

qof_book_merge

**'Query Object Framework: Design and
direction.'**

Neil Williams

qof_book_merge: 'Query Object Framework: Design and direction.'

by Neil Williams

Copyright © 2004-2005 Neil Williams

The GNU GENERAL PUBLIC LICENSE Version 2, June 1991

This documentation is part of QOF.

QOF 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.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

The GNU General Public Licence (GPL.txt).

Table of Contents

Preface: Dependencies	vi
1. QOF dependencies.....	vi
2. GnuCash dependencies.....	vi
2.1. GnuCash for Gnome2: gnucash-gnome2-dev	vi
2.2. Building GnuCash	vii
1. Introduction	1
1.1. Terms and definitions.....	1
1.1.1. QOF: Query Object Framework (http://qof.sourceforge.net/).....	1
1.1.2. Pilot-QOF: Querying Palm databases as objects (http://pilot-qof.sourceforge.net/).....	1
1.1.3. QOF-gen: QOF Object Generator (http://qof-gen.sourceforge.net/).....	1
1.1.4. Data Freedom: Liberate your data from the application (http://www.data-freedom.org/).....	2
1.1.5. What's a QofBook?	2
1.1.6. QSF - QOF Serialization Format.....	2
1.2. Other versions of this documentation.....	2
2. Background	4
2.1. What kind of merge?.....	4
2.2. How to identify one entity.....	4
2.3. The requirements for a merge operation.....	4
3. Source Documentation	6
3.1. Doxygen Documentation.....	6
3.1.1. qof_book_merge Doxygen documentation (doxygen/group__BookMerge.html).....	6
3.1.2. QSF Doxygen documentation (doxygen/group__QSF.html).....	6
3.1.3. QOF Doxygen documentation (http://qof.sourceforge.net/doxy/).....	6
3.1.4. GnuCash Doxygen documentation (http://cvs.gnucash.org/docs/HEAD/).....	6
3.1.5. GnuCash Gnome2 port Doxygen documentation (gnome2/).....	6
3.2. Other general documentation.....	6
3.2.1. Gnucash Design documentation.....	7
3.2.2. GnuCash Tutorial and Concepts Guide.....	7
3.2.3. GnuCash Help Manual	7
3.2.4. KVP Values used By GnuCash.....	7
3.3. qof_book_merge Source Code.....	7
3.4. QSF Source code.....	8
3.4.1. QSF and qof_book_merge tarballs	9
4. Creating GnuCash Invoices using a Palm PDA	10
4.1. Beginnings.....	10
4.2. Getting into GnuCash / QOF	10
4.2.1. Why the GnuCash File QofBackend needs changing	12
4.2.2. Tips on debugging within GnuCash	12
4.3. Building QOF onto pilot-link.....	13
4.3.1. Converting existing objects to QOF	14
4.4. pilot-qof manpages.....	17
pilot-qof	17
pilot-qof.....	21

pilot-qof	23
5. QSF - QOF Serialization Format.	26
5.1. What is QSF?	26
5.1.1. Features of QSF	26
5.1.2. Requirements of QSF	27
5.1.3. Validation of QSF.	28
5.1.4. QSF examples	28
5.2. Mapping QSF objects between QOF applications	30
5.2.1. The QSF Map file	30
5.2.2. Relating the map to the QSF objects	32
5.3. Writing new QSF maps	33
6. Merging QofBook's.....	37
6.1. Preparing the rule set.....	37
6.2. Draft of a rule set framework.	38
6.3. Design of the merge	42
6.3.1. Example programs for qof_book_merge.....	43
6.4. Using qof_book_merge with new and existing QOF objects	44
6.5. Known problems	46
6.6. Design improvements.....	48
7. Data Mining and freedom of access.	52
7.1. Data mining within QOF.....	52
7.2. Data Freedom within QOF.....	53

List of Tables

6-1. Struct members (Engine) without set_fcn, get_fcn pairs	46
6-2. Struct members (Business) without set_fcn, get_fcn pairs	47
6-3. Struct members (Engine) with resolved issues.....	49
6-4. Struct members (Business) with resolved issues.....	49

Preface: Dependencies

1. QOF dependencies.

2. GnuCash dependencies.

To build GnuCash from CVS HEAD using Debian unstable, the following packages were required:

- libgwrapguile-dev
- guile-1.6-dev
- libguppi-dev
- liboaf-dev
- libghttp-dev
- libzvt-dev
- libgtkhtml-dev
- libgal-dev
- libofx-dev
- libltdl3-dev
- automake1.7
- intltool
- libpopt-dev

2.1. GnuCash for Gnome2: gnuCash-gnome2-dev

To build the gnuCash-gnome2-dev branch, you also need:

- libgnome2-dev
- libgnomeui-dev
- libgnomeprint2.2-dev
- libgnomeprintui2.2-dev
- libgtkhtml3.2-dev
- libbz2-dev
- libgsf-gnome-1

These packages also brought in a range of other dependencies.

2.2. Building GnuCash

To build from GnuCash CVS, the sequence is:

- `cvs -d :pserver:cvs@cvs.gnucash.org:/home/cvs/cvsroot login`

The password is 'guest'.

- `cvs -d :pserver:cvs@cvs.gnucash.org:/home/cvs/cvsroot checkout gnucash`
- `cd gnucash`
- `./autogen.sh --enable-opt-style-install --enable-doxygen --disable-nls --prefix=/opt/gnucash/`
- `make`
- `make install`

Common mistake of mine #1: Don't be tempted to use `cd src/` before issuing the make command or you will find the build halts with an error building `../lib/libc/libc-missing.la` required by `libgncmmod-engine.la`. Always issue the make command from the top level `gnucash/` directory.

Chapter 1. Introduction

This mini-project was started to add extra functionality to GnuCash (<http://www.gnucash.org/>) to make it easier to import data from various sources into GnuCash to allow users to manage personal or business finances using Free Software.

Various import filters exist in GnuCash but it became clear that a capability to merge two accounting books (specifically two QofBook objects) was desirable. QofBook comes from the Query Object Framework (<http://qof.sourceforge.net/>) - an offshoot from the GnuCash Project. QOF is basically the GnuCash "engine" with all of the financial objects removed. As such, QOF is "tried and true", and has proven its worth.

1.1. Terms and definitions.

1.1.1. QOF: Query Object Framework (<http://qof.sourceforge.net/>).

The idea behind QOF is to provide a generic framework so that any query can be executed, including queries designed by the end-user. Normally, this is possible only if you use a database that supports SQL, and then only if you deeply embed the database into your application. QOF provides simpler, more natural way to work with objects.

1.1.2. Pilot-QOF: Querying Palm databases as objects (<http://pilot-qof.sourceforge.net/>).

Pilot-QOF converts Palm databases to queryable objects. A Palm database record is an instance of the object, called an Entity. Objects are collated to form one data source, called books, that can contain any number of different objects - depending on the application running QOF. Pilot-QOF writes selected palm data to QSF XML offline storage and running SQL-type queries on the live data or XML file.

Pilot-QOF currently supports the four main applications (Expenses, Contacts, Calendar and ToDo) and work is now progressing to support other free software Palm databases like PCash and FreeCoins.

1.1.3. QOF-gen: QOF Object Generator (<http://qof-gen.sourceforge.net/>).

Generating new objects for the Query Object Framework is repetitive, tedious and time consuming.

qof-generator automates this process in PHP or perl to build a working test program, linked against QOF.

1.1.4. Data Freedom: Liberate your data from the application (<http://www.data-freedom.org/>).

Data-freedom encourages the development of implementation architectures and conceptual foundations to synchronise arbitrary tree-structured data.

The goal is to encourage generic data handling using extensible mechanisms that are inter-related to allow free exchange of data between disparate applications, systems, architectures and platforms.

Each implementation is free to choose the most suitable method of providing data and receiving data. All that is required is that the method chosen is as open, extensible and generic as possible. Naturally, the eXtensible Markup Language - XML - features strongly.

1.1.5. What's a QofBook?

A QOF Book is a dataset within a QOF framework. It provides a single handle through which all the various collections of entities can be found. In particular, given only the type of the entity, the collection can be found. Books also provide the 'natural' place to working with a storage backend, as a book can encapsulate everything held in storage. It is therefore the obvious start point for any data merge in QOF.

1.1.6. QSF - QOF Serialization Format.

An XML serialization format for QOF - i.e. it lays out a QOF object in a series of XML tags.

QSF object files will contain user data and are to be exported from QOF applications under user control or they can be hand-edited and can be used as an export or offline storage format for QOF applications (although long term storage may be best performed using separate (non-XML) methods, depending on the application).

Anyone is free to create their own QSF objects, subject to the GNU GPL. It is intended that QSF can be used as the flexible, open and free format for QOF data - providing all that is lacking from a typical CSV export with all the validation benefits of XML and the complex data handling of QOF. The QSF object and map formats remain under the GNU GPL licence and QSF is free software.

1.2. Other versions of this documentation.

This documentation created using DocBook <http://www.docbook.org/tdg/en/html/docbook.html> (<http://www.docbook.org/tdg/en/html/docbook.html>). This allows other versions to be created from the one source.

The permanent URL for the HTML version of this document is: <http://code.neil.williamsleesmill.me.uk/> (<http://code.neil.williamsleesmill.me.uk/>).

So far, there is a PDF version: http://code.neil.williamsleesmill.me.uk/qof_book_merge.pdf (http://code.neil.williamsleesmill.me.uk/qof_book_merge.pdf).

Also a single HTML file ([qof_book_merge.html](#)).

On request, TXT, TEXI, LaTeX, DVI, RTF and PostScript versions can be generated, or a HTML version with all pages in one file. A tarball of this DocBook and associated XML files is also available:

http://code.neil.williamsleesmill.me.uk/qof_book_merge.tar.gz
(http://code.neil.williamsleesmill.me.uk/qof_book_merge.tar.gz)

Chapter 2. Background

2.1. What kind of merge?

I decided on a rule based merge - building one object in memory that contained all the essential data to run the comparison and store the result. Certain comparisons cannot be resolved automatically and the user needs to be offered a choice of solutions. Each rule that gave an ambiguous comparison result could then be presented to the user when all the comparisons had been done. This allows the user to see a known number of reports and monitor their progress - the idea of presenting the user with a seemingly never-ending list of reports by trying to compare, resolve and merge each object individually was not appealing. When all necessary results have been resolved, the code would then prepare to merge the data - leaving the user the opportunity to abort the merge at any point prior to the final commit.

Each rule would describe one entity - a single object of a specific QOF type containing known QOF parameters, discrete from all other objects of the same type in the QofBook. Rules would then be built into a single ruleset that can be queried, updated and provide the data for the final commit.

2.2. How to identify one entity.

QOF uses a unique identifier to identify single entities, independently of the contained data or object type; the Globally Unique ID - GUID. Used to provide a way to uniquely identify some thing. A GUID is a unique, cryptographically random 128-bit value. The identifier is so random that it is safe to assume that there is no other such item on the planet Earth, and indeed, not even in the Galaxy or beyond.

QOF GUID's can be used independently of any other subsystem in QOF. In particular, they do not require the use of other parts of the object subsystem.

2.3. The requirements for a merge operation.

The purpose of the code was described on the GnuCash development mailing list as:

"The bulk of the task is the merging. qof_book_merge needs to determine what data in the import maps to the existing data, what data is new, and what data is a duplication. There's been a good deal of work to do this with Transactions, but not with Accounts, or any of the businessfeatures."

qof_book_merge needs to determine for each object in import if it's:

```
"the same"           (guids match)
"maybe the same"    (guids don't match but something else matches,
                    like maybe the account name or invoice owner/date)
```

```
"new"                ("clearly" new)
```

What we're testing here is if the object refers to the same semantic concept. e.g. there is a semantic concept of the top-level Asset account. But if I'm merging your account tree into my account tree the guids will differ, but semantically they are the same. Hence, these accounts are "maybe the same". qof_book_merge probably needs user intervention here to properly map the "maybe the same" objects.

If objects are "the same" or "maybe the same" then qof_book_merge might need to determine whether they contain different data. For example, if I import my OWN account tree again (say, merging a backup file), I might have changed a description in an account, or I might have changed a transaction.

qof_book_merge also needs to keep all the references correct.

Finally, qof_book_merge must allow plug in rules per object type.

