aLTAG3D Documentation

Version 1.2

Consortium 3D

sept. 04, 2017

Contenu:

CHAPITRE 1

Introduction

À propos de aLTAG3D

aLTAG3D permet de générer des archives de vos projets 3D, compatible avec le service d'archivage du CINES et du Consortium 3D.

Le fonctionnement d'aLTAG3D cherche à être le plus automatique possible, et donc à réduire le plus possible le nombre d'informations que vous aurez à remplir manuellement.

aLTAG3D utilise un système de graphes pour permettre un remplissage plus intuitifs des données d'archivage.

> **Attention :** Ceci est la documentation développeur du logiciel aLTAG3D, si vous cherchez la documentation utilisateur, elle est disponible ici !

CHAPITRE 2

Modification du XSD

Le fichier XSD utilisé par aLTAG3D pour fonctionner est maintenu par les chercheurs du Consortium 3D. Cependant, aLTAG3D étant un logiciel Open-Source, il est possible que vous cherchiez à modifier ce fichier XSD par vous même.

Le fichier XSD actuel est disponible ici.

Parsage du fichier XSD

Le fichier XSD n'est pas utilisable tel quel par aLTAG3D, il faut en effet effectuer une requête xquery particulière sur le fichier XSD afin d'en extraire un fichier XML utilisable par le logiciel.

Cette requête était au départ effectuée directement dans aLTAG3D, un problème de version de XQuery dans Qt nous a forcé à déplacer cette étape hors du logiciel.

Le parseur XQuery utilisé est Saxon-HE-9.

```
java -classpath saxon9he.jar net.sf.saxon.Query -q:altag.xq inputDocument=mdacst3d.
↪xsd -o:mdacst3d.xml
```

Le fichier altag.xq est disponible dans les sources.

Modification des fichiers de configuration

En plus du fichier XSD et de son équivalent parsé, aLTAG3D utilise un fichier de configuration appelé altag.json.

Ce fichier permet de rajouter des informations concernant le XSD sans pour autant le surcharger de données nécessaire uniquement à aLTAG3D et non à l'archivage en lui même.

Code source 2.1 – altag.json :

```
1 {
2     "version": "1.0",
3     "indicators": {"maillage":"nomMaillage","groupeSource":"description","objetVirtuel
↪":"vignette"},
```

```
    "mapToSIP": {"creator":"entiteResponsable","subject":"sujet",
                 "description":"descriptionArcheologique","date":"dateProjet",
                 "coverage":["dateArcheologique","lieuConservation","lieuDecouverte"],
                 "fichier":"cheminFichier","identifiantDocProducteur":"createur",
                 "compression":"compression","encodage":"encodage",
                 "formatFichier":"formatFichier","nomFichier":"cheminFichier",
                 "empreinteOri":"empreinteOri","noteFichier":"note","structureFichier
↪":"structureFichier",
                 "structureDocument":"structureDocument"
                },
    "fichMeta":["fichier3DGeometrie","fichier3DTexture","fichierArchive",
                "fichierDonneesVolumiques","fichierLasergrammetrie",
                "fichierParadonnees","fichierPhotogrammetrie"],
    "autoCompletion":["nombreFichiers","structureDocument"]
}
```

Utilisations du fichier de configuration par aLTAG3D :

- **version** : détermine la version du fichier XSD
- **indicators** : indique les noms des données utilisées comme “labels” sur certaines boîtes
- **mapToSIP** : permet de fixer en dur certains noms dans le code et de pouvoir simplement les modifier plus tard (utilisé lors de la création du SIP.xml)
- **fichMeta** : ensemble des boîtes utilisant des fichiers (utilisé lors de la création du SIP.xml)
- **autoCompletion** : ensemble des boîtes pouvant être auto-complétées (hors fichiers)

CHAPITRE 3

Création d'un template de rapport

aLTAG3D utilise le moteur de templates **Mustache** dont la documentation est disponible ici. N'importe quel template mustache utilisant comme mots-clés les noms utilisés dans le XSD est donc compatible avec notre système de rapport. Ce template pourra alors être écrit en LaTeX, HTML, MD, TXT, etc.

Utilisation de *Mustache*

Mustache fonctionne en remplaçant des mots-clés dans un template par des valeurs issues d'un jeu de données. Il est dit « Logic-less » car il ne dispose d'aucun mot-clé spécifique (if, else, for, etc.), il utilise seulement les mots-clés présents dans vos données pour fonctionner.

