des données. Un INTEGER ne pourra pas être transformé en REAL, et son TAG ne pourra pas être changé. Tout comme les objets d'un SEQUENCE ne pourront être réordonnés ou supprimés.

La structure est bien figée à cette étape, seules les données variables peuvent être modifiées. Pour modifier la structure, il faut revenir aux étapes précédentes.

Il est ainsi possible dans l'arbre de :

- changer une valeur. Par exemple, un IA5String peut être modifié pour passer de « aaa » à « bbb »
- ignorer un noeud optionnel. En effet, si la grammaire accepte que le noeud soit présent ou non (OPTIONAL), la structure n'est pas modifiée, il s'agit en quelque sorte d'une valeur « nulle ».
- ajouter des objets aux SEQUENCE OF. Ici encore, l'ajout d'objet ne modifie pas la structure de données, qui elle indique une liste, sans en préciser le nombre d'éléments. Le nombre d'éléments ainsi que leur valeurs sont bien des données.
- réordonner les éléments dans un SEQUENCE OF
- choisir le choix pour un CHOICE
- remplir le champ « description », qui n'est pas lié au standard ASN.1 et ne sera pas sérialisé. Il peut cependant faciliter le travail de l'utilisateur, par exemple en affichant de manière claire à quoi correspond le noeud.

La fenêtre se compose de deux panneaux. A gauche, l'arbre, et à droite, un champ de texte.

Le champ de texte ne peut contenir que des caractères hexadécimaux. Il affichera la sortie de sérialisation, mais servira également d'entrée pour la désérialisation. Il convient donc de prendre soin de copier ses données dans un éditeur de texte externe pour éviter les écrasements. Dans la barre d'action, un menu propose d'ouvrir un fichier contenant de l'hexadécimal, ou de sauvegarder le contenu du champ.

Pour sérialiser, il faut cliquer sur l'action correspondante (la petite feuille avec du binaire). Tout l'arbre est sérialisé et le résultat s'affiche dans le champ texte. Il est possible de ne sérialiser qu'une branche de l'arbre par un clic droit sur un noeud de l'arbre, en sélectionnant l'action pour sérialiser à partir de ce noeud.

Pour désérialiser, il faut cliquer sur le livre dans la barre d'action. Le contenu du champ texte est alors lu par l'arbre qui va tenter d'en extraire les données. Cette opération peut échouer si le contenu lu n'a pas été généré par la même structure de données, et peut donc renvoyer une erreur. Si l'opération est un succès, les valeurs lues sont directement visible dans l'arbre. De la même manière que pour la sérialisation, il est possible de ne lire qu'à partir d'une branche. Le champ texte ne doit alors contenir que le contenu attendu par la branche.

L'ASN.1 travaillant avec des octets (deux chiffres hexadécimaux), si le champ texte comporte un nombre impair de caractères, il sera proposé à l'utilisateur d'ajouter un 0 au début (ce 0 n'a pas pour but de rendre la valeur correcte. Seulement la rendre utilisable. Mais il y a de fortes chances que ça ne fonctionne pas, et que la valeur ait été partiellement effacée…)

La troisième action (les deux fichiers avec les flèches), permet de comparer le contenu du champ texte avec le contenu de l'arbre. Ni l'arbre ni le champ texte ne seront modifiés. Si les deux sont égaux, un petit message l'indique dans la barre d'état en bas de la fenêtre. Si la comparaison trouve des différences, elles seront indiquées dans une fenêtre pop-up.

La boîte à outil permet de calculer rapidement le résultat d'une conversion « type ASN.1 » vers hexadécimal ou vice versa, sans avoir à passer par une grammaire. Ici, seule la valeur est convertie, il n'y a ni tag, ni longueur. Il n'est pas possible de convertir des noeuds complexes, puisque ceux-ci nécessiteraient une grammaire pour les définir…

L'arbre propose plusieurs fonctionnalités :

- la recherche. Si la checkbox contenu n'est pas sélectionnée, la valeur sera recherchée dans le nom, tag et description. Ceci permet par exemple, si on recherche un tag en particulier, disons '81', de ne pas le noyer dans les données, qui pourraient contenir de multiples fois '81', sans que ça n'ait de rapport avec le tag recherché.
- afficher ou cacher les noeuds CHOICE. En effet, si l'utilisateur ne souhaite pas modifier les CHOICE, ceux-ci peuvent devenir encombrants et compliquer la lecture. Cette option les cache, mais les noeuds restent bien présents, et toujours éditables si on les ré-affiche.
- étendre ou réduire l'arbre
- exécuter des actions spécifiques à chaque noeud. Les actions des SEQUENCE OF et CHOICE possèdent des raccourcis dans le panneau gauche de l'arbre, si un noeud d'un de ces types est sélectionne. Ces actions sont aussi accessible par clic droit sur un de ces noeuds dans l'arbre. Le CHOICE propose de choisir un élément parmi sa liste de choix. Le SEQUENCE OF permet de déplacer un noeud de sa liste vers le haut ou le bas, de le supprimer, ou d'en ajouter un.