Chapter 3. Source Documentation

3.1. Doxygen Documentation.

3.1.1. qof_book_merge Doxygen documentation (doxygen/group__BookMerge.html).

The source code documentation is generated automatically using Doxygen.

3.1.2. QSF Doxygen documentation (doxygen/group__QSF.html).

The source code documentation is generated automatically using Doxygen.

3.1.3. QOF Doxygen documentation (<http://qof.sourceforge.net/doxy/>).

<http://qof.sourceforge.net/doxy/>

3.1.4. GnuCash Doxygen documentation (<http://cvs.gnucash.org/docs/HEAD/>).

3.1.5. GnuCash Gnome2 port Doxygen documentation (gnome2/).

The gnucash-gnome2-dev branch has all the QSF code and is hosted here for ease of use.

3.2. Other general documentation

3.2.1. GnuCash Design documentation

The design documentation is out of date in some areas. Always refer to the Doxygen output for details of any specific API.

single HTML file (gnucash-design.html) generated using texi2html.

individual HTML files (texi/gnucash-design.html)

PDF file (texi/gnucash-design.pdf)

DVI file (texi/gnucash-design.dvi)

3.2.2. GnuCash Tutorial and Concepts Guide

single HTML file (gnucash-guide.html) from the gnucash-docs package.

PDF file (gnucash-guide.pdf) from the gnucash-docs package.

3.2.3. GnuCash Help Manual

single HTML file (gnucash-help.html) from the gnucash-docs package.

PDF file (gnucash-help.pdf) from the gnucash-docs package.

3.2.4. KVP Values used By GnuCash

The use of keys in the key-value pair system (kvp_doc.txt) used by the GnuCash Application.

Other text files in GnuCash. Text files describing design principles and giving overviews of the main GnuCash principles have been folded into the Doxygen output.

The rest of this documentation describes the design behind the code, reasons behind the current design and how the code can be utilised.

3.3. qof_book_merge Source Code

For the latest versions, see the CVS. QOF CVS: <http://cvs.sourceforge.net/viewcvs.py/qof/qof/>
(<http://cvs.sourceforge.net/viewcvs.py/qof/qof/>)

GnuCash CVS: <http://cvs.gnucash.org/cgi-bin/cvsweb.cgi/gnucash/src/engine/>
(<http://cvs.gnucash.org/cgi-bin/cvsweb.cgi/gnucash/src/engine/>)

- qof_book_merge.c
- qof_book_merge.h

- **Test and example programs.** This test file shows an implementation of the qof_book_merge API:

test-book-merge.c

This example files shows an implementation using GnuCash objects:

example-gncBookMerge.c

- The example now uses external XML files as data sources:

druid-business.gnc

druid-simple.gnc

These are simple GnuCash files, created using the New File druid.

- A GUI example is also under development - to allow the New Account Hierarchy druid to be run over an existing file.
- druid-merge.c - small changes required to main-window.c to allow for the menu item to call the druid.

merge.glade - the Glade code to describe the druid resource.

3.4. QSF Source code

For the latest versions, see the CVS. QOF CVS: <http://cvs.sourceforge.net/viewcvs.py/qof/qof/>
(<http://cvs.sourceforge.net/viewcvs.py/qof/qof/>)

qsf-backend.c

qsf-dir.h.in

qsf-xml.c

qsf-xml.h

qsf-xml-map.c

3.4.1. QSF and qof_book_merge tarballs

A tarball was being regularly generated for all qof_book_merge and QSF code. However, all patches except pilot-link are now in the respective CVS trees for each project. This will be the LAST tarball! Once the final v0.12 of pilot-link is released, this patch too will be added to it's project CVS and these tarballs will become unnecessary. As a result, only this final tarball is now hosted. The code in the CVS trees is not finished yet, but the code in the old tarballs is simply too old (and contains too many errors!).

QSF no longer uses md5. For better security, those without GnuPG should use sha1sum to verify the hash below. GnuPG/PGP signatures are signed with 0x28BCB3E3, available from key servers.

Note: These tarballs were NOT incremental, they were generated against the entire CVS HEAD tree for each project as it was when the tree was updated from CVS prior to the generation of the patch. If you try to apply these patches to a later CVS tree, you can expect problems. Now that the code is in the project CVS, please use that code. There will be no further tarballs hosted here.

1. **Wed Feb 2 11:43:10 UTC 2005.** This is the LAST tarball, all future code will be committed to the CVS tree of the project concerned. Development is far from complete, but it is at a stage that allows each tree to build a test environment and operate the basic code without errors.

Tarball: qof-qsf02-02-05.tar.gz (29K)

Signature: qof-qsf02-02-05.tar.gz.asc

Checksum: qof-qsf02-02-05.tar.gz.sha1sum (qof-qsf02-02-05.tar.gz.sha1sum
)97ac9e49d6ea5429f7d3d5ca6fe662a47fa05c3b

Chapter 4. Creating GnuCash Invoices using a Palm PDA

4.1. Beginnings

This is where this entire saga began. I needed to collate data from my Palm PDA (using Palm OS 5) and create an invoice in GnuCash with the minimum of extra typing. The PDA databases contain details of the hours to be billed (Calendar), incidental expenses incurred during the invoiced work (Expenses) and contact details for the client and workplace (Contacts). By adding a user-configurable set of defaults, the other essential elements of an invoice are ready, item type (hours, material etc.), accounts to use for each type and the account to use for posting the invoice, if the user chooses to let the program do the post. (Not advisable until the user is fully familiar with the program!) This area is now under development.

The mechanism was not considered at that stage. I didn't really care how the data would be passed from PDA to GnuCash and considered XML/XSLT, CSV and QIF before agreeing to do the hard bit of writing the generic merge routine. The benefits are huge but the work has been long and a steep learning curve. The main reason for writing this documentation is to see if it can help others join the GnuCash team. Since starting `qof_book_merge`, the Doxygen source code documentation has been rendered directly from the CVS code on a nightly basis. These pages are therefore designed to cover the other questions; where did I start, how can new developers get under the skin of GnuCash quickly and without hassling other developers as I did!

- Initial discussions (<https://lists.gnucash.org/pipermail/gnucash-devel/2004-April/011219.html>) about Invoices from a PDA.
- Discussions about a merge (<https://lists.gnucash.org/pipermail/gnucash-devel/2004-April/011250.html>) function for GNCBook*, a synonym for QofBook.
- My first mistakes (<https://lists.gnucash.org/pipermail/gnucash-devel/2004-April/011255.html>).
- Learning QOF (<https://lists.gnucash.org/pipermail/gnucash-devel/2004-May/011346.html>).
- Function pointer problems (<https://lists.gnucash.org/pipermail/gnucash-devel/2004-June/011607.html>).
- QofSetterFunc (<https://lists.gnucash.org/pipermail/gnucash-devel/2004-July/011833.html>) changes to aid `qof_book_merge`.

4.2. Getting into GnuCash / QOF

This section is meant to be helpful and useful - nothing here is intended as a rule or to be interpreted as the only way to do things. It is not meant as patronising either, I made mistakes when starting this mini-project because this kind of project was so unfamiliar to me.

The critical areas for me are highlighted above. The up to date source documentation provided by Doxygen is always the first place to look for answers. General overviews and design texts do exist but you need to use and understand the program before most of this will make any sense. All documentation must choose a starting level and mine will mimic my own start:

- Experience of C++ more than C - able to write and compile console programs, design C++ classes and implement inheritance. Includes some gaps in C knowledge, specifically function pointers.
- A real need, not just curiosity. GnuCash is a large program and trying to learn it all is more than most people need. To pick up GnuCash, QOF and the vagaries of using only C, I can only recommend a clear objective that relates to an existing GnuCash objective - you can't expect a lot of assistance from GnuCash developers if you are not working on a project that is seen to be useful to the rest of GnuCash.
- Experience of GnuCash itself, preferably in situations where the figures are real and the calculations matter. A tax return focuses the mind very well.
- Time - always more than you had originally planned.
- Patience. If you don't have much patience, you probably wouldn't still be reading!
- <http://www.gnucash.org/>. Read the "Developer Information" section and join the gnucash-devel mailing list.
- Make sure what you want GnuCash / QOF to do coincides with what the rest of the developers consider useful. Explain your ideas and like my experience above, be prepared to:
 1. Alter your own plans substantially.
 2. Provide good reasons for not altering the bits you want to keep.
 3. Acknowledge your strengths *AND* weaknesses and ask for help.
 4. Keep to what you know you can achieve - don't be tempted into taking on a larger project than you can manage.
 5. Keep it simple, concise and informative - other developers are busy too.
- Download the CVS HEAD and build it.
- Spend lots of time understanding the current Doxygen output.

There are lots of sites that have helped me understand QOF, GnuCash, automake, doxygen, glib, libxml and Glade. In the hope that some might be useful . . .

- <http://qof.sourceforge.net/>
- <http://www.gnu.org/manual/manual.html> GNU Manuals Online, autoconf, autogen and automake.
- <http://developer.gnome.org/doc/API/2.0/glib/index.html> GLib Reference Manual - GLib is a general-purpose utility library, which provides many useful data types, macros, type conversions, string utilities, file utilities, a main loop abstraction, and so on. It works on many UNIX-like platforms, Windows, OS/2 and BeOS. GLib is released under the GNU Library General Public License (GNU LGPL).
- QOF does *NOT* require Gtk or any GUI elements. These are provided to aid GnuCash development.

- <http://developer.gnome.org/doc/API/2.0/gtk/index.html> GTK+ Reference Manual
- <http://glade.gnome.org/> Glade is a free user interface builder for GTK+ and GNOME. It is released under the GNU General Public License (GPL).
- <http://www.yolinux.com/TUTORIALS/GnomeLibXml2.html> YoLinux Tutorial - XML and Gnome libXML2
- <http://www.w3.org/TR/xmlschema-0/primer.html#Intro> XML Schema Part 0: Primer Second Edition, Introduction.
- <http://www.xmlsoft.org/> The XML C parser and toolkit of Gnome.
- <http://www.eskimo.com/~scs/cclass/int/sx10a.html> Declaring, Assigning, and Using Function Pointers.

4.2.1. Why the GnuCash File QofBackend needs changing

The current XML data store format used within GnuCash has served well but is slated for removal in a future release. XML is better suited to data interchange than permanent data storage and QSF represents the best way to continue the use of XML for interchange between QOF programs using existing (and trusted) code. The current GnuCash XML format is too hierarchical for generic QOF use and uses non QOF structures to write out the file.

4.2.2. Tips on debugging within GnuCash

- Use `src/engine/gnc-trace.c gnc_log_modules` to your advantage. The version in CVS tends to favour the area of largest development need. This will produce debugging logs that concentrate on those sections of the codebase. If you are working on another section, change the log levels to suit your needs and compile. The sequence is laid out in `gncLogLevel` in the header file:
`src/engine/gnc-trace.h - GNC_LOG_WARNING` will ignore all but the most serious errors. To see more of what is going on, change the log level for the module to a more inclusive setting like `GNC_LOG_DEBUG`. Keep the comments intact to identify which line affects which module! The output is written, by default, to `/tmp/gnucash.trace` although this can also be changed in `gnc-trace`. Remember NOT to commit these changes back to CVS!
- Use `ENTER (" ");`, `LEAVE (" ");` and `PINFO(" ");` in your code to create your own log messages like:
`Info: qof_session_load(): new book=0x84266b8`

Note that the function name is added automatically. To output variable values, use `printf()` syntax. e.g. the command to generate that log message is:

```
PINFO ("new book=%p", newbook);
```

Try to avoid using log functions in callback routines or iterative loops - at least in the code committed to CVS. The log output needs to be clear for others, not bloated with hundreds of identical entries.

- Use the comments in HACKING to link into GDB or Valgrind. You can also use gdb by using two terminal windows:

```
term1 gnuccash/bin>$ ./gnucash
term2 ~>$ ps waux | grep guile
term2 ~>$ gdb
gdb> attach 1234
gdb> continue
```

Replace 1234 with the process ID of the guile process identified by `ps`. GDB will then load a lot of debugging symbols before you can enter the `continue` command. The advantage of this method is that the initialisation of GnuCash does not operate within the supervision of GDB and therefore runs at normal speed. You can use GnuCash to create a test environment, load the relevant files etc., without the penalty of `gdb` inspecting every call until you need it. I use this method to find the cause of segmentation faults within the program more quickly than using `gdb` from the beginning.