Exemple de fonctionnement :

Code source 3.1 – Données :

```
{
  "objets": [
    { "nom": "boîte", "format":"ply" },
    { "nom": "statue", "format":"ply" },
    { "nom": "cadre", "format":"dae" }
  ]
}
```

Code source 3.2 – Template :

```
<ul>
  {{#objets}}
  <li>{{nom}} ({{format}})</li>
  {{/objets}}
</ul>
```

Code source 3.3 – Résultat :

```
<ul>
  <li>boîte (ply)</li>
  <li>statue (ply)</li>
  <li>cadre (dae)</li>
</ul>
```

Ici le mot-clé **{{#objets}}** joue le rôle d'une condition **if** et d'une boucle **for**.

Pour plus d'informations concernant la syntaxe et le fonctionnement de *Mustache*, la documentation est disponible ici.

Mots-clés spécifiques à aLTAG3D

Dans aLTAG3D, les données (et donc les mots-clés) sont directement extraites de l'arbre de votre projet. La partie template est personnalisable lors de la création d'un rapport par un simple copier/coller dans la partie gauche (template) de la fenêtre.

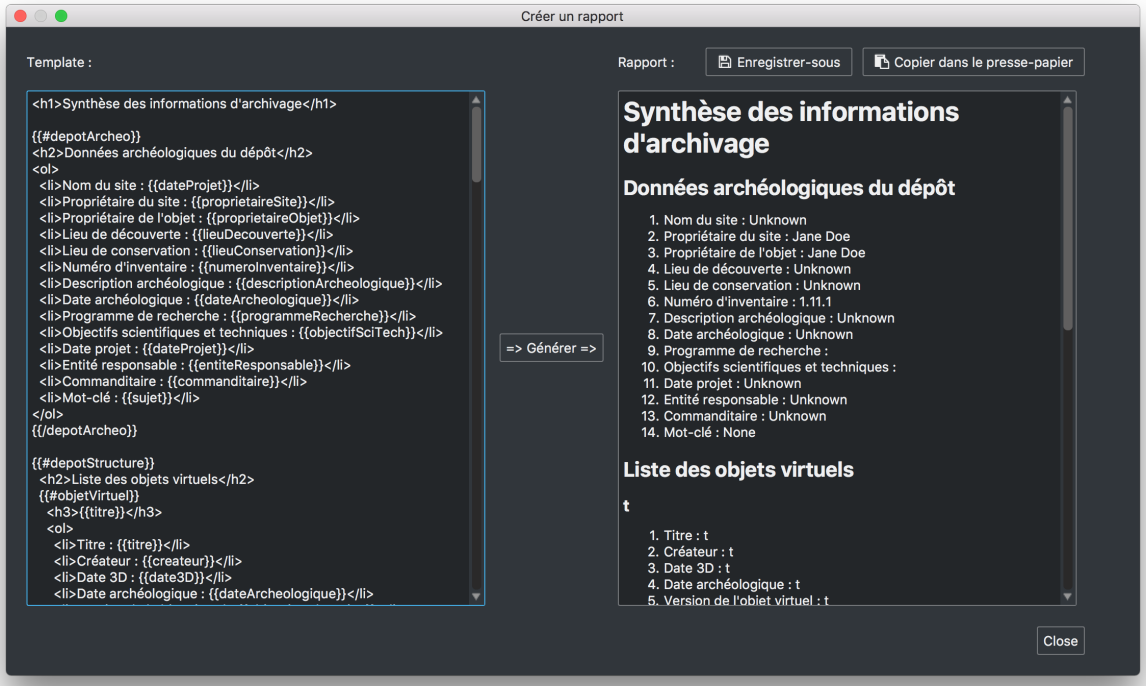

Fig. 3.1 – Exemple de création d'un rapport

L'ensemble des mots-clés et de leurs labels (i. e. : le texte qui apparaît dans aLTAG3D à la place du mot-clé) est disponible ci-dessous ou bien dans le fichier XSD.

