

SUPPORT DE LA NORME OFX POUR LOGICIELS LIBRES

PROJET DE FIN D'ÉTUDES - INF4199 et INF4196

DÉPARTEMENT DE GÉNIE INFORMATIQUE

Benoit Grégoire

1018168

Été 2002

ÉCOLE POLYTECHNIQUE

DE MONTRÉAL

DÉPARTEMENT DE GÉNIE INFORMATIQUE

SUPPORT DE LA NORME OFX POUR LOGICIELS LIBRES

Rapport de projet de fin d'études soumis
comme condition partielle à l'obtention du
diplôme de baccalauréat en ingénierie.

Présenté par: Benoit Grégoire

Matricule: 1018168

Directeur de projet: Michel Dagenais

Date: 18/8/2002

1) Sommaire

La norme OFX¹ permet la communication de données comptables entre une institution financière et un client. Elle est populaire pour télécharger des relevés de compte afin d'effectuer une conciliation bancaire, mais aucun logiciel libre supportant cette norme n'est disponible. Ce projet consiste donc à ajouter les fonctionnalités nécessaires à un logiciel de finances personnelles existant. Le logiciel choisi est GnuCash², le logiciel de finances personnelles le plus en vue du monde du logiciel libre. Il est distribué sous licence GPL³.

Étant donné le processus ouvert employé, il a été nécessaire d'utiliser, en plus des méthodes habituelles de conception et de programmation propres au génie informatique, différents moyens de communication et méthodes propres aux projets ouverts. Ce projet a donc été fait en collaboration avec d'autres développeurs que je n'ai jamais rencontrés et situés sur trois continents. Peu après le début du projet, il est devenu évident qu'il valait mieux implanter les fonctionnalités OFX comme telles dans une librairie partagée, afin de maximiser sa valeur pour la communauté et faciliter le déverminage. Ce projet a donc donné naissance à un logiciel indépendant: LibOFX.

Ce projet a été couronné de succès. Les éléments qui ont été produits sont entre autres: la librairie LibOFX (conçue pour permettre à n'importe quelle application de supporter la norme OFX), l'utilitaire ofxdump (permet le déverminage de la librairie), l'utilitaire ofx2qif (un convertisseur de fichiers OFX=>QIF⁴), le site <http://step.polymtl.ca/~bock/libofx/>, un manuel du programmeur, gnc-ofx-import (un module d'importation OFX pour le logiciel GnuCash), et gnc-generic-import (une architecture générique d'importation pour GnuCash).

Ceux-ci permettent pour la première fois au logiciel GnuCash et bientôt à plusieurs autres logiciels d'importer avec succès des relevés bancaires utilisant la norme OFX.

¹ Open Financial Exchange: <http://www.ofx.net/ofx/default.asp>

² [Http://www.gnucash.org/](http://www.gnucash.org/)

³ GNU General Public License: <http://www.gnu.org/licenses/licenses.html#TOCGPL>

⁴ Quicken Interchange Format

2) Table des matières

1) Sommaire.....	I
2) Table des matières.....	II
3) Remerciements.....	IV
4) Liste des figures.....	V
5) Liste des tableaux.....	V
6) Introduction et problématique.....	6
6.1) La problématique.....	6
6.2) Le projet.....	6
6.3) Quelques mots sur la norme OFX:.....	7
6.4) Structure du document.....	8
7) Les méthodes et contraintes propres au processus de développement ouvert.....	9
7.1) Communication avec les autres développeurs et les usagers.....	10
7.2) Publication auprès des développeurs et des usagers.....	11
7.3) La documentation.....	11
7.4) La compatibilité aux environnements diversifiés.....	13
8) Les composantes du projet.....	14
8.1) Le parseur SGML.....	15
8.2) Le logiciel cible: GnuCash.....	17
8.3) La librairie partagée LibOFX.....	20
8.3.1) L'API.....	20
8.3.2) Le fonctionnement général.....	21
8.4) L'utilitaire ofxdump.....	23
8.5) L'utilitaire ofx2qif.....	25
8.6) Le module gnuccash gnc-generic-import.....	26
8.6.1) La problématique des modules d'importation.....	26
8.6.2) La solution.....	27
8.7) Le module gnuccash gnc-ofx.....	29
9) Discussion et conclusion.....	31
9.1) Les résultats du projet.....	31

9.2) Statistiques et performance:.....	32
9.2.1) Taille du code.....	32
9.2.2) Performance.....	33
9.3) Retour sur la planification.....	35
9.4) Travaux à réaliser dans les prochaines versions.....	36
9.5) Conclusion.....	37
10) Bibliographie.....	40
11) Annexes.....	41
11.1) Annexe 1: Résumé des fichiers et principales fonctions du projet LibOFX.....	41
11.2) Annexe 2: Document de design présenté aux développeurs de GnuCash.....	44
11.3) Annexe 3: Document de présentation et de planification original.....	47
11.4) Annexe 4: Le fichier lbofx.h définissant l'API.....	52
11.5) Annexe 5: Manuel du programmeur.....	59

3) Remerciements

Michel Dagenais, directeur de projet de fin d'études pour ce projet.

Christian Stimmings, développeur du module HBCI de GnuCash, pour son aide et ses commentaires.

Christian Stimmings, Derek Atkins et Linas Vepstas pour la confiance qu'ils m'ont accordée.

La communauté de développeurs de GnuCash, pour leur accueil chaleureux.

John Klar, Scot Drenan, Neil B. Cohen, Jake Gibbons, qui ont testé et/ou corrigé des défauts de la librairie.

4) Liste des figures

Figure 8.1: Composantes du projet	14
Figure 8.2: Structures de données de GnuCash	19
Figure 8.3: Hiérarchie des conteneurs définis dans ofx_proc_rs.cpp	22
Figure 8.4: Dialogue de sélection de compte	28
Figure 8.5: Exemple de transactions OFX téléchargées	30

5) Liste des tableaux

Tableau 1: Taille du code source	32
Tableau 2: Temps d'exécution de ofxdump pour différents fichiers	33

6) Introduction et problématique

6.1) *La problématique*

La norme OFX⁵ est un standard permettant la communication de données comptables entre une institution financière et un client. Elle est particulièrement populaire pour télécharger des relevés de compte afin d'effectuer une conciliation bancaire. C'est cette norme qu'utilise généralement les logiciels Quicken (Intuit), Money (Microsoft) et Personnel (Makisoft) pour effectuer ces opérations. Malheureusement, aucun logiciel libre supportant cette norme n'était disponible. Le seul logiciel qui possédait cette fonctionnalité et tournant sur Linux ou les différents BSD est un logiciel propriétaire: Kapital (theKompany.com).

Or, le manque d'un logiciel personnel perçu comme complet par le grand public est considéré comme un obstacle à l'adoption de Linux pour les ordinateurs des particuliers. L'une des fonctionnalités souvent demandée par les usagers avant d'utiliser les logiciels disponibles actuellement est justement le support de la norme OFX. Ce projet visait donc à ajouter les fonctionnalités nécessaires à un logiciel de finances personnelles existant. Le logiciel choisi est GnuCash⁶, le logiciel de finances personnelles le plus en vue du monde du logiciel libre. Il est distribué sous licence GPL⁷.

6.2) *Le projet*

L'écriture d'un module de communication OFX général avait été tenté par les développeurs de GnuCash. Il était prévu que le module implante la norme OFX presque au complet (conciliation bancaire, paiement de factures, transferts bancaires, transactions boursières, etc.). Ce projet a malheureusement échoué il y a déjà deux ans. L'estimation du temps nécessaire par les développeurs était de 12 mois-personnes. Le projet ne comptant que sur des bénévoles, il n'avancait plus.

⁵ Open Financial Exchange: <http://www.ofx.net/ofx/default.asp>

⁶ <Http://www.gnucash.org/>

⁷ GNU General Public License: <http://www.gnu.org/licenses/licenses.html#TOCGPL>

Je me suis donc lancé dans un projet un peu moins ambitieux : une implantation partielle de la norme OFX, soit la partie nécessaire pour le traitement de ce qui est communément appelé un « fichier OFX » soit un fichier de conciliation bancaire en format SGML⁸ suivant la norme OFX. Cela permet d'implanter la fonction la plus courante et la plus désirée de la norme OFX, soit de concilier un compte de GnuCash avec les transactions fournies par une institution financière à partir d'un compte bancaire ou d'un compte de carte de crédit.

6.3) *Quelque mots sur la norme OFX:*

La norme OFX (Open Financial eXchange) est une norme élaborée par un consortium formé de CheckFree, Intuit et Microsoft en 1997. Elle vise à supporter toutes les opérations financières susceptibles d'être effectuées en ligne entre un consommateur et une banque ou une entreprise. Quelques exemples sont le téléchargement de relevés d'opérations, la présentation et le paiement de factures ainsi que la gestion de comptes d'actions. C'est une norme client-serveur, couvrant la totalité des interactions incluant le protocole de transfert et la sécurité. Cependant, il est rare qu'une banque permette d'envoyer directement des requêtes OFX. Généralement, une interface WWW permet de paramétrer les informations demandées, et la banque nous transfère un fichier contenant la réponse OFX.

Cette norme est une norme complexe (la spécification que j'ai utilisée fait 659 pages) qui existe en plusieurs versions. La version 1.02 (basée sur SGML) est la plus couramment implantée par les banques. La version 1.6 (également basée sur SGML) est la version la plus complète (et la plus complexe). Elle supporte toutes les fonctionnalités de la norme 1.02. Il semble cependant qu'elle n'a pas connu un grand succès, peu d'institutions financières l'ayant déployée. Sans doute dans le but de rendre OFX plus populaire, le consortium a publié la version 2.01 de la norme, une version simplifiée et au goût du jour: basée sur XML⁹ (un sous-ensemble de SGML). La norme 2.01 ne supporte donc pas les versions précédentes.

⁸ Structured Generalised Markup Language

⁹ eXtensible Markup Language

Fort heureusement, je n'ai à ce jour trouvé aucune incompatibilité importante empêchant les fichiers basés sur la norme 2.01 d'être traités par un logiciel basé sur la version 1.6. Tous les fichiers peuvent donc être supportés à partir d'une même base de code construite à partir de la norme 1.6, avec quelques ajustements mineurs. Il me faut impérativement un moyen de me générer des fichiers tests, mais je pourrai utiliser ceux de mon institution financière (le service Accès-D de Desjardins) qui exporte ses fichiers dans le format OFX 1.02. Mon module est donc testable et utile à la majorité des utilisateurs en le basant sur la norme OFX 1.6 (la plus complète, également la plus complexe: 659 pages).

6.4) Structure du document

L'évolution de ce projet s'est poursuivie au-delà de certains des objectifs que je m'étais fixés. Il a donné naissance à plusieurs éléments reliés mais distincts. Afin d'éviter un discours décousu, la suite de ce rapport suivra la structure des différents modules qui composent le projet final.

Je discuterai d'abord globalement de la méthodologie propre aux projets ouverts que j'ai employée.

Suivra ensuite une section pour chacun des éléments importants du projet. Chacune de ces sections contiendra une description plus ou moins brève de la problématique à résoudre, de la méthodologie employée et une description du résultat final (le logiciel ou le document produit).

Finalement, je discuterai plus globalement des résultats de ce projet, des problèmes encore à résoudre ainsi que de ses suites probables.

J'espère que cette légère déviation de la structure habituelle des rapports de projets de fin d'études permettra au lecteur une lecture plus facile.

7) Les méthodes et contraintes propres au processus de développement ouvert

Le processus de développement choisi pour ce projet est un processus ouvert. Le concept consiste à publier le logiciel accompagné de son code source. Une licence est accordée à toute personne désirant se le procurer. Dans ce cas-ci, c'est la licence GPL qui est utilisée. Celle-ci permet à toute personne de modifier le produit, mais toute distribution du logiciel modifié doit être accompagnée du code source des modifications. Le but est de créer une communauté d'intérêts autour du logiciel afin que les personnes et organismes qui la constitue partagent leurs ressources pour développer et déverminer le logiciel. Cette philosophie à laquelle je crois fermement permet, lorsqu'elle fonctionne, la création d'un logiciel plus stable et plus complet que tous ceux qui auraient pu être produits par chacun des membres de la communauté travaillant en isolation. L'idée est donc de ne pas réinventer continuellement la roue.

Évidemment, pour que ce portrait idyllique se réalise, il faut d'abord créer la communauté. D'innombrables projets tentent de démarrer avec cette philosophie, mais avortent parce que la « communauté » n'a jamais été plus grande que la ou les personnes ayant lancé le projet. Plusieurs facteurs aident à rallier une telle communauté.

