Présentation du tool QtClient du socle

AUTEUR	Nicolas BELLAICHE | SIDT / IDI
MOTS-CLES	GPAO, socle
RESUME	Ce document décrit le fonctionnement du tool QtClient générique et comment l'utiliser pour générer une application avec IHM construite sur le module GPAO du socle

SOMMAIRE

Besoin

Le module GPAO du socle permet de développer rapidement des clients de GPAO permettant de générer et suivre une base de GPAO. Plusieurs applications ont été développées sur cette base :

OrientExpress, MoSAR, MicMacMgr, DeDAL...

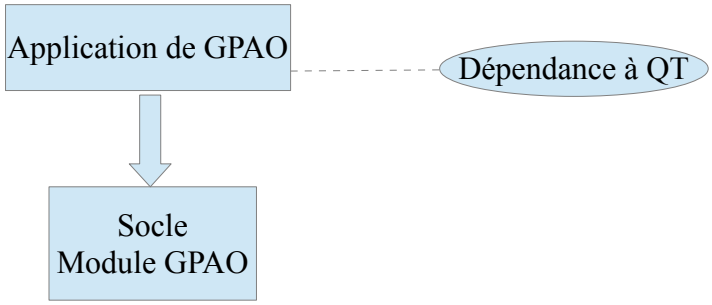
Toutefois, à l'usage, il est apparu qu'une partie du code pouvait être assez facilement factorisée entre toutes ces applications, de manière à en simplifier l'usage et à faciliter la maintenance du code :

- développement de la fenêtre d'interface permettant de remplir les paramètres de création de la GPAO
- Gestion de l'import d'un dictionnaire de paramètres pour remplir automatiquement l'IHM
- Gestion des boutons de création et suivi de progression d'une nouvelle base.
- Classes spécifiques à la manipulation des produits d'une GPAO lors de la création de chantier.
- Génération des scripts d'installation sous MacOSX et Windows.

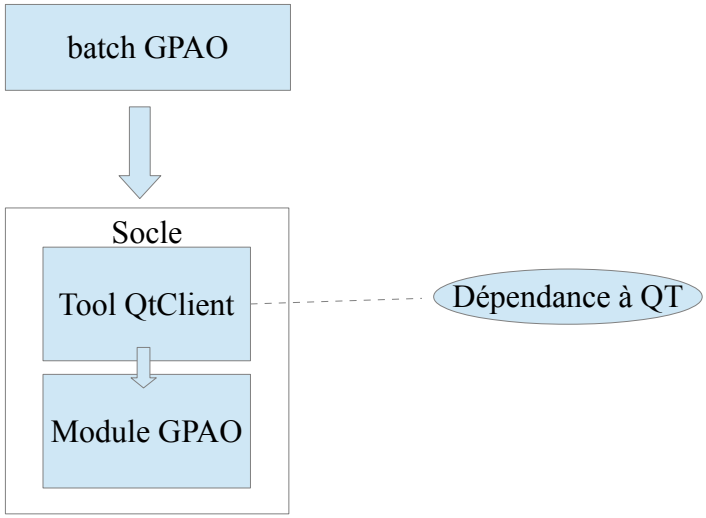
Les développements permettant de répondre à ces besoins ont été réalisés au fil de l'eau durant les développements de MosAR, DeDAL et lors de la refonte de MicMacMgr (V2) tout au long de l'année 2016. Ce document vise à décrire les résultats obtenus et la manière d'utiliser ces nouvelles fonctionnalités du socle.

Outil QtClient

Les GPAO « à l'ancienne » sont développées sur le modèle suivant :



Avec QtClient, l'architecture devient :



Le développeur qui utilise ces mécanismes n'a donc plus besoin de se linke à QT et de manipuler des concepts complexes (objets d'IHM, signaux, slots...). Il doit simplement développer un batch qui respecte certaines conventions (nom des paramètres et résultats attendus)

Batch GPAO

Le batch GPAO est un outil en ligne de commande qui reste à la charge du développeur de l'application de GPAO. Il communique avec l'IHM de QtClient par l'intermédiaire de fichiers texte qui sont écrits dans un dossier temporaire entré en paramètre de QtClient au lancement de l'application.

Le batch reste donc à la charge du développeur utilisateur, mais celui-ci n'a plus besoin de gérer lui-même la partie 'interface'.

