SimFQT

1.00.1

Generated by Doxygen 1.6.1

Sun Jun 21 22:10:03 2015

Contents

1 SimFQT Documentation

1.1 Getting Started

- Main features
- Installation
- Linking with SimFQT
- Users Guide
- Tutorials
- Copyright and License
- Make a Difference
- Make a new release
- People

1.2 SimFQT at SourceForge

- `Project page`
- `Download SimFQT`
- `Open a ticket for a bug or feature`
- `Mailing lists`
- `Forums`
  - `Discuss about Development issues`
  - `Ask for Help`
  - `Discuss SimFQT`

1.3 SimFQT Development

- `Git Repository` (Subversion is deprecated)
- Coding Rules
- Documentation Rules
- Test Rules

1.4 External Libraries

- Boost (C++ STL extensions)
- Python
- MySQL client
- SOCI (C++ DB API)

1.5 Support SimFQT

1.6 About SimFQT

SimFQT is a C++ project of airline pricing classes and functions, mainly targeting simulation purposes. N

SimFQT makes an extensive use of existing open-source libraries for increased functionality, speed and accuracy. In particular Boost (*C++ STL Extensions*) library is used.

The SimFQT project originates from the department of Operational Research and Innovation at Amadeus, Sophia Antipolis, France. SimFQT is released under the terms of the GNU Lesser General Public License (LGPLv2.1) for you to enjoy.

SimFQT should work on GNU/Linux, Sun Solaris, Microsoft Windows (with Cygwin, MinGW/MSYS, or Microsoft Visual C++ .NET) and Mac OS X operating systems.

Note:

(N) - The SimFQT library is **NOT** intended, in any way, to be used by airlines for production systems. If you want to report issue, bug or feature request, or if you just want to give feedback, have a look on the right-hand side of this page for the preferred reporting methods. In any case, please do not contact Amadeus directly for any matter related to SimFQT.

2 People

2.1 Project Admins (and Developers)

- Gabrielle Sabatier <gsabatier@users.sourceforge.net> (N)
- Denis Arnaud <denis_arnaud@users.sourceforge.net> (N)
- Anh Quan Nguyen <quannaus@users.sourceforge.net> (N)

2.2 Retired Developers

- Mehdi Ayouni <mehdi.ayouni@gmail.com>
- Son Nguyen Kim <snguyenkim@users.sourceforge.net> (N)

2.3 Contributors

- Emmanuel Bastien <ebastien@users.sourceforge.net> (N)

2.4 Distribution Maintainers

- `Fedora`/`RedHat`: Denis Arnaud <`denis_arnaud@users.sourceforge.net`> (N)
- `Debian`: Emmanuel Bastien <`ebastien@users.sourceforge.net`> (N)

Note:

(N) - `Amadeus` employees.

3 Coding Rules

In the following sections we describe the naming conventions which are used for files, classes, structures, local variables, and global variables.

3.1 Default Naming Rules for Variables

Variables names follow Java naming conventions. Examples:

- `lNumberOfPassengers`
- `lSeatAvailability`

3.2 Default Naming Rules for Functions

Function names follow Java naming conventions. Example:

- `int myFunctionName (const int& a, int b)`

3.3 Default Naming Rules for Classes and Structures

Each new word in a class or structure name should always start with a capital letter and the words should be separated with an under-score. Abbreviations are written with capital letters. Examples:

- `MyClassName`
- `MyStructName`

3.4 Default Naming Rules for Files

Files are named after the C++ class names.

Source files are named using `.cpp` suffix, whereas header files end with `.hpp` extension. Examples:

- `FlightDate.hpp`
- `SegmentDate.cpp`

3.5 Default Functionality of Classes

All classes that are configured by input parameters should include:

- default empty constructor
- one or more additional constructor(s) that takes input parameters and initializes the class instance
- setup function, preferably named `'setup'` or `'set_parameters'`

Explicit destructor functions are not required, unless they are needed. It shall not be possible to use any of the other member functions unless the class has been properly initiated with the input parameters.

4 Copyright and License

4.1 GNU LESSER GENERAL PUBLIC LICENSE

4.1.1 Version 2.1, February 1999

```
Copyright (C) 1991, 1999 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL.  It also counts
as the successor of the GNU Library Public License, version 2, hence
the version number 2.1.]
```

4.2 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

4.3 TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

4.3.1 NO WARRANTY

4.3.2 END OF TERMS AND CONDITIONS

4.4 How to Apply These Terms to Your New Programs

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library 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
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the
library `Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it!

Source

5 Documentation Rules

5.1 General Rules

All classes in SimFQT should be properly documented with Doxygen comments in include (.hpp) files. Source (.cpp) files should be documented according to a normal standard for well documented C++ code.

An example of how the interface of a class shall be documented in SimFQT is shown here:

```
/*!
 * \brief Brief description of MyClass here
 *
 * Detailed description of MyClass here. With example code if needed.
 */
```

```
class MyClass {
public:
  //! Default constructor
  MyClass(void) { setup_done = false; }

  /*!
   * \brief Constructor that initializes the class with parameters
   *
   * Detailed description of the constructor here if needed
   *
   * \param[in] param1 Description of \a param1 here
   * \param[in] param2 Description of \a param2 here
   */
  MyClass(TYPE1 param1, TYPE2 param2) { setup(param1, param2); }

  /*!
   * \brief Setup function for MyClass
   *
   * Detailed description of the setup function here if needed
   *
   * \param[in] param1 Description of \a param1 here
   * \param[in] param2 Description of \a param2 here
   */
  void setup(TYPE1 param1, TYPE2 param2);

  /*!
   * \brief Brief description of memberFunction1
   *
   * Detailed description of memberFunction1 here if needed
   *
   * \param[in]     param1 Description of \a param1 here
   * \param[in]     param2 Description of \a param2 here
   * \param[in,out] param3 Description of \a param3 here
   * \return Description of the return value here
   */
  TYPE4 memberFunction1(TYPE1 param1, TYPE2 param2, TYPE3 &param3);

private:

  bool _setupDone;         /*!< Variable that checks if the class is properly
                                initialized with parameters */
  TYPE1 _privateVariable1; //!< Short description of _privateVariable1 here
  TYPE2 _privateVariable2; //!< Short description of _privateVariable2 here
};
```

5.2 File Header

All files should start with the following header, which include Doxygen's `\file`, `\brief` and `\author` tags, `$Date$` and `$Revisions$` CVS tags, and a common copyright note:

```
/*!
 * \file
 * \brief Brief description of the file here
 * \author Names of the authors who contributed to this code
 * \date Date
 *
 * Detailed description of the file here if needed.
 *
 * ----------------------------------------------------------------------
 *
 * SimFQT - C++ Standard Airline IT Object Library
 *
 * Copyright (C) 2009-2010  (\see authors file for a list of contributors)
 *
 * \see copyright file for license information
 *
 * ------------------------------------------------------------------------
 */
```

5.3 Grouping Various Parts

All functions must be added to a Doxygen group in order to appear in the documentation. The following code example defines the group `'my_group'`:

```
/*!
 * \defgroup my_group Brief description of the group here
 *
 * Detailed description of the group here
*/
```

The following example shows how to document the function `myFunction` and how to add it to the group `my_group:`

```
/*!
 * \brief Brief description of myFunction here
 * \ingroup my_group
 *
 * Detailed description of myFunction here
 *
 * \param[in] param1 Description of \a param1 here
 * \param[in] param2 Description of \a param2 here
 * \return Description of the return value here
 */
TYPE3 myFunction(TYPE1 param1, TYPE2 &param2);
```

6 Main features

A short list of the main features of SimFQT is given below sorted in different categories. Many more features and functions exist and for these we refer to the reference documentation.

6.1 Fare calculation

- Calculation of fare from statistics on tickets/coupons

6.2 Fare rule engine

- Fare rules: storage, engine, management

6.3 Fare retrieval

- Retrieval of fares for specific booking requests or product assesment

6.4 Other features

- CSV input file parsing
- Memory handling

7 Make a Difference

Do not ask what SimFQT can do for you. Ask what you can do for SimFQT.

You can help us to develop the SimFQT library. There are always a lot of things you can do:

- Start using SimFQT
- Tell your friends about SimFQT and help them to get started using it
- If you find a bug, report it to us. Without your help we can never hope to produce a bug free code.
- Help us to improve the documentation by providing information about documentation bugs
- Answer support requests in the SimFQT discussion forums on SourceForge. If you know the answer to a question, help others to overcome their SimFQT problems.
- Help us to improve our algorithms. If you know of a better way (e.g., that is faster or requires less memory) to implement some of our algorithms, then let us know.
- Help to port SimFQT to new platforms. If you manage to compile SimFQT on a new platform, then tell how you did it.
- Send us your code. If you have a good SimFQT compatible code, which you can release under the LGPL, and you think it should be included in SimFQT, then send it to the communauty.
- Become an SimFQT developer. Send us an e-mail and tell what you can do for SimFQT.

8 Make a new release

8.1 Introduction

This document describes briefly the recommended procedure of releasing a new version of SimFQT using a Linux development machine and the SourceForge project site.

The following steps are required to make a release of the distribution package.