Tout d'abord, il faut qu'il y ait un intérêt commun. Idéalement le logiciel doit répondre à un besoin déjà exprimé. Ensuite, il faut éviter l'effet de dilution. Le logiciel ne doit donc pas être le quinzième logiciel similaire. Le logiciel doit avoir quelque chose à offrir, idéalement on doit donc le présenter à la communauté dans un état à tout le moins partiellement fonctionnel, afin d'offrir une base sur laquelle d'autres pourront poursuivre le développement. Il faut faciliter au maximum l'entrée dans la communauté, par une installation facile, un processus de compilation structuré, une bonne documentation, etc.

Il est à noter qu'à l'origine, je désirais simplement me joindre à une communauté existante, celle du logiciel GnuCash. Mais en cours de route, je me suis rendu compte que ce que j'étais en train de créer pouvait être utile à plusieurs logiciels. J'ai donc décidé

de rendre toute la partie s'occupant de la norme OFX comme telle indépendante. Ainsi est né le projet LibOFX.

Aucune librairie partagée n'était prévue dans la planification d'origine. La décision de rendre ce projet indépendant et l'ambition d'en faire le standard de facto pour supporter la norme OFX a rendu beaucoup plus importante la charge de travail reliée à plusieurs des tâches suivantes, typiques des projets ouverts.

7.1) Communication avec les autres développeurs et les usagers

Mon projet doit être intégré à un logiciel existant. Il faut donc que je travaille avec les personnes présentement en charge, et que je me conforme à certaines de leurs normes et procédés. De plus, le modèle de développement ouvert utilisé par les logiciels libres implique que je dois demander des modifications, implanter de nouvelles fonctionnalités ou éliminer des erreurs dans des modules maintenus par d'autres développeurs et que d'autres ont à faire de même pour mes modules.

Le travail collaboratif entre plusieurs développeurs éloignés géographiquement et qui ne se sont jamais rencontrés est rendu possible par l'utilisation de plusieurs outils. Voici ceux que j'ai utilisés, mais il en existe bien sûr une multitude d'autres.

- Le courrier électronique: Il permet de discuter en privé et d'échanger des fichiers. Plus d'une centaine de messages ont été échangés au cours de ce projet.
- Les listes de diffusion: Elles permettent de discuter de façon publique du développement du logiciel. Ainsi, les autres développeurs et les utilisateurs qui le désirent peuvent être tenus au courant du travail de chacun, échanger leurs idées sur l'évolution future du projet et rapporter des déficiences.
- CVS: Un système de contrôle de versions. Contrairement aux logiciels classiques comme PVCS et RCS, il ne fonctionne pas sous un modèle « verrouiller le fichier => modifier => déverrouiller ». Il fonctionne plutôt sur un modèle « modifier =>

fusionner ». C'est ce logiciel qui permet en pratique de travailler à plusieurs sur une même base de code sans avoir de communication directe régulière.

- Bugzilla: Un outil de gestion des défauts.
- #gnucash: Un canal IRC¹⁰ sur GIMPnet. Il permet la communication non-structurée et en temps réel.

Au début de ce projet, j'ai contacté la communauté de développeurs de GnuCash, et j'y ai été bien reçu. LibOFX est encore jeune, mais une quinzaine de personnes ont déjà envoyé des rapports de défauts ou des suggestions. De plus, les développeurs des logiciels de finances personnelles Grisbi¹¹ et KmyMoney2 ont manifesté leur intérêt à utiliser la librairie.

7.2) Publication auprès des développeurs et des usagers

Depuis que j'ai décidé de créer un projet indépendant pour LibOFX, il est devenu essentiel de tenir les usagers au courant des progrès de son développement. De plus, dans le but de développer une communauté et que le projet attire vers lui la majorité des développeurs qui s'intéressent à la question, il faut en faire la publicité auprès des auteurs de logiciels et de leurs usagers.

Les moyens utilisés pour ce faire sont la création et la mise à jour d'un site web (<http://step.polymtl.ca/~bock/libofx/>), la création d'une archive ordonnée des différentes révisions (il y en a 12 de publiées à ce jour), la mise en ligne de la documentation, l'annonce des nouvelles versions sur le site très fréquenté <http://www.freshmeat.net> et finalement contacter directement les auteurs de différents logiciels afin que nous ne soyons pas plusieurs à travailler indépendamment à un même objectif.

7.3) La documentation

La documentation de mauvaise qualité est souvent l'une des principales raisons pour

¹⁰ Internet Relay Chat

¹¹ <http://www.grisbi.org/>

laquelle les projets ouverts échouent. Ces projets sont souvent démarrés par des bénévoles dans leur temps libre, et il arrive souvent que la documentation soit complètement ignorée, ou remise à plus tard. Deux types de documentation sont pourtant à mon avis essentiels:

Documentation pour l'utilisateur: Constituée des différents fichiers à « Lisez moi », des instructions de compilation, du manuel de l'utilisateur s'il y a lieu. Dans mon cas, elle est constituée d'un fichier « lisez-moi » (libofx/README) et d'un manuel d'installation (libofx/INSTALL) incluant une foire aux questions.

Documentation pour les développeurs: J'ai eu beaucoup à me plaindre du manque de documentation à jour pour les développeurs du logiciel GnuCash. J'essaie donc de ne pas répéter la même erreur. La documentation est présentement constituée de libofx/README, de notes d'implantation, d'une liste de bugs connus, du fichier libofx.h (qui documente très bien l'API), du code source ofxdump.cpp et ofx2qif.c qui servent respectivement d'exemples de code C++ et C. Mais surtout j'ai rédigé, à même le code source, une documentation complète et à jour de l'API, de la structure interne de la librairie et des utilitaires ofxdump et ofx2qif. Cette documentation est ensuite extraite par l'excellent logiciel Doxygen¹² (un outil vaguement semblable à JavaDoc). Celui-ci comprend la syntaxe d'un logiciel en C, C++, Java ou PHP et, si l'on suit certaines règles, génère une documentation facile à parcourir des fonctions, structures de données et fichiers du logiciel. Il peut la générer en plusieurs formats. Le résultat en format PostScript de ce travail est le manuel du programmeur qui est disponible à l'annexe 5.

Trop de programmeurs assument que lire les sources d'un logiciel est suffisant pour que leurs collègues intéressés au projet parviennent rapidement à les comprendre. C'est peut-être parfois vrai pour un développeur très motivé désireux d'ajouter des fonctionnalités majeures au projet, mais ce n'est généralement pas pour un programmeur simplement intéressé à régler un défaut mineur, à adapter le logiciel à son système ou à ajouter des fonctionnalités mineures.

¹² <http://www.stack.nl/~dimitri/doxygen/>

7.4) La compatibilité aux environnements diversifiés

Malgré tous les efforts des développeurs et la disponibilité de la suite d'outils standard GNU¹³, il reste souvent très difficile de produire un logiciel facile à compiler et à installer dans les environnements divers des usagers.

Les principal problèmes que j'ai rencontrés sont de subtiles différences au niveau des compilateurs gcc 3.0 et 2.96. Afin d'éviter ces problèmes, je compile maintenant ma librairie systématiquement une fois avec chaque compilateur.

Un autre problème d'installation important est en fait un défaut commun des distributions Linux. Celui-ci fait en sorte que les bibliothèques installées dans /usr/lib ont priorité sur celles installées dans usr/local/lib, même si ces dernières sont plus récentes. L'utilisateur devra donc parfois s'assurer d'installer les bibliothèques de soutien dans /usr/lib.

¹³ GNU's Not Unix, un organisme ombrelle pour un grand nombre de projets libres.

8) Les composantes du projet

Voici l'architecture finale du projet; La majorité de ces éléments sont décrits dans les sections qui suivent:

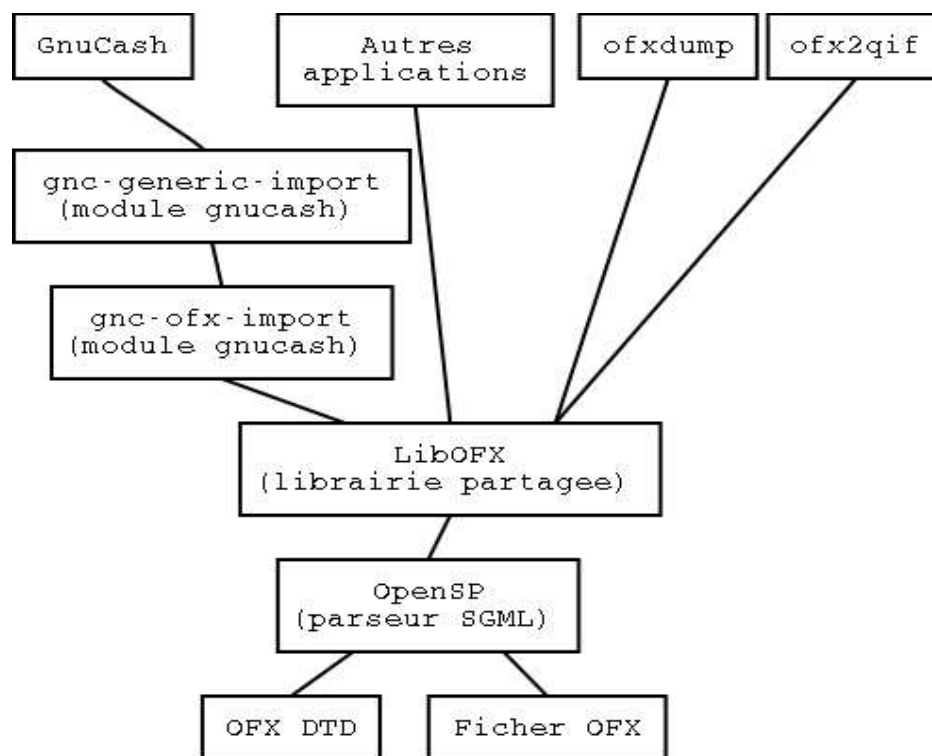


Figure 8.1: Composantes du projet

8.1) Le parseur SGML

Un document SGML est composé d'un DTD¹⁴ décrivant la structure du document, et du document lui-même. Ce dernier contient des éléments (on dirait des « tags » en HTML) représentant des entités logiques.

Un DTD pour la norme OFX 1.6 est fourni par le consortium. Celui-ci fait près de 5000 lignes. Il me fallait donc trouver ou programmer un parseur capable de parcourir ce monstre. Il aurait certes été envisageable d'écrire à partir de la spécification un parseur « manuel » recherchant linéairement dans le document les éléments supportés. Cependant, cette approche aurait donné naissance à un logiciel fragile, incomplet et difficile à maintenir et étendre. La première question architecturale à régler était donc de choix du parseur SGML capable d'interpréter le DTD et de servir de base à un parseur OFX. XML étant un sous-produit de SGML, la norme OFX 2.0 basée sur XML devrait pouvoir être supportée par le même parseur.

Il aurait été très facile de trouver un tel logiciel si la norme était basée uniquement sur XML. XML étant présentement la grande mode, il existe une multitude d'outils et de parseurs pour la traiter. Encore mieux, l'interface avec les parseurs XML est généralement régie par l'une de deux normes: SAX¹⁵ ou DOM¹⁶, ce qui permet d'écrire un logiciel indépendamment du parseur utilisé. L'interface SAX est une interface simple qui appelle des fonctions visiteurs à mesure que les différents éléments sont rencontrés, tandis que la norme DOM construit un arbre de données et offre les fonctions nécessaires pour le parcourir.

Malheureusement, dans le petit monde SGML, je n'ai réussi qu'à trouver un seul parseur qui remplisse les deux conditions que je jugeais essentielles, soit d'être activement maintenu et de supporter également les fichiers XML

Le parseur choisi est OpenSP¹⁷, descendant de sgmls, écrit par James Clark. Il est à noter

¹⁴ Document Type Definition

¹⁵ Simple API for XML

¹⁶ Document Object Model

¹⁷ <http://openjade.sourceforge.net/>

que ce dernier est une sommité du monde SGML/XML: il est présentement « technical lead » du comité définissant la norme XML. OpenSP inclut une interface C++ générique et documentée similaire à SAX.

La suite OpenSP n'est malheureusement pas sans défauts.

Tout d'abord, son interface a été conçue pour être incluse dans un logiciel au moment de la compilation. Or, la taille d'un caractère pour OpenSP peut être de 1, 2 ou 4 octets, selon les options de compilation choisies. Or, il n'y a aucune fonction permettant de connaître cette taille au moment de l'exécution, ce qui a occasionné d'importants maux de tête aux personnes qui ont testé la librairie au début du projet. Je suis finalement parvenu à écrire une fonction déterminant cette taille en inspectant les premiers octets d'une chaîne de caractère, et qui fonctionne pour les chaînes de toutes tailles.