Pour être fonctionnel, ce batch doit pouvoir répondre aux trois commandes suivantes :

Commande « info »

Permet de donner le nom et la version de l'application de GPAO :

BatchGpao --info --output info.json

Le résultat est un dictionnaire exporté au format json :

{"info":{"name":"MicMacMgr","version":"2.0.0.6352 | socle 8540"}}

Commande «ihm»

Permet de décrire l'IHM de la page de création des traitements sous la forme d'un dictionnaire exporté au format json

BatchGpao --ihm --output ihm.json

Exemple de résultat :

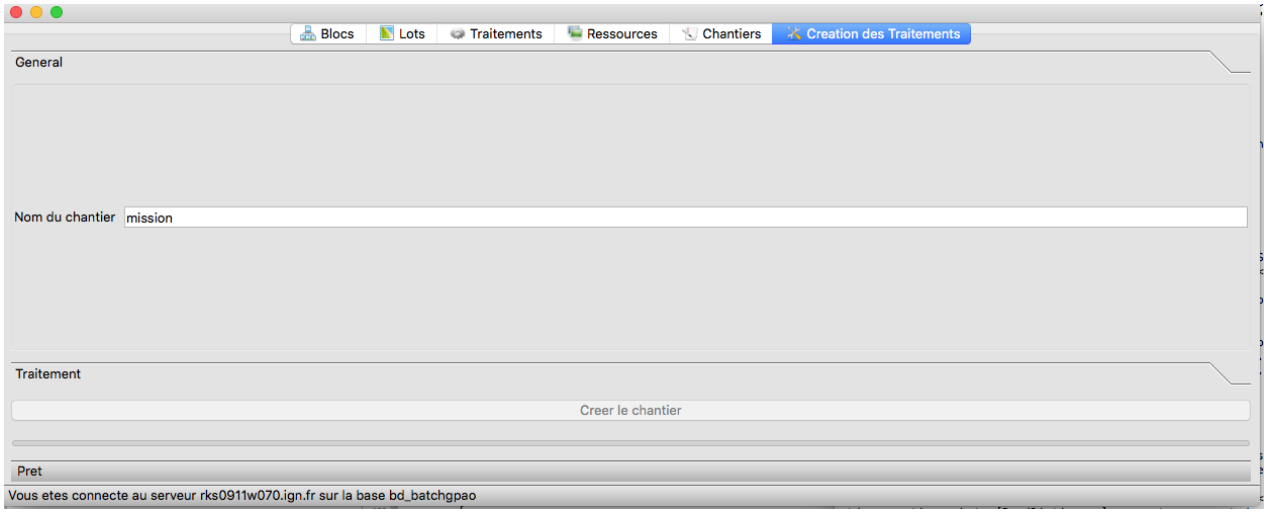
{"0":{"Name":"General","Type":"Page","2":
{"Name":"","Type":"Group","GroupType":"VerticalGroup","3":{"Name":"Nom du chantier","Key":"kMissionName","Value":"mission","Type":"LineEdit"}}},"1":
{"Name":"Traitement","Type":"Page"}}

Ce dictionnaire peut être généré grâce à la classe IhmBuilder du module GPAO. Cette classe ne dépendant pas de QT, il est possible d'utiliser cette classe sans que le batch résultant ait besoin d'être linké aux bibliothèques QT

Exemple de code permettant de générer le dictionnaire ci-dessus :

```
ign::data::Object ihm;

ign::gpao::IHMBuilder ihmBuilder;

ign::data::Object& globalPage = ihmBuilder.addPage("General", ihm);

ign::data::Object& globalGroup
      = ihmBuilder.addGroup("", ign::gpao::IHMBuilder::GroupTypeVertical, globalPage);

ihmBuilder.addLineEdit("Nom du chantier","kMissionName", "mission", globalGroup);

ihmBuilder.addPage("Traitement", ihm);

std::ofstream fic("ihm.json");

if (fic.good()) fic<<ihm.toString();
```

Copie d'écran de l'interface générée dans QtClient :



La classe ign::gpao ::IhmBuilder permet de décrire la plupart des objets courants d'une IHM, mais les styles sont relativement limités en l'état de la librairie. Des enrichissements sont envisageables en fonction des besoins rencontrés.