Mot-clé	Label
depotArcheo	Données archéologiques du dépôt
dateProjet	Date projet

Suite sur la page suivante

Tableau 3.1 – Suite de la page précédente

Mot-clé	Label
descriptionArcheologique	Description archéologique
entiteResponsable	Entité responsable
proprietaireObjet	Propriétaire de l'objet
proprietaireSite	Propriétaire du site
siteNom	Nom du site
sujet	Mot-clé
commanditaire	Commanditaire
dateArcheologique	Date archéologique
lieuConservation	Lieu de conservation
lieuDecouverte	Lieu de découverte
numeroInventaire	Numéro d'inventaire
objectifsScientifiquesTechniques	Objectifs scientifiques et techniques
paradonnees	Fichier de paradonnées
programmeRecherche	Programme de recherche
depotStructure	Structure du dépôt
nombreFichiers	Nombre de fichiers
tailleProjet	Taille du projet
structureDocument	Structure du document
objetVirtuel	Objet virtuel
createur	Créateur
date3D	Date 3D
objetVirtuelVersion	Version de l'objet virtuel
titre	Titre
description	Description
note	Note
vignette	Vignette
maillage	Maillage
nombrePolygones	Nombre de polygones
nomMaillage	Nom du maillage
cheminFichier	Chemin du fichier
dateFichier	Date du fichier
formatFichier	Format du fichier
compression	Compression
empreinteOri	Empreinte ORI
encodage	Encodage
structureFichier	Structure du fichier
fichier3DGeometrie	Fichier 3D géométrie
axeOrientation	Axe orientation
axeVertical	Axe vertical
uniteMesure	Unité de mesure
logicielTraitement	Logiciel de traitement
Dimension_x	Dimension X
Dimension_y	Dimension Y
Dimension_z	Dimension Z
fichier3DTexture	Fichier 3D texture
typeTexture	Type de texture
groupeSource	Groupe de fichiers sources
fichierArchive	Fichier d'archive
editeur	Editeur

Suite sur la page suivante

Tableau 3.1 – Suite de la page précédente

Mot-clé	Label
droits	Droits
source	Source
contributeur	Contributeur
typeRessource	Type de ressource
fichierDonneesVolumiques	Fichier de données volumiques
marqueCapteur	Marque du capteur
modeleCapteur	Modèle du capteur
profondeurStockeeVoxel	Profondeur stockée du voxel
profondeurUtiliseeVoxel	Profondeur utilisée du voxel
resolution	Résolution
fichierLasergrammetrie	Fichier de lasergrammétrie
nombrePoints	Nombre de points
texture	Texture
systemCoordonneesXY	Système de coordonnées XY
systemCoordonneesZ	Système de coordonnées Z
fichierParadonnees	Fichier de paradonnées
langue	Langue
fichierPhotogrammetrie	Fichier de photogrammétrie
pointTopo	Point topo
exif	EXIF
geoTag	Geo tag

CHAPITRE 4

Création d'un plugin aLTAG3D

Afin de permettre à aLTAG3D de remplir de manière automatique le plus grand nombre d'informations possible, un système de plugin a été mis en place.

Lors de l'importation d'un fichier dans le projet d'archivage, aLTAG3D cherche dans la liste des plugins disponible celui étant compatible avec le format du fichier importé. Si il existe, ce plugin pourra communiquer au logiciel un ensemble d'informations concernant le fichier, qui viendront compléter les données du projet.

Le système de plugins mis en place est celui de Qt, dont la documentation est disponible ici.

L'interface plugin dans aLTAG3D

```
#ifndef INTERFACES_H
#define INTERFACES_H

#include <QtPlugin>

QT_BEGIN_NAMESPACE
class QString;
class QStringList;
class QJsonObject;
QT_END_NAMESPACE

class PluginInterface
{
public:
    virtual ~PluginInterface() {}
    virtual QString author() const = 0;
    virtual QString version() const = 0;
    virtual QString title() const = 0;
    virtual QStringList exportableData() const = 0;
    virtual QStringList formats() const = 0;
    virtual QJsonObject extractData(const QString &filename, const QStringList &
↪dataToExtract, QWidget *parent) = 0;
```

```
    virtual QString fileDescription() const = 0;
};

QT_BEGIN_NAMESPACE
#define PluginInterface_iid "fr.cnrs.archeovision.aLTAG3d.PluginInterface"

Q_DECLARE_INTERFACE(PluginInterface, PluginInterface_iid)
QT_END_NAMESPACE

#endif
```

Exemple de plugin simple

Code source 4.1 – Fichier .pro :

```
#-------------------------------------------------
#
# Exemple de fichier .pro pour un plugin altag3d
# inspiré de la documentation Qt
#
#-------------------------------------------------

TEMPLATE      = lib
CONFIG       += plugin
QT           += widgets
INCLUDEPATH  += inc
HEADERS       = exampleplugin.h \
                inc/interfaces.h
SOURCES       = exampleplugin.cpp
TARGET        = altag3d_example_plugin
DESTDIR       = ../plugins

target.path = ../../plugins
INSTALLS += target

CONFIG += install_ok  # Do not cargo-cult this!
uikit: CONFIG += debug_and_release
```