Ensuite, une version relativement ancienne de OpenSP est communément distribuée avec la suite OpenJade, dans un seul paquetage RPM. Or, cette version (1.3.1) est incapable de traiter correctement le DTD OFX. Il faut donc que l'utilisateur installe une version plus récente, ce qui peut causer des problèmes étant donné que le paquetage RPM de OpenJade est insécable (un seul RPM, du moins sur Linux Mandrake). J'ai finalement réussi à assurer une certaine compatibilité avec cette version, mais ce n'est pas très fiable, et il n'est pas certain que cela soit maintenu dans le futur.

Finalement, au début de ce projet OpenSP avait un défaut d'une ligne dans les sources (corrigé depuis) qui empêchait la compilation sur les compilateurs récents (`gcc >= 2.96`).

8.2) Le logiciel cible: GnuCash

GnuCash est un logiciel extrêmement complet et beaucoup plus complexe que j'avais prévu au début du projet. Pour ne nommer que quelques unes de ses possibilités, il offre la comptabilité à double-entrée, les comptes d'actions et de fonds mutuels, l'importation de fichiers QIF, supporte une quinzaine de langues, le téléchargement sur le web de la valeur des actions et fonds mutuels, l'impression de chèques personnalisés, la possibilité d'enregistrer ses données dans un système SQL multi-usagers, et la génération d'une quantité énorme de rapports différents.

Au total, l'intégration à GnuCash allait prendre encore plus de temps que l'écriture du parseur OFX lui-même, alors qu'il devait être la partie la plus courte.

Tout d'abord, le logiciel était en plein changement d'architecture interne pour les modules, ce que je ne pouvais pas savoir et qui ne m'a guère facilité la tâche. La nouvelle architecture n'était pas encore vraiment documentée. Je ne disposais d'aucun exemple de code car les autres modules d'importation ne l'utilisaient pas. Heureusement, les développeurs étaient très disponibles pour répondre à mes différentes questions.

Un second problème était au niveau des langages de programmation utilisés. Je croyais que le logiciel était écrit en C. Le moteur l'est effectivement, mais la majorité du reste du logiciel (dont l'interface graphique et l'importation de fichiers) est écrit en Scheme (un langage semblable à Lisp) avec lequel je ne suis pas familier. Sans vouloir évaluer les mérites et défauts de ce langage par lequel jurent la majorité des développeurs, il a à tout le moins un défaut majeur pour quelqu'un qui est un nouveau développeur: il n'y a pas un équivalent des fichiers .h, ce qui fait en sorte que la majorité des fonctions ne sont pas documentées (pas même leur existence) autrement que par le code source. Considérant qu'il y a présentement 538 fichiers .c ou .scm dans les sources de GnuCash, on peut se douter que je me suis souvent fait dire « Oh, mais il y avait déjà une fonction qui fait exactement ça, mais il fallait le savoir...».

Il est nécessaire de prendre ici le temps de définir plusieurs termes utilisés pour décrire le modèle financier et les structures de données de GnuCash:

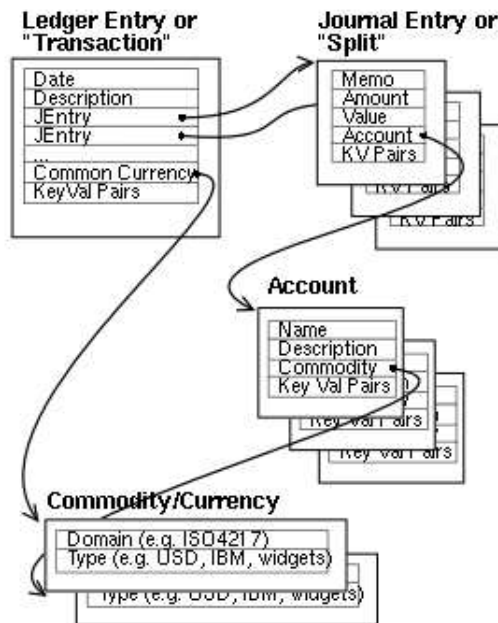
- Livre (Book): Contient des comptes et des transactions
- Compte (Account): Contient des entrée de journal, dont la somme doit être 0
- Entrée de livre (Transaction): Contient des entrées de journal, dont la somme doit également être 0
- Entrée de journal (Split): Mouvement d'argent d'un compte à un autre

Les quatre structures précédentes peuvent contenir des structures de données arbitraires nommées « KVP frames », qui permettent de stocker des données supplémentaires sans devoir modifier tout le code d'entrée-sortie et les prototypes de fonctions.

- Commodity (Commodity): Élément pouvant être fractionné de façon finie. Cela peut être de l'argent (la plus petite fraction est 1/100, et elle vaut 0,01\$) ou autre chose, comme une action. GnuCash ne mémorise jamais de valeurs numériques comme telle. Il mémorise le nombre de fractions, et sa valeur.

Accounting Data Structures

Support the Accounting Equation:
 $0 == \text{sum total of all splits in txn}$



12 octobre 2000 21:22:33

Figure 8.2: Structures de données de GnuCash

Le logiciel GnuCash est organisé en plusieurs sous-systèmes. Les sous systèmes avec lesquels j'ai eu a interagir sont:

- Le moteur
- L'interface de création et d'édition de comptes
- Les fonctions utilitaires pour les interfaces graphiques

8.3) La librairie partagée LibOFX

La librairie LibOFX est l'élément principal du projet. À l'origine, il n'était pas prévu de créer une librairie partagée. Cependant, en cours de développement il est rapidement devenu nécessaire d'avoir un exécutable indépendant afin de tester et ajouter des fonctionnalités au code faisant l'interprétation des données OFX proprement dites (ce rôle est maintenant rempli par l'utilitaire ofxdump). Comme il était nécessaire d'avoir une librairie, il était ensuite facile de la rendre partagée.

8.3.1) L'API

La facilité d'utilisation par le programmeur est le but de toute librairie partagée. Voici donc quelques critères que j'ai tenté de suivre:

- OFX est une norme très complexe, mais peu de logiciels ont besoin de toute cette complexité. Dans la mesure du possible, il faut éviter de forcer l'utilisation de structure de données hiérarchiques. Une seule structure devrait correspondre à un compte, une transaction, etc. Dans cette structure, toutes les données devraient être disponibles directement.
- Essayer d'utiliser les abstractions les plus larges possibles. Par exemple, il devrait être possible de supporter des comptes bancaires, de cartes de crédit ou d'actions avec la même structure.
- Tenter de faire toutes les conversions de données possibles à l'intérieur de la librairie: conversion entre deux monnaies, conversion des dates au fuseau horaire local, etc.
- Il est impossible de savoir exactement de quel format de données les logiciels utilisant la librairie auront besoin. Afin de leur simplifier la tâche, toutes les données sont dans une représentation directement utilisable dans le langage C (et C++). Une date est un `time_t`, un montant d'argent est un double float, une chaîne de caractères est un `char[]`, etc.

C'est en suivant ces principes que j'ai créé l'interface définie dans `libofx.h` (disponible à

l'annexe 4, voir également le manuel du programmeur à l'annexe 5). Cette interface définit plusieurs structures de données correspondant à une entité financière logique tel un état de compte, un compte bancaire ou une transaction. Ces structures sont envoyées au client à l'aide de fonctions visiteurs. Ceci a l'avantage que la majorité des clients n'auront pas besoin de mettre l'information dans un tampon ou de gérer le flux de contrôle.

8.3.2) Le fonctionnement général

Pour faciliter la compréhension de cette section, je vous recommande de vous référer au résumé des principales fonctions et fichiers, disponibles à l'annexe 1. Ce n'est cependant pas un substitut au manuel du programmeur complet, disponible à l'annexe 5.

En débutant, le nom fichier OFX à traiter est reçu en paramètre par la fonction `ofx_proc_file`. Les en-têtes OFX sont éliminées pour ne laisser que le contenu SGML. Les éléments OFX propriétaires sont ensuite éliminés. La norme OFX prévoit un format pour les extensions propriétaires à une compagnie. Celles-ci sont dans le format `<PRE.NOM>` où PRE est un préfixe de trois lettres enregistré par la compagnie auprès du consortium, et NOM est le nom de l'élément. Or, ceux-ci provoquent des problèmes de parsing, car ces éléments ne font pas partie du DTD, et sont de toute façon inutiles pour LibOFX puisqu'ils ne sont pas documentés. Le résultat de ces pré-traitements est ensuite écrit dans un fichier temporaire. Finalement, la librairie trouve le fichier DTD approprié, et envoie celui-ci et le fichier temporaire à la fonction `ofx_proc_sgml`.

Celle-ci passe les deux fichiers à la librairie OpenSP, qui prend en charge le flux de contrôle jusqu'à la fin du processus.

L'interface générique de cette librairie définit différents objets que l'on peut implanter pour recevoir les événements de la librairie. C'est ce que fait le fichier `ofx_sgml.cpp`. Les événements qu'il traite sont les suivants: ouverture d'un élément, fermeture d'un élément, réception de données et erreur. C'est à partir du nom et du type de chaque élément SGML reçu que LibOFX est en mesure de savoir quel objet créer, détruire ou remplir.

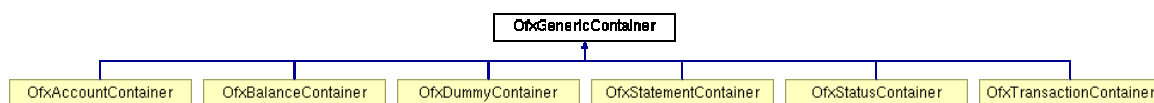


Figure 8.3: Hiérarchie des conteneurs définis dans *ofx_proc_rs.cpp*

Le fichier *ofx_proc_rs* définit plusieurs objets qui servent de conteneurs pour les différents éléments ofx supportés. Lorsque *ofx_proc_sgml* rencontre un élément auquel correspond un conteneur, il crée une nouvelle instance de l'objet approprié. Tous les objets implémentent la méthode `add_atribute`. Chaque objet doit savoir quoi faire avec les différents éléments de données qu'il est susceptible de recevoir. L'objet spécial `OfxDummyContainer` est utilisé pour les éléments inconnus. Un fois que l'élément SGML ayant mené à l'instanciation d'un objet est fermé, l'événement approprié défini dans *libofx.h* est généré avec la structure de données remplie par l'objet, puis l'objet est détruit.

8.4) L'utilitaire ofxdump

Ofxdump imprime sur la console tout ce que la librairie arrive à comprendre d'un fichier de réponse OFX. Il envoie également à la sortie de messages d'erreurs (stderr) les différents messages transmis par la librairie.

Ofxdump est à la fois le principal outil de déverminage, une démonstration des possibilités de la librairie et un exemple de code en C++. Il utilise toutes les fonctions et structures de données de l'API de LibOFX.

Utilisation: ofxdump chemin_fichier_ofx/fichier_ofx

Exemple de fonctionnement (tronqué):

```
[bock@bock fichiers_ofx]$ ofxdump selrelevc_2.cmd
LibOFX INFO: sanitize_proprietary_tags() removed: <INTU.BID>00012
LibOFX STATUS: find_dtd():DTD found: /usr/local/share/libofx/dtd/ofx160.dtd
LibOFX ERROR: OpenSP parser: otherError (misc parse error)
Error msg: /usr/local/share/libofx/dtd/ofx160.dtd:3058:5:E: content model is
ambiguous: when no tokens have been matched, both the 1st and 3rd occurrences of
"SIGNONMSGSET" are possible

LibOFX ERROR: OpenSP parser: otherError (misc parse error)
Error msg: /usr/local/share/libofx/dtd/ofx160.dtd:3058:5:E: content model is
ambiguous: when no tokens have been matched, both the 2nd and 4th occurrences of
"PROFMSGSET" are possible

LibOFX INFO: Created OfxDummyContainer to hold unsupported aggregate OFX
LibOFX INFO: Created OfxDummyContainer to hold unsupported aggregate
SIGNONMSGSRSV1
LibOFX INFO: Created OfxDummyContainer to hold unsupported aggregate SONRS
ofx_proc_status():
  Ofx entity this status is relevent to: SONRS
  Severity: INFO
  Code: 0, name: Success
  Description: The server successfully processed the request.
  Server Message: OK

LibOFX INFO: Created OfxDummyContainer to hold unsupported aggregate
BANKMSGSRSV1
LibOFX INFO: Created OfxDummyContainer to hold unsupported aggregate STMTTRNRS
ofx_proc_status():
  Ofx entity this status is relevent to: STMTTRNRS
  Severity: INFO
  Code: 0, name: Success
  Description: The server successfully processed the request.
  Server Message: OK

ofx_proc_account():
  Account ID: 70000010000234210023421815-30480-0023421-EOP
  Account type: CHECKING
  Currency: CAD

ofx_proc_transaction():
```

Account ID : 70000010000234210023421815-30480-0023421-EOP
Transaction type: DEBIT: Generic debit
Date posted: Thu Jan 24 08:00:00 2002 EST
Amount: -140.00
Financial institution's ID for this transaction: jhKbOAO3v
Name of payee or transaction description: Retrait au GA/H.E.C
Extra transaction information (memo): IAGA

8.5) L'utilitaire ofx2qif

Il est difficile de stimuler l'intérêt pour une nouvelle librairie. En effet, à moins d'être utilisée par un logiciel celle-ci ne fait strictement rien. Il est donc utile de distribuer avec la librairie un ou des logiciels qui sont directement utiles pour l'utilisateur. Ceux-ci peuvent également servir d'exemple de code.