Commande «make»

Permet de générer une description json de la gpao à partir d'un fichier de paramètres au format json

BatchGpao --make --output --input parameters.json gpao.json --error error.txt --log log.txt --progress progress.txt

Le fichier parameters.json est généré automatiquement par QtClient et correspond à un dictionnaire (clef, valeur) où chaque clef correspond à la clef de l'objet d'interface définie dans le dictionnaire d'IHM et la valeur correspond à la valeur rentrée par l'utilisateur dans l'interface. Dans l'exemple ci-dessus, si l'utilisateur ne modifie pas la valeur par défaut, on aura donc une entrée (« kMissionName », « chantier ») dans parameters.json.

Le fichier gpao.json est une description json de la structure de gpao interne au socle. Elle se génère de la manière suivante :

```
ign::gpao::Constructeur gpaoConstructeur(« NomBaseGpao »);

//on remplit le constructeur de gpao à l'aide du dictionnaire de paramètres (c'est le gros du travail, et dépend de chaque processus) :

GpaoBuilder builder(dicoParameters); // → initialisation d'une classe de construction de la base a partir des paramètres utilisateur

builder.run(gpaoConstructeur); // → remplissage de la structure de la base GPAO

gpaoConstructeur.exportAsDictionary(chantier);  // → export du resultat au format json
```

Le fichier error.txt contient les éventuelles erreurs générées. Il est relu par QtClient si le batch renvoie une autre valeur que 0 et envoyé dans une fenêtre d'alerte.

Le fichier log.txt sert à tracer le déroulement du traitement.

Le fichier progress.txt sert à renseigner le statut et la barre de progression du client QT. Il est relu toutes les 5 secondes dans un thread à part et les dernières lignes « STATUS » et « PROGRESS » sont interprétées par le client pour mettre à jour l'IHM.

Exemple de fichier Progress :

```
STATUS Transfert des donnees (MNT, TA...) sur le serveur
STATUS Lecture du TA
STATUS Calcul des emprise au sol des images
STATUS Calcul du bloc de correlation image
PROGRESS 16
PROGRESS 33
PROGRESS 50
PROGRESS 66
PROGRESS 83
PROGRESS 100
STATUS Calcul du bloc de fusion
PROGRESS 3
PROGRESS 7
```

Ces trois derniers fichiers sont optionnels. S'ils ne sont pas générés par le batch, cela ne perturbera pas le traitement. En revanche, il est nécessaire que les options –error, --log et –progress ne génèrent pas d'erreur au lancement du batch.

Classes utilitaires pour aider à remplir la base

Des classes de manipulation de produits (données résultant de la GPAO n'existant pas au moment de la création de chantier) ont été développées dans MosAR et MicMacMgrV2. Elles n'ont pas été remontées dans le module gpao du socle pour différentes raisons, mais il est recommandé de s'en inspirer largement pour faciliter le développement d'une nouvelle GPAO.

La remontée de ces classes dans le socle via un léger refactoring peut être estimé à une semaine de charge + 2 jours par application.

Déploiement de l'application

Sous MacOSX

Un script générique de déploiement, nommé « QtClientGpao_MacOSXInstaller.sh » est généré dans tools/gpao/QtClientGpao/setup. Il permet de construire un Bundle valide et de le compresser en image disque DMG.

Il suffit de le compléter avec un script « chapeau » définissant les variables qui correspondent aux éléments dépendant de chaque GPAO :

Variables du script à définir

appname : Nom du Bundle

version : Version de l'application

clientname : Nom du client QT

iconname : Path de l'icône de l'application (fichier icns)

param : Nom du fichier json permettant de définir les variables d'initialisation du client QT (batch de génération de la gpao, dossier temporaire où écrire les fichiers d'échange...)

batchgpao : Path du batch de génération de la GPAO

batchlist : liste des batches permettant d'exécuter les lignes de commande décrites par la GPAO

docname : la liste des fichiers documentant la GPAO

Exemple de script de configuration :

```
export appname=${TOOLNAME}

export version=${${PROJECT_NAME}_VERSION}.${${PROJECT_NAME}_WC_REVISION}

export clientname=QtClient

export iconname=../data/macosx/${TOOLNAME}.icns

export param=../data/macosx/local_param.json

export batchgpao=${IgnSocle_DIR}/bin/tool-IgnSocle_gpao_BatchGpao

export batchlist="${IgnSocle_DIR}/bin/tool-IgnSocle_filesystem_FileTransfert ${IgnSocle_DIR}/bin/tool-IgnSocle_filesystem_DirDelete"

export docname=""

sh ./QtClientGpao_MacOSXInstaller.sh $1
```

Utilisation des scripts

Il suffit de se placer dans le dossier setup où se trouve le script « chapeau » à la charge du développeur et de le lancer avec comme unqiue argument l'endroit où le DMG doit être créé.