8.2 Initialisation

Clone locally the full `Git project`:

```
cd ~
mkdir -p dev/sim
cd ~/dev/sim
git clone git://simfqt.git.sourceforge.net/gitroot/simfqt/simfqt simfqtgit
cd simfqtgit
git checkout trunk
```

8.3 Release branch maintenance

Switch to the release branch, on your local clone, and merge the latest updates from the trunk. Decide about the new version to be released.

```
cd ~/dev/sim/simfqtgit
git checkout releases
git merge trunk
```

Update the version in the various build system files, replacing the old version numbers by the correct ones:

```
vi CMakeLists.txt
vi autogen.sh
vi README
```

Update the version, add some news in the `NEWS` file, add a change-log in the `ChangeLog` file and in the RPM specification files:

```
vi NEWS
vi ChangeLog
vi simfqt.spec
```

8.4 Commit and publish the release branch

Commit the new release:

```
cd ~/dev/sim/simfqtgit
git add -A
git commit -m "[Release 0.5.0] Release of the 0.5.0 version of SimFQT."
git push
```

8.5 Create distribution packages

Create the distribution packages using the following command:

```
cd ~/dev/sim/simfqtgit
git checkout releases
rm -rf build && mkdir -p build
cd build
export INSTALL_BASEDIR=/home/user/dev/deliveries
export LIBSUFFIX_4_CMAKE="-DLIB_SUFFIX=64"
cmake -DCMAKE_INSTALL_PREFIX=${INSTALL_BASEDIR}/simfqt-0.5.0 \
 -DWITH_STDAIR_PREFIX=${INSTALL_BASEDIR}/stdair-stable \
 -DWITH_AIRRAC_PREFIX=${INSTALL_BASEDIR}/airsched-stable \
 -DWITH_AIRRAC_PREFIX=${INSTALL_BASEDIR}/airrac-stable \
 -DWITH_RMOL_PREFIX=${INSTALL_BASEDIR}/rmol-stable \
 -DWITH_RMOL_PREFIX=${INSTALL_BASEDIR}/airinv-stable \
 -DWITH_RMOL_PREFIX=${INSTALL_BASEDIR}/simfqt-stable \
 -DCMAKE_BUILD_TYPE:STRING=Debug -DINSTALL_DOC:BOOL=ON \
 ${LIBSUFFIX_4_CMAKE} ..
make check && make dist
make install
```

This will configure, compile and check the package. The output packages will be named, for instance, `simfqt-0.5.0.tar.gz` and `simfqt-0.5.0.tar.bz2`.

8.6 Upload the HTML documentation to SourceForge

In order to update the Web site files, either:

- `synchronise them with rsync and SSH`: Upload the just generated HTML (and PDF) documentation onto the `SourceForge Web site`.

  ```
  cd ~/dev/sim/simfqtgit/build
  git checkout releases
  rsync -aiv ${INSTALL_BASEDIR}/simfqt-0.5.0/share/doc/simfqt-0.5.0/html/ \
   your_sf_user,simfqt@web.sourceforge.net:htdocs/
  ```

where `-aiv` options mean:

  - `-a`: archive/mirror mode; equals `-rlptgoD` (no `-H`, `-A`, `-X`)
  - `-v`: increase verbosity
  - `-i`: output a change-summary for all updates
  - Note the trailing slashes (/) at the end of both the source and target directories. It means that the content of the source directory (`doc/html`), rather than the directory itself, has to be copied into the content of the target directory.

- or use the `SourceForge Shell service`.

8.7 Generate the RPM packages

Optionally, generate the RPM package (for instance, for `Fedora`/`RedHat`):

```
cd ~/dev/sim/simfqtgit/build
git checkout releases
make dist
```

To perform this step, rpm-build, rpmlint and rpmdevtools have to be available on the system.

```
cp ../simfqt.spec ~/dev/packages/SPECS \
  && cp simfqt-0.5.0.tar.bz2 ~/dev/packages/SOURCES
cd ~/dev/packages/SPECS
rpmbuild -ba simfqt.spec
cd ~/dev/packages
rpmlint -i SPECS/simfqt.spec SRPMS/simfqt-0.5.0-1.fc16.src.rpm \
  RPMS/noarch/simfqt-* RPMS/i686/simfqt-*
```

8.8 Update distributed change log

Update the `NEWS` and `ChangeLog` files with appropriate information, including what has changed since the previous release. Then commit and push the changes into the `SimFQT's Git repository`.

8.9 Create the binary package, including the documentation

Create the binary package, which includes HTML and PDF documentation, using the following command:

```
cd ~/dev/sim/simfqtgit/build
git checkout releases
make package
```

The output binary package will be named, for instance, `simfqt-0.5.0-Linux.tar.bz2`. That package contains both the HTML and PDF documentation. The binary package contains also the executables and shared libraries, as well as C++ header files, but all of those do not interest us for now.

8.10 Upload the files to SourceForge

Upload the distribution and documentation packages to the SourceForge server. Check `SourceForge help page on uploading software`.

8.11 Make a new post

- submit a new entry in the SourceForge project-related news feed
- make a new post on the SourceForge hosted WordPress blog
- and update, if necessary, Trac tickets.

8.12 Send an email on the announcement mailing-list

Finally, you should send an announcement to simfqt-announce@lists.sourceforge.net (see https://lists.sourceforge.net/lists/listinfo/simfqt-announce for the archives)

9 Installation

9.1 Table of Contents

9.2 Fedora/RedHat Linux distributions

Note that on Fedora/RedHat Linux distributions, RPM packages are available and can be installed with your usual package manager. For instance:

```
yum -y install simfqt-devel simfqt-doc
```

RPM packages can also be available on the SourceForge download site.

9.3 SimFQT Requirements

SimFQT should compile without errors or warnings on most GNU/Linux systems, on UNIX systems like Solaris SunOS, and on POSIX based environments for Microsoft Windows like Cygwin or MinGW with MSYS. It can be also built on Microsoft Windows NT/2000/XP/Vista/7 using Microsoft's Visual C++ .NET, but our support for this compiler is limited. For GNU/Linux, SunOS, Cygwin and MinGW we assume that you have at least the following GNU software installed on your computer:

- GNU Autotools:
  - `autoconf`,
  - `automake`,
  - `libtool`,
  - `make`, version 3.72.1 or later (check version with '`make` --version')
- `GCC` - GNU C++ Compiler (g++), version 4.3.x or later (check version with '`gcc` --version')
- `Boost` - C++ STL extensions, version 1.35 or later (check version with '`grep` "define BOOST_-LIB_VERSION" /usr/include/boost/version.hpp')
- `MySQL` - Database client libraries, version 5.0 or later (check version with '`mysql` --version')
- `SOCI` - C++ database client library wrapper, version 3.0.0 or later (check version with '`soci-config` --version')

Optionally, you might need a few additional programs: `Doxygen`, `LaTeX`, `Dvips` and `Ghostscript`, to generate the HTML and PDF documentation.

We strongly recommend that you use recent stable releases of the GCC, if possible. We do not actively work on supporting older versions of the GCC, and they may therefore (without prior notice) become unsupported in future releases of SimFQT.

9.4 Basic Installation

Briefly, the shell commands '`./cmake .. && make install`' should configure, build, and install this package. The following more-detailed instructions are generic; see the '`README`' file for instructions specific to this package. Some packages provide this '`INSTALL`' file but do not implement all of the features documented below. The lack of an optional feature in a given package is not necessarily a bug. More recommendations for GNU packages can be found in the info page corresponding to "Makefile Conventions: (standards)Makefile Conventions".

The '`cmake`' shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a '`Makefile`' in each directory of the package. It may also create one or more '.h' files containing system-dependent definitions. Finally, it creates a '`CMakeCache`.txt' cache file that you can refer to in the future to recreate the current configuration, and a file '`CMakeFiles`' containing compiler output (useful mainly for debugging '`cmake`').

It can also use an optional file (typically called '`config`.cache' and enabled with '`--cache-file=config`.cache' or simply '`-C`') that saves the results of its tests to speed up reconfiguring. Caching is disabled by default to prevent problems with accidental use of stale cache files.

If you need to do unusual things to compile the package, please try to figure out how 'configure' could check whether to do them, and mail diffs or instructions to the address given in the '`README`' so they can be considered for the next release. If you are using the cache, and at some point '`config`.cache' contains results you don't want to keep, you may remove or edit it.

The file `CMakeLists.txt' is used to create the `Makefile' files.

The simplest way to compile this package is:

1. `cd' to the directory containing the package's source code and type `./cmake ..' to configure the package for your system. Running `cmake' is generally fast. While running, it prints some messages telling which features it is checking for.
2. Type `make' to compile the package.
3. Optionally, type `make check'to run any self-tests that come with the package, generally using the just-built uninstalled binaries.
4. Type `make install' to install the programs and any data files and documentation. When installing into a prefix owned by root, it is recommended that the package be configured and built as a regular user, and only the `make install' phase executed with root privileges.
5. You can remove the program binaries and object files from the source code directory by typing `make clean'. To also remove the files that `configure' created (so you can compile the package for a different kind of computer), type `make distclean'. There is also a `make maintainer-clean' target, but that is intended mainly for the package's developers. If you use it, you may have to get all sorts of other programs in order to regenerate files that came with the distribution.
6. Often, you can also type `make uninstall' to remove the installed files again. In practice, not all packages have tested that uninstallation works correctly, even though it is required by the GNU Coding Standards.

9.5 Compilers and Options

Some systems require unusual options for compilation or linking that the `cmake' script does not know about. Run `./cmake --help' for details on some of the pertinent environment variables.

You can give `cmake' initial values for configuration parameters by setting variables in the command line or in the environment. Here is an example:

```
./cmake  CC=c99 CFLAGS=-g LIBS=-lposix
```

See also:

Defining Variables for more details.