En double cliquant sur un noeud dans l'arbre, on ouvre la fenêtre d'édition de la valeur. S'il s'agit d'un CHOICE ou d'un SEQUENCE OF, les actions citées ci-dessus sont retrouvées ici. Pour les autres types, il y aura un champ pour éditer leur valeur, qui sera différent en fonction du type. Il n'est pas possible de modifier directement la valeur en hexadécimal. Pour les noeuds optionnels, la checkbox 'ignorer la valeur', la retirera lors de la sérialisation. Les liens vers les noeuds enfants ou parent permettent une visualisation en lecture seulement, pour les éditer, il faut les ouvrir directement depuis l'arbre.

Enfin, par un clic droit sur un noeud, un menu propose un raccourci vers l'outil de conversion correspondant au type de noeud. La valeur du noeud n'est pas affectée par ce que l'utilisateur saisi dans le convertisseur, il ne s'agit que d'un raccourci qui peut s'avérer pratique.

Intégration de Q-ASN.1

Q-ASN.1 étant composé de plusieurs modules, il faut choisir ceux dont vous aurez besoin :

- ASN1_lib.dll : générer une structure, à partir d'une grammaire compilée (celle-ci peut être écrite à la main, ou générée par Q-ASN.1)
- ASN1_parser.dll : le compilateur ASN.1. A partir d'un script au format ASN.1, il génère un script JavaScript utilisable par ASN1_Qt_Script_lib.dll (voir ci-dessous), ou un fichier C++, utilisable par ASN1_lib.dll (voir ci-dessus)
- ASN1_Qt_Script_lib.dll : Lire un script JavaScript, et retourner la structure de données, utilisable ensuite avec ASN1_lib.dll
- ASN1_Qt_gui.exe : l'exécutable graphique qui permet d'expérimenter les fonctionnalités citées ci-dessus
- ASN1_console.exe : Exécutable de test, qui regroupe un certain nombre d'exemples

Il est à noter que ASN1_lib.dll et ASN1_parser.dll n'ont pas de dépendance avec les dlls Qt. ASN1_Qt_Script_lib.dll nécessite QtCore.dll, et QtScript.dll pour fonctionner. Évidemment, ASN1_lib.dll est nécessaire pour faire fonctionner ASN1_parser.dll et ASN1_Qt_Script_lib.dll.

Les sources étant disponibles, il est possible de tout recompiler, de manière à pouvoir utiliser directement les objets C++. Mais il est aussi possible, grâce à l'interface C de la dll, de la lier dynamiquement. Un fichier header récupérant tous les pointeurs de fonction est déjà disponible dans les sources du test : DLL_Interface.h. Il est écrit en C++ et peut donc nécessiter quelques modifications pour d'autres langages.

Le code source de ASN1_console.exe propose des exemples avec les deux méthodes.

Une fois la structure de données instanciée à partir de la grammaire, il n'y a plus qu'à la remplir, ou à la lire !

Son remplissage se fait noeud par noeud. Suivant le type de noeud, de fonctions différentes sont accessibles (par exemple: SetBooleanValue(const bool& val), ne sera accessible que sur un noeud de type BOOLEAN, et permet de changer sa valeur).

Pour parcourir l'arbre (ce qui équivaut à accéder aux éléments de SEQUENCE, SET, SEQUENCE OF, CHOICE), des fonctions sont également accessibles sur ces noeuds (par exemple GetObjectAt(unsigned int i), qui existe sur SEQUENCE, SET, SEQUENCE OF, ou encore GetSelectedChoice(), pour CHOICE)

Pour sérialiser l'arbre, il suffit d'appeler WriteIntoBuffer(ByteArray& buffer) sur le noeud racine. Plus généralement, l'appel à cette fonction sur un noeud quelconque sérialisera la branche.

Plutôt que de remplir des données pour les sérialiser, on peut vouloir les lire à partir d'un buffer. Celui-ci devra être du type ByteArray (constructible facilement à partir d'une chaine de caractères, pour peu qu'elle ne soit composée que de caractères hexadécimaux). L'appel à ReadFromBuffer(const ByteArray& buffer, char* error, size_t errorBufferSize) fera le reste. Cette fonction devra être appelée sur le même noeud que celui utilisé pour générer le buffer d'entrée (la plupart du temps, le noeud racine de la grammaire). Comme la lecture peut échouer, la fonction écrit dans un buffer contenant les erreurs rencontrées, et retourne un booléen indiquant si c'est un échec ou non.

Une fois les données désérialisées, elles sont accessibles dans l'arbre sur chaque noeud (par exemple: GetIntegerValue(), sur un noeud INTEGER)

Le type ByteArray possède une place assez centrale puisqu'il est toujours utilisé pour représenter l'hexadécimal. Des fonctions pratiques sont ajoutées sur ce type de manière à pouvoir manipuler les chaine hexadécimales. A noter que Size() retourne le nombre d'octets contenus, et non le nombre de caractères (nombre d'octets = 2 * nombre de caractère).