Ofx2qif a donc été écrit dans ce but. Ofx2qif est un convertisseur de fichiers du format OFX vers le format QIF¹⁸. Il a été originalement écrit en C++ pour stimuler l'intérêt pour la librairie. Il a subséquemment été ré-écrit en C afin de servir d'exemple de code en C et assurer que la compatibilité C soit maintenue.

Le format QIF est malheureusement très limité, ce qui fait que la conversion est incomplète. Ce n'est donc pas une bonne façon de supporter la norme OFX. Cependant, il permet à des usagers utilisant un logiciel ne supportant pas la librairie mais supportant le format QIF de se dépanner.

Utilisation: `ofx2qif chemin_fichier_ofx/fichier_ofx > fichier_sortie.qif`

Exemple de fonctionnement (tronqué):

```
[bock@bock fichiers_ofx]$ ofx2qif selrelevc_2.cmd
!Account
N70000010000234210023421815-30480-0023421-EOP
TBank
DOFX online account
$806.69
^
!Type:Bank
D24/1/2002
T-140.00
PRetrait au GA/H.E.C
MIAGA
LGeneric debit
^
```

¹⁸ Quicken Interchange Format

8.6) Le module *gnucash gnc-generic-import*

8.6.1) La problématique des modules d'importation

Lorsque j'ai débuté ce projet, GnuCash ne comportait qu'un seul module d'importation opérationnel: le module *gnc-qif-import*. Cependant, plusieurs autres sont en développement. Tout d'abord, *qif-io-core*, un module plus sophistiqué censé remplacer éventuellement *qif-import*, mais dont le développement est présentement arrêté. Ensuite, il y a bien sûr *gnc-ofx* dont je parlerai dans la prochaine section, et *gnc-hbci*, développé par Christian Stimmings pour supporter la norme allemande HBCI¹⁹.

Or, plusieurs fonctions d'un module d'importation de données financières sont communes. Voici les tâches qu'un tel module devrait idéalement être en mesure d'accomplir:

La première est l'identification et la création des comptes GnuCash correspondant aux comptes importés. Dans le cas d'OFX, ceux-ci sont identifiés par leur numéro de transit, mais tout autre identificateur unique peut être utilisé. Si le numéro de transit du compte en question n'existe pas dans les autres comptes, le logiciel offre à l'utilisateur de choisir le compte à utiliser ou d'en créer un nouveau. Il ajoute le numéro de transit au compte choisi. Si deux comptes ont le même numéro de transit, un message doit être affiché.

La seconde et plus importante fonction commune des processus d'importation est la fusion des transactions importées avec les transactions existantes. J'espérais pouvoir récupérer des fonctions de GnuCash pour la réaliser, mais ce fut en bonne partie impossible.

Cette fusion consiste d'abord à prendre les transactions obtenues du module d'importation et trouver le compte correspondant. Une fois le compte identifié, pour chaque transaction, le module :

- cherche une transaction ayant le même identificateur, la même date et le même

¹⁹ Home Banking Computer Interface

montant. S'il en trouve une et une seule, il considère l'opération comme conciliée.

- s'il n'en a pas trouvé, il cherche une transaction ayant le même identificateur et le même montant dans les 4 jours précédant l'opération conciliée, ou une opération ayant le même montant et la même date. S'il en trouve une, il offre de faire correspondre.
- S'il n'en a pas trouvé ou si l'utilisateur a refusé la correspondance, il offre d'ajouter l'opération au compte.

La gestion des duplicatas est importante. Si le module trouve plus d'une opération correspondant à une opération téléchargée, le module informera l'utilisateur d'une erreur potentielle. Quicken a une faiblesse à ce niveau. Il fait correspondre toute opération téléchargée avec la plus récente opération ayant le même identificateur et le même montant, même si elle est plus vieille de plusieurs mois! Il ne faudrait pas reproduire ce problème. Limiter à un certain nombre de jours comme je l'ai fait plus haut règle une partie du problème, mais j'aimerais aussi régler le problème posé par deux retraits d'un même montant dans un guichet automatique faits la même journée. Mais je ne suis pas sûr qu'il y ait une solution plus élégante dans ce cas que d'offrir à l'utilisateur d'ajouter les deux opérations. De plus, il ne faut pas rejeter complètement l'approche de Quicken, car pour les cartes de crédit, il peut arriver de recevoir la transaction plusieurs semaines après un achat. (J'ai déjà payé un repas dans un aéroport d'Europe et la transaction est apparue plus d'un mois plus tard)

8.6.2) La solution

Afin de régler ce problème, j'ai décidé de soumettre une proposition de design pour une architecture générique sur la liste de diffusion des développeurs de GnuCash. Cette proposition a été bien reçue. Vous pouvez lire la troisième version de ce document à l'Annexe 2.

J'ai implanté une partie de cette proposition. Présentement, toute la partie sur la sélection de comptes est implantée. Lorsqu'un module d'importation appelle la fonction `gnc_import_select_account` avec en paramètre un identificateur unique, le module

générique cherche dans le livre GnuCash un compte comportant ce numéro d'identification dans son KVP `frame online_id`. Si un tel compte est trouvé, le compte GnuCash est retourné immédiatement. Dans le cas contraire, l'utilisateur se fait présenter le dialogue suivant:

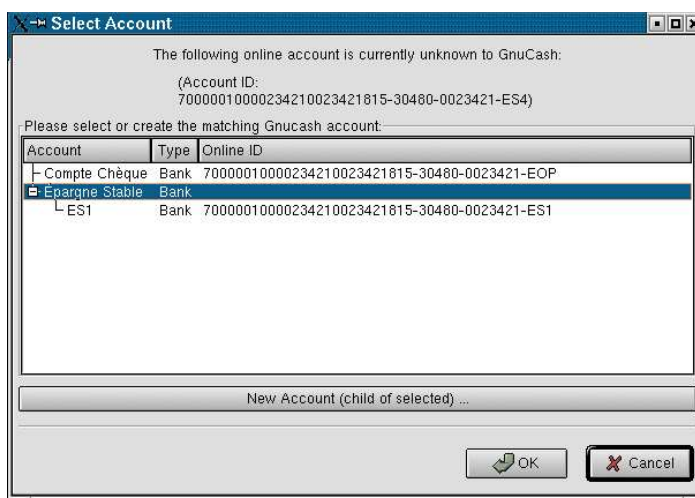


Figure 8.4: Dialogue de sélection de compte

Si aucun compte GnuCash ne correspond au compte à partir duquel on a téléchargé un relevé, l'utilisateur a l'option d'en créer un nouveau.

Malheureusement, la correspondance entre les transactions n'est pas encore totalement implantée. L'ajout des transactions est fonctionnel et le module détecte les transactions téléchargées deux fois et ne créera pas de duplicatas. Par contre, si une transaction a été entrée précédemment à la main, elle ne sera pas encore détectée comme correspondant à la transaction téléchargée.

Dans ce domaine, je n'ai donc pas atteint les objectifs originaux de mon projet. Il est possible de télécharger ses relevés de banque, mais il n'est pas encore possible de concilier des transactions existantes.

8.7) Le module *gnucash gnc-ofx*

Le module *gnc-ofx* permet de faire le pont entre la librairie LibOFX et les structures internes de GnuCash. Bien que son développement ait été assez complexe, maintenant que le module générique est fonctionnel, son rôle et son fonctionnement sont conceptuellement très simples.

Au démarrage, GnuCash tente de charger le module. Si le chargement réussit, le module ajoute une entrée « OFX/QFX import » dans la section import/export du menu « File » de GnuCash. Une bonne part des difficultés se sont situées à ce niveau, l'architecture de chargement des modules de GnuCash n'étant pas encore complétée.

Lorsqu'il reçoit un événement *ofx_proc_account* de LibOFX, il appelle la fonction *gnc_import_select_account* du module générique, afin de s'assurer qu'un compte est créé ou identifié pour recevoir les transactions à venir.

Lorsqu'il reçoit un événement *ofx_proc_transaction* de LibOFX, il crée une transaction *gnucash*, convertit les informations en une répartition *gnuCash*, associe la répartition au compte *gnucash* approprié et passe le tout au module générique à l'aide de la fonction *gnc_import_add_trans*.

Le module n'a aucune interface usager qui lui est propre, mais voici le résultat final:

Compte Chèque - Register

File Edit View Actions Reports Tools Help

Close Enter Cancel Delete Duplicate Schedule Split Blank Jump Transfer Find Report Print

Present: -CAD 1,961.98 Future: -CAD 1,961.98 Cleared: CAD 0.00 Reconciled: CAD 0.00

Date	Num	Description	Transfer	R	Deposit	Withdrawal	Balance
Action		Notes					
01/31/2002		Frais d'utilisation/ OFX ext. info: [Trans type:Generic debit Memo:ADM	n			3.30	-1,579.37
02/01/2002		Paielement facture - AccèsD Intern OFX ext. info: [Trans type:Generic debit Memo:PWV	n			106.43	-1,685.80
02/01/2002	Num	Paielement/MARITIME LIFE ASSURANCE	Transfer	n	Deposit	47.03	-1,732.83
	Action	OFX ext. info: [Trans type:Generic debit Memo:RA]					
02/01/2002		Retrait au GA/POLYTECHNIQUE/MTL OFX ext. info: [Trans type:Generic debit Memo:AGA	n			100.00	-1,832.83
02/04/2002		Dépôt au comptoir/CP STE-THERESE OFX ext. info: [Trans type:Generic credit Memo:IDSL	n		18.00		-1,814.83
02/06/2002		Retrait au GA/POLYTECHNIQUE/MTL OFX ext. info: [Trans type:Generic debit Memo:AGA	n			100.00	-1,914.83
02/08/2002	82	Chèque/ OFX ext. info: [Trans type:Check Memo:DCN	n			345.00	-2,259.83
02/11/2002		Achat/SUBWAY OFX ext. info: [Trans type:Generic debit Memo:ACH	n			8.61	-2,268.44
02/12/2002		Retrait au GA/POLYTECHNIQUE/MTL OFX ext. info: [Trans type:Generic debit Memo:AGA	n			100.00	-2,368.44
02/13/2002		Dépôt au GA/POLYTECHNIQUE/MTL OFX ext. info: [Trans type:Generic credit Memo:IDGA	n		190.25		-2,178.19
02/18/2002		Paielement facture - AccèsD Intern OFX ext. info: [Trans type:Generic debit Memo:PWV	n			783.79	-2,961.98
02/18/2002		Virement - AccèsD Internet/ OFX ext. info: [Trans type:Generic credit Memo:VWV	n		1,000.00		-1,961.98
08/12/2002			n				

OFX ext. info: [Trans type:Generic debit|Memo:RA]

Figure 8.5: Exemple de transactions OFX téléchargées

9) Discussion et conclusion

9.1) *Les résultats du projet*

Globalement, ce projet a permis de générer un grand nombre d'éléments utiles et fonctionnels. Il convient d'en faire ici la liste:

- **LibOFX:** La librairie devant permettre à n'importe quelle application utilisant la licence GPL de supporter les conciliations bancaires utilisant le protocole OFX. C'est à ma connaissance la première fois que la norme OFX (partie client) a été implémentée avec succès dans le monde du logiciel libre. 12 versions du projet publiées à ce jour.
- **ofxdump:** L'utilitaire C++ de déverminage et test de la librairie
- **ofx2qif:** Un convertisseur de fichier OFX=>QIF écrit en C permettant d'utiliser les logiciels ne supportant pas encore la librairie.
- **Le manuel du programmeur:** Écrit à même les sources afin d'être toujours maintenue à jour. Disponible en ligne en format HTML (plus pratique que la version PostScript à l'annexe 5).
- **<http://step.polymtl.ca/~bock/libofx/>:** Le site www du projet pour distribuer et faire connaître LibOFX
- **Un document de design pour une architecture d'importation générique pour le logiciel GnuCash:**
- **gnc-generic-import:** Le module d'importation générique de GnuCash.
- **Gnc-ofx-import:** Un client de LibOFX pour le logiciel GnuCash.
- **Bon nombre de Makefiles, fichiers « lisez-moi », et autres éléments appréciés mais souvent oubliés.**

9.2) Statistiques et performance:

9.2.1) Taille du code

Module	Lignes de code
LibOFX	1387
ofxdump	220
ofx2qif	171
gnc_generic_import	345
gnc_ofx	269
Total:	2392

Tableau 1: Taille du code source

- Les statistiques sur la taille du code sont générées par l'utilitaire sloccount²⁰. Ce logiciel saute les commentaires et les lignes vides.
- J'ai jugé qu'il n'était pas pertinent de compter les lignes des fichiers XML de description d'interface graphique générés par l'outil Glade.
- À titre de comparaison, l'arbre de développement complet de GnuCash représente 254 244 lignes de code.