9.6 Compiling For Multiple Architectures

You can compile the package for more than one kind of computer at the same time, by placing the object files for each architecture in their own directory. To do this, you can use GNU `make'. `cd' to the directory where you want the object files and executables to go and run the 'configure' script. 'configure' automatically checks for the source code in the directory that 'configure' is in and in '..'. This is known as a "VPATH" build.

With a non-GNU 'make', it is safer to compile the package for one architecture at a time in the source code directory. After you have installed the package for one architecture, use 'make distclean' before reconfiguring for another architecture.

On MacOS X 10.5 and later systems, you can create libraries and executables that work on multiple system types--known as "fat" or "universal" binaries--by specifying multiple '-arch' options to the compiler but only a single '-arch' option to the preprocessor. Like this:

```
./configure CC="gcc -arch i386 -arch x86_64 -arch ppc -arch ppc64" \
            CXX="g++ -arch i386 -arch x86_64 -arch ppc -arch ppc64" \
            CPP="gcc -E" CXXCPP="g++ -E"
```

This is not guaranteed to produce working output in all cases, you may have to build one architecture at a time and combine the results using the 'lipo' tool if you have problems.

9.7 Installation Names

By default, 'make install' installs the package's commands under '/usr/local/bin', include files under '/usr/local/include', etc. You can specify an installation prefix other than '/usr/local' by giving 'configure' the option '--prefix=PREFIX', where PREFIX must be an absolute file name.

You can specify separate installation prefixes for architecture-specific files and architecture-independent files. If you pass the option '--exec-prefix=PREFIX' to 'configure', the package uses PREFIX as the prefix for installing programs and libraries. Documentation and other data files still use the regular prefix.

In addition, if you use an unusual directory layout you can give options like '--bindir=DIR' to specify different values for particular kinds of files. Run 'configure --help' for a list of the directories you can set and what kinds of files go in them. In general, the default for these options is expressed in terms of '${prefix}', so that specifying just '--prefix' will affect all of the other directory specifications that were not explicitly provided.

The most portable way to affect installation locations is to pass the correct locations to 'configure'; however, many packages provide one or both of the following shortcuts of passing variable assignments to the 'make install' command line to change installation locations without having to reconfigure or recompile.

The first method involves providing an override variable for each affected directory. For example, 'make install prefix=/alternate/directory' will choose an alternate location for all directory configuration variables that were expressed in terms of '${prefix}'. Any directories that were specified during 'configure', but not in terms of '${prefix}', must each be overridden at install time for the entire installation to be relocated. The approach of makefile variable overrides for each directory variable is required by the GNU Coding Standards, and ideally causes no recompilation. However, some platforms have known limitations with the semantics of shared libraries that end up requiring recompilation when using this method, particularly noticeable in packages that use GNU Libtool.

The second method involves providing the 'DESTDIR' variable. For example, 'make install DESTDIR=/alternate/directory' will prepend '/alternate/directory' before all installation names. The approach of 'DESTDIR' overrides is not required by the GNU Coding Standards, and does not work on platforms that have drive letters. On the other hand, it does better at avoiding recompilation issues, and works well even when some directory options were not specified in terms of '${prefix}' at 'configure' time.

9.8 Optional Features

If the package supports it, you can cause programs to be installed with an extra prefix or suffix on their names by giving 'cmake' the option '--program-prefix=PREFIX' or '--program-suffix=SUFFIX'.

Some packages pay attention to '--enable-FEATURE' options to 'configure', where FEATURE indicates an optional part of the package. They may also pay attention to '--with-PACKAGE' options, where PACKAGE is something like 'gnu-as' or 'x' (for the X Window System). The 'README' should mention any '--enable-' and '--with-' options that the package recognizes.

For packages that use the X Window System, 'configure' can usually find the X include and library files automatically, but if it doesn't, you can use the 'configure' options '--x-includes=DIR' and '--x-libraries=DIR' to specify their locations.

Some packages offer the ability to configure how verbose the execution of 'make' will be. For these packages, running './configure --enable-silent-rules' sets the default to minimal output, which can be overridden with 'make V=1'; while running './configure --disable-silent-rules' sets the default to verbose, which can be overridden with 'make V=0'.

9.9 Particular systems

On HP-UX, the default C compiler is not ANSI C compatible. If GNU CC is not installed, it is recommended to use the following options in order to use an ANSI C compiler:

```
./configure CC="cc -Ae -D_XOPEN_SOURCE=500"
```

and if that doesn't work, install pre-built binaries of GCC for HP-UX.

On OSF/1 a.k.a. Tru64, some versions of the default C compiler cannot parse its '<wchar.h>' header file. The option '-nodtk' can be used as a workaround. If GNU CC is not installed, it is therefore recommended to try

```
./configure CC="cc"
```

and if that doesn't work, try

```
./configure CC="cc -nodtk"
```

On Solaris, don't put '/usr/ucb' early in your 'PATH'. This directory contains several dysfunctional programs; working variants of these programs are available in '/usr/bin'. So, if you need '/usr/ucb' in your 'PATH', put it _after_ '/usr/bin'.

On Haiku, software installed for all users goes in '/boot/common', not '/usr/local'. It is recommended to use the following options:

```
./cmake -DCMAKE_INSTALL_PREFIX=/boot/common
```

9.10 Specifying the System Type

There may be some features 'configure' cannot figure out automatically, but needs to determine by the type of machine the package will run on. Usually, assuming the package is built to be run on the _same_ architectures, 'configure' can figure that out, but if it prints a message saying it cannot guess the machine type, give it the '--build=TYPE' option. TYPE can either be a short name for the system type, such as 'sun4', or a canonical name which has the form CPU-COMPANY-SYSTEM

where SYSTEM can have one of these forms:

- OS
- KERNEL-OS

See the file 'config.sub' for the possible values of each field. If 'config.sub' isn't included in this package, then this package doesn't need to know the machine type.

If you are _building_ compiler tools for cross-compiling, you should use the option '--target=TYPE' to select the type of system they will produce code for.

If you want to _use_ a cross compiler, that generates code for a platform different from the build platform, you should specify the "host" platform (i.e., that on which the generated programs will eventually be run) with '--host=TYPE'.

9.11 Sharing Defaults

If you want to set default values for 'configure' scripts to share, you can create a site shell script called 'config.site' that gives default values for variables like 'CC', 'cache_file', and 'prefix'. 'configure' looks for 'PREFIX/share/config.site' if it exists, then 'PREFIX/etc/config.site' if it exists. Or, you can set the 'CONFIG_-SITE' environment variable to the location of the site script. A warning: not all 'configure' scripts look for a site script.

9.12 Defining Variables

Variables not defined in a site shell script can be set in the environment passed to 'configure'. However, some packages may run configure again during the build, and the customized values of these variables may be lost. In order to avoid this problem, you should set them in the 'configure' command line, using 'VAR=value'. For example:

```
./configure CC=/usr/local2/bin/gcc
```

causes the specified 'gcc' to be used as the C compiler (unless it is overridden in the site shell script).

Unfortunately, this technique does not work for 'CONFIG_SHELL' due to an Autoconf bug. Until the bug is fixed you can use this workaround:

```
CONFIG_SHELL=/bin/bash /bin/bash ./configure CONFIG_SHELL=/bin/bash
```

9.13 'cmake' Invocation

'cmake' recognizes the following options to control how it operates.

- '--help', '-h' print a summary of all of the options to 'cmake', and exit.
- '--help=short', '--help=recursive' print a summary of the options unique to this package's 'configure', and exit. The 'short' variant lists options used only in the top level, while the 'recursive' variant lists options also present in any nested packages.
- '--version', '-V' print the version of Autoconf used to generate the 'configure' script, and exit.
- '--cache-file=FILE' enable the cache: use and save the results of the tests in FILE, traditionally 'config.cache'. FILE defaults to '/dev/null' to disable caching.
- '--config-cache', '-C' alias for '--cache-file=config.cache'.
- '--quiet', '--silent', '-q' do not print messages saying which checks are being made. To suppress all normal output, redirect it to '/dev/null' (any error messages will still be shown).
- '--srcdir=DIR' look for the package's source code in directory DIR. Usually 'configure' can determine that directory automatically.
- '--prefix=DIR' use DIR as the installation prefix.

See also:

Installation Names for more details, including other options available for fine-tuning the installation locations.

- '--no-create', '-n' run the configure checks, but stop before creating any output files.

'cmake' also accepts some other, not widely useful, options. Run 'cmake' --help' for more details.