Code source 4.2 – exampleplugin.cpp :

```
#include "exampleplugin.h"

#include <QtWidgets>
#include <QMessageBox>

QString
ExamplePlugin::author() const
{
    //Auteur du plugin
    return QString("Archeovision");
}

QString
ExamplePlugin::version() const
{
```

```
16      //Version du plugin
17      return QString("v1.0");
18  }
19
20  QString
21  ExamplePlugin::title() const
22  {
23      //Nom du plugin
24      return QString("JPEG/JPG plugin example");
25  }
26
27  QStringList
28  ExamplePlugin::exportableData() const
29  {
30      //La liste des données renvoyées par le plugin
31      QStringList list;
32      //Données "Fichier"
33      list << tr("Chemin du fichier");
34      list << tr("Créateur");
35      list << tr("Date du fichier");
36      list << tr("Format du fichier");
37      list << tr("Empreinter ORI");
38      //Données spécifique au JPEG
39      //EXIF par exemple
40      return list;
41  }
42
43  QStringList
44  ExamplePlugin::formats() const
45  {
46      //Les formats que le plugin traite
47      QStringList list;
48      list << "jpg";
49      list << "jpeg";
50      return list;
51  }
52
53  QJsonObject
54  ExamplePlugin::extractData(const QString &filename, const QStringList &dataToExtract,␣
    ↪QWidget *parent)
55  {
56      QJsonObject object;
57      QFileInfo fi = QFileInfo(filename);
58      if (dataToExtract.contains("cheminFichier"))
59      {
60          object["cheminFichier"] = filename;
61      }
62      if (dataToExtract.contains("createur"))
63      {
64          object["createur"] = fi.owner();
65      }
66      if (dataToExtract.contains("formatFichier"))
67      {
68          //fi.completeSuffix().toUpper() en principe mais pas ici : "jpg" != "jpeg"
69          object["formatFichier"] = "JPEG";
70      }
71      if (dataToExtract.contains("dateFichier"))
72      {
```

```
        object["dateFichier"] = fi.created().toString();
    }
    if (dataToExtract.contains("empreinteOri"))
    {
        QString hash = QString("MD5@");
        QByteArray hashHex = fileChecksum(filename,QCryptographicHash::Md5);
        hash += QString::fromUtf8(hashHex);
        object["empreinteOri"] = hash;
    }
    //On retourne l'objet JSON
    return object;
}

QString
ExamplePlugin::fileDescription() const
{
    //"Un fichier <type> au format ..."
    return QString("image");
}
```

Code source 4.3 – exampleplugin.h :

```
#ifndef EXAMPLEPLUGIN_H
#define EXAMPLEPLUGIN_H

#include <interfaces.h>

#include <QObject>
#include <QtPlugin>
#include <QStringList>
#include <QFileInfo>
#include <QByteArray>
#include <QCryptographicHash>
#include <QString>
#include <QDateTime>

class ExamplePlugin : public QObject, public PluginInterface
{
    Q_OBJECT
    Q_PLUGIN_METADATA(IID "fr.cnrs.archeovision.aLTAG3d.PluginInterface" FILE
↪"pluginsfilters.json")
    Q_INTERFACES(PluginInterface)

public:
    QString author() const override;
    QString version() const override;
    QString title() const override;
    QStringList exportableData() const override;
    QStringList formats() const override;
    QJsonObject extractData(const QString &filename, const QStringList &dataToExtract,
↪ QWidget *parent) override;
    QString fileDescription() const override;

private:
    QByteArray fileChecksum(const QString &filename,
                            QCryptographicHash::Algorithm hashAlgorithm)
    {
        //Permet de générer une empreinte pour un fichier (Attention aux gros␣
↪fichiers)
```

```
        QFile f(filename);
        if (f.open(QFile::ReadOnly)) {
            QCryptographicHash hash(hashAlgorithm);
            if (hash.addData(&f)) {
                return hash.result().toHex();
            }
        }
        return QByteArray();
    }
};

#endif // EXAMPLEPLUGIN_H
```

Code source 4.4 – pluginsfilters.json :

```
{}
```

Le fichier *pluginsfilters.json* est presque vide, mais nécessaire à la compilation.

CHAPITRE 5

Rechercher

— search