²⁰ <http://www.dwheeler.com/sloccount/>

9.2.2) Performance

Afin de mesurer la performance de la librairie, j'ai produit différents fichiers de test par copier-coller. La commande `time` permet ensuite de calculer le temps d'exécution de `ofxdump` pour différents nombres de transactions.

Nb. de transactions	Temps usager (s)	Temps système (s)	Temps écoulé (s)
0	0,13	0,02	0,23
200	0,59	0,14	2,01
2090	4,66	0,94	16,08
20900	45,92	9,37	156,52

Tableau 2: Temps d'exécution de ofxdump pour différents fichiers

Le 0,13 seconde nécessaire pour traiter le fichier vide représente le coût fixe de traitement du DTD et d'initialisation de la librairie. Le test suivant avec 200 transactions représente à mon avis une borne supérieure représentative d'une utilisation normale, puisqu'il faudrait faire 100 transactions par mois et attendre 2 mois avant de les télécharger pour arriver à un tel fichier. Le temps réel écoulé de 2,01 secondes m'apparaît donc acceptable. On peut constater par les tests suivants que l'augmentation de temps est linéaire. Il aurait été fort surprenant qu'il n'en soit pas ainsi, puisque chaque transaction est détruite une fois traitée.

J'ai tenté de raffiner ces résultats par profilage. En recompilant `ofxdump` avec la librairie incluse de façon statique, il devient possible de faire un profilage avec l'option de compilation `-pg` et l'utilitaire `gproc`. À première vue, les résultats sont quelque peu surprenants.

Le temps d'UCT rapporté pour les fonctions profilées avec le fichier de test de 20900 transactions est de 2,83 secondes au total, soit 6,2% des 45,92 secondes de temps d'UCT usager donné par `time`. Cela laisse entendre que la majorité du temps est passé dans les fonctions non-profilées, soit donc celles d'OpenSP. Ayant de la difficulté à faire confiance à ces chiffres, j'ai utilisé `nsgmls` (l'outil de OpenSP) sur le DTD et le fichier de 20900 transactions, et selon `time`, seulement 2,05 secondes d'UCT usager sont utilisées au

total. Certes, les auteurs d'OpenSP mentionnent que l'interface générique de OpenSP occasionnait une « légère » perte de performance, mais il est difficile de croire que l'énorme manque à gagner de $45,92 - (2,05 + 2,83) = 41,04$ secondes est du seulement à l'interface.

Il apparaît donc clairement que la granularité du profilage (0,01 secondes entre les échantillons selon le rapport), soit nettement insuffisante pour prendre en compte avec une erreur acceptable tout le temps passé dans les différentes courtes fonctions de conversion de texte de la librairie.

Le profilage nous apprend tout de même que la fonction la plus coûteuse (33% du temps rapporté) est `sanitize_proprietary_tags`. Comme elle est appelée sur chaque ligne du fichier en entrée et ne sera qu'à éliminer quelques lignes, il est probable qu'une ronde d'optimisation sur cette fonction serait très profitable pour la performance générale.

En résumé, la performance est présentement adéquate pour une utilisation normale de la librairie. Il serait donc plus judicieux d'investir du temps pour ajouter les fonctionnalités mentionnées à la section 8.4, que de l'investir dans une optimisation précoce.

9.3) Retour sur la planification

Même si ce projet a beaucoup changé depuis sa planification originale, il reste utile de faire un retour sur le temps initialement alloué aux différentes tâches. Le document de planification original est disponible à l'annexe 3.

- ✓ Familiarisation avec le code source du logiciel et la norme OFX. Il s'agissait de prendre contact avec les développeurs de GnuCash et de discuter des modifications nécessaires au logiciel actuel afin de faciliter l'intégration. Je prévoyais 20 heures pour cette étape préliminaire. Ce temps a été respecté, mais les objectifs n'ont pas été rencontrés. Il est clairement apparu que ma familiarité avec le fonctionnement interne du logiciel était encore insuffisante à cette étape, et que les modifications à apporter au logiciel étaient plus grandes que prévues.
- ✓ Ajout du numéro de transit aux structures de données des comptes de GnuCash. Il s'agissait d'ajouter les structures de données nécessaires et l'interface programmeur pour accéder et modifier ces données. C'est le seul changement à GnuCash qui a été réalisé dans le temps prévu, soit ici sept heures.
- ✓ Parseur du fichier OFX. Ce devait être le travail le plus complexe et le plus important de ce projet. Si l'on exclut le développement d'un API structure et tout le travail nécessaire pour rendre la librairie indépendante et dans un format distribuable, les 125 heures prévues ont probablement été respectées avant l'atteinte d'un fonctionnement de base. Par contre, beaucoup plus de temps a en réalité été consacré à la librairie au cours de l'été.
- ✓ Fonction de fusion. Il s'agissait plus ou moins de développer les fonctionnalités aujourd'hui contenues dans gnc-generic-import et gnc-ofx. À peine 75 heures étaient prévues. Les problèmes d'intégration au logiciel GnuCash ont fait en sorte que ce temps a été dépassé simplement pour la création des modules, leur intégration dans les sources et l'environnement de compilation et les menus de GnuCash. Donc 75 heures ont été brûlées avant même l'écriture des fonctionnalités. Le temps total consacré approche probablement les 200 heures, et les fonctionnalités n'ont pas encore atteint le

niveau originellement désiré.

- ✓ Documentation : 10 heures devaient être consacrées à la documentation. Cependant, la publication indépendante de LibOFX et le design et la création de gnc-generic-import ont fortement augmenté les besoins en documentation.
- ✓ Tests et déverminage : Cette activité étant répartie sur toute la durée du projet, il est difficile de dire si les 25h prévues ont été respectées.
- ✓ Rapport : 20 heures étaient prévues. Étant donné l'augmentation de la taille du projet, ce temps va probablement doubler.

9.4) Travaux à réaliser dans les prochaines versions

Il est bien évident qu'un projet de cette nature ne se termine jamais. Voici quelques-unes des améliorations susceptibles d'être apportées dans un futur plus ou moins éloigné.

- LibOFX: Passer à un environnement de compilation GNU complet pour libofx et ses utilitaires en utilisant les logiciels automake, autoconf et libtool est nécessaire à court terme. Ce changement devrait permettre une plus grande portabilité du projet, un meilleur environnement pour accueillir les contributions des futurs développeurs et surtout permettre la génération facile de paquetages binaires.
- LibOFX: Implanter le support des transactions d'actions et des comptes de placements. C'est une fonctionnalité très demandée, et probablement mon prochain projet pour la librairie.
- Fonctionnalité d'exportation de fichiers OFX. Fonctionnalité suggérée par des développeurs de GnuCash. Présentement, les fichiers QIF sont généralement utilisés pour transférer des données financières entre les applications. Or, le format QIF, bien que populaire, n'est pas documenté, et souffre d'une importante carence au niveau de l'identification des transactions. Ce projet pourrait donner l'occasion de faire les changements architecturaux à la librairie pour permettre éventuellement la communication OFX bidirectionnelle. Cependant, il est beaucoup plus probable que la

voie facile sera employée: simplement traduire par un traitement textuel les structures de données et les entourer des marqueurs préalablement mémorisés. Le client n'aurait alors qu'à remplir chacune des structures de données de `libofx.h` autant de fois que nécessaire puis appeler une fonction pour générer le fichier. Dans tous les cas, c'est un projet intéressant, mais il est peu pertinent tant que LibOFX (ou du moins le support de la norme OFX) n'est pas implanté dans un grand nombre d'applications.

- Support de l'exportation directe de fichiers QIF. Analogue au précédent, mais presque trivial à réaliser, puisque la majeure partie du code nécessaire se trouve déjà dans `ofx2qif`.
- Support de l'importation de fichiers QIF. Malgré tous ses défauts, la norme QIF est présentement très utilisée par une panoplie de logiciels divers. Or, cette norme pourrait être supportée par les clients à l'aide de l'API actuel de LibOFX. Conjugé aux deux propositions précédentes, cela ferait de LibOFX une solution complète d'importation/exportation de fichiers. Certaines applications financières simples pourraient même utiliser LibOFX pour leurs propres données, plutôt que de définir leur propre format.
- Permettre d'enregistrer les fonctions visiteurs à appeler par la librairie plutôt que de forcer les usagers à implanter toutes les fonctions en utilisant les prototypes de `libofx.h`. C'est plus élégant d'un point de vue architectural, et cela permettrait plus de flexibilité aux clients, entre autre en permettant d'appeler plus d'une fonction.
- Chargement dynamique de la librairie par le module `gnc-ofx` afin de permettre de créer des exécutables de GnuCash qui fonctionneront même sans la librairie installée.

9.5) Conclusion

Sur le plan technique, ce projet est clairement un succès. Il est maintenant possible de traiter d'une façon fiable les fichiers OFX. Le parseur OFX est tolérant des malformations des fichiers. Non seulement il ne plante pas, il est même souvent capable d'extraire tout de même la majorité des informations d'un fichier endommagé.

Ce projet de fin d'études a acquis une vie propre. Bien des ajustements ont du être apportés à la planification originale en fonction des besoins exprimés par ses futurs usagers, des contraintes du développement de GnuCash et des nouveaux objectifs que je lui ai donnés. Mais c'est là la nature même des projets ouverts.

Contrairement aux projets classiques où les objectifs sont fixes (ainsi qu'en théorie le temps et le budget disponibles), les projets ouverts sont influencés par les objectifs, ressources, talents et opinions différentes et fluides de ses usagers et développeurs. Il n'est donc pas possible de faire une planification rigide. Être flexible est bien sûr un avantage, mais il m'apparaît évident que le ratio résultat obtenu sur mois-personne investi par tous les contributeurs est plus mauvais que celui d'un projet classique. Ce n'est donc que si un bon nombre de personnes mettent en commun leurs ressources qu'il y aura un gain d'efficacité pour chacun des participants (le résultat ne se divise pas, mais l'effort si).

Il est d'ailleurs trop tôt pour dire à quel point l'usage d'un processus ouvert aura contribué au succès de ce projet. Bien que plusieurs personnes aient donné des suggestions utiles et réglé quelques défauts, il est évident que jusqu'à maintenant le processus ouvert a jusqu'à un certain point ralenti sa progression. Pour ne citer qu'un exemple, 90% de la documentation qui a été produite aurait été parfaitement inutile si j'étais demeuré seul.

On pourrait mesurer son succès en fonction des objectifs énoncés dans la planification originale. De ce point de vue, les objectifs ont été en grande partie rencontrés. Cependant, il reste une partie manquante: la détection des transaction correspondantes entrées à la main. Il est à noter qu'il avait déjà été discuté avec le superviseur, peu après le début du projet, que l'emphase plus grande mise sur la documentation et la librairie risquait d'avoir des conséquences sur la partie interface graphique. Cependant, le design de cette partie est déjà avancé, les prototypes sont complets, ainsi qu'une partie de l'interface. Il est probable qu'elle sera achevée au cours des prochaines semaines, avant la sortie de la prochaine version stable de GnuCash (1.8), d'autant plus qu'elle se fera en commun avec Christian Stimmings, le développeur du module HBCI de GnuCash.

On pourrait également tenter de mesurer le succès en fonction des nouveaux objectifs que j'ai fixés pour ce projet, soit de faire de LibOFX le standard de facto pour l'importation de fichiers OFX, il est trop tôt pour se prononcer. Il m'apparaît évident que j'ai présentement une excellente base pour réussir. L'architecture est à mon avis structurée et documentée, et je n'ai pas eu de commentaires négatifs au sujet de l'API. Il reste maintenant à voir si un effet d'entraînement se créera: que d'autres logiciels intégreront LibOFX et que d'autres développeurs y contribueront. Si cela ne se produit pas, tous les efforts supplémentaires mis dans l'API, dans l'architecture et dans la documentation auront été inutiles.