The 'cmake' script produces an ouput like this:

```
-- Requires Git without specifying any version
cmake -DCMAKE_INSTALL_PREFIX=/home/user/dev/deliveries/simfqt-0.5.0 \
 -DWITH_STDAIR_PREFIX=/home/user/dev/deliveries/stdair-stable \
 -DLIB_SUFFIX=64 -DCMAKE_BUILD_TYPE:STRING=Debug -DINSTALL_DOC:BOOL=ON ..
-- Current Git revision name: 0e31d63879056d26f01eb09757d232d247c42164 trunk
-- Requires Boost-1.41
-- Found Boost version: 1.44.0
-- Requires Readline without specifying any version
-- Found Readline version: 6.1
-- Requires MySQL without specifying any version
-- Using mysql-config: /usr/bin/mysql_config
-- Found MySQL version: 5.1.56
-- Requires SOCI-3.0
-- Using soci-config: /usr/bin/soci-config
-- SOCI headers are buried
-- Found SOCI with MySQL back-end support version: 3.0.0
-- Requires StdAir-0.35
-- Found StdAir version: 99.99.99
-- Requires Doxygen without specifying any version
-- Found Doxygen version: 1.7.4
-- Had to set the linker language for 'simfqtlib' to CXX
-- Test 'FQTTestSuite' to be built with 'FQTTestSuite.cpp'
--
-- ============================================================
-- --------------------------------
-- ---     Project Information    ---
-- --------------------------------
-- PROJECT_NAME .................. : simfqt
-- PACKAGE_PRETTY_NAME ........... : SimFQT
-- PACKAGE ....................... : simfqt
-- PACKAGE_NAME .................. : SIMFQT
-- PACKAGE_BRIEF ................. : C++ Simulated Fare Quote System Library
-- PACKAGE_VERSION ............... : 99.99.99
-- GENERIC_LIB_VERSION ........... : 99.99.99
-- GENERIC_LIB_SOVERSION ......... : 99.99
--
-- --------------------------------
-- ---     Build Configuration    ---
-- --------------------------------
-- Modules to build .............. : simfqt
-- Libraries to build/install ..... : simfqtlib
-- Binaries to build/install ...... : simfqt;fareQuote
-- Modules to test ............... : simfqt
-- Binaries to test .............. : FQTTestSuitetst
--
-- * Module ...................... : simfqt
--   + Layers to build ........... : .;basic;bom;factory;command;service
--   + Dependencies on other layers :
--   + Libraries to build/install . : simfqtlib
--   + Executables to build/install : simfqt;fareQuote
--   + Tests to perform .......... : FQTTestSuitetst
--
-- BUILD_SHARED_LIBS ............. : ON
-- CMAKE_BUILD_TYPE ............... : Debug
--  * CMAKE_C_FLAGS ............... :
--  * CMAKE_CXX_FLAGS ............. : -Wall -Werror
--  * BUILD_FLAGS ................. :
--  * COMPILE_FLAGS ............... :
-- CMAKE_MODULE_PATH .............. : /home/localoriuser/dev/sim/simfqt/simfqtgit/config/
-- CMAKE_INSTALL_PREFIX ........... : /home/localoriuser/dev/deliveries/simfqt-99.99.99
--
-- * Doxygen:
--   - DOXYGEN_VERSION ............. : 1.7.4
--   - DOXYGEN_EXECUTABLE ........... : /usr/bin/doxygen
--   - DOXYGEN_DOT_EXECUTABLE ....... : DOXYGEN_DOT_EXECUTABLE-NOTFOUND
--   - DOXYGEN_DOT_PATH ............. :
--
-- ----------------------------------
-- --- Installation Configuration ---
-- ----------------------------------
-- INSTALL_LIB_DIR ................ : /home/localoriuser/dev/deliveries/simfqt-99.99.99/lib
-- INSTALL_BIN_DIR ................ : /home/localoriuser/dev/deliveries/simfqt-99.99.99/bin
-- INSTALL_INCLUDE_DIR ............ : /home/localoriuser/dev/deliveries/simfqt-99.99.99/include
-- INSTALL_DATA_DIR ............... : /home/localoriuser/dev/deliveries/simfqt-99.99.99/share
-- INSTALL_SAMPLE_DIR ............. : /home/localoriuser/dev/deliveries/simfqt-99.99.99/share/simfqt/sampl
-- INSTALL_DOC .................... : ON
--
-- ----------------------------------
-- ---   Packaging Configuration  ---
-- ----------------------------------
-- CPACK_PACKAGE_CONTACT .......... : Denis Arnaud <denis_arnaud - at - users dot sourceforge dot net>
-- CPACK_PACKAGE_VENDOR ........... : Denis Arnaud
-- CPACK_PACKAGE_VERSION .......... : 99.99.99
-- CPACK_PACKAGE_DESCRIPTION_FILE . : /home/localoriuser/dev/sim/simfqt/simfqtgit/README
-- CPACK_RESOURCE_FILE_LICENSE .... : /home/localoriuser/dev/sim/simfqt/simfqtgit/COPYING
-- CPACK_GENERATOR ................ : TBZ2
-- CPACK_DEBIAN_PACKAGE_DEPENDS ... :
-- CPACK_SOURCE_GENERATOR ......... : TBZ2;TGZ
-- CPACK_SOURCE_PACKAGE_FILE_NAME . : simfqt-99.99.99
--
-- ---------------------------------
-- ---     External libraries    ---
-- ---------------------------------
--
-- * Boost:
--   - Boost_VERSION ............. : 104400
--   - Boost_LIB_VERSION .......... : 1_44
--   - Boost_HUMAN_VERSION ........ : 1.44.0
--   - Boost_INCLUDE_DIRS ......... : /usr/include
--   - Boost required components .. : program_options;date_time;iostreams;serialization;filesystem;unit_te
--   - Boost required libraries ... : optimized;/usr/lib/libboost_iostreams-mt.so;debug;/usr/lib/libboost_
--
-- * Readline:
--   - READLINE_VERSION ........... : 6.1
--   - READLINE_INCLUDE_DIR ....... : /usr/include
--   - READLINE_LIBRARY ........... : /usr/lib/libreadline.so
--
-- * MySQL:
--   - MYSQL_VERSION .............. : 5.1.56
--   - MYSQL_INCLUDE_DIR .......... : /usr/include/mysql
--   - MYSQL_LIBRARIES ............ : /usr/lib/mysql/libmysqlclient_r.so
--
-- * SOCI:
--   - SOCI_VERSION ............... : 3.0.0
--   - SOCI_INCLUDE_DIR ........... : /usr/include/soci
--   - SOCIMYSQL_INCLUDE_DIR ...... : /usr/include/soci
--   - SOCI_LIBRARIES ............. : /usr/lib/libsoci_core.so
--   - SOCIMYSQL_LIBRARIES ........ : /usr/lib/libsoci_mysql.so
--
-- * StdAir:
```

```
--   - STDAIR_VERSION ............ : 99.99.99
--   - STDAIR_BINARY_DIRS ......... : /home/localoriuser/dev/deliveries/stdair-0.3.0/bin
--   - STDAIR_EXECUTABLES ......... : stdair
--   - STDAIR_LIBRARY_DIRS ........ : /home/localoriuser/dev/deliveries/stdair-0.3.0/lib
--   - STDAIR_LIBRARIES ........... : stdairlib;stdairuicllib
--   - STDAIR_INCLUDE_DIRS ........ : /home/localoriuser/dev/deliveries/stdair-0.3.0/include
--   - STDAIR_SAMPLE_DIR .......... : /home/localoriuser/dev/deliveries/stdair-0.3.0/share/stdair/samples
--
-- Change a value with: cmake -D<Variable>=<Value>
-- ============================================================
--
-- Configuring done
-- Generating done
-- Build files have been written to: /home/localoriuser/dev/sim/simfqt/simfqtgit/build
```

It is recommended that you check if your library has been compiled and linked properly and works as expected. To do so, you should execute the testing process 'make check'. As a result, you should obtain a similar report:

```
[  0%] Built target hdr_cfg_simfqt
[ 90%] Built target simfqtlib
[100%] Built target FQTTestSuitetst
Test project /home/localoriuser/dev/sim/simfqt/simfqtgit/build/test/simfqt
    Start 1: FQTTestSuitetst
1/1 Test #1: FQTTestSuitetst ..................   Passed    0.43 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   0.47 sec
[100%] Built target check_simfqttst
[100%] Built target check
```

Check if all the executed tests PASSED. If not, please contact us by filling a bug-report.

Finally, you should install the compiled and linked library, include files and (optionally) HTML and PDF documentation by typing:

```
make install
```

Depending on the PREFIX settings during configuration, you might need the root (administrator) access to perform this step.

Eventually, you might invoke the following command

```
make clean
```

to remove all files created during compilation process, or even

```
cd ~/dev/sim/simfqtgit
rm -rf build && mkdir build
cd build
```

to remove everything.

10 Linking with SimFQT

10.1 Table of Contents

10.2 Introduction

There are two convenient methods of linking your programs with the SimFQT library. The first one employs the `pkg-config' command (see http://pkgconfig.freedesktop.org/), whereas the second one uses `simfqt-config' script. These methods are shortly described below.

10.3 Dependencies

The SimFQT library depends on several other C++ components.

10.3.1 StdAir

Among them, as for now, only StdAir has been packaged. The support for StdAir is taken in charge by a dedicated M4 macro file (namely, `stdair.m4'), from the configuration script (generated thanks to `configure.ac').

10.4 Using the pkg-config command

`pkg-config' is a helper tool used when compiling applications and libraries. It helps you insert the correct compiler and linker options. The syntax of the `pkg-config' is as follows:

```
pkg-config <options> <library_name>
```

For instance, assuming that you need to compile an SimFQT based program `my_prog.cpp', you should use the following command:

```
g++ `pkg-config --cflags simfqt` -o my_prog my_prog.cpp \
 `pkg-config --libs simfqt`
```

For more information see the `pkg-config' man pages.

10.5 Using the simfqt-config script

SimFQT provides a shell script called `simfqt-config`, which is installed by default in `$prefix/bin` (`/usr/local/bin`) directory. It can be used to simplify compilation and linking of SimFQT based programs. The usage of this script is quite similar to the usage of the `pkg-config` command.