Les fonction définies dans le fichier Utils.h permettent d'effectuer des conversions, ou verification usuelles en lien avec l'ASN.1.

Seuls les objets dérivant de la classe ASN1_Object sont exposés à l'utilisateur. Il n'est en aucun cas possible d'accéder directement à la classe ASN1_Value ou ses dérivées. Ceci s'explique par le fait que l'utilisateur ne doit pas accéder à des fonctions manipulant les données en hexadécimal du noeud. Pour lire ou écrire une valeur, il faudra toujours passer par le type de la valeur. ASN1_Object fonctionne comme une interface permettant de masquer le contenu plus technique.

Développer Q-ASN.1

ASN1_lib

Cette librairie se compose de deux parties : les classes qui héritent de ASN1_Value, et celles qui héritent de ASN1_Object. Comme indiqué dans le chapitre précédent, ASN1_Object est l'interface exposée dans la dll, et accesible à l'extérieur. ASN1_Value correspond au fonctionnement interne de la librairie (encodage et décodage) et ne doit pas être accessible à l'extérieur de la librairie.

Pour ajouter un type ASN.1 qui ne serait pas déjà dans Q-ASN.1, c'est assez simple.

Sous le dossier ASN1_Value :

- ASN1_Value_Nodes.h, à la fin, ajouter le nouveau type à l'aide de la macro NODE_CLASS (nom, type c++, tag par défaut, valeur par défaut). Le nom sera utilisé pour composer le nom de la classe, le type C++ doit être celui qui correspond le mieux au type ASN.1 (par exemple, il conviendra d'utiliser std::string pour tout type String en ASN.1), le tag par défaut doit être celui du standard ASN.1, et la valeur par défaut est arbitraire, elle sera utilisée à la création du noeud pour le pré-remplir.
- sous le dossier ASN1_Value, créer un fichier pour le nouveau type : ASN1_ValueXXX.cpp (ou XXX correspond exactement au nom donné ci-dessus). Dedans, implémenter les méthodes de conversion statiques ASN1_ValueXXX::XXXToHex(const TYPE& input, ByteArray& output, std::string& error) et ASN1_ValueXXX::HexToXXX(const ByteArray& input, TYPE& output, std::string& error). Ce sont les conversions entre le type C++ (qui correspond au type ASN.1) et la valeur héxadécimale (du type ByteArray).
- enfin, ajoute la classe d'interface sous le dossier Object. Ici encore, grâce aux macro déjà en place, c'est très simple, il suffit d'ajouter un appel à OBJECT_CLASS_DECL(nom, type C++), dans ASN1_Object_Nodes.h, et OBJECT_CLASS_IMPL(nom, type C++) dans ASN1_Object_Nodes.cpp.
- étendre éventuellement C_Interface.h et C_Interface.cpp avec des appels respectivement à OBJECT_INTERFACE_TYPE et OBJECT_INTERFACE_TYPE_IMPL. Pour le type C++ std::string, il faudra utiliser la variante avec BUFFER, qui générera automatiquement des fonctions compatibles C avec une variable char* et un int pour la longueur de la chaine.

Le noeud est désormais disponible et utilisable à l'extérieur de la dll.

ASN1_parser

Il s'agit du compilateur ASN.1. Il est basé sur une grammaire flex/bison. Le fichier ASN1_parser.h contient les fonctions accessibles de l'extérieur.

Pour ajouter une conversion dans un langage autre que JavaScript ou C++, il faudra créer un nouveau générateur, qui héritera de la classe Generateur, laquelle contient toute les informations utiles après la lecture du code ASN.1.

Les versions de flex/bison utilisée sont les 2.4.8/2.5.8.

ASN1_Qt_Script_lib

Pour étendre la librairie script, il faut ajouter le nouveau noeud aux fichiers Node/ASN1_Script_Basic_Nodes.h/.cpp. Il ne s'agit ici que de définir l'interface entre le noeud de la librairie ASN1_lib avec l'objet du script.

ASN1_Qt_gui

Pour étendre l'interface graphique avec un nouveau type de noeud, le plus simple reste de copier ce qui est fait pour un autre noeud. Le noeud doit être ajouté à :

- la création de l'interface graphique du noeud (créer une classe sous Nodes)
- ajouter un lien vers cette interface graphique (Nodes/UI_ASN1_Value_Main.h / .cpp)
- optionnellement créer un convertisseur (sous le dossier Converters)
- ajouter le convertisseur au menu (UI_Editor_Menus.cpp)
- ajouter un lien vers le convertisseur dans le menu contextuel de l'arbre (UI_TreePanel.cpp)
- UI_GrammarJS et UIGrammarASN1 pour ajouter le noeud aux éditeurs ASN1 et JavaScript
- JSHighlighter et ASNHighlighter pour ajouter la coloration syntaxique