Les événements des prochaines semaines risquent d'être déterminants pour le succès futur de LibOFX. Tout d'abord, il y aura la publication de GnuCash 1.8 qui devrait donner beaucoup d'exposition à la librairie. Ensuite, une fonctionnalité très attendue de la librairie est le support des comptes d'investissements. Si celui-ci arrive trop tard, ce manque sera sans doute à l'origine de plus de critiques que celles engendrées par d'éventuels défauts dans la librairie ou son API. Finalement, les logiciels Grisby et KmyMoney2 ont manifesté leur intérêt pour l'intégration de LibOFX. Si cela se fait rapidement, l'annonce par trois logiciels différents du support de la norme OFX par une même librairie assurera presque certainement que celle-ci deviendra le standard de facto.

10) Bibliographie

- (1) CheckFree Corp. et al. Open Financial Exchange Specification 1.6, 18 octobre 1999
- (2) Browne, Christopher B. *Finances, Linux, and Stuff*.
<http://cbbrowne.com/info/finances.html> . juin 2002.
- (3) Intuit Inc. *QIF Definition*. http://www.respmech.com/mym2qifw/qif_new.htm . 1996
- (4) Projet GnuCash. *QIF file format*.
<http://www.gnucash.org/lxr/gnucash/source/src/import-export/qif-import/file-format.txt>
- (5) Sperberg-McQueen, C. M. et Lou Burnard, A. *Gentle Introduction to SGML*.
<http://www-tei.uic.edu/orgs/tei/sgml/teip3sg/index.html>
- (6) World Wide Web Consortium. *Comparison of SGML and XML*. 1
<http://www.w3.org/TR/NOTE-sgml-xml> . 5/12/1997
- (7) World Wide Web Consortium. *Extensible Markup Language (XML)*,
<http://www.w3.org/XML/>
- (8) Leppänen, Marko. *Differences between SGML, XML and HTML*. .
<http://matwww.ee.tut.fi/~onykane/courses/prj/xmlsem2000/ht/leppanen.htm>
- (9) O'Reilly & Associates, Inc. *XML from the inside Out*. <http://www.xml.com/> . 2002
- (10) OASIS Open inc. *XML White Papers*.
http://www.xml.org/xml/resources_whitepapers.shtml
- (11) Walsh, Norman. *Converting an SGML DTD to XML*.
<http://www.xml.com/pub/a/98/07/dtd/> . 8 juillet 1998
- (12) Bosak, Jon. *XML - Questions & Answers*. <http://www.isgmlug.org/n3-1/n3-1-18.htm>

11) Annexes

11.1) Annexe 1: Résumé des fichiers et principales fonctions du projet LibOFX

Voici une description sommaire des fichiers composant le projet LibOFX. Notez que ce n'est pas un substitut pour le manuel du programmeur que l'on retrouve plus loin à l'annexe 5.

Fichier:	libofx.h
Description:	Fichier d'inclusion principal, il contient l'API. Il doit être inclus par les logiciels client. Toutes les fonctions qu'il définit (excepté ofx_proc_file) sont des fonctions visiteurs qui doivent être implantées par le code du client mais qui peuvent être vides.
Principales fonctions:	<pre>int ofx_proc_file(int argc, char *argv[]) Démarre le processus de traitement</pre> <pre>int ofx_proc_status(struct OfxStatusData data) La structure contient un message de la part du serveur OFX</pre> <pre>int ofx_proc_account(struct OfxAccountData data) La structure contient les informations de base permettant d'identifier un compte et ses caractéristiques (no. d'identification, devise, type de compte). Il y aura un appel pour chaque compte, avant les transactions et le relevé afin de permettre à l'application d'identifier le compte ou d'en créer un nouveau.</pre> <pre>int ofx_proc_transaction(struct OfxTransactionData data) La structure contient toutes les informations disponibles pour une transaction particulière. Un appel par transaction.</pre> <pre>int ofx_proc_statement(struct OfxStatementData data) La structure contient un relevé des informations sur l'état actuel d'un compte: le solde, le montant disponible pour retrait, les dates de validité des transactions qui viennent d'être envoyées, etc.</pre>

Fichier:	messages.h
Description:	Gestion des différents messages envoyés à l'utilisateur.
Principales fonctions:	<pre>int message_out(OfxMsgType type, const string message); OfxMsgType est une énumération permettant de préciser le type de message (DEBUG2, STATUS, INFO, etc.) Ceux-ci seront ou ne seront pas affichés à l'utilisateur selon la configuration de la variable associée à ce type de message.</pre>

Fichier:	ofx_error_msg.h
Description:	Information sur les différents codes d'erreurs OFX. Contient toutes les erreurs connues, passées et présentes.

Fichier:	ofx_error_msg.h
Principales fonctions:	const ErrorMessage find_error_msg(int param_code); Prend un numéro d'erreur en paramètre et retourne une description détaillée

Fichier:	ofx_preproc.cpp
Description:	Contient le code nécessaire pour le traitement des fichiers avant de les envoyer au parseur SGML. Reçoit le fichier en paramètre, trouve le fichier DTD approprié, éliminer les entête, les éléments OFX propriétaires et les commentaires.
Principales fonctions:	int ofx_proc_file(int argc, char *argv[]) Point d'entrée de la librairie (voir libofx.h). string sanitize_proprietary_tags(string input_string) Élimine les éléments OFX propriétaires et les commentaires. string find_dtd(const int requested_version = 160); Trouve le DTD approprié au fichier.

Fichier:	ofx_proc_sgml.cpp
Description:	Code spécifique au parseur OpenSP. En théorie, pour changer de parseur, presque tous les changements seraient dans ce fichier. C'est ici que sont construits les différents objets de ofx_proc_rs.cpp en fonction des éléments et des données SGML rencontrés.
Principales méthodes:	void startElement (const StartElementEvent &event) Début d'un élément SGML -void endElement (const EndElementEvent &event) Fin d'un élément SGML. -void data (const DataEvent &event) Données ASCII provenant d'un élément SGML -void error (const ErrorEvent &event) Erreurs du parseur

Fichier:	ofx_proc_rs.cpp
Description:	Une série d'objets (des contenants) représentant plus ou moins l'un des éléments SGML OFX. Ils sont construits et détruits au fur et à mesure par ofx_proc_sgml.cpp. Ces objets incluent les fonctions nécessaires pour remplir les structure de données correspondantes de libofx.h.

Fichier:	ofx_proc_rs.cpp
Principaux objets:	<p>OfxGenericContainer: Tous les autres contenants en héritent. Attributs et méthodes communes, dont la très importante fonction virtuelle <code>add_attribute(const string identifier, const string value)</code> qui permet d'ajouter les données brutes SGML d'une manière générique.</p> <p>OfxDummyContainer: Sert à contenir les éléments OFX SGML que la librairie ne connaît pas. Permet aussi de ne pas avoir à supporter explicitement des éléments ne contenant pas eux-mêmes de données. <code><OFX></code> par exemple.</p> <p>OfxAccountContainer: Une abstraction d'un compte bancaire créée à partir de <code><BANKACCTFROM></code> (compte bancaire) or <code><CCACCTFROM></code> (compte de cartes de crédit).</p> <p>OfxStatementContainer: Une abstraction d'un relevé de compte créée à partir d'un <code><STMTRS></code> (relevé de compte bancaire) ou un <code><CCSTMTRS></code> (relevé de compte de cartes de crédit).</p> <p>OfxStatusContainer: Permet de supporter les différents messages d'erreur que le serveur OFX peut envoyer.</p> <p>OfxTransactionContainer: Permet de supporter les transactions telles quelles. Créé à partir d'un <code><STMTRN></code></p>

Fichier:	ofx_utilities.cpp
Description:	Différentes fonctions utilitaires nécessaires au bon fonctionnement de la librairie.
Principales fonctions:	<pre>-ostream & operator<< (ostream &os, SGMLApplication::CharString s) -wchar_t * CharStringtowchar_t (SGMLApplication::CharString source, -wchar_t *dest) -string CharStringtostring (const SGMLApplication::CharString source, string &dest) -string AppendCharStringtostring (const SGMLApplication::CharString source, string &dest) Différentes fonctions permettant de convertir les chaînes de caractères OpenSP en chaînes STL²¹.</pre> <p><code>time_t ofxdate_to_time_t (const string ofxdate)</code> Convertit une chaîne de caractères contenant une date en format OFX en une structure C <code>time_t</code>. La conversion est précise jusqu'à la seconde. (OFX permet une précision jusqu'à une milliseconde)</p> <p>double ofxamount_to_double (const string ofxamount) Conversion d'un montant d'argent en représentation point flottant.</p> <p><code>string strip_whitespace (const string para_string)</code> Permet d'éliminer les espaces blancs et une série de caractères ASCII étranges pouvant être mêlés aux chaînes de caractères de OpenSP (retour arrière, tabulation, fin de ligne, etc.)</p>

²¹ Standard Template Library

11.2) Annexe 2: Document de design présenté aux développeurs de GnuCash.

This a draft of a design proposal for a generic import architecture. The objective is to maximize code sharing between the QIF, HBCI and OFX modules.

The most important area of potential code sharing is the account and transaction matching code. This code has 3 distinct roles:

- Finding the source account.
- Finding and eliminating transactions downloaded twice in the source account.
- Finding the destination account(s), and finding the matching transactions(s) if it/they exist(s).

The Online System specific module is responsible for any steps necessary for obtaining and processing the data. During this process, it must use the generic-import module to:

- Identify and if necessary create the source account: The account is identified using the account number for OFX and HBCI, and the account name or description for qif, if available. The account number or identifier is stored in a kvp_string with key account_online_id. The online system can use and format this frame however it sees fit, but it must be a kvp_string. The account can be found using the function:

```
Account * gnc_import_select_account(char * account_online_id_value,
                                   char * account_human_description,
                                   gnc_commodity * new_account_default_commodity,
                                   GNCAccountType new_account_default_type);
```

If no account is found with a matching online_id, the generic-module gives the user the option to select an existing account from a list, or create a new one. The account_online_id is then stored in the selected or created account's kvp_frame. The last 3 parameters of the function are defaults for new account creation and are optional.

- The Online System specific module is then responsible for creating a GnuCash transaction and adding the source split (associated with the source account, possibly created above), and filling it with as much information as it has as it's disposal (much info is available for ofx, little for qif). If a unique transaction id is available from the online system, it is stored in the splits kvp_frame, using key transaction_online_id. No transaction matching or duplicate detection is done at this stage.

The generic module then receives the Transaction for the online system specific module using function:

```
void gnc_import_add_trans(Transaction *trans);
```

(We do not use GUID, because in all cases, the transaction was just created)

The functions defines the following enum:

```
enum gnc_match_probability{
    CERTAIN,
```

```

PROBABLE,
LIKELY,
POSSIBLE,
UNLIKELY,
IMPOSSIBLE
}

```

Here is the pseudocode of the `gnc_import_add_trans` function:

Variables: `matches` (a list of possible match with likelihood)

`split_to_match` = `trans`'s first split.

In `split_to_match`'s parent account; for each split where `date >= split_to_match.date - 2 months`:

```

if transaction_online_id match
    add to matches using CERTAIN
    if preferences dictate: end search here
if amount match
    if transaction_online_id exists but doesn't match
        add to matches using UNLIKELY (not IMPOSSIBLE, could be
protocol change or prior error)
    else if memo match and date within 4 days
        add to matches using PROBABLE
    else if date within 24 hours
        add to matches using LIKELY
    else if date within 10 days
        add to matches using POSSIBLE
    else
        add to matches using UNLIKELY

```

Present the list of matches to the user in decreasing order of likelihood. User has the option of selecting one of the match or creating a new transaction.

Add `transaction_online_id` to selected split

Erase from other CERTAIN splits

if transaction not balanced

 TODO: `gnc_balance_transaction(Transaction *trans)`

commit changes

return

`gnc_balance_transaction((Transaction *trans)` add's or matches other splits until the transaction is balanced, using whatever user interaction and heuristics are appropriate. Since I haven't really used `gnucash`'s current transaction matching before, I would like someone else to contribute the description of the process to match unbalanced transactions.

Remarks and things to remember:

-Credit card transactions can sometimes appear over a month after the purchase (clerk

lost the paper, international transaction not always fast, etc.)