Assuming that you need to compile the program `my_prog`.cpp' you can now do that with the following command:

```
g++ `simfqt-config --cflags` -o my_prog my_prog.cpp `simfqt-config --libs`
```

A list of `simfqt-config` options can be obtained by typing:

```
simfqt-config --help
```

If the `simfqt-config` command is not found by your shell, you should add its location `$prefix/bin` to the `PATH` environment variable, e.g.:

```
export PATH=/usr/local/bin:$PATH
```

10.6 M4 macro for the GNU Autotools

A M4 macro file is delivered with SimFQT, namely 'simfqt.m4', which can be found in, e.g., '/usr/share/a-clocal'. When used by a 'configure' script, thanks to he `AM_PATH_SIMFQT` macro (specified in the M4 macro file), the following Makefile variables are then defined:

- `SIMFQT_VERSION` (e.g., defined to 0.2.0)
- `SIMFQT_CFLAGS` (e.g., defined to `-I${prefix}/include`)
- `SIMFQT_LIBS` (e.g., defined to `-L${prefix}/lib -lsimfqt`)

10.7 Using SimFQT with dynamic linking

When using static linking some of the library routines in SimFQT are copied into your executable program. This can lead to unnecessary large executables. To avoid having too large executable files you may use dynamic linking instead. Dynamic linking means that the actual linking is performed when the program is executed. This requires that the system is able to locate the shared SimFQT library file during your program execution. If you install the SimFQT library using a non-standard prefix, the `LD_LIBRARY_-PATH` environment variable might be used to inform the linker of the dynamic library location, e.g.:

```
export LD_LIBRARY_PATH=<SimFQT installation prefix>/lib:$LD_LIBRARY_PATH
```

11 Test Rules

This section describes rules how the functionality of the SimFQT library should be verified. In the `tests` subdirectory test files are provided. All functionality should be tested using these test files.

11.1 The Test File

Each new SimFQT module/class should be accompanied with a test file. The test file is an implementation in C++ that tests the functionality of a function/class or a group of functions/classes called modules. The test file should test relevant parameter settings and input/output relations to guarantee correct functionality of the corresponding classes/functions. The test files should be maintained using version control and updated whenever new functionality is added to the SimFQT library.

The test file should print relevant data to a standard output that can be used to verify the functionality. All relevant parameter settings should be tested.

The test file should be placed in the `'tests'` subdirectory and should have a name ending with `'_test.cpp'`.

11.2 The Reference File

Consider a test file named `'module_test.cpp'`. A reference file named `'module_test.ref'` should accompany the test file. The reference file contains a reference printout of the standard output generated when running the test program. The reference file should be maintained using version control and updated according to the test file.

11.3 Testing SimFQT Library

One can compile and execute all test programs from `'tests'` subdirectory by typing

```
% make check
```

after successful compilation of the SimFQT library.

12 Users Guide

12.1 Table of Contents

12.2 Introduction

The `SimFQT` library contains classes for fare rule management. This document does not cover all the aspects of the `SimFQT` library. It does however explain the most important things you need to know in order to start using `SimFQT`.

12.3 Get Started

12.3.1 Get the SimFQT library

Clone locally the full `Git project`:

```
cd ~
mkdir -p dev/sim
cd ~/dev/sim
git clone git://simfqt.git.sourceforge.net/gitroot/simfqt/simfqt simfqtgit
cd simfqtgit
git checkout trunk
```

12.3.2 Build the SimFQT project

Link with StdAir, create the distribution package (say, 0.5.0) and compile using the following commands:

```
cd ~/dev/sim/simfqtgit
rm -rf build && mkdir -p build
cd build
cmake -DCMAKE_INSTALL_PREFIX=~/dev/deliveries/simfqt-0.5.0 \
 -DWITH_STDAIR_PREFIX=~/dev/deliveries/stdair-stable \
 -DCMAKE_BUILD_TYPE:STRING=Debug -DINSTALL_DOC:BOOL=ON ..
make
```

12.3.3 Run the Tests

After building the SimFQT project, the following commands run the tests:

```
cd ~/dev/sim/simfqtgit
cd build
make check
```

As a result, you should obtain a similar report:

```
[  0%] Built target hdr_cfg_simfqt
[ 90%] Built target simfqtlib
[100%] Built target FQTTestSuitetst
```

```
Test project /home/localoriuser/dev/sim/simfqt/simfqtgit/build/test/simfqt
    Start 1: FQTTestSuitetst
1/1 Test #1: FQTTestSuitetst ..................   Passed    0.15 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   0.16 sec
[100%] Built target check_simfqttst
[100%] Built target check
```

12.3.4 Install the SimFQT Project (Binaries, Documentation)

After the step Build the SimFQT project, to install the library and its header files, type:

```
cd ~/dev/sim/simfqtgit
cd build
make install
```

You can check that the executables and other required files have been copied into the given final directory:

```
cd ~dev/deliveries/simfqt-0.5.0
```

To generate the SimFQT project documentation, the commands are:

```
cd ~/dev/sim/simfqtgit
cd build
make doc
```

The SimFQT project documentation is available in the following formats: HTML, LaTeX. Those documents are available in a subdirectory:

```
cd ~/dev/sim/simfqtgit
cd build
cd doc
```

12.4 Input file of SimFQT Project

The fare input file structure should look like the following sample:

```
\textcolor{comment}{// Fares: fare ID; OriginCity; DestinationCity; TripType; DateRangeStart; DateRan
      geEnd; DepartureTimeRangeStart; DepartureTimeRangeEnd; POS; CabinCode; Channel; A
      dvancePurchase; SaturdayNight; ChangeFees; NonRefundable; MinimumStay; Price; nb
      Segments}
\textcolor{comment}{// Segment: AirlineCode; Class; }
1; SIN; BKK; OW; 2010-01-15; 2010-12-31; 00:00; 23:59; SIN; Y; IN; 7; T; T; T; 3;
       150.0; SQ; Y;
2; SIN; BKK; OW; 2010-01-15; 2010-12-31; 00:00; 23:59; BKK; Y; IN; 7; T; T; T; 3;
       150.0; SQ; Y;
3; SIN; HND; OW; 2010-01-15; 2010-12-31; 00:00; 23:59; SIN; Y; IN; 7; T; T; T; 3;
       150.0; SQ; Y;
4; SIN; HND; OW; 2010-01-15; 2010-12-31; 00:00; 23:59; HND; Y; IN; 7; T; T; T; 3;
       150.0; SQ; Y;
5; SIN; HND; OW; 2010-01-15; 2010-12-31; 00:00; 23:59; ROW; Y; IN; 7; T; T; T; 3;
       150.0; SQ; Y;
6; SIN; BKK; OW; 2010-01-15; 2010-12-31; 00:00; 23:59; SIN; Y; IF; 7; T; T; T; 3;
       150.0; SQ; Y;
7; SIN; BKK; OW; 2010-01-15; 2010-12-31; 00:00; 23:59; BKK; Y; IF; 7; T; T; T; 3;
       150.0; SQ; Y;
```

```
8; SIN; HND; OW; 2010-01-15; 2010-12-31; 00:00; 23:59; SIN; Y; IF; 7; T; T; T; 3;
       150.0; SQ; Y;
9; SIN; HND; OW; 2010-01-15; 2010-12-31; 00:00; 23:59; HND; Y; IF; 7; T; T; T; 3;
       150.0; SQ; Y;
10; SIN; HND; OW; 2010-01-15; 2010-12-31; 00:00; 23:59; ROW; Y; IF; 7; T; T; T; 3
      ; 150.0; SQ; Y;
```

Each line represents a fare rule (see `SIMFQT::FareRuleStruct`), i.e., each line tells us the price a customer will be asked according to a lot of criteria such as:

- the origin and destination of his travel (for instance from Singapour to Bangok in the first fare rule).
- the type of his travel, i.e. one-way "OW" or round trip "RT".
- the date and time he is willing to travel (each fare rule has a date range and a time range of validity).
- the place where he is buying the ticket, i.e. the point of sale.
- his prefered cabin.
- the channel of the booking described by a two letters code: direct(D)/indirect(I) and online(N)/offline(F).
- the date when he wants to buy the ticket, i.e. the advanced purchase required in number of days.
- the saturday night stay option, i.e. is he staying a staturday night between his inbound trip and his outbound one? "T" stands for true and "F" stands for false.
- the change fees option, i.e. are there fees to change his ticket? "T" stands for true and "F" stands for false.
- the refundable criterion, i.e. is the ticket refundable? "T" stands for true and "F" stands for false.
- the number of days he is willing to stay at the destination location (each fare rule has a minimum stay requirement in number of days).

Some fare input examples (including the example above named fare01.csv) are given in the stdair::samples directory.

12.5 The fare quoting BOM Tree

The Fare Quoting Business Object Model (BOM) tree is a structure permitting to store all the `SIMFQT::FareRuleStruct` objects of the simulation. That is why, the BOM tree is built parsing the fare file containing all the fare rules (as described in the previous section Input file of SimFQT Project). For convenience and first use of SimFQT (the input fare file building can be long and heavy), SimFQT API enables to build a small default BOM tree.

12.5.1 Build of the fare quoting BOM tree

First, a BOM root object (i.e., a root for all the classes in the project) is instantiated by the `stdair::STDAIR_ServiceContext` context object, when the `stdair::STDAIR_Service` is itself instantiated, that is to say during the instanciation of the `simfqt::SIMFQT_Service` object. The corresponding `type` (class) `stdair::BomRoot` is defined in the StdAir library.