Exemple:

```
mac1012w026:setup nbellaiche$ sh ./MicMacMgr_MacOSXScript.sh /Temp/
```

Résultat :

```
Version = 2.0.0.6352
command extras: cp -Rn /Volumes/DONNEES/Developpements/DVP_IGN/MERCURIAL/culture3d/include /Temp/MiMacMgr.app/Contents/MacOS/ && cp ../data/MM-Malt.xml /Temp/MicMacMgr.app/Contents/MacOS/include/XML_MicMac && cp ../config/command.sh /Temp/MicMacMgr.app/Contents/MacOS/
creation d'un package MicMacMgr dans /Temp/
CXX: /usr/bin/clang++
-- creation des sous-dossiers du bundle
-- copie de l'icone de l'app
-- copie du dictionnaire du bundle
-- copie des executables de l'application
cp: ../data/macosx/local_param.xml: No such file or directory
copie de /Volumes/DONNEES/Developpements/DVP_IGN/SVN/sd-socle/trunk/bin/tool-IgnSocle_orientation_GeoHeader dans le bundle
copie de /Volumes/DONNEES/Developpements/DVP_IGN/SVN/sd-socle/trunk/bin/tool-IgnSocle_filesystem_FileTransfert dans le bundle
copie de /Volumes/DONNEES/Developpements/DVP_IGN/SVN/sd-socle/trunk/bin/tool-IgnSocle_filesystem_DirDelete dans le bundle
copie de /Volumes/DONNEES/Developpements/DVP_IGN/SVN/sd-socle/trunk/bin/tool-IgnSocle_photogrammetry_OriConvertor dans le bundle
copie de ../bin/ControlDSM dans le bundle
copie de ../bin/ImportDSM dans le bundle
copie de ../bin/MakeParam dans le bundle
copie de ../bin/PrepareMALT dans le bundle
copie de ../bin/OriConvertorMicMac dans le bundle
copie de /Volumes/DONNEES/Developpements/DVP_IGN/MERCURIAL/culture3d/bin/mm3d dans le bundle
copie de /Volumes/DONNEES/Developpements/autres/make/bin/make dans le bundle
on lance les commandes custom
-- copie de la documentation
-- copie des librairies dynamiques (plugins Qt, Pq, odbc)
-- copie des frameworks QT
-- copie du plugin cocoa
-- suppression des versions debug des frameworks Qt et chmod +rw des versions release
-- recherche du prefixe QT dans les frameworks
-- passage des frameworks QT en chemin relatif (MicMacMgr)= QtCore QtGui QtPrintSupport QtWidgets QtSql QtNetwork QtPrintSupport
install (QtCore, QtCore)
install (QtCore, QtGui)
(...)
-- copie des ressources du socle IGN
-- creation du lien symbolique vers /Applications
-- creation de l'installer MicMacMgr dans /Temp/
created: /Temp//MicMacMgr_installer.dmg
fin de creation de MicMacMgr dans /Temp/
-- DMG cree: /Temp//setup-MicMacMgr-2.0.0.6352-macosx-qt5.dmg
```

Sous Windows

Un script permettant de faire un setup Windows sous « Inno Setup Compiler » est généré dans

tools/gpao/QtClientGpao/setup. Il faut le compléter avec deux fichiers à placer dans le même dossier « setup » :

Un fichier « CustomBatches.txt » comportant la liste des batches permettant d'exécuter les lignes de commande décrites par la GPAO sous la forme suivante:

```
Source: "${IgnSocle_DIR}/bin/tool-IgnSocle_filesystem_FileTransfert.exe"; DestDir: "{app}"; Flags: ignoreversion
```

Source: "${IgnSocle_DIR}/bin/tool-IgnSocle_filesystem_DirDelete.exe"; DestDir: "{app}"; Flags: ignoreversion

Un fichier « AppId » comportant un identifiant unique (que l'on peut générer sous Inno Setup), permettant d'identifier l'application dans la base de registre.

AppId={{7DFC207B-BCC2-485B-8F57-59A96F8400AG}

Commencer à développer sur ce modèle

Un tool « Template » a été placé dans le socle sur le modèle décrit : BatchGpao.

Les applications MicMacMgr V2 et DeDAL sont également développés sur ce modèle.