-void `gnc_import_add_trans(Transaction *trans)` should return as soon as possible (BEFORE user interaction) so that for systems that maintain a connection (such as HBCI) the user won't run into timeouts. For example, `gnc_import_add_trans` could check if it's main dialog is open, and open it if it isn't, add to the list and return immediately. The dialog is closed automatically once the list is empty.

-We may want to implement the function in such a way that it won't match any transaction that have been added as part of the current import process (flag them volatile or something). This will solve the problems of multiple interac withdrawals in the same day for QIF, (possibly HBCI too?).

-The transaction passed to `gnc_import_add_trans` will have only one split for OFX and HBCI, but 1 or more for QIF.

11.3) Annexe 3: Document de présentation et de planification original

Proposition de PFE : Module de conciliation bancaire OFX

Résumé :

Ce projet consiste à écrire et intégrer un module de conciliation bancaire pour le logiciel financier GnuCash. Le logiciel GnuCash est un logiciel de finances personnelles similaire à Quicken, mais imposant les pratiques comptables de la double entrée. Il est distribué sous licence GPL²².

Le projet :

L'écriture d'un module de communication OFX²³ général avait été tentée par les développeurs de GnuCash. Celui-ci était prévu pour supporter la norme OFX au complet (conciliation bancaire, paiement de factures, transferts bancaires, transactions boursières, etc.). Ce projet a échoué. Les développeurs estimaient à 12 mois-personnes le travail nécessaire. Le projet ne comptant que sur des bénévoles, il n'avancait plus.

Je propose donc un projet moins ambitieux : n'implanter qu'une partie de la norme OFX : L'importation de ce qui est communément appelé un « fichier OFX » soit un fichier de conciliation bancaire suivant la norme OFX. Cela permet d'implanter la fonction la plus désirée de la norme OFX, soit de concilier un compte GnuCash avec des transactions téléchargées à partir d'un compte bancaire ou d'une carte de crédit. Je crois ce projet réalisable comme un PFE de 6 crédits.

Considérations techniques :

Mon module sera basé sur la norme OFX 1.6. C'est la version la plus récente des « anciennes » normes OFX, basées sur SGML²⁴, et elle est compatible avec les versions précédentes. Il existe une nouvelle norme, OFX2.01, mais celle-ci est basée sur XML²⁵, et est encore très peu déployée par les institutions financières. Utiliser OFX2.01 poserait probablement des problèmes de compatibilité arrière. Par contre, les deux normes sont des cousines rapprochées²⁶ et il est possible de passer de l'une à l'autre. Comme il existe une multitude d'outils pour travailler avec XML, et moins pour SGML, il est envisageable qu'il soit plus simple de convertir le DTD au format XML, et qu'il soit ensuite simple de supporter les deux normes. Mais à cette étape préliminaire, je n'ai

²² GNU General Public License: <http://www.gnu.org/licenses/licenses.html#TOCGPL>

²³ Open Financial Exchange: <http://www.ofx.net/ofx/default.asp>

²⁴ Standard Generalized Markup Language

²⁵ Extensible Markup Language

²⁶ Comparison of SGML and XML, World Wide Web Consortium Note 15/12/1997, <http://www.w3.org/TR/NOTE-sgml-xml>

aucun moyen de le savoir sans une étude plus approfondie des deux normes et des différents outils. De plus, mon institution financière (le service Accès-D de Desjardins) exporte ses fichiers dans le format OFX 1.02, et il me faut impérativement un moyen de me générer des fichiers tests. Donc, pour que mon module soit utile à la majorité des utilisateurs et que je puisse le tester, je m'en tiendrai à priori à OFX1.6 dans le cadre de ce PFE.

Mon projet devra être intégré à un logiciel existant. Il faudra donc que je travaille avec les personnes présentement en charge, et que je me conforme à certaines de leurs normes et procédés. De plus le modèle de développement ouvert utilisé par les logiciels libres implique que je devrai demander des modifications ou éliminer des erreurs dans des modules maintenus par d'autres développeurs, et que d'autres feront sans doute de même pour les miens. J'ai déjà contacté la communauté de développeurs de GnuCash, et j'ai été bien reçu.

Implantation et temps prévu :

Familiarisation avec le code source du logiciel et la norme OFX. À cette étape, je prévois également discuter des modifications au logiciel actuel afin de faciliter l'intégration aux dernières étapes. Je prévois 20 h pour cette étape préliminaire.

Ajout du numéro de transit aux structures de données des comptes de GnuCash. L'importation de fichiers OFX dans le populaire logiciel Quicken a des faiblesses qui laissent une ambiguïté quant au compte que l'on désire concilier si l'on met à jour le logiciel ou si notre service bancaire en ligne fait certaines modifications. Je ne veux pas répéter ce problème. Le numéro de transit étant unique, il n'y aura plus d'ambiguïté quant au compte à concilier. Cependant, les comptes de cartes de crédit n'ont pas de numéro de transit, un autre identificateur unique devra être utilisé. Je devrai aussi ajouter l'interface usager et l'interface programmeur pour accéder et modifier ces données. Le changement devrait être relativement trivial. Temps prévu : 7h.

Parseur du fichier OFX. C'est bien sûr le travail le plus complexe et le plus important de ce projet. Plusieurs approches sont possibles. Il serait possible de simplement parcourir linéairement le fichier à la recherche des jetons définissant un compte bancaire, puis d'interpréter directement les jetons qui suivent. Ce serait probablement une approche assez rapide, mais extrêmement risquée, car elle rendrait quasi-impossible la détection d'erreurs. De plus le nombre d'exceptions à gérer serait très élevé, et la maintenance très difficile. Il vaudrait probablement mieux utiliser une librairie SGML ou XML existante, l'utiliser pour parser le DTD et accéder ensuite aux éléments du fichier en parcourant l'arbre généré. Cette approche offre la validation, et donnera également une excellente base pour que d'autres développeurs puissent éventuellement étendre mon module pour ajouter d'autres fonctions que la conciliation bancaire (paiement en ligne, etc.). Un bon nombre d'heures d'analyse restent à passer pour choisir la meilleure implantation de cette fonction.

Ce module transformera les données contenues dans le fichier OFX en un compte temporaire de GnuCash, et affichera les autres données d'intérêt (Version, date, banque,

transit, etc.). 125h prévues.

Fonction de fusion. La fusion des données importées devrait être séparée de l'importation elle-même pour des raisons de robustesse et de lisibilité du code. Ainsi on ne fera pas des opérations de copie et de comparaison sur des structures de données différentes, et on pourra utiliser les fonctions de copie et de recherche qui font déjà partie du logiciel. De plus, ce module a d'autres utilisations potentielles, et pourra être réutilisé dans d'autres parties du logiciel.

La fusion consiste à prendre le compte temporaire et le fusionner avec le compte correspondant (identifié par son no. de transit) de GnuCash. Si le no. de transit du compte en question n'existe pas dans les autres comptes, le logiciel offre à l'utilisateur de choisir le compte à utiliser, ou en créer un nouveau. Il ajoute le numéro de transit au compte choisi. Si deux comptes ont le même numéro de transit, un message d'erreur est affiché.

Un fois le compte identifié, pour chaque transaction, le module :

-cherche une transaction ayant le même identificateur, la même date et le même montant. S'il en trouve une et une seule, il considère l'opération comme conciliée.

-S'il n'en a pas trouvé, il cherche une transaction ayant le même identificateur et le même montant dans les 4 jours précédent l'opération conciliée, ou une opération ayant le même montant et la même date. S'il en trouve une, il offre de faire correspondre.

-S'il n'en a pas trouvé ou si l'utilisateur a refusé la correspondance, il offre d'ajouter l'opération au compte.

La gestion des duplicatas est importante. C'est là une autre faiblesse du logiciel Quicken que je ne voudrais pas reproduire. S'il trouve plus d'une opération correspondante à une opération téléchargée, le module informera l'utilisateur d'une erreur potentielle. Par défaut Quicken fait correspondre toute opération téléchargée avec la récente opération ayant le même identificateur et le même montant, même si elle est plus vieille de plusieurs mois! Limiter à un certain nombre de jours comme je l'ai fait plus haut règle une partie du problème, mais j'aimerais aussi régler le problème posé par deux retraits d'un même montant dans un guichet automatique faits la même journée. Mais je ne suis pas sûr qu'il y ait une solution plus élégante dans ce cas que d'offrir à l'utilisateur d'ajouter les deux opérations. De plus, il ne faut pas rejeter complètement l'approche de Quicken, car pour les cartes de crédit, il peut arriver de recevoir la transaction plusieurs semaines après un achat. (Récemment j'ai payé un repas dans un aéroport d'Europe et j'ai reçu la transaction plus d'un mois après)

Une partie des fonctions nécessaires à la fusion étant déjà offerte par GnuCash, cette partie sera probablement intégrée à un module existant. C'est à ce niveau que se situe la majorité de l'intégration au reste du logiciel, c'est l'une des principales difficultés pressenties. 75h prévues.

Documentation: Une partie de tout développement structuré est d'écrire de la

documentation à l'intention des autres développeurs ainsi que des utilisateurs. 10h prévues.

Tests et déverminage : L'échelle et le temps alloué à ce projet ne me permet pas de prévoir autre chose que des tests unitaires et quelques tests fonctionnels. Je vais devoir écrire des tests pour exercer les fonctions du module de fusion (particulièrement pour la vérification des duplicatas). Les tests fonctionnels seront très limités, puisque je n'ai qu'une source de fichiers OFX pour l'instant (il est cependant possible que d'autres développeurs acceptent de me fournir des fichiers OFX). Je ne prévois pas générer une campagne de tests automatisés. 25h prévues

Rapport : Une partie de la documentation produite et de la correspondance échangée pourra être récupérée pour rédiger le rapport, j'y alloue donc peu de temps : 20h prévues.

Total : 282 heures

Liens et références :

GnuCash

Le logiciel GnuCash <http://www.gnucash.org/>

OFX

Le consortium OFX, <http://www.ofx.net/ofx/default.asp>

XML et SGML

-A Gentle Introduction to SGML, C. M. Sperberg-McQueen and Lou Burnard, <http://www-tei.uic.edu/orgs/tei/sgml/teip3sg/index.html>

-Comparison of SGML and XML, World Wide Web Consortium Note, 15/12/1997, <http://www.w3.org/TR/NOTE-sgml-xml>

-Differences between SGML, XML and HTML (avec graphiques) <http://matwww.ee.tut.fi/~onykane/courses/prj/xmlsem2000/ht/leppanen.htm>

Extensible Markup Language (XML), <http://www.w3.org/XML/>

-XML from the inside Out, <http://www.xml.com/>

-XML White Papers http://www.xml.org/xml/resources_whitepapers.shtml

-Converting an SGML DTD to XML, Norman Walsh, July 08, 1998, <http://www.xml.com/pub/a/98/07/dtd/>

-XML - Questions & Answers, Jon Bosak, Sun Microsystems, <http://www.isgmlug.org/n3-1/n3-1-18.htm>

Outils

-OpenSP Suite d'outils SGML <http://openjade.sourceforge.net/doc/index.htm>

-Free SGML Tools, Christopher Browne, <http://www.ntlug.org/~cbbrowne/sgmlfreestuff.html>

-PSGML: Edition de fichiers SGML avec Emacs http://www.lysator.liu.se/projects/about_psgml.html

11.4) Annexe 4: Le fichier libofx.h définissant l'API

```

/*****
    libofx.h - Main header file for the libofx API
    -----
    copyright      : (C) 2002 by Benoit Grégoire
    email         : bock@step.polymtl.ca
*****/
/**@file
 * \brief Main header file containing the LibOfx API
 *
 * This file should be included for all applications who use this API. This
 * header file will work with both C and C++ programs. The entire API is
 * made of the following structures and functions.
 *
 * All of the following ofx_proc_* functions are callbacks (Except
 * ofx_proc_file which is the entry point). They must be implemented by
 * your program, but can be left empty if not needed. They are called each
 * time the associated structure is filled by the library.
 *
 * Important note: The variables associated with every data element have a
 * *_valid companion. Always check that data_valid == true before using.
 * Not only will you ensure that the data is meaningful, but also that
 * pointers are valid and strings point to a null terminated string.
 * Elements listed as mandatory are for information purpose only, do not
 * trust the bank not to send you non-conforming data...
 */
/*****
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 *****/

#ifndef LIBOFX_H
#define LIBOFX_H
#include <time.h>

#ifdef __cplusplus
#define CFCT extern "C"
#else
#define CFCT
#define true 1
#define false 0
#endif

#define OFX_ELEMENT_NAME_LENGTH      100
#define OFX_SVRTID2_LENGTH           36 + 1
#define OFX_CHECK_NUMBER_LENGTH      12 + 1
#define OFX_REFERENCE_NUMBER_LENGTH  32 + 1
#define OFX_FITID_LENGTH             255 + 1
#define OFX_TOKEN2_LENGTH            36 + 1
#define OFX_MEMO2_LENGTH             390 + 1
#define OFX_BALANCE_NAME_LENGTH      32 + 1
#define OFX_BALANCE_DESCRIPTION_LENGTH 80 + 1
#define OFX_CURRENCY_LENGTH          3 + 1 /* In ISO-4217 format */
#define OFX_BANKID_LENGTH            9
#define OFX_BRANCHID_LENGTH          22
#define OFX_ACCTID_LENGTH            22
#define OFX_ACCTKEY_LENGTH           22
/* Must be MAX of <BANKID>+<BRANCHID>+<ACCTID> or <ACCTID>+<ACCTKEY> */