Then, the BOM root can be either constructed thanks to the `simfqt::SIMFQT_Service::buildSampleBom()` method:

```
    \textcolor{keywordtype}{void} buildSampleBom();
```

or can be constructed using the fare dump file described above thanks to the `simfqt::SIMFQT_-Service::parseAndLoad (const stdair::Filename_T&)` method:

```
    \textcolor{keywordtype}{void} parseAndLoad (\textcolor{keyword}{const} FareFilePath& iFareFilename);
```

12.5.2 Display of the fare quoting BOM tree

The fare quoting BOM tree can be displayed as done in the `batches::simfqt.cpp` program:

When the default bom tree is used (`-b` option of the main program `simfqt.cpp`), the fare quoting BOM tree display should look like:

```
===============================================================
BomRoot:  -- ROOT --
===============================================================
+++++++++++++++++++++++++++++++++++++++++++++++++
AirportPair: LHR, SYD
+++++++++++++++++++++++++++++++++++++++++++++++++
------------------------------------------
DatePeriod: [2011-Jan-15/2011-Dec-30]
------------------------------------------
******************************************
PosChannel: LHR,DN
******************************************
----------------------------------------
TimePeriod: 00:00:00-23:00:00
----------------------------------------
--------------------------------------
Fare-Features: RT -- 0-1-1-1-0
--------------------------------------
------------------------------------
AirlineClassList: BA Y
------------------------------------
```

Here the fare quoting BOM tree is just composed of one fare rule.

12.5.3 Structure of the fare quoting BOM tree

As one can guess looking at the BOM tree display above, the tree is constructed as follow:

- At the top of the tree, we find a `stdair::BomRoot` object (i.e., a root for all the classes in the project).
- Just under the root, at the first level, we find `stdair::AirportPair` objects (i.e., all the possible combinations of origin-destination). In the instance above, the only combination possible is from London to Sydney.
- At the next level, under a particular `stdair::AirportPair`, we find all the date periods of the fare rules applicable for this origin-destination.
- Then, under a particular `stdair::DatePeriod`, we find all the possible combinations of point-of-sale and channel applicable.

- Under a particular `stdair::PosChannel` object, we have the correponding `stdair::TimePeriod` objects.
- At the next-to-last level, we have `stdair::FareFeatures` objects, that is to say the trip type, the advanced purchase and stay duration required, ...
- Finally we find the code of the airline publishing the current fare rule and the applicable class code.

12.6 The fare quoting procedure

The project SimFQT aims at fare quoting a list of `travel solutions` corresponding to a `booking request`. The fare quoter looks for all the fare rules matching a travel solution: when a fare rule matches, it creates a `fare option` object and adds this object to the current travel solution.

A few steps:

- Instanciate the default booking request
- Instanciate the default travel solution list
- Fare Quoting a list of travel solution

12.6.1 Instanciate the default booking request

A default booking request can be built using the `simfqt::SIMFQT_Service::buildBookingRequest` method:

```
stdair::BookingRequestStruct buildBookingRequest(\textcolor{keyword}{const} \textcolor{keywordtype}{bo
  ;
```

12.6.2 Instanciate the default travel solution list

In the following sample, a list of travel solutions is given as input/output parameter of the `simfqt::SIMFQT_Service::buildSampleTravelSolutions` method:

```
\textcolor{keywordtype}{void} buildSampleTravelSolutions (stdair::TravelSolutionList\_T&);
```

12.6.3 Fare Quoting a list of travel solution

Once a booking request, its correponding list of travel solutions and the fare Quote BOM tree are constructed, the main fonction of the module can be called:

```
\textcolor{keywordtype}{void} quotePrices (\textcolor{keyword}{const} stdair::BookingRequestStruct&,
                  stdair::TravelSolutionList\_T&);
```

For each travel solution of the list, the applicable fare rules are picked from the BOM tree (information such as the trip type or the booking request date are only contained into the booking request, that is why we need this object too).

Each chosen fare rule enables to create a fare option structure which is finally stored into the travel solution.

12.7 Error Messages

This section lists the fatal errors you may encounter when using SimFQT:

- Fare input file not found
- Fare input file can not be parsed
- Error Messages for missing fare rules

12.7.1 Fare input file not found

In this case, the output error message will be similar to:

```
terminate called after throwing an instance of 'SIMFQT::FareInputFileNotFoundException'
  what():  The fare input file '~/<YourFileName>.csv' does not exist or can not be read
Aborted
```

You can check:

- the given path to your input file is correct.
- the specified file name <YourFileName> is correct.
- the file permission settings: is the file "readable"?.

12.7.2 Fare input file can not be parsed

This error message means that your input file has been opened but has not been fully read.

```
terminate called after throwing an instance of 'SIMFQT::FareFileParsingFailedException'
  what():  Parsing of fare input file: ~/<YourFileName>.csv failed
Aborted
```

Your input file structure is somehow incorrect. See the tutorial section How to build a fare input file?.

12.7.3 Error Messages for missing fare rules

If you obtain one of the error messages below and you are currently using your own input file, that means it has been fully read. However, at least one fare rule is missing to complete the fare quote.

- If your error message is about a missing airport pair, you should obtain a similar report:

  ```
  terminate called after throwing an instance of 'SIMFQT::AirportPairNotFoundException'
    what():  No available fare rule for the Origin-Destination pair: xxx, xxx
  Aborted
  ```

  You need to be sure that all your travel solutions have at least one corresponding origin-destination fare rule. It seems you should add one origin-destination (i.e., xxx, xxx) fare rule into your input file.

- If your error message is about a missing fare rule for a flight date, you should obtain a similar report:

  ```
  terminate called after throwing an instance of 'SIMFQT::FlightDateNotFoundException'
    what():  No available fare rule for the flight date x, xxxx-xxx-xx and to the Origin-Destination pa
  Aborted
  ```

You need to be sure that all your travel solutions have at least one corresponding fare rule: same origin-destination and valid date range. It seems you should add/change a fare rule with the Origin-Destination pair: xxx, xxx: its date range must include the flight date xxxx-xxx-xx.

- If your error message is about a missing fare rule for a point-of sale and/or channel, you should obtain a similar report:

```
terminate called after throwing an instance of 'SIMFQT::PosOrChannelNotFoundException'
  what():  No available fare rule for the point of sale xxx, the channel xx, the flight date x, xxxx-
Aborted
```

You need to be sure that all your travel solutions have at least one corresponding fare rule: same origin-destination, valid date range, same point-of-sale and same channel. It seems you should add/change a fare rule to have the same combination as given in the output error message: "the point of sale xxx, the channel xx, the flight date x, xxxx-xxx-xx and the Origin-Destination pair: xxx, xxx".

- If your error message is about a missing fare rule for a flight time, you should obtain a similar report:

```
terminate called after throwing an instance of 'SIMFQT::FlightTimeNotFoundException'
  what():  No available fare rule corresponding to 'xx; x, xxxx-xxx-xx; xxx, xxx; xx:xx' (parsed key)
Aborted
```

You need to be sure that all your travel solutions have at least one corresponding fare rule: same origin-destination, valid date range, same point-of-sale, same channel and valid time range. Add/change a fare rule if necessary.

- If your error message is about a missing fare rule for some features, you should obtain a similar report:

```
terminate called after throwing an instance of 'SIMFQT::FeaturesNotFoundException'
  what():  No available fare rule corresponding to a trip type xx, to a stay duration of x, to a requ
Aborted
```

You need to be sure that all your travel solutions have at least one corresponding fare rule: same origin-destination, valid date range, same point-of-sale, same channel, valid time range and valid features. The features are:

  - the trip type. Maybe you need both "OW" (One-Way) and "RT" (Round-trip) fare rules?
  - the minimum stay duration. You can try "0" for this parameter to include all the possible stay durations.
  - the advance purchase. You can try "0" for this parameter to include all the booking requests up to departure date.

- If your error message is about a missing fare rule for an airline, you should obtain a similar report:

```
terminate called after throwing an instance of 'SIMFQT::AirlineNotFoundException'
  what():  No available fare rule corresponding to 'xx; x, xxxx-xxx-xx; xxx, xxx; xx:xx' (parsed key)
Aborted
```

At least one of your fare rules is correct except that the fare into question must be defined by the airline operating (see the first two letters of the parsed key in the error message to know which airline is operating).

13 Supported Systems

13.1 Table of Contents

13.2 Introduction

This page is intended to provide a list of SimFQT supported systems, i.e. the systems on which configuration, installation and testing process of the SimFQT library has been sucessful. Results are grouped based on minor release number. Therefore, only the latest tests for bug-fix releases are included. Besides, the information on this page is divided into sections dependent on the operating system.

Where necessary, some extra information is given for each tested configuration, e.g. external libraries installed, configuration commands used, etc.

If you manage to compile, install and test the SimFQT library on a system not mentioned below, please let us know, so we could update this database.

13.3 SimFQT 3.10.x

13.3.1 Linux Systems

13.3.1.1 Fedora Core 4 with ATLAS

- **Platform**: Intel Pentium 4

- **Operating System**: Fedora Core 4 (x86)
- **Compiler**: g++ (GCC) 4.0.2 20051125
- **SimFQT release**: 3.10.0
- **External Libraries**: From FC4 distribution:
  - `fftw3.i386-3.0.1-3`
  - `fftw3-devel.i386-3.0.1-3`
  - `atlas-sse2.i386-3.6.0-8.fc4`
  - `atlas-sse2-devel.i386-3.6.0-8.fc4`
  - `blas.i386-3.0-35.fc4`
  - `lapack.i386-3.0-35.fc4`
- **Tests Status**: All tests PASSED
- **Comments**: SimFQT configured with:

  ```
  % CXXFLAGS="-O3 -pipe -march=pentium4" ./configure
  ```

- **Date**: March 7, 2006
- **Tester**: Tony Ottosson

13.3.1.2 Gentoo Linux with ACML

- **Platform**: AMD Sempron 3000+
- **Operating System**: Gentoo Linux 2006.0 (x86 arch)
- **Compiler(s)**: g++ (GCC) 3.4.5
- **SimFQT release**: 3.10.1
- **External Libraries**: Compiled and installed from portage tree:
  - `sci-libs/acml-3.0.0`
- **Tests Status**: All tests PASSED
- **Comments**: BLAS and LAPACK libs set by using the following system commands:

  ```
  % eselect blas set ACML
  % eselect lapack set ACML
  ```

  SimFQT configured with:

  ```
  % export CPPFLAGS="-I/usr/include/acml"
  % ./configure --with-blas="-lblas"
  ```

- **Date**: March 31, 2006
- **Tester**: Adam Piatyszek (ediap)

13.3.1.3 Gentoo Linux with ATLAS

- **Platform**: Intel Pentium M Centrino
- **Operating System**: Gentoo Linux 2006.0 (x86)
- **Compiler**: g++ (GCC) 3.4.5
- **SimFQT release**: 3.10.1
- **External Libraries**: Compiled and installed from portage tree:
  - `sci-libs/fftw-3.1`
  - `sci-libs/blas-atlas-3.6.0-r1`
  - `sci-libs/lapack-atlas-3.6.0`
- **Tests Status**: All tests PASSED
- **Comments**: BLAS and LAPACK libs set by using the following system commands:

  ```
  % eselect blas set ATLAS
  % eselect lapack set ATLAS
  ```

  SimFQT configured with:

  ```
  % ./configure --with-blas="-lblas"
  ```

- **Date**: March 31, 2006
- **Tester**: Adam Piatyszek (ediap)

13.3.1.4 Gentoo Linux with MKL

- **Platform**: Intel Pentium M Centrino
- **Operating System**: Gentoo Linux 2006.0 (x86 arch)
- **Compiler**: g++ (GCC) 3.4.5
- **SimFQT release**: 3.10.0
- **External Libraries**: Intel Math Kernel Library (MKL) 8.0.1 installed manually in the following directory: `/opt/intel/mkl/8.0.1`
- **Tests Status**: All tests PASSED
- **Comments**: SimFQT configured using the following commands:

  ```
  % export LDFLAGS="-L/opt/intel/mkl/8.0.1/lib/32"
  % export CPPFLAGS="-I/opt/intel/mkl/8.0.1/include"
  % ./configure
  ```

- **Date**: February 28, 2006
- **Tester**: Adam Piatyszek (ediap)

13.3.1.5 Gentoo Linux with NetLib's BLAS and LAPACK

- **Platform**: Intel Pentium M Centrino
- **Operating System**: Gentoo Linux 2006.0 (x86)
- **Compiler**: g++ (GCC) 3.4.5
- **SimFQT release**: 3.10.1
- **External Libraries**: Compiled and installed from portage tree:
  - `sci-libs/fftw-3.1`
  - `sci-libs/blas-reference-19940131-r2`
  - `sci-libs/cblas-reference-20030223`
  - `sci-libs/lapack-reference-3.0-r2`
- **Tests Status**: All tests PASSED
- **Comments**: BLAS and LAPACK libs set by using the following system commands:

  ```
  % blas-config reference
  % lapack-config reference
  ```

  SimFQT configured with:

  ```
  % ./configure --with-blas="-lblas"
  ```

- **Date**: March 31, 2006
- **Tester**: Adam Piatyszek (ediap)

13.3.1.6 Red Hat Enterprise Linux with SimFQT External

- **Platform**: Intel Pentium 4
- **Operating System**: Red Hat Enterprise Linux AS release 4 (Nahant Update 2)
- **Compiler**: g++ (GCC) 3.4.4 20050721 (Red Hat 3.4.4-2)
- **SimFQT release**: 3.10.0
- **External Libraries**: BLAS, CBLAS, LAPACK and FFTW libraries from SimFQT External 2.1.1 package
- **Tests Status**: All tests PASSED
- **Date**: March 7, 2006
- **Tester**: Erik G. Larsson

13.3.1.7 SUSE Linux 10.0 with NetLib's BLAS and LAPACK

- **Platform**: Intel Pentium 4 CPU 3.20GHz (64-bit)
- **Operating System**: SUSE Linux 10.0 (x86_64)
- **Compiler(s)**: g++ (GCC) 4.0.2
- **SimFQT release**: 3.10.0
- **External Libraries**: BLAS, LAPACK and FFTW libraries installed from OpenSuse 10.0 RPM repository:
  - `blas-3.0-926`
  - `lapack-3.0-926`
  - `fftw3-3.0.1-114`
  - `fftw3-threads-3.0.1-114`
  - `fftw3-devel-3.0.1-114`
- **Tests Status**: All tests PASSED
- **Comments**: SimFQT configured with:

```
% export CXXFLAGS="-m64 -march=nocona -O3 -pipe"
% ./configure --with-lapack="/usr/lib64/liblapack.so.3"
```

- **Date**: March 1, 2006
- **Tester**: Adam Piatyszek (ediap)

13.3.1.8 SUSE Linux 10.0 with MKL

- **Platform**: Intel Pentium 4 CPU 3.20GHz (64-bit)
- **Operating System**: SUSE Linux 10.0 (x86_64)
- **Compiler(s)**: g++ (GCC) 4.0.2
- **SimFQT release**: 3.10.0
- **External Libraries**: Intel Math Kernel Library (MKL) 8.0.1 installed manually in the following directory: `/opt/intel/mkl/8.0.1`
- **Tests Status**: All tests PASSED
- **Comments**: SimFQT configured with:

```
% export CXXFLAGS="-m64 -march=nocona -O3 -pipe"
% export LDFLAGS="-L/opt/intel/mkl/8.0.1/lib/em64t"
% export CPPFLAGS="-I/opt/intel/mkl/8.0.1/include"
% ./configure
```

- **Date**: March 1, 2006
- **Tester**: Adam Piatyszek (ediap)

13.3.2 Windows Systems

13.3.2.1 Microsoft Windows XP with Cygwin

- **Platform**: AMD Sempron 3000+
- **Operating System**: Microsoft Windows XP SP2, Cygwin 1.5.19-4
- **Compiler(s)**: g++ (GCC) 3.4.4 (cygming special)
- **SimFQT release**: 3.10.1
- **External Libraries**: Installed from Cygwin's repository:
  - `fftw-3.0.1-2`
  - `fftw-dev-3.0.1-1`
  - `lapack-3.0-4`
- **Tests Status**: All tests PASSED
- **Comments**: Only static library can be built. SimFQT configured with:

  ```
  % ./configure
  ```
- **Date**: March 31, 2006
- **Tester**: Adam Piatyszek (ediap)

13.3.2.2 Microsoft Windows XP with Cygwin and ATLAS

- **Platform**: AMD Sempron 3000+
- **Operating System**: Microsoft Windows XP SP2, Cygwin 1.5.19-4
- **Compiler(s)**: g++ (GCC) 3.4.4 (cygming special)
- **SimFQT release**: 3.10.1
- **External Libraries**: Installed from Cygwin's repository:
  - `fftw-3.0.1-2`
  - `fftw-dev-3.0.1-1`

  ATLAS BLAS and LAPACK libraries from SimFQT External 2.1.1 package configured using:

  ```
  % ./configure --enable-atlas --disable-fftw
  ```
- **Tests Status**: All tests PASSED
- **Comments**: Only static library can be built. SimFQT configured with:

  ```
  % export LDFLAGS="-L/usr/local/lib"
  % ./configure
  ```
- **Date**: March 31, 2006
- **Tester**: Adam Piatyszek (ediap)

13.3.2.3 Microsoft Windows XP with Cygwin and ACML

- **Platform**: AMD Sempron 3000+
- **Operating System**: Microsoft Windows XP SP2, Cygwin 1.5.19-4
- **Compiler(s)**: g++ (GCC) 3.4.4 (cygming special)
- **SimFQT release**: 3.10.2
- **External Libraries**: ACML version 3.1.0 (`acml3.1.0-32-win32-g77.exe`) installed into a default directory, i.e. "`c:\Program Files\AMD\acml3.1.0`"
- **Tests Status**: All tests PASSED
- **Comments**: Only static library can be built. SimFQT configured with:

```
% export LDFLAGS="-L/cygdrive/c/Progra~1/AMD/acml3.1.0/gnu32/lib"
% export CPPFLAGS="-I/cygdrive/c/Progra~1/AMD/acml3.1.0/gnu32/include"
% ./configure --enable-debug
```

- **Date**: May 15, 2006
- **Tester**: Adam Piatyszek (ediap)

13.3.2.4 Microsoft Windows XP with MinGW, MSYS and ACML

- **Platform**: AMD Sempron 3000+
- **Operating System**: Microsoft Windows XP SP2, MinGW 5.0.2, MSYS 1.0.10
- **Compiler(s)**: g++ (GCC) 3.4.4 (mingw special)
- **SimFQT release**: 3.10.2
- **External Libraries**: ACML version 3.1.0 (`acml3.1.0-32-win32-g77.exe`) installed into a default directory, i.e. "`c:\Program Files\AMD\acml3.1.0`"
- **Tests Status**: All tests PASSED
- **Comments**: Only static library can be built. SimFQT configured with:

```
% export LDFLAGS="-L/c/Progra~1/AMD/acml3.1.0/gnu32/lib"
% export CPPFLAGS="-I/c/Progra~1/AMD/acml3.1.0/gnu32/include"
% ./configure --enable-debug
```

- **Date**: May 15, 2006
- **Tester**: Adam Piatyszek (ediap)

13.3.2.5 Microsoft Windows XP with MinGW, MSYS and SimFQT External

- **Platform**: AMD Sempron 3000+
- **Operating System**: Microsoft Windows XP SP2, MinGW 5.0.2, MSYS 1.0.10
- **Compiler(s)**: g++ (GCC) 3.4.4 (mingw special)
- **SimFQT release**: 3.10.5
- **External Libraries**: BLAS, CBLAS, LAPACK and FFTW libraries from SimFQT External 2.2.0 package
- **Tests Status**: All tests PASSED
- **Comments**: Only static library can be built. SimFQT configured with:

```
% export LDFLAGS="-L/usr/local/lib"
% export CPPFLAGS="-I/usr/local/include"
% export CXXFLAGS="-Wall -O3 -march=athlon-tbird -pipe"
% ./configure --disable-html-doc
```

- **Date**: August 11, 2006
- **Tester**: Adam Piatyszek (ediap)

13.3.2.6 Microsoft Windows XP with MS Visual C++ and Intel MKL

- **Platform**: AMD Sempron 3000+
- **Operating System**: Microsoft Windows XP SP2
- **Compiler(s)**: Microsoft Visual C++ 2005 .NET
- **SimFQT release**: 3.10.5
- **External Libraries**: Intel Math Kernel Library (MKL) 8.1 installed manually in the following directory: `"C:\Program Files\Intel\MKL\8.1"`
- **Tests Status**: Not fully tested. Some SimFQT based programs compiled and run with success.
- **Comments**: Only static library can be built. SimFQT built by opening the `"win32\simfqt.vcproj"` project file in MSVC++ and executing `"Build -> Build Solution"` command from menu.
- **Date**: August 11, 2006
- **Tester**: Adam Piatyszek (ediap)

13.3.3 Unix Systems

13.3.3.1 SunOS 5.9 with SimFQT External

- **Platform**: SUNW, Sun-Blade-100 (SPARC)
- **Operating System**: SunOS 5.9 Generic_112233-10
- **Compiler(s)**: g++ (GCC) 3.4.5

- **SimFQT release**: 3.10.2
- **External Libraries**: BLAS, CBLAS, LAPACK and FFTW libraries from SimFQT External 2.1.1 package. The following configuration command has been used:

```
% export CFLAGS="-mcpu=ultrasparc -O2 -pipe -funroll-all-loops"
% ./configure
```

- **Tests Status**: All tests PASSED
- **Comments**: SimFQT configured with:

```
% export LDFLAGS="-L/usr/local/lib"
% export CPPFLAGS="-I/usr/local/include"
% export CXXFLAGS="-mcpu=ultrasparc -O2 -pipe"
% ./configure --enable-debug
```

- **Date**: May 15, 2006
- **Tester**: Adam Piatyszek (ediap)

14 SimFQT Supported Systems (Previous Releases)

14.1 SimFQT 3.9.1

14.2 SimFQT 3.9.0

14.3 SimFQT 3.8.1

15 Tutorials

15.1 Table of Contents

15.2 Preparing the SimFQT Project for Development

The source code for these examples can be found in the `batches` and `test/simfqt` directories. They are compiled along with the rest of the `SimFQT` project. See the Users Guide for more details on how to build the `SimFQT` project.

15.3 Your first fareQuote

15.3.1 Summary of the different steps

All the steps below can be found in the same order in the batch simfqt.cpp program.

First, we instanciate the simfqtService object:

```
  std::ofstream logOutputFile;
  \textcolor{keyword}{const} stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);
  \hyperlink{classSIMFQT_1_1SIMFQT__Service}{SIMFQT::SIMFQT_Service} simfqtService (lLogParams);
```

Then, we construct a default sample list of travel solutions and a default booking request (as mentionned in Instanciate the default booking request and Instanciate the default travel solution list parts):

```
 stdair::TravelSolutionList\_T& ioInteractiveTravelSolutionList,
    \textcolor{keywordflow}{return} ioBookingRequestStruct;
```

For basic use, the default BOM tree can be built using:

```
    simfqtService.buildSampleBom();
```

The main step is the fare quoting (see The fare quoting procedure):

```
        simfqtService.quotePrices (lInteractiveBookingRequest,
```

15.3.2 Result of the Batch Program

When the simfqt.cpp program is run (with the -b option), the log output file should look like:

```
[D]../../../simfqt/batches/simfqt.cpp:186: Welcome to Simfqt
[D]../../../simfqt/batches/simfqt.cpp:212: Travel solutions:
    [0] [0] BA, 9, 2011-06-10, LHR, SYD, 21:45 ---  ---
[D]../../../simfqt/command/FareQuoter.cpp:519: Segment path: BA; 9, 2011-06-10; L
      HR, SYD; 21:45. A corresponding fare option \textcolor{keywordflow}{for} the \textcolor{stringlitera
      Y; 450 EUR; conditions: 1 1 1
[D]../../../simfqt/service/SIMFQT\_Service.cpp:352: Fare Quote retrieving: 0.00140
      3 - SIMFQT\_ServiceContext -- Owns StdAir service: 1
[D]../../../simfqt/batches/simfqt.cpp:214: BOM tree:
===============================================================
BomRoot:  -- ROOT --
===============================================================
+++++++++++++++++++++++++++++++++++++++++++++++++
AirportPair: LHR, SYD
+++++++++++++++++++++++++++++++++++++++++++++++++
------------------------------------------
DatePeriod: [2011-Jan-15/2011-Dec-30]
------------------------------------------
******************************************
PosChannel: LHR,DN
******************************************
----------------------------------------
TimePeriod: 00:00:00-23:00:00
----------------------------------------
--------------------------------------
Fare-Features: RT -- 0-1-1-1-0
```

```
-------------------------------------
-----------------------------------
AirlineClassList: BA Y
-----------------------------------

[D]../../../simfqt/batches/simfqt.cpp:219: Travel solutions:
    [0] [0] BA, 9, 2011-06-10, LHR, SYD, 21:45 --- Y, 450, 1 1 1 ---
```

What is interesting is to compare the travel solution list (here reduced to a single travel solution) displayed before:

```
    [0] [0] BA, 9, 2011-06-10, LHR, SYD, 21:45 ---  ---
```

and after the fare quoting:

```
    [0] [0] BA, 9, 2011-06-10, LHR, SYD, 21:45 --- Y, 450, 1 1 1 ---
```

Between the two groups of dashes, we can see that a fare option structure has been added by the fare quoter: the price is 450 EUR for the Y class, the ticket is refundable but there are exchange fees and the customer must stay over on saturday night.

Let's return to our default BOM tree display: the only fare rule stored was a match for the travel solution into consideration (same origin airport, same destination airport, flight date included in the fare rule date range, same airline "BA", ...).

By looking at the fare rule trip type "RT", we can guess we face a round trip fare: that means the price given in the default bom tree construction in `stdair::CmdBomManager.hpp` has been divided by 2 because we are considering either an inbound trip or an outbound one.

15.4 Fare quoting with an input file

15.4.1 How to build a fare input file?

The objective here is to build a fare input file to fare quote the default travel solution list built using:

```
 stdair::TravelSolutionList\_T& ioInteractiveTravelSolutionList,
```

This travel solution list, reduced to a singleton, can be displayed as done before:

```
    [0] [0] BA, 9, 2011-06-10, LHR, SYD, 21:45 ---  ---
```

We deduce:

- we need a fare rule whose origin-destination couple is "LHR, SYD".
- the date range must include the date "2011-06-10".
- the time range must include the time "21:45".
- the airline operating is "BA", so it must be the airline pricing.

We can deduce a part of our fare rule file :

```
\textcolor{comment}{// Fares: fare ID; OriginCity; DestinationCity; TripType; DateRangeStart; DateRan
      geEnd; DepartureTimeRangeStart; DepartureTimeRangeEnd; POS; CabinCode; Channel; A
      dvancePurchase; SaturdayNight; ChangeFees; NonRefundable; MinimumStay; Price; nb
      Segments}
\textcolor{comment}{// Segment: AirlineCode; Class; }
1; LHR; SYD; ??; 2011-01-01; 2011-12-31; 00:00; 23:59; ???; ?; ??; ?; ?; ?; ?; ?;
       ????; BA; ?;
```

We have no information about stay duration and advance purchase (such information are contained into the booking request): so let us put "0" to embrace all the requests possible.

No information for the point-of-sale and the channel too: let us consider all the channels ("IN", "DN", "IF" and DF") and all the points of sale (the origin "LHR", the destination "SYD" and the rest-of-the-world "ROW) existing. To access this information, we could look into the default booking request.

The input file is now:

```
\textcolor{comment}{// Fares: fare ID; OriginCity; DestinationCity; TripType; DateRangeStart; DateRan
      geEnd; DepartureTimeRangeStart; DepartureTimeRangeEnd; POS; CabinCode; Channel; A