- GnuCash saves user space settings in `~/.gnucash/` and in `~/.gnome/GnuCash` - if you use the `gnucash-gnome2-dev` branch, it uses `~/.gnome2/GnuCash`. General configuration settings go in `~/.gnucash`, file history and others go in `~/.gnome` or `~/.gnome2`

4.3. Building QOF onto pilot-link

QOF and pilot-link. Merging QofBook's within GnuCash is one thing, to achieve my initial goal, I have to get the PDA to talk in QOF as well. I've now begun the work of QOF enabling pilot-link by wrapping existing structures in a QOF environment.

- `qof-expenses.h`
- `qof-expenses.c`
- `qof-address.h`
- `qof-address.c`
- `qof-datebook.c`
- `qof-datebook.h`

The principle is to wrap the existing struct inside a new QOF-compatible struct. The essential addition is `QofInstance inst`; then define a `get` and a `set` routine for each parameter to be supported by QOF. In each routine, cast the address of the wrapped struct to a suitable pointer and use that to store or retrieve the real data via QOF.

Localtime is determined by the process running QOF.: There are also issues of dates and times between QOF and pilot-link. Pilot-link usually works on localtime - the Palm only sets the timezone in Preferences, it doesn't use it internally - so the link to QOF will have to make this crucial assumption. When converting Palm localtimes into QOF universal (UTC) times, the time will be converted by QOF using the localtime of the QOF process.

In real terms, QOF will be running as a framework around pilot-link so the offline storage, (QSF XML), will be written by pilot-link. Pilot-link itself does not directly support users who cross timezones with their Palm - it may be wise that users are reminded that if timezone issues are important to their DateBook or Expenses data, the user should take the sensible precaution of changing the Palm Preferences to use the same timezone as the computer running pilot-link. Once in QOF, all dates and times are UTC so the QSF XML files can be freely exchanged across timezones without complication.

QOF has now been configured to build as a framework around pilot-link when compiled from source. Using the `--enable-qof=<path>` option to `./autogen.sh`, the QOF code is built into `libpilotqof.so` and a `pilot_qof` executable will be installed. Full documentation for `pilot_qof` is included in this documentation as manpages for `pilot_qof`.

The ability to import QSF data directly into the Palm will follow at a later date.

4.3.1. Converting existing objects to QOF

Existing objects need to describe themselves to QOF and this can be achieved by using QOF as an external framework around the existing structs. This is how pilot-link is now able to use QOF. The ToDo database will be used as an example. Not all the steps will be shown here, see the source for full information.

1. Create a new struct - this will be used to contain your existing structs as well as adding the necessary `QofInstance` instance that will provide the GUID that is fundamental to QOF. You can add other variables if you like.

```
typedef struct {
    QofInstance inst;
    ToDo_t wrap;
}qof_todo;
```

2. Identify the parameters to use with QOF by comparing your struct variables with the basic QOF data types:

- `QOF_TYPE_STRING` - suitable for `char*`, `const char*`, `gchar*`, `GString->str` and enumerators (see note).

```
char* todo_getNote(qof_todo*);
void todo_setNote(qof_todo*, const char*);
```

These would be represented in the `QofClass` parameter list as:

```
{TODO_NOTE, QOF_TYPE_STRING, (QofAccessFunc)todo_getNote, (QofSetterFunc)todo_setNote},
```

- `QOF_TYPE_GUID` - suitable for the QOF GUID itself as well as use as references to other QOF objects.

- QOF_TYPE_BOOLEAN - suitable for gboolean and other boolean values. Note that QSF (the XML file backend for QOF) uses a single value for each boolean value: false or true - lowercase. If you set or get a QOF_TYPE_BOOLEAN, for conversion to or from a string etc., FALSE, TRUE, 1, 0 etc. may cause errors. A QOF_TYPE_DEBCRED is also available for financial purposes - it is essentially a boolean with values for debit or credit.
- QOF_TYPE_NUMERIC - suitable for all non-integer numeric values - numerics provide rational number handling with rounding error control. See the QOF documentation for more information. Routines exist to convert to and from double and string. QOF_TYPE_DOUBLE exists but has essentially been replaced by numeric.
- QOF_TYPE_DATE - suitable for all date values. All QOF times are in UTC and are converted to use the full date and time (even if you only need the date or just the time). The example shows how to convert a QOF Timespec into a struct tm used by the underlying struct variable.

```
Timespec todo_getDateDue(qof_todo *t)
{
    ToDo_t *qt;
    Timespec ts;

    ts.tv_sec = 0;
    ts.tv_nsec = 0;
    g_return_val_if_fail(t != NULL, ts);
    qt = &t->wrap;
    timespecFromTime_t(&ts, mktime(&qt->due));
    return ts;
}

void todo_setDateDue(qof_todo *t, Timespec ts)
{
    ToDo_t *qt;
    time_t s;

    g_return_if_fail(t != NULL);
    qt = &t->wrap;
    s = timespecToTime_t(ts);
    qt->due = *localtime(&s);
}

```

The QofClass parameter for these functions would be:

```
{TODO_DATE, QOF_TYPE_DATE, (QofAccessFunc)todo_getDateDue, (QofSetterFunc)todo_setDateDue}
```

- QOF_TYPE_INT32 and QOF_TYPE_INT64 - integer values.

Guidance on enumerators.: Use strings instead of numerical enumerators for values that will appear in the QSF XML. This increases readability of the XML and guards against future changes in the enumerator itself. Configure enumerator parameters to be set and fetched as strings - clear and concise descriptions of the meaning of the enumerator value - and do the conversion inside the object.

For example:

```
<string type="expense_type">Airfare</string>
```

is more easily understood than:

```
<gint32 type="expense_type">0</gint32>.
```

- QOF_TYPE_CHAR - suitable for single character values. e.g. GnuCash uses char for the flag that indicates whether a transaction has been fully reconciled.
 - QOF_TYPE_KVP - Key:Value pairs. Sets of associations between character strings (keys) and KvpValue structures. A KvpValue is a union with possible types enumerated in the KvpValueType enum, and includes, among other things, ints, doubles, strings, guid's, lists, time and numeric values. KvpValues may also be other frames, so KVP is inherently heirarchical.
3. Now decide on how these parameters should be used. The simple decision is whether a parameter should be calculated or set. For example, an account object has a description string that would be set when creating a new account. However, the balance of that account cannot be set, it is calculated from the transactions held within the account. Values that should be set need a set function defined in QOF, these use QofSetterFunc prototypes, as well as a get function (QofAccessFunc). Calculated values should only be provided with a QofAccessFunc - a get function. Use the examples of the pilot-link structs where necessary.
 4. Create the functions that will provide the link between QOF and your existing functions. You'll need a get function for each parameter and a set function for relevant parameters. These functions can handle any necessary conversions, e.g. in pilot-link, dates are converted to UTC, doubles and floats converted to numerics.

```
void exp_setAmount(qof_exp *e, gnc_numeric h) {
    Expense_t *qe;
    double exp_d;

    g_return_if_fail(e != NULL);
    qe = &e->wrap;
    exp_d = gnc_numeric_to_double(h);
    qe->amount = g_strdup_printf("%f", exp_d);
}

gnc_numeric exp_getAmount(qof_exp *e) {
    Expense_t *qe;
    gnc_numeric amount;
    double pi_amount;

    amount = gnc_numeric_zero();
    g_return_val_if_fail(e != NULL, amount);
    qe = &e->wrap;
    /* floating point as a string converts to gnc_numeric */
    pi_amount = strtod(qe->amount, NULL);
    amount = double_to_gnc_numeric(pi_amount, 0, GNC_HOW_DENOM_EXACT);
    if(gnc_numeric_check(amount) == GNC_ERROR_OK) {
        return amount;
    }
    return gnc_numeric_zero();
}
```

5. Use the pilot-link examples to create a QofObject definition and a create function that properly initialises each parameter.

The object type has strict syntax

Although you can have spaces and punctuation in the type_label (which is intended to be descriptive), the e_type cannot include spaces OR hyphens. You can use underscores in the e_type. Remember also that the e_type may be visible as well as the description and is more often used to identify an object type - users may have to type the object type in certain applications, so make it usable!

valid types include: pilotAddress, pilot_address, pilotaddress

Some of the types that will fail: pilot address, pilot-address, pilot.address, pilot/address
pilot=address, pilot(address)

Similarly, a QofClass set of parameters that sets out the parameter names, the parameter type and each of the get and set functions. If no set function exists, set the final value to NULL. Each object definition should include two special cases:

```
{QOF_PARAM_BOOK,QOF_ID_BOOK,(QofAccessFunc)qof_instance_get_book,NULL},  
{QOF_PARAM_GUID,QOF_TYPE_GUID,(QofAccessFunc)qof_instance_get_guid,NULL},
```

These are part of how the object links into the framework and all objects need to include them. Note that you should not expect to set the book or GUID - this is done by QOF when merging books and/or loading from files.

4.4. pilot-qof manpages

pilot-qof

Name

pilot-qof — Interface with the optional QOF external framework.

Section

pilot-link: Conduits

synopsis

```
pilot-qof [--version] | [-q|--quiet] | [-?|--help] | [--usage] | [-o|--cold-query filename] |  
[-a|--hot-query] [-l|--list] | [-p|--port <port>] [-c|--category string] [-d|--database  
string] [-t|--date string] [-e|--exclude string] [-s|--sql string] [-f|--sql-file  
filename] [-w|--write filename]
```

Description

This is data abstraction layer of the pilot-link suite.

Commands

Use exactly one of -oal.

`-o, --cold-query filename`

Query the offline storage in `filename` without requiring a HotSync.

`-a, --hot-query`

Activate/HotSync and query the Palm, ignoring the offline storage.

`-l, --list`

Lists all databases supported by the current QOF framework. Any options are ignored.

Options

pilot-qof options

Use `-cdte`, optionally with either `-s` or `-f`, followed by `-w`.

`-c, --category string`

Shorthand to only query objects that are set to the specified category.

`-d, --database string`

Shorthand to only query objects within one specific supported database.

`-t, --date string`

Shorthand to only query objects that contain the specified date. Specify dates using YY-MM-DD, YY-MM or just YY. Single digits can omit the leading zero, e.g. 04-12-1, 1st December 2004 - years less than 100 are assumed to be in the 21st century. Years prior to 2000 can be specified as YYYY-MM-DD etc. This value is taken as a range, 05-03-01 includes all times between Tue Mar 1 00:00:00 UTC 2005 and Tue Mar 1 23:59:59 UTC 2005. 05-03 includes all dates and times between Tue Mar 1 00:00:00 UTC 2005 and Thu Mar 31 23:59:59 UTC 2005. 05 includes all dates and times in 2005.

`-e, --exclude string`

Shorthand to exclude a supported database from the query.

`-s, --sql string`

Specify a SQL query on the command line.

`-f, --sql-file filename`

Specify one or more SQL queries contained in a file.

`-w, --write filename`

Write the results of any query to the file.

conduit options

`-p, --port <port>`

Use device file `port` to communicate with the Palm handheld. If this is not specified, *pilot-qof* will look for the `$PILOTPORT` environment variable. If neither are found or supplied, *pilot-qof* will print the usage information.

`-q, --quiet`

Suppress 'Hit HotSync button' message

`-v, --version`

Display version of *pilot-qof*.

help options

`-h, --help`

Display the help synopsis for *pilot-qof*.

`--usage`

Display a brief usage message and exit without connecting.

Examples

List all currently supported databases.

```
pilot-qof -l
```

Print all objects in all supported databases that have the category **Business** using offline storage, to STDOUT.

```
pilot-qof -o offline.xml -c "Business"
```

HotSync to the Palm and create the offline storage (all databases, all objects). This will overwrite the existing file, if any.

```
pilot-qof -a -w offline.xml
```

Command line SQL query of offline storage.

```
pilot-qof -o offline.xml -s "SELECT * from pilot-address where city = 'London%';"
```

Author

pilot-qof was written by Neil Williams <linux@codehelp.co.uk>.

This manual page was written by Neil Williams <linux@codehelp.co.uk>

Reporting Bugs

Report bugs at: <http://bugs.pilot-link.org>

Copyright

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.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

See Also

pilot-link (7) *pilot-qof* (5) *pilot-qof* (7)

<http://code.neil.williamsleesmill.me.uk/> <http://qof.sourceforge.net/> <http://sourceforge.net/projects/qof/>
<http://www.gnucash.org/>

pilot-qof

Name

pilot-qof — Query Object Framework Serialisation Format: QSF

Section

pilot-link: Reference

What is QSF?

QSF - QOF Serialization Format - lays out a QOF object in a series of XML tags. The format will consist of two component formats:

- qof-qsf for the QSF object data and
- qsf-map to map sets of QSF objects between QOF applications.

QSF object files will contain user data and are to be exported from QOF applications under user control or they can be hand-edited. QSF maps contain application data and can be created by application developers from application source code. Tools may be created later to generate maps interactively but maps require application support as well as an understanding of QOF objects in the import and output applications and are intended to remain within the scope of application developers rather than users.

A QSF file written by one QOF application will need an appropriate QSF map before the data can be accessed by a different application using QOF. Any QSF objects that are not defined in the map will be ignored. QSF files written and accessed by the same application can use maps if necessary or can simply import the QSF data as a whole.

Author

pilot-qof was written by Neil Williams <linux@codehelp.co.uk>.

This manual page was written by Neil Williams <linux@codehelp.co.uk>

Reporting Bugs

Report bugs at: <http://bugzilla.gnome.org/> (GnuCash) or on the gnuCash-devel mailing list: <https://lists.gnucash.org/mailman/listinfo/gnucash-devel> or direct to Neil Williams <linux@codehelp.co.uk>. Please do NOT report bugs in QSF to pilot-link.

Copyright

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.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

See Also

pilot-link (7) *pilot-qof* (1) *pilot-qof* (7)

<http://code.neil.williamsleesmill.me.uk/> <http://qof.sourceforge.net/> <http://sourceforge.net/projects/qof/>
<http://www.gnucash.org/>

pilot-qof

Name

`pilot-qof` — Installing the optional QOF external framework for `pilot-link`.

Section

`pilot-link`: Reference

What is QOF?

QOF provides a set of C language utilities for performing generic structured complex queries on a set of data held by a set of C/C++ objects. This framework allows programmers to add query support to their applications without having to hook into a SQL Database.

QOF provides the framework, the new QOF XML backend (QSF) provides the off-line storage and a format for a data stream between different applications, including converting QOF objects.

What does QOF have to do with pilot-link?

QOF frees the data from the application, allowing queries, data mining, abstraction and data interchange without having to recompile the original application, let alone program the filters. Central to QOF are the ideas of databases and objects. A Palm database is an object in QOF. A Palm database record is an instance of the object, called an Entity. QOF then collates all supported objects to form one data source that can be queried. QOF allows the programmer to easily find all other instances of any one type, just by having one instance of that type. Objects are collected in books and a book can contain any number of different objects - depending on the application running QOF. The book is the central data source within QOF and a book can be queried or written out to XML using the QOF Serialisation Format: QSF. See *pilot-qof* (5).

In real terms, QOF will be running as a framework around pilot-link so the offline storage, (QSF XML), will be written by pilot-link. Pilot-link itself does not directly support users who cross timezones with their Palm - it may be wise that users are reminded that if timezone issues are important to their DateBook or Expenses data, the user should take the sensible precaution of changing the Palm Preferences to use the same timezone as the computer running pilot-qof. Once in QOF, all dates and times are UTC so the QSF XML files can be freely exchanged across timezones without complication.

QOF has now been configured to build as a framework around pilot-link when compiled from source. Download the QOF CVS source from <http://sourceforge.net/projects/qof/> - you can choose to install in /opt/ using the `--enable-opt-style-install` option to `autogen.sh` and then use a prefix, `--prefix=/opt/qof/`. Once QOF is built and installed, download the pilot-link source and use the `--enable-qof=<path>` option to `./autogen.sh`. The QOF code will be built into `libpilotqof.so` and a `pilot-qof` executable will be installed.

A little personal story

This all started when I needed to collate data from my Palm PDA (using Palm OS 5) and create an invoice in GnuCash with the minimum of extra typing. The PDA databases contain details of the hours to be billed (Calendar), incidental expenses incurred during the invoiced work (Expenses) and contact details for the client and workplace (Contacts). By adding a user-configurable set of defaults, the other essential elements of an invoice are ready, item type (hours, material etc.), accounts to use for each type and the account to use for posting the invoice later.

Author

pilot-qof was written by Neil Williams <linux@codehelp.co.uk>.

This manual page was written by Neil Williams <linux@codehelp.co.uk>

Reporting Bugs

Report bugs at: <http://bugzilla.gnome.org/> (GnuCash) or on the gnuCash-devel mailing list: <https://lists.gnucash.org/mailman/listinfo/gnucash-devel> or direct to Neil Williams <linux@codehelp.co.uk>. Please do NOT report bugs in QSF to pilot-link.

Copyright

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.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

See Also

pilot-link (7) *pilot-qof* (1) *pilot-qof* (5)

<http://code.neil.williamsleesmill.me.uk/> <http://qof.sourceforge.net/> <http://sourceforge.net/projects/qof/>
<http://www.gnucash.org/>

Chapter 5. QSF - QOF Serialization Format.

5.1. What is QSF?

After some discussion (<http://lists.gnucash.org/pipermail/gnucash-devel/2004-October/012201.html>) on the GnuCash lists, an XML serialization format has been created - i.e. it lays out a QOF object in a series of XML tags. The format will be QSF - QOF Serialization Format and will consist of two component formats:

1. `qof-qsf` for the QSF object data and
2. `qsf-map` to map sets of QSF objects between QOF applications.

QSF object files will contain user data and are to be exported from QOF applications under user control or they can be hand-edited. QSF maps contain application data and can be created by application developers from application source code. Tools may be created later to generate maps interactively but maps require application support as well as an understanding of QOF objects in the import and output applications and are intended to remain within the scope of application developers rather than users.

A QSF file written by one QOF application will need an appropriate QSF map before the data can be accessed by a different application using QOF. Any QSF objects that are not defined in the map will be ignored. QSF files written and accessed by the same application can use maps if necessary or can simply import the QSF data as a whole.

I've created a Schema that covers all QSF files - it's generic and uses tags based only on the QOF data types, so until new types are defined, it should cope with all QOF XML needs.

5.1.1. Features of QSF

- QSF itself is now being built into the QOF library for use with pilot-link to allow Palm objects to be described in QOF, written to XML as QSF and imported directly into GnuCash and other QOF-compliant applications. As a generic format, it does not depend on any pre-defined objects - as the current GnuCash XML format depends on AccountGroup. Instead, QSF is a simple container for all QOF objects.
- QSF grew from the `qof_book_merge` code base and uses the `qof_book_merge` code that is now part of QOF. Any `QofBook` generated by QSF still needs to be merged into the existing application data using `qof_book_merge`.
- QSF can be used as an export or offline storage format for QOF applications (although long term storage may be best performed using separate (non-XML) methods, depending on the application).
- QSF is designed to cope with partial `QofBooks` at the `QofObject` level. There is no requirement for specific objects to always be defined, as long as each QOF object specified is fully defined, no orphan or missing parameters are allowed.

- Work is continuing on supporting QSF in GnuCash and QOF. QSF is a very open format - the majority of the work will be in standardising object types and creating maps that convert between objects. Applications that read QSF should ignore any objects that do not match the available maps and warn the user about missing data. The example file outlines three objects defined in pilot-link and in the pilot-qsf-GnuCashInvoice.xml QSF map to retrieve the data from pilot-link to create a GnuCash invoice. The range of QSF objects and QSF maps will then be increased from that base.
- Anyone is free to create their own QSF objects, subject to the GNU GPL. It is intended that QSF can be used as the flexible, open and free format for QOF data - providing all that is lacking from a typical CSV export with all the validation benefits of XML and the complex data handling of QOF. The QSF object and map formats remain under the GNU GPL licence and QSF is free software.

5.1.2. Requirements of QSF

- The data is described as genuine QOF objects,
- References are written as the GUID of the parent or child
- The type attribute of the object tag must match an existing QOF object `e_type` string so that the object can be matched, either against the same QOF object or against a QSF map.
- Object parameters are written out in a defined sequence:
 1. All string parameters, any order
 2. All GUID parameters, including references to parent or child objects, in any order.
 3. all boolean parameters, any order
 4. all numeric parameters, any order
 5. all date parameters, any order
 6. all int32 parameters, any order
 7. all int64 parameters, any order
 8. all double parameters, any order
 9. all char parameters, any order
 10. The KVP frame will be supported soon.

Note that KVP is not included in QSF at this stage, but support is being organised.

The example XML file demonstrates this pattern.

- The XML itself is valid, according to the QSF schema. Keep any comments to between the `xml` version and the first `qof-qsf` tag or after the closing `qof-qsf` tag.
- QSF uses `xsd:dateTime` formatting for times. This imposes strict limits on the formatting of time strings in the QSF XML. `xsd:dateTime` follows the ISO 8601 (<http://www.w3.org/TR/NOTE-datetime>) standard in the Coordinated Universal Time (UTC) syntax to be timezone independent. This generates timestamps of the form: `2004-11-29T19:15:34Z` - you

can reproduce the same timestamps with the GNC C Library formatting string

`%Y-%m-%dT%H:%M:%SZ` - remember to use `gmtime()` *NOT* `localtime()`!. From the command line, use the `-u` switch with the date command: `date -u +%Y-%m-%dT%H:%M:%SZ`

- Every QSF file must contain just one book and at least one QOF object but can contain more - including QOF objects from more than one QOF application.
- A QSF file written by one QOF application will need an appropriate QSF map before the data can be accessed by a different application using QOF. Any QSF objects that are not defined in the map will be ignored. QSF files written and accessed by the same application can use maps if necessary or can simply import the QSF data as a whole.

5.1.2.1. Guidance on enumerators.

Wherever possible, use strings instead of numerical enumerators for values that will appear in the QSF XML. This increases readability of the XML and guards against future changes in the enumerator itself. The only practical way to convert a numerical enumerator to a string is within the object. Object developers should configure enumerator parameters to be set and fetched as strings - clear and concise descriptions of the meaning of the enumerator value - and do the conversion inside the object. If you still want to retrieve an enumerator parameter, define a suitable `QofAccessFunc` but leave `QofSetterFunc` as `NULL` - this will prevent the parameter being duplicated. A suitable setter function for use with the enumerator can still exist in the source, just not included in the `QofClass` parameter list.

For example:

```
<string type="expense_type">Airfare</string>
```

is more easily understood than:

```
<gint32 type="expense_type">0</gint32>.
```

5.1.3. Validation of QSF.

All QSF files must be valid XML according to the schema above, but XML validation will *NOT* be the only restriction on QSF data. XML files that validate against the schema will not necessarily be importable into any QOF application because the application needs a qsf-map to understand the objects defined in the QSF. The map is selected according to the object type, expressed in the `type` attribute of the object tag in the XML.

The example file therefore contains `pilot-expenses`, `pilot-datebook` and `pilot-address` objects. If the receiving application only had a map defined for `pilot-expenses`, the other two objects would be ignored and the user warned. Note that any one QSF file can contain data from different objects. If a suitable map exists, there is nothing to prevent one QSF file containing objects from different QOF programs.

5.1.4. QSF examples

- Example XML file (qof-qsf.xml) using pilot-link objects. More objects will be added as support improves.
- Second Example (qsf-gnc.xml) using GnuCash objects. Note that the range of GnuCash objects is limited initially and some parameters are not yet fully supported, notably KVP as well as objects that are currently unsupported by QOF itself, like commodity and pricedb. This QSF example was hand edited (which is why it is small and consists solely of account objects!) and final QSF files written by QOF/GnuCash are likely to contain more parameters for each supported object. e.g. in a qof_book_merge, typical GnuCash Account objects have eight parameters. The actual strings used for the type attributes can be adjusted in the GnuCash source. It may be possible to have an XSLT stylesheet to do more transformations. Note also how references are described as obj:guid with type="reference", type=guid" being reserved for the guid of the object itself. Overall, QSF may add a small overhead to the existing GnuCash XML format but this is not anticipated as a problem - QSF is intended as an intermediary format between applications, not long term storage.
- The GnuCash XML v2 file (childcare.gnc) for the second example QSF above. Note how the QSF file uses the one object tag and obj namespace for all supported objects. The obj namespace tags are standardised to the underlying QOF type for that parameter and the type takes on the descriptive / human readable name for the QOF parameter. Note also the strict sequence in a QSF file - string, GUID, boolean, numeric, date, int: (SGBNDI). No neat acronym but still important!
- qsf-object.xsd.xml - The QSF Object Schema.

qsf-map.xsd.xml - The QSF Map Schema.

A simple parser (qsf-xml.c) written in C is also available, based on a tutorial for libxml (<http://www.yolinux.com/TUTORIALS/GnomeLibXml2.html>). This has been adapted to remove all dependencies on Gnome or Gtk and enhanced to cope with all QOF data types currently supported by QSF.

A reader utility in C using the Sixtp parser used by GnuCash was tested but proved inadequate for the QSF format - it would appear not to be possible for a single tag to be both parsed and contain children. The test file that was produced contained a large amount of unnecessary code and the parser itself added unnecessary bulk to the routines. QSF - as implemented in QOF and GnuCash - will therefore use only direct calls to libxml and QOF itself.

To compile `qsf-xml.c`, use the command:

```
gcc -g -Wall `xml2-config --cflags --libs` -o qsf-xml qsf-xml.c
```

The default looks for the qof-qsf.xml file in the current directory. It is also capable of reading the qsf-gnc.xml file above. Schema validation has also been added to the test routine - it looks for qsd.xsd in the current directory. When finished, it will look in `install_dir/share/qsf/`.

5.2. Mapping QSF objects between QOF applications

5.2.1. The QSF Map file

qsf-map is an XML format used to map a series of QOF objects from one application to a series of different QOF objects in another application. The map works by defining the steps of the calculations involved, including conditionals, pattern matching, defaults, simple assignments and arithmetic operators. The example snippet below does not include all necessary data for a working map for reasons of brevity but does try to include at least one example of each QSF map tag and construct.

```

1: <definition qof_version="3">
2:   <define e_type="qof-expenses">Pilot-link QOF expenses</define>
3:   <define e_type="qof_datebook">Pilot-link QOF datebook</define>
4:   <define e_type="gncInvoice">Invoice</define>
5:   <define e_type="Trans">Transaction</define>
6:   <define e_type="gncEntry">Order/Invoice/Bill Entry</define>
7:   <default name="mileage_rate" type="numeric" value="28/100"/>
8: </definition>
9: <object type="gncEntry">
10:  <calculate type="string" value="action">
11:    <if type="qof-expenses">
12:      <set>Material</set>
13:    </if>
14:    <else type="qof-datebook">
15:      <set>Hours</set>
16:    </else>
17:  </calculate>
18:  <calculate type="numeric" value="iprice">
19:    <if type="string" value="expense_type">
20:      <equals type="string" value="etMileage">
21:        <set>mileage_rate</set>
22:      </equals>
23:    </if>
24:  </calculate>
25:  <calculate type="string" value="desc">
26:    <if boolean="use_weekday_descriptor">
27:      <set format="%A">qsf_time_string</set>
28:    </if>
29:    <else>
30:      <set object="qof-expenses">expense_vendor</set>
31:    </else>
32:  </calculate>
33: </object>

```

Preset values. Some default values are always defined:

- `qsf_time_now` : Format: `QOF_TYPE_DATE`. The current time taken from the moment the QSF object file is loaded at runtime.
- `qsf_time_string` : Format: `QOF_TYPE_STRING`. The current timestamp taken from the moment the QSF object file is loaded at runtime. This form is used when the output parameter needs a formatted date string, not an actual date object. The format is determined by the optional format attribute of the set tag which takes the same operators as the GNU C Library for `strftime()` and output may therefore be dependent on the locale of the calling process - take care. Default value is `%F`, used when `qsf_time_string` is set without the format attribute.
- `qsf_time_precision` : Format: `QOF_TYPE_STRING`. This dictates how precisely date objects are matched. e.g. in the test environment of a conversion from pilot-link to GnuCash invoices, events and expenses that occur on the same day should be considered part of the same invoice - even if the actual timestamps are several hours apart. The preset default uses the GNU C Library formatting and is set to `%j` - the same day of the year. This default can be overridden by specifying it directly in the map as a local default. If you do change this default, take care in case the calling process is not using the same locale. Other useful options for this value might be `%U` for a weekly precision, `%C` for annual precision, `%F` for an absolute day match, `%c` for an absolute match including timezone. Note that this value is always compared as the string returned by `strftime()` using that format. Formats that return a NULL or invalid string and formats that don't specify a % will cause the map to revert to the preset `%j` default. Complex formats using ordinary characters and/or multiple format specifiers are not supported.
- `qsf_lookup_from_string` : Format: `QOF_TYPE_BOOLEAN`. Always true. Used to allow references to QOF objects outside the QSF object data. e.g. The `gncInvoice` includes a reference to the account to use for posting the invoice plus accounts to use for certain entry types. These cannot be specified by GUID in the QSF map in order to retain portability. Instead, specify the `qsf_lookup_from_string` and add the descriptive name of the account to use. `qof_book_merge` will create the account if appropriate.

Line 1: Interface Version. All QOF objects used in the map must use the same QOF interface version. Back

Line 2 to 6: Definitions. The definition tag defines the objects from both applications that will be used in the map. The `e_type` and descriptive contents of each description must appear exactly as defined in the `QofObject` definitions within the relevant application. Defined objects must also be unique: the `e_type` values and descriptive contents from different applications must not overlap within the map. If there is a name collision between two applications, a request should be made to the application developers to modify the settings of one of the two applications. If subsequent lines define suitable calculations, the map can be used in either direction. Back

Line 7: Defaults. Defaults are independent of object definitions. The calculations can assign the default to any parameter that matches the required QOF type. Back

Line 9: Map Objects. Each defined object must have one calculation defined in the map for each `QofClass` parameter that has both a `get()` and a `set()` routine defined as not NULL (a principle established with `qof_book_merge`). Each calculation must end with a set tag. Empty calculations for required parameters will cause the map to fail. Back

Line 10: Calculations. Each calculation is defined by specifying the type and name of the parameter in the current output object to be *SET*. The map is processed by the receiving application which understands the output object. Back

Line 11: Conditionals. Conditionals can reference objects as if within the original application. In

operation, the map is overlaid across both sets of defined objects, an import object in the source application and an output object for the destination object. The current import and output QSF objects are therefore always available to the map. This conditional specifies that if the type of the current import QSF object is qof-expenses, the current output gncEntry object should use the action parameter defined within the conditional. Conditionals can reference parameter as well as object values. Back

Line 12: Assignments. Map assignments can use the native values within the output object. The output object must support setting the relevant parameter using the value exactly as given in the map because the relevant set() function will be called using this value. This may reduce the readability of the map but the relevant application could also be modified to support a more readable set function. Back

Line 14: Nesting. if(){ } else{ } is also supported. Nesting of conditionals causes problems for validating the final map against any sensible XML Schema and a map that doesn't validate will be rejected. When editing conditionals in a QSF map, *ALWAYS* validate the map using `xmlLint`. If necessary, define a variable at the foot of the definitions block, using a similar syntax to a default, then use that variable in another conditional. Back

```
<variable name="my_rate" type="numeric" value="0/1"/>
```

Line 21: Conversions. A value of one type can be used to set a suitable value of a different parameter with a different type but converting between types is *NOT* supported. You must define a function within one of the applications that can handle the conversion from the relevant import type. Any request to set a parameter with a value that does not match the required type will be ignored. The only exceptions to this are for GUID's and enumerator values. Back

- GUID: The GUID for reference objects cannot be specified directly in the QSF map in order to retain portability. Instead, a default value is available. `qsf_lookup_from_string` provides the descriptive name of the account to use. QSF will lookup the appropriate GUID using QOF.
- enumerator values: Setting the plain integer decreases the human readability of the map and increases the chance of errors. By defining the enumerator values in the map, the readable versions can be used in the calculation.

Both GUID's and enumerator values use the option attribute of the set tag. This allows QSF to do these strictly limited type conversions before calling the set function for the parameter.

Line 26: Boolean preferences. Use a default boolean value to adjust strings like descriptions and notes according to user preference. Back

Line 30: QSF objects. When the map needs to draw in a real value from the QSF objects, use the type attribute to specify the QSF object to use. Back

An example QSF map file, `pilot-qsf-GnuCashInvoice.xml`, and the QSF Map Schema (`qsf-map.xsd`) are now available.

5.2.2. Relating the map to the QSF objects

The simple QSF parser (`qsf-xml.c`) shows how the QSF map can be built into a `xmlHashTable` (the QOF

version of this parser will use GHashTable which has certain advantages) that can be used to lookup the calculation needed for each tag in the output QSF object file. The parser reads the input QSF object file into a hash then iterates over the QSF map to build the definitions and default hashes. Each calculation is then parsed to create the output tags in a new QSF XML tree. When all calculations are complete, the output tree is written to the output file using xmlDocDump.

The pilot-qsf-GnuCashInvoice QSF map is not yet complete - none of the datebook or address data is currently handled. The QSF map Schema and format may change slightly during this process - development is ongoing.

5.3. Writing new QSF maps

Until tools can be developed to design the map, each QSF map will have to be edited by hand. The process requires detailed knowledge of the QOF objects of both applications as well as a general understanding the source code for each program. This section contains the notes I used to create the pilot-qsf-GnuCashInvoice QSF map.

The expenses QOF object in pilot-link can be described as follows. This text is copied directly from the QofObject and QofClass parameter definitions in the source code.

```
struct tm date    EXP_DATE, QOF_TYPE_DATE,
enum ExpenseType type EXP_TYPE, QOF_TYPE_INT32,
enum ExpensePayment payment EXP_PAYMENT, QOF_TYPE_INT32,
int currency EXP_CURRENCY, QOF_TYPE_INT32,
char *amount EXP_AMOUNT, QOF_TYPE_STRING,
char *vendor EXP_VENDOR, QOF_TYPE_STRING,
char *city EXP_CITY, QOF_TYPE_STRING,
char *attendees EXP_ATTENDEES, QOF_TYPE_STRING,
char *note EXP_NOTE, QOF_TYPE_STRING,

enum ExpenseType {
etAirfare, etBreakfast, etBus, etBusinessMeals,
etCarRental, etDinner,
etEntertainment, etFax, etGas, etGifts, etHotel,
etIncidentals,
etLaundry,
etLimo, etLodging, etLunch, etMileage, etOther, etParking,
etPostage,
etSnack, etSubway, etSupplies, etTaxi, etTelephone, etTips,
etTolls,
etTrain
};

enum ExpensePayment {
epAmEx, epCash, epCheck, epCreditCard, epMasterCard,
epPrepaid, epVISA,
epUnfiled
```

```
};
```

Note that although a struct `tm` in the code, date is considered as a standard `QOF_TYPE_DATE` - a Timespec. The use of an integer for currency and a string for the amount could also cause problems in GnuCash.

- Date is used in the query to obtain the expenses for the days covered by the invoice. Date maps to `gncInvoice::INVOICE_OPENED`
- Type is used to determine any mileage rate, to calculate other reimbursement parameters and to complete the description of this transaction within the invoice. `Type(as string)` maps to `Transaction::TRANS_DESCRIPTION`. `Type` (as an enum) is used to retrieve the rate that maps to `gncEntry::ENTRY_IPRICE`.
- Payment indicates the method of payment. This can be incorporated into the description. In other circumstances, it would be used to set the GnuCash account to be debited but that will not be used for my test case involving invoices. Other maps can use this field as the user sees fit.
- currency needs experimentation to see how it is used by pilot-link and how it can be translated into GnuCash. With `gncCommodity` still being a problem within QOF, this may simply be left to the GnuCash default.
- Amount - interesting that it is used as a string, it will need careful conversion to a `gnc_numeric`. This will be used as the Amount of the expenses transaction within the invoice. Currency problems could reappear here too. Amount maps to `gncEntry::ENTRY_QTY`.
- Vendor - this may appear optional but could be an important check that the correct expense is being allocated to the correct invoice by comparing with the `gncJob` or `gncCustomer`. Not directly mapped.
- City - again, can be used to check that the correct expense has been identified by date.
- Attendees - not used in this test map, may be useful to others.
- Note - not used in this test map, may be useful to others.

Datebook is defined as follows (items not used in the map omitted):

```
int event; /* Is this a timeless event? */
DATEBOOK_EVENT, QOF_TYPE_INT32,
struct tm begin; /* When does this appointment start? */
DATEBOOK_BEGIN, QOF_TYPE_DATE,
struct tm end; /* When does this appointment end? */
DATEBOOK_END, QOF_TYPE_DATE,
enum repeatTypes repeatType;
/* How should this appointment be repeated, if at all?*/
DATEBOOK_REPEAT_TYPE, QOF_TYPE_INT32,
int repeatForever;
/* Do the repetitions end at some date or carry on forever? */
DATEBOOK_REPEAT_FOREVER, QOF_TYPE_INT32,
(this may be changed to QOF_TYPE_BOOLEAN).
struct tm repeatEnd; /* What date do they end on? */
DATEBOOK_REPEAT_END, QOF_TYPE_DATE,
int repeatFrequency;
```

```

/* Should I skip an interval for each repetition? */
DATEBOOK_REPEAT_FREQUENCY, QOF_TYPE_INT32,
enum DayOfMonthType repeatDay; /* for repeatMonthlyByDay */
DATEBOOK_REPEAT_DAY, QOF_TYPE_INT32,
int repeatWeekstart;
/* What day did the user decide starts the week? */
DATEBOOK_REPEAT_WEEK_START, QOF_TYPE_INT32,
int exceptions;
/* How many repetitions are there to be ignored? */
DATEBOOK_EXCEPTIONS, QOF_TYPE_INT32,
struct tm *exception; /* What are they? */
DATEBOOK_EXCEPTION, QOF_TYPE_DATE,
char *description;
/* What is the description of this appointment? */
DATEBOOK_DESCRIPTION, QOF_TYPE_STRING,
char *note; /* Is there a note to go along with it? */
DATEBOOK_NOTE, QOF_TYPE_STRING,

```

Timeless events are anniversaries, birthdays or other marker events that are set in the PDA as 'No time'. If event is set, the datebook object needs to be ignored for the purposes of an invoice.

begin and end will be used to calculate the number of hours to charge on the invoice. end - begin maps to gncEntry::ENTRY_QTY.

Repeating events may need care to identify the actual date of the event. Check the repeatType: repeatNone, repeatDaily, repeatWeekly, repeatMonthlyByDay, repeatMonthlyByDate or repeatYearly. Check the repeatForever value and the repeatEnd Timespec. Check the repeatFrequency, repeatDay and repeatWeekStart only if appropriate. If still within the repeatEnd, check if any dates are to be ignored and find out which ones. Finally, if the event survives all checks, map to gncInvoice::INVOICE_OPENED and gncEntry::ENTRY_DATE, gncEntry::ENTRY_DATE_ENTERED. (Other maps are free to map to just one or two of those.)

Description should match with the gncJob but this is a simple check.

Some items are not read by pilot-link as of 0.11.8. There is a category setting and a Location setting. User intervention will be required to confirm that the correct event has been selected, in the absence of either of these settings, if there is any mis-match in the Description.

Finally for this map, the AddressBook object is defined as a list of all strings:

```

(concat(ADDR_FIRST_NAME,ADDR_LAST_NAME) <==> ADDRESS_NAME ||
ADDR_COMPANY, <==> ADDRESS_NAME)
ADDR_PHONE_ONE, <==> ADDRESS_PHONE (if regexp matches)
ADDR_PHONE_TWO, <==> ADDRESS_FAX (if regexp matches)
ADDR_PHONE_THREE, <==> ADDRESS_EMAIL (if regexp matches)
ADDR_PHONE_FOUR,
ADDR_PHONE_FIVE,

```

```
ADDR_ADDRESS, <==> ADDRESS_ONE
ADDR_CITY, <==> ADDRESS_TWO
ADDR_STATE, <==> ADDRESS_THREE
ADDR_ZIP, <==> ADDRESS_FOUR (if regexp matches)
ADDR_COUNTRY,
ADDR_TITLE,
ADDR_CUSTOM_ONE,
ADDR_CUSTOM_TWO,
ADDR_CUSTOM_THREE,
ADDR_CUSTOM_FOUR,
ADDR_NOTE,
ADDR_CATEGORY,
```

Note how this map is not fixed. In the last summary, I've tried to summarise the process using familiar symbols. The map will be intelligent and will always check the incoming content against a regular expression of what the mapped field normally contains. In most cases, an address is only created once. More commonly, the details of the address will be used to check against the existing addresses to locate the correct customer. As invoices will be almost always new items, a GUID check is not practical. It will also be possible to map the custom fields on a per-user basis, perhaps to contain the rates to charge etc.

Chapter 6. Merging QofBook's

'Merging two QofBook* objects with collision handling'

QSF is only a part of `qof_book_merge`. Processing any QSF file generates a second QofBook in a second QofSession. The data from the QSF file then needs to be imported into the active QofBook for that application using `qof_book_merge`. Every QSF operation requires a complimentary `qof_book_merge`.

However, `qof_book_merge` can be used without QSF. Any time that a QOF application has two QofBook's when one is needed, `qof_book_merge` can be used. The following sections now centre on the merge process itself.

6.1. Preparing the rule set

The rule set for `qof_book_merge` needs to be open and extendable. The difficulty with this method is handling errors. If the rule set is wrong, errors at compile and debug time could be difficult to trace because `qof_book_merge` needs to rely on the rule set to decide which variables can be assigned to which structs. These errors will appear the same as errors arising from invalid values in existing structures. In order to allow `qof_book_merge` to run smoothly in the final program, the rule set needs to be definitive and robust. Each object declares its own parameters in the `qofclass` object definitions.

These parameters include:

- **The QOF object type.** A unique name for the object that identifies all other objects of the same type in any QofBook.
- **A full list of all useful parameters held in the object.** Note that this list is crucial to `qof_book_merge` but is actually defined by the object developer. `qof_book_merge` can only work with objects that describe themselves to QOF clearly and with all necessary parameters sensibly defined. This is the price of a generic merge.
- **The QOF type of each parameter.** The type is essential if `qof_book_merge` is to know how to compare and merge the parameter data into the target QofBook.
- **Templates for functions.** To be used to get and set suitable parameters. This was the hardest part of the merge operation for me to understand initially. I was unused to typed function pointers and with GnuCash using C, not C++, casting a function pointer to retrieve a parameter value from an object without knowing in advance what object is being queried, seemed like a black art. I missed the `virtual` keyword like never before!

The merge code needs to determine the type of object, the relative importance of the data and to set a sequence for the query that precedes the match. In any merge, some parameter data is inconsequential to the final result of the merge. Values that relate only to the import environment and values that are calculated from other parts of the import environment need to be recalculated in the final merge; the actual value of the calculation in the import is irrelevant to the final calculation after the merge. What

matters is that all relevant components of the calculation are retained and merged into any existing components in the final book. Merging books necessarily causes the balance of merged accounts to be different from the balance of the import or the pre-merge book. The final value should be consistent with the pre-merge values (by addition and/or subtraction), but cannot be set explicitly, it must be determined from the real data as it exists after the merge.

A simple premise is therefore used to determine the relative importance of parameters within an object:

If a parameter cannot be set, it is ignored by the merge.: (The object is still compared using other parameters).

This is simply because a value that does not have a set function declared in the object definition should be defined as a calculated value or similar that should never be set. e.g. An account holds many transactions, each with a value. Although the value of a transaction clearly does need to be set, the balance of the account is a calculated value, dependent on the transactions contained within the account. Setting an account balance artificially would generate invalid data. So although the balance of an account can be retrieved (it has a get function), the lack of a set function determines that the balance parameter will not be compared as part of the merge. The comparison is based on the other parameters for that object. Two identical accounts are still identical objects, even if they contain different transactions and therefore a different balance. The transactions differ, the account does not. To consider a bank account or credit card account, each month the transactions and the balance change but the account is still the same account.

This clearly illustrates the need for objects to be described sensibly and logically! It also allows `qof_book_merge` to be used with other kinds of objects in the wider QOF design - the parameters that determine the details of the merge are set by the implementation of the code, rather than the code itself, subject to the implementation remaining a valid QOF environment.

In all rules, the presence or lack of a GUID (`../doxygen/group__GUID.html`) must be of primary importance in any merge because an exact match in GUID allows `qof_book_merge` to import the attached data, overwriting that section of the existing QofBook.

`qof_book_merge` only accepts input data as a QofBook* structure and therefore all import objects will be assigned a new GUID if one does not exist already when the data is inserted into the import QofBook. If the GUID in the imported data does not match the target QofBook*, `qof_book_merge` must use the rule set to proceed to match the contained data. A sequence of equality checks are required against existing data. In allowing fast processing of these checks, the rule set must balance the desire to not generate thousands of collision queries for the user to resolve against the need to limit the amount of manual editing of the final file.

6.2. Draft of a rule set framework.

A rough draft of a rule set

1. *Validate the import book.*

To generate the ruleset, `qof_book_merge` will parse all objects and all parameters in the import book. If any parameters contain unexpected data, illegal values, non-existent parameter types etc., `qof_book_merge` will use the reserved value of `INVALID` and will immediately abort, freeing the import book in memory. The target book will be untouched and an error code will be returned. Procedures that call `qof_book_merge` code must check the error value set by `qof_book_mergeInit` (`../doxygen/group__BookMerge.html`). Any parameters or objects that are not correctly described for `qofclass` at compile time will be ignored.

2. *Identify the GUID.*

If a GUID match exists, this is a semantic match - the two objects are the same, even if the data content has changed. The object will be tagged as an `ABSOLUTE` match if there are no changes in any relevant object parameters. See Preparation for the discussion of what parameters are relevant and why. If any relevant parameters differ, the object is tagged as `UPDATE`. If the import contains an account with the same GUID, for example, a change in the account description would be inserted into the target book (overwriting the previous value). Semantic matches are not reported to the user. `ABSOLUTE` matches will be ignored - leaving the target book untouched.

If the import object GUID does not match an existing object, the parameter values of the object are compared to other objects of the same type in the target book. If the same data exists in the target book with a different GUID, the object is tagged as `DUPLICATE`. All objects tagged as `DUPLICATE` are ignored in the final merge. If the data has changed, the object is tagged as `REPORT`.

3. *Identifying fields that may result in a collision.*

To determine whether an object should be tagged as `ABSOLUTE` or `UPDATE`, `DUPLICATE` or `REPORT`, the parameter data within each object needs to be compared with the target book to check for a collision. `qof_book_merge` queries the target book for all objects of the same object type and compares the parameter values.

The object type can be easily determined as a result of the QOF framework; objects are passed to the `qof_book_merge` comparison routine in type order. Therefore, a single check on all existing objects of the same type in the target book is required to see if an object exists that would cause a collision.

4. *Tag remaining objects*

Any objects that cannot be tagged as `ABSOLUTE`, `UPDATE`, `DUPLICATE` or `REPORT` are tagged as `NEW`. These contain GUID's that do not exist in the target book and contain no data that conflicts with existing objects of the same type. Unlike semantic matches, a `NEW` object does not overwrite

existing data in the target book. An Account object, for example, tagged as NEW, will be inserted into the target book as a new account. Transactions tagged as NEW will be appended to the appropriate account. Objects tagged as NEW are not reported to the user.

5. *User Intervention.*

All REPORT items are made available to the user intervention dialog which then controls how each conflict is resolved. All REPORT items need to be re-assigned to UPDATE, DUPLICATE or NEW. In order to maintain the integrity of the amended target book, objects in the import book cannot be ignored. Note that because import data must be passed to `qof_book_merge` as a `GNCBook*`, certain default data will be created. Allowing certain objects to be ignored could lead to unexpected results.

If the user aborts the dialog, the dialog control procedure should set the last/current object to INVALID - this will cause `qof_book_merge` to abort without changing the target book. Until all REPORT items have been re-assigned, the target book remains unchanged.

6. *Altering the target book*

All objects tagged as ABSOLUTE or DUPLICATE are ignored. Before processing the objects tagged as UPDATE, all NEW objects will be added to the target book.

The rule set itself is built from the QOF registration definitions of each component of the book. GnuCash uses QOF to provide all internal query functions. QoF in turn uses definitions within the code of each object to determine the kind of data contained in the object and how that data can be addressed and modified.

```
static QofObject commodity_table_object_def =
{
    interface_version: QOF_OBJECT_VERSION,
    e_type:            GNC_ID_COMMODITY_TABLE,
    type_label:       "CommodityTable",
    book_begin:       commodity_table_book_begin,
    book_end:         commodity_table_book_end,
    is_dirty:         NULL,
    mark_clean:       NULL,
    foreach:          NULL,
    printable:        NULL,
};

gboolean
gnc_commodity_table_register (void)
{
    gnc_quote_source_init_tables();
    return qof_object_register (&commodity_table_object_def);
}
```

A more complex definition will help to show how the rule sets from the first draft will be applied into the existing QoF structures:

```

static QofObject gncInvoiceDesc =
{
    interface_version:  QOF_OBJECT_VERSION,
    e_type:              _GNC_MOD_NAME,
    type_label:         "Invoice",
    book_begin:         NULL,
    book_end:           NULL,
    is_dirty:           qof_collection_is_dirty,
    mark_clean:         qof_collection_mark_clean,
    foreach:            qof_collection_foreach,
    printable:          _gncInvoicePrintable,
};
gboolean gncInvoiceRegister (void)
{
    static QofParam params[] = {
        { INVOICE_ID, QOF_TYPE_STRING, (QofAccessFunc)gncInvoiceGetID, NULL },
        { INVOICE_OWNER, GNC_ID_OWNER, (QofAccessFunc)gncInvoiceGetOwner, NULL },
        { INVOICE_OPENED, QOF_TYPE_DATE, (QofAccessFunc)gncInvoiceGetDateOpened, NULL },
        { INVOICE_DUE, QOF_TYPE_DATE, (QofAccessFunc)gncInvoiceGetDateDue, NULL },
        { INVOICE_POSTED, QOF_TYPE_DATE, (QofAccessFunc)gncInvoiceGetDatePosted, NULL },
        { INVOICE_IS_POSTED, QOF_TYPE_BOOLEAN, (QofAccessFunc)gncInvoiceIsPosted, NULL },
        { INVOICE_IS_PAID, QOF_TYPE_BOOLEAN, (QofAccessFunc)gncInvoiceIsPaid, NULL },
        { INVOICE_BILLINGID, QOF_TYPE_STRING, (QofAccessFunc)gncInvoiceGetBillingID, NULL },
        { INVOICE_NOTES, QOF_TYPE_STRING, (QofAccessFunc)gncInvoiceGetNotes, NULL },
        { INVOICE_ACC, GNC_ID_ACCOUNT, (QofAccessFunc)gncInvoiceGetPostedAcc, NULL },
        { INVOICE_POST_TXN, GNC_ID_TRANS, (QofAccessFunc)gncInvoiceGetPostedTxn, NULL },
        { INVOICE_POST_LOT, GNC_ID_LOT, (QofAccessFunc)gncInvoiceGetPostedLot, NULL },
        { INVOICE_TYPE, QOF_TYPE_STRING, (QofAccessFunc)gncInvoiceGetType, NULL },
        { INVOICE_TERMS, GNC_ID_BILLTERM, (QofAccessFunc)gncInvoiceGetTerms, NULL },
        { INVOICE_BILLTO, GNC_ID_OWNER, (QofAccessFunc)gncInvoiceGetBillTo, NULL },
        { QOF_QUERY_PARAM_ACTIVE, QOF_TYPE_BOOLEAN, (QofAccessFunc)gncInvoiceGetActive, NULL },
        { QOF_QUERY_PARAM_BOOK, QOF_ID_BOOK, (QofAccessFunc)qof_instance_get_book, NULL },
        { QOF_QUERY_PARAM_GUID, QOF_TYPE_GUID, (QofAccessFunc)qof_instance_get_guid, NULL },
        { NULL },
    };

    qof_class_register (_GNC_MOD_NAME, (QofSortFunc)gncInvoiceCompare, params);
    reg_lot ();
    reg_txn ();

    return qof_object_register (&gncInvoiceDesc);
}

```

Note that the initial config of Account had a NULL where the QofSetterFunc could have been declared.

A problem arises in Transaction where a transaction is voided. There are three QofAccessFunc routines for each part of the void but only one function that could be used as a QofSetterFunc which sets all three values, as well as a second set function that undoes the void operation.

Invoice: unable to set the due date - no set_fcn?

Lots are also potential problems - this will need to be monitored.

In Jobs, gncJobGetActive is defined in two separate parameters?

The files in use with QofSetterFunc are:

- engine/Account.c
- engine/Transaction.c
- gnc-pricedb.c
- business/business-core/gncInvoice.c
- business/business-core/gncAddress.c
- business/business-core/gncBillTerm.c
- business/business-core/gncCustomer.c
- business/business-core/gncEntry.c
- business/business-core/gncJob.c
- business/business-core/gncOrder.c
- business/business-core/gncOwner.c
- business/business-core/gncTaxTable.c
- business/business-core/gncVendor.c

6.3. Design of the merge

The qof_book_merge code (../doxygen/qof__book__merge_8h.html) is already documented.

A sample test program would use the three qof_book_merge API routines:

•

```
qof_book_merge *mergeData;
mergeData = qof_merge_bookInit(QofBook *import, QofBook *target);
```

Make sure you check that mergeData is not NULL before proceeding with the user intervention loop. A NULL indicates a failure and your code MUST abort.

- Write your loop to present the output for the user and take the results, ready to pass to qof_book_mergeUpdateResult(). Offer the user options to abort the merge at any/all stages of this loop - nothing in the target book is changed until the code proceeds to the qof_book_mergeCommit stage.
- Only when all REPORT items have been re-assigned to NEW, DUPLICATE or UPDATE should you call qof_book_mergeCommit(mergeData).

Commit cannot be stopped or undone!: Remember that the commit is a *ONE WAY* operation - there is NO going back or "unmerging" the books. Your user involvement routine must make it clear to the user that the user must take responsibility for either making a backup or unpicking the merge manually if they have chosen the wrong import file etc. Commit cannot be aborted - forcing a shutdown carries significant risks of corrupting the target book.

This is the pseudo-code for the Init function where most of the work is done.

```

Processing starts with Init
Init calls ForeachType (once for each registered type of object)
ForeachType calls ForeachParam
parameter added to list of parameters for the object
Return to ForeachType
ForeachType calls Foreach (each entity in turn)
Foreach creates a new rule and looks up an absolute match in the target book
using Compare
If no absolute match, calls ForeachTypeTarget
Checks that the entity type matches and calls ForeachTarget
ForeachTarget calls orphan_check to prioritise the match
returns to ForeachTypeTarget
returns to Foreach
Sets target(s) or sets MERGE_NEW
returns to ForeachType
returns to Init

```

```

mergeData = qof_book_mergeInit( QofBook *importBook, QofBook *targetBook)
qof_object_foreach_type(qof_book_mergeForeachType, mergeData);
qof_class_param_foreach(merge_obj->e_type, qof_book_mergeForeachParam , mergeData);
qof_object_foreach(merge_obj->e_type, mergeData->mergeBook, qof_book_mergeForeach, mergeData);
mergeRule = g_new(qof_book_mergeRule,1);
targetEnt = qof_collection_lookup_entity (
qof_book_get_collection (mergeData->targetBook, mergeEnt->e_type), g);
g_return_if_fail(qof_book_mergeCompare(mergeData) != -1);
qof_object_foreach_type(qof_book_mergeForeachTypeTarget, mergeData);
if(safe_strcmp(merge_obj->e_type, currentRule->importEnt->e_type) == 0) {
qof_object_foreach(currentRule->importEnt->e_type, mergeData->targetBook,
qof_book_mergeForeachTarget, mergeData);
qof_book_merge_orphan_check(difference, mergeRule, mergeData);
}
}
mergeData->mergeList = g_slist_prepend(mergeData->mergeList,mergeRule);

```

Note that qof_book_mergeForeach is the first time real user data is available.

6.3.1. Example programs for qof_book_merge

The old examples have been removed - it was proving time-consuming to keep them up to date with changes in the merge API. For examples, please see the GnuCash source tree:

```
gnucash/src/engine/test/test-book-merge.c
```

```
gnucash/src/gnome/druid-merge.c
```

6.4. Using qof_book_merge with new and existing QOF objects

qof_class_register parameters. A successful merge operation relies on adequate support from new and existing QOF and GnuCash objects. A generic merge must rely on the specific objects to describe themselves and provide the functionality to get and set the relevant data. Each object needs to utilise a full set of qof_class parameters and a standard set of functions to get and set all and any parameters that cannot or should not be calculated.

The test-book-merge.c program demonstrates a full and complete qof_class parameter and function description:

```
static QofObject obj_object_def = {
    interface_version:    QOF_OBJECT_VERSION,
    e_type:               TEST_MODULE_NAME,
    type_label:          TEST_MODULE_DESC,
    create:               (gpointer)obj_create,
    book_begin:          NULL,
    book_end:             NULL,
    is_dirty:             NULL,
    mark_clean:          NULL,
    foreach:              qof_collection_foreach,
    printable:           NULL,
    version_cmp:         (int (*)(gpointer,gpointer)) qof_instance_version_cmp,
};

gboolean myobjRegister (void)
{
    static QofParam params[] = {
    { OBJ_NAME, QOF_TYPE_STRING, (QofAccessFunc)obj_getName, (QofSetterFunc)obj_setName
    },
    { OBJ_AMOUNT, QOF_TYPE_NUMERIC, (QofAccessFunc)obj_getAmount, (QofSetterFunc)obj_setAmo
    },
    { OBJ_GUID, QOF_TYPE_GUID, (QofAccessFunc)obj_getGUID, (QofSetterFunc)obj_setGUID
    },
    { OBJ_DATE, QOF_TYPE_DATE, (QofAccessFunc)obj_getDate, (QofSetterFunc)obj_setDate
    },
    }
```

```

{ OBJ_DISCOUNT, QOF_TYPE_DOUBLE, (QofAccessFunc)obj_getDiscount, (QofSetterFunc)obj_setDisc
{ OBJ_ACTIVE, QOF_TYPE_BOOLEAN, (QofAccessFunc)obj_getActive, (QofSetterFunc)obj_setAct
},
{ OBJ_VERSION, QOF_TYPE_INT32, (QofAccessFunc)obj_getVersion, (QofSetterFunc)obj_setVersi
{ OBJ_MINOR, QOF_TYPE_INT64, (QofAccessFunc)obj_getMinor, (QofSetterFunc)obj_setMinor
},
{ QOF_PARAM_BOOK, QOF_ID_BOOK, (QofAccessFunc)qof_instance_get_book, NULL },
{ QOF_PARAM_GUID, QOF_TYPE_GUID, (QofAccessFunc)qof_instance_get_guid, NULL },
{ NULL },
};

qof_class_register (TEST_MODULE_NAME, NULL, params);

return qof_object_register (&obj_object_def);
}

```

The prototypes for each get and set function illustrate the standard mechanisms for use by the merge operation:

```

/* getter functions */
void obj_setName(myobj*, char*);
void obj_setGUID(myobj*, const GUID*);
void obj_setAmount(myobj*, gnc_numeric);
void obj_setDate(myobj*, Timespec h);
void obj_setDiscount(myobj*, double);
void obj_setActive(myobj*, gboolean);
void obj_setVersion(myobj*, gint32);
void obj_setMinor(myobj*, gint64);

/* setter functions */
gnc_numeric obj_getAmount(myobj*);
char* obj_getName(myobj*);
const GUID* obj_getGUID(myobj*);
Timespec obj_getDate(myobj*);
double obj_getDiscount(myobj*);
gboolean obj_getActive(myobj*);
gint32 obj_getVersion(myobj*);
gint64 obj_getMinor(myobj*);

```

Please note the return types and parameter types. To use the merge operation, your object should use the same prototype for each relevant QOF_TYPE.

In particular, please keep the same pattern for pointer parameters - strings should be char (or gchar) pointers, GUID should be constant pointers and all other types are passed and returned without using pointers.

Declaring functions that use additional parameters, unexpected pointers or other discrepancies is likely to cause a segmentation fault in the merge.

Any parameter that does NOT have a get and a corresponding set function will be ignored by the merge operation - this rule should be used to discriminate between parameters that should only ever be calculated (e.g. account balances) and parameters that must always be set by the backend, user or merge (e.g. account names).

Note that all objects require a foreach AND a create function in the object definition. If any object is not fully QOF compliant, ensure that either the foreach or create functions are NULL or you will get errors when `qof_book_merge` runs, even if no objects of that type exist in either book.

6.5. Known problems

GncCommodity and QOF. The system used for commodities does not follow the object handling and identification system (GUID's, Entities, etc.) that the other parts of GnuCash use. The API really should be ported over. This would allow us to get rid of the commodity table routines defined in v1.8

As a result, `qof_book_merge` does not currently handle commodities.

A problem arises in Transaction where a transaction is voided. There are three `QofAccessFunc` routines for each part of the void but only one function that could be used as a `QofSetterFunc` which sets all three values, as well as a second set function that undoes the void operation.

Lots are also potential problems - this will need to be monitored.

Comparisons without a GUID match. `qof_book_mergeUpdateRule` only sets a failed match if ALL objects fail to match. When `absolute` is FALSE, all suitable target objects are compared and `mergeResult` is not set until all targets have been checked. The closest match is identified using a difference rank. This avoids using non-generic tests for object similarities. `difference` has a maximum value of the total number of comparable parameters and the value closest to zero is used. In the case of a tie, it is currently first-come-first-served. This needs to be improved.

In the following table, when the specified object is merged, data in the specified struct member variables will not be read, compared or set. Existing values in the target `QofBook` will not be changed. Where the object is QOF compatible, if the object is new (MERGE_NEW) the listed variable will be set to the default value given in the table.

Table 6-1. Struct members (Engine) without set_fcn, get_fcn pairs

Type	Name	Default	Reason / Plan
Account			
<code>gnc_commodity*</code>	<code>commodity</code>	NULL	Incompatible with QOF
<code>SplitList*</code>	<code>splits</code>	NULL	QOF_TYPE_COLLECT: Pending
<code>LotList*</code>	<code>lots</code>	NULL	QOF_TYPE_COLLECT: Pending

Type	Name	Default	Reason / Plan
GNCPolicy*	policy	xaccGetFIFOPolicy()	Unknown - does this need to be fixed?
Split			
Split*	gains_split	NULL	May need a call to xacc-SplitDetermineGainStatus - please report any issues
Transaction			
gnc_commodity*	common_currency	NULL	Outstanding issue with gnc_commodity types
GSList*	splits	NULL	QOF_TYPE_COLLECT: Pending
Lot			
Incomplete QOF registration needs to be fixed			Pending
Price			
gnc_commodity*	commodity	set in GncPriceDB	unresolved - please report any issues
gnc_commodity*	currency	set in GncPriceDB	unresolved - please report any issues

Table 6-2. Struct members (Business) without set_fcn, get_fcn pairs

Type	Name	Default	Reason / Plan
gncCustomer			
gnc_commodity*	currency	undefined	Unresolved.
GSList*	jobs	NULL	QOF_TYPE_COLLECT: Pending
gncEmployee			
GncAddress*	addr	gncAddressCreate (book, &cust->inst.entity)	Resolved - could still conflict in a merge, monitoring.
gnc_commodity*	currency	undefined	Unresolved.
gncEntry			
GList*	i_tax_values	undefined	QOF_TYPE_COLLECT: Pending
gncInvoice			
GncOwner	owner	undefined	QOF_TYPE_COLLECT: Pending
GncOwner	billto	GNC_OWNER_CUSTOM	QOF_TYPE_COLLECT: Pending
gnc_commodity*	currency	undefined	Unresolved.
gncJob			

Type	Name	Default	Reason / Plan
GncOwner	owner	undefined	QOF_TYPE_COLLECT: Pending
gncOrder			
GncOwner	owner	undefined	QOF_TYPE_COLLECT: Pending
GList*	entries	undefined	QOF_TYPE_COLLECT: Pending

Design problems. A brief note about books & sessions: A book encapsulates the datasets manipulated by GnuCash. A book holds the actual data. By contrast, the session mediates the connection between a book (the thing that lives in virtual memory in the local process) and the datastore (the place where book data lives permanently, e.g., file, database).

In the current design, a session may hold multiple books. For now, exactly what this means is somewhat vague, and code in various places makes some implicit assumptions: first, only one book is 'current' and open for editing. Next, it's assumed that all of the books in a session are related in some way. i.e. that they are all earlier accounting periods of the currently open book. In particular, the backends probably make that assumption, in order to store the different accounting periods in a clump so that one can be found, given another.

6.6. Design improvements

qof_book_merge needs a method to add references to entire QOF objects into other QOF objects as variables. e.g. Account and GncAddress objects are often referenced by other objects. qof_class definitions exist under GNC_ID types and merge will be extended to get and retrieve such object references.

Each GNC_ID is defined in QOF anyway - propose a check on the mergeType before the comparison.

Use the unknown type to reproduce the links that already exist in the import book for objects that contain the unknown type. i.e. Set get_fcn and set_fcn for object parameters that take or return objects that are already registered with QofObject/QofClass. Then when that parameter is compared, the GNC_ID... will fail to match any of the QOF_TYPE... fundamental data types - this is the unknown type. Store the QofEntity of that type within the rule for the 'parent' object.

Later, run a secondary loop to match up the child QofEntity with the qof_book_mergeResult of the actual object, use that to modulate the qof_book_mergeResult of the parent object.

e.g.

```
{ INVOICE_ACC, GNC_ID_ACCOUNT, (QofAccessFunc)gncInvoiceGetPostedAcc, NULL },
```

This parameter could use (QofSetterFunc)gncInvoiceSetPostedAcc - which takes an Account object - not a fundamental QOF_TYPE.

In the rule that is dealing with the Invoice QofEntity, use the QofObject and QofClass definitions to obtain the parameter of the INVOICE_ACC account. qof_class_is_registered, QofInstance -> QofEntity.

Table 6-3. Struct members (Engine) with resolved issues.

Type	Name	Default	Reason / Plan
SchedXactions			
All parameters now QOF-compatible.			Resolved.
FreqSpec			
All parameters compatible with QOF.			Resolved.
Account			
AccountGroup	parent, children	NULL	Resolved as a reference
QofInstance	inst	set by create:	OK
gint32	version	0	Not suitable for merge.
guint32	version	0	Not suitable for merge.
Split			
QofEntity	entity	qof_entity_init()	OK
QofBook*	book	set be create:	OK
Account*	acc	NULL	Resolved
GNCLot*	lot	NULL	Resolved
Transaction*	parent	NULL	Resolved
unsigned char	gains	NULL	Set via xaccSplitSetSharePrice - please report any issues
Transaction			
QofInstance	inst	set by create:	OK
gint32	version	0	Not suitable for merge.
guint32	version	0	Not suitable for merge.
Transaction*	orig	NULL	Resolved.
Price			
QofInstance	inst	set by create:	OK
GncPriceDB*	db	not set	This appears to be handled internally - please report any issues
gint32	version	0	Not suitable for merge.
guint32	version	0	Not suitable for merge.

Table 6-4. Struct members (Business) with resolved issues.

Type	Name	Default	Reason / Plan
gncAddress			
QofBook*	book	set by create:	OK
gboolean	dirty	FALSE	internal - OK
gncBillTerm			
QofInstance	inst	set by create:	OK
GncBillTermType	type	undefined	Resolved.
gncCustomer			
QofInstance	inst	set by create:	OK
GncTaxTable*	taxtable	undefined	Resolved
GncTaxIncluded	taxincluded	undefined	Resolved
GncBillTerm*	terms	undefined	Resolved.
gncEmployee			
QofInstance	inst	set by create:	OK
Account*	ccard_acc	undefined	Resolved.
Account*	i_account	undefined	Resolved.
GncTaxTable*	i_taxtable	undefined	Resolved
Account*	b_account	undefined	Resolved.
GncTaxTable*	b_taxtable	undefined	Resolved
gboolean	values_dirty	TRUE	internal value, not to be set.
gnc_numeric	i_value	undefined	calculated value, not to be set.
gnc_numeric	i_value_rounded	undefined	calculated value, not to be set.
gnc_numeric	i_tax_value	undefined	calculated value, not to be set.
gnc_numeric	i_tax_value_rounded	undefined	calculated value, not to be set.
gnc_numeric	b_value	undefined	calculated value, not to be set.
gnc_numeric	b_value_rounded	undefined	calculated value, not to be set.
gnc_numeric	b_tax_value	undefined	calculated value, not to be set.
gnc_numeric	b_tax_value_rounded	undefined	calculated value, not to be set.
Timespec	b_taxtable_modtime	undefined	calculated value, not to be set.
gncInvoice			
QofInstance	inst	set by create:	OK

Type	Name	Default	Reason / Plan
char*	printname	NULL	internal value. Not set.
GncBillTerm*	terms	undefined	Resolved.
GList*	entries	undefined	Internal variable, not to be set.
GncJob*	job	undefined	Resolved.
gnc_numeric	to_charge_amount	gnc_numeric_zero	calculated value, not to be set.
Account*	posted_acc	undefined	Resolved.
Transaction*	posted_txn	undefined	Resolved.
GNCLot*	posted_lot	undefined	Resolved.

Chapter 7. Data Mining and freedom of access.

QOF separates the data from the application. Data can be handled, queried, modified, transformed, processed and calculated outside the limits of what the source application can support. This provides two main benefits:

- **Data Mining.** The ability to query the data behind the top level figures and find out why certain results occur.

QOF supports recursive queries on the data set but for more refined data mining, see the DWI (<http://sourceforge.net/projects/dwi>) project that works directly with QOF, or specialised data mining sites (<http://www.the-data-mine.com/>).

- **Data Freedom** (<http://www.data-freedom.org/>). The ability of the user to access their own data in ways that are too specialised, customised or user-specific to be supported by the main application. The data freedom website gives examples (<http://www.data-freedom.org/explain.html>) of how current data sharing models lack a generic, data-centric interface and how QOF can provide the solution.

Handling generic data. Users often need their data presented in specialised or unusual ways. It is a waste of resources for the application to implement these customised formats and it is inefficient for the user to have to wait for the application to support ever more specialised reports. QOF can build applications around the data using the QOF generator (<http://qof-gen.sourceforge.net/>) project. QOF provides the engine and a backend (QSF XML), and the generated code provides the front-end - a command line interface with SQL-type query handling. The command line interface itself is a cut-down version of pilot-QOF (<http://pilot-qof.sourceforge.net/>), man pages are included in this documentation.

Any QOF application can use any QOF object - once the application code is removed. QOF generator will be able to recreate enough C code to handle any QOF object based on the QSF XML exported by the original application, producing a C file that describes the QOF object and which can be used by any other QOF application. This should make data mining easier as customised applications can be developed quickly and easily. Create the C code for the objects from the XML, download a mini-application that links that C code against QOF, insert other objects from other sources and you can do whatever you need to do with the data. The custom application will be able to load any QSF file that describes the supported objects - no matter which application originally wrote the file.

7.1. Data mining within QOF

Recursive queries. The results of each query made within QOF can be saved to XML which can then be used to create the data set for the next query.

Additive queries. Data sets (QOfBooks) can be built up using a series of queries, constructing a larger data set. This allows fine control over exactly which records (entities) are included and excluded from the data set

Object transformations. QOF uses a generic format to describe objects from different applications in a standard way. The objects can also be translated from one application to another, allowing one data set to

be constructed from disparate applications. Objects themselves can be copied between applications. Future releases of the QOF generator (<http://qof-gen.sourceforge.net/>) project will recreate the objects in C from exported XML. This will allow customised applications that handle existing objects natively, including objects from more than one source application.

7.2. Data Freedom within QOF

Exchanging data is not just exchanging a document or sharing a file. Users need more than simply passing data as a file between applications. This is useful but requires both programs to be locked into the one shared format.

Exchanging data is more than asking a procedure to provide a fixed data result. Users need more than a fixed result. Calling a specific function in a specific CORBA or RPC compatible program will only give the result designed by the original developer. If the user wants data that is not specifically supported by the RPC calls, the data is inaccessible.

Data-centric model. Data should be available on-demand to whichever process can use it. QOF provides an engine that can work with multiple front ends and multiple backends, with the same data being mapped or translated between them. Each backend is generic and part of QOF. Therefore any program that can exchange data with QOF can use whichever backend is suitable. Any other QOF compatible program can read the data, even if it doesn't use that backend normally. This increases platform independence, data accessibility and program flexibility.