```

```

#define OFX_ACCOUNT_ID_LENGTH OFX_BANKID_LENGTH + OFX_BRANCHID_LENGTH +
OFX_ACCTID_LENGTH + 1
#define OFX_MARKETING_INFO_LENGTH 360 + 1
#define OFX_TRANSACTION_NAME_LENGTH 32 + 1

/**
 * \brief ofx_proc_file is the entry point of the library.
 *
 * ofx_proc_file must be called by the client, with a list of 1 or more OFX
 * files to be parsed in command line format.
 */
CFCT int ofx_proc_file(int argc, char *argv[]);

/**
 * \brief An abstraction of an OFX STATUS element.
 *
 * The OfxStatusData structure represents a STATUS OFX element sent by the
 * OFX server. Be carefull, you do not have much context except the entity
 * name so your application should probably ignore this status if code==0.
 * However, you should display a message if the status is non-zero,
 * since an error probably occurred on the server side.
 *
 * In a future version of this API, OfxStatusData structures might be
 * linked from the OFX structures they are related to.
 */
struct OfxStatusData{
    /** @name Additional information
     * To give a minimum of context, the name of the OFX SGML elemet where
     * this <STATUS> is located is available.
     */
    char ofx_element_name[OFX_ELEMENT_NAME_LENGTH];/** Name of the OFX element
                                                    this status is relevant to */
    int ofx_element_name_valid;

    /** @name OFX mandatory elements
     * The OFX spec defines the following elements as mandatory. The associated
     * variables should all contain valid data but you should not trust the servers.
     * Check if the associated *_valid is true before using them. */
    int code; /**< Status code */
    char* name; /**< Code short name */
    char* description; /**< Code long description, from ofx_error_msg.h */
    int code_valid; /**< If code_valid is true, so is name and description
                    (They are obtained from a lookup table) */
    /** Severity of the error */
    enum Severity{INFO, /**< The status is an informational message */
                  WARN, /**< The status is a warning */
                  ERROR /**< The status is a true error */
    } severity;
    int severity_valid;

    /** @name OFX optional elements
     * The OFX spec defines the following elements as optional. If the
     * associated *_valid is true, the corresponding element is present and the
     * associated variable contains valid data. */

    char* server_message; /**< Explanation given by the server for the Status
    Code.
                            Especially important for generic errors. */
    int server_message_valid;
    /**@}*/
};

```

```

/**
 * \brief The callback function for the OfxStatusData stucture.
 *
 * An ofx_proc_status event is sent everytime the server has generated a OFX
 STATUS element. As such, it could be received at any time (but not during
 other events). An OfxStatusData structure is passed to this even.
 */
CFCT int ofx_proc_status(struct OfxStatusData data);

/**
 * \brief An abstraction of an account
 *
 * The OfxAccountData structure gives information about a specific account,
 including it's type, currency and unique id.
 */
struct OfxAccountData{

    /** @name OFX mandatory elements
     * The OFX spec defines the following elements as mandatory. The associated
     variables should all contain valid data but you should not trust the servers.
     Check if the associated *_valid is true before using them. */

    /** The account_id is actually a concatenation of
     <BANKID><BRANCHID><ACCTID> for a bank account, and <ACCTID><ACCTKEY>
     for a credit card account. Together, they form a worldwide OFX unique
     identifier wich can be used for account matching, even in system with
     multiple users.*/
    char account_id[OFX_ACCOUNT_ID_LENGTH];

    int account_id_valid;
    /** account_type tells you what kind of account this is. See the AccountType
    enum */
    enum AccountType{
        OFX_CHECKING, /**< A standard checking account */
        OFX_SAVINGS, /**< A standard savings account */
        OFX_MONEYMRKT, /**< A money market account */
        OFX_CREDITLINE, /**< A line of credit */
        OFX_CMA, /**< Cash Management Account */
        OFX_CREDITCARD /**< A credit card account */
    } account_type;
    int account_type_valid;
    char currency[OFX_CURRENCY_LENGTH]; /**< The currency is a string in ISO-4217
    format */
    int currency_valid;

};

/**
 * \brief The callback function for the OfxAccountData structure.
 *
 * The ofx_proc_account event is always generated first, to allow the
 application to create accounts or ask the user to match an existing
 account before the ofx_proc_statement and ofx_proc_transaction event are
 received. An OfxAccountData is passed to this event.
 *
 Note however that this OfxAccountData structure will also be available as
 part of OfxStatementData structure passed to ofx_proc_statement event.
 */
CFCT int ofx_proc_account(struct OfxAccountData data);

/**
 * \brief An abstraction of a transaction in an account.
 *

```

```

* The OfxTransactionData structure contains all available information about
an actual transaction in an account.
*/
struct OfxTransactionData{

    /** @name OFX mandatory elements
     * The OFX spec defines the following elements as mandatory. The associated
     variables should all contain valid data but you should not trust the servers.
     Check if the associated *_valid is true before using them. */

    char account_id[OFX_ACCOUNT_ID_LENGTH];/**< Use this for matching with
                                             the relevant account in your
                                             application */

    int account_id_valid;
    enum TransactionType{
        OFX_CREDIT,    /**< Generic credit */
        OFX_DEBIT,     /**< Generic debit */
        OFX_INT,       /**< Interest earned or paid (Note: Depends on signage of
amount) */
        OFX_DIV,       /**< Dividend */
        OFX_FEE,       /**< FI fee */
        OFX_SRVCHG,    /**< Service charge */
        OFX_DEP,       /**< Deposit */
        OFX_ATM,       /**< ATM debit or credit (Note: Depends on signage of
amount) */
        OFX_POS,       /**< Point of sale debit or credit (Note: Depends on signage
of amount) */
        OFX_XFER,      /**< Transfer */
        OFX_CHECK,     /**< Check */
        OFX_PAYMENT,   /**< Electronic payment */
        OFX_CASH,      /**< Cash withdrawal */
        OFX_DIRECTDEP, /**< Direct deposit */
        OFX_DIRECTDEBIT,/**< Merchant initiated debit */
        OFX_REPEATPMT, /**< Repeating payment/standing order */
        OFX_OTHER      /**< Somer other type of transaction */
    } transactiontype;
    int transactiontype_valid;
    time_t date_posted;/**< Date the transaction took effect (ex: date it
appeared on your credit card bill) */
    int date_posted_valid;
    double amount;     /**< Actual money amount of the transaction, signage
will determine if money went in or out */
    int amount_valid;
    char fi_id[256];   /**< Generated by the financial institution (fi),
unique id of the transaction, to be used to detect
duplicate downloads */

    int fi_id_valid;

    /** @name OFX optional elements
     * The OFX spec defines the following elements as optional. If the
     associated *_valid is true, the corresponding element is present and
     the associated variable contains valid data. */

    time_t date_initiated;    /**< Date the transaction was initiated (not
always provided) (ex: date you bought
something in a store) */
    int date_initiated_valid;
    time_t date_funds_available;/**< Date the funds are available (not always
provided) (ex: the date you are allowed to
withdraw a deposit */
    int date_funds_available_valid;
    /** IMPORTANT: if fi_id_corrected is present, this transaction
is meant to replace or delete the transaction with this fi_id. See
OfxTransactionData::fi_id_correction_action to know what to do. */
    char fi_id_corrected[256];

```

```

int fi_id_corrected_valid;
/** The OfxTransactionData::FiIdCorrectionAction enum contains the action
    to be taken */
enum FiIdCorrectionAction{
    DELETE, /**< The transaction with a fi_id matching fi_id_corrected should
            be deleted */
    REPLACE /**< The transaction with a fi_id matching fi_id_corrected should
            be replaced with this one */
} fi_id_correction_action;
int fi_id_correction_action_valid;

/** Used for user initiated transaction such as payment or funds transfer.
    Can be seen as a confirmation number. */
char server_transaction_id[OFX_SVRTID2_LENGTH];
int server_transaction_id_valid;
/** The check number is most likely an integer and can probably be
    converted properly with atoi(). However the spec allows for up to
    12 digits, so it is not guaranteed to work */
char check_number[OFX_CHECK_NUMBER_LENGTH];
int check_number_valid;
/** Might present in addition to or instead of a check_number.
    Not necessarily a number */
char reference_number[OFX_REFERENCE_NUMBER_LENGTH];
int reference_number_valid;
long int standard_industrial_code;/**< The standard industrial code can have
                                at most 6 digits */
int standard_industrial_code_valid;
char payee_id[OFX_SVRTID2_LENGTH];/**< The identifier of the payee */
int payee_id_valid;
char name[OFX_TRANSACTION_NAME_LENGTH];/**< Can be the name of the payee or
                                the description of the transaction */
int name_valid;
char memo[OFX_MEMO2_LENGTH];/**< Extra information not included in name */
int memo_valid;

/***** NOT YET COMPLETE!!! *****/
};

/**
 * \brief The callback function for the OfxTransactionData stucture.
 *
 * An ofx_proc_transaction event is generated for every transaction in the
 * ofx response. An OfxTransactionData structure is passed to this event
 */
CFCT int ofx_proc_transaction(struct OfxTransactionData data);

/**
 * \brief An abstraction of an account statement.
 *
 * The OfxStatementData structure contains information about your account
 * at the time the ofx response was generated, including the balance. A
 * client should check that the total of his recorded transactions matches
 * the total given here, and warn the user if they dont.
 */
struct OfxStatementData{

    /** @name OFX mandatory elements
     * The OFX spec defines the following elements as mandatory. The
     * associated variables should all contain valid data but you should not
     * trust the servers. Check if the associated *_valid is true before using
     * them.
     */

    char currency[OFX_CURRENCY_LENGTH]; /**< The currency is a string in ISO-4217

```

```

format */
    int currency_valid;
    char account_id[OFX_ACCOUNT_ID_LENGTH];/**< Use this for matching this
statement with
                                the relevant account in your application
*/
    struct OfxAccountData account; /**< Full account structure, see OfxAccountData
*/
    int account_id_valid;
    /** The actual balance, according to the FI. The user should be warned
        of any discrepancy between this and the balance in the application */
    double ledger_balance;
    int ledger_balance_valid;
    time_t ledger_balance_date;/**< Time of the ledger_balance snapshot */
    int ledger_balance_date_valid;

/** @name OFX optional elements
 * The OFX spec defines the following elements as optional. If the
associated *_valid is true, the corresponding element is present and the
associated variable contains valid data. */

double available_balance; /**< Amount of money available from the account.
                            Could be the credit left for a credit card,
                            or amount that can be withdrawn using INTERAC) */
int available_balance_valid;
time_t available_balance_date;/** Time of the available_balance snapshot */
int available_balance_date_valid;
/** The start time of this statement.
 *
All the transactions between date_start and date_end should have been
provided */
time_t date_start;
int date_start_valid;
/** The end time of this statement.
 *
If provided, the user can use this date as the start date of his next
statement request. He is then assured not to miss any transactions. */
time_t date_end;
int date_end_valid;
/** marketing_info could be special offers or messages from the bank,
    or just about anything else */
char marketing_info[OFX_MARKETING_INFO_LENGTH];
int marketing_info_valid;
};

/**
 * \brief The callback function for the OfxStatementData stucture.
 *
 * The ofx_proc_statement event is sent after all ofx_proc_transaction
events have been sent. An OfxStatementData is passed to this event.
*/
CFCT int ofx_proc_statement(struct OfxStatementData data);

/**
    \brief NOT YET SUPPORTED
*/
struct OfxCurrency{
    char currency[OFX_CURRENCY_LENGTH]; /**< Currency in ISO-4217 format */
    double exchange_rate; /**< Exchange rate from the normal currency of the
account */
    int must_convert; /**< true or false */
};

#endif

```


11.5) Annexe 5: Manuel du programmeur

Version PostScript du Manuel. Pour une version à jour en format HTML avec les liens aux sources, voir <http://step.polymtl.ca/~bock/libofx/html/index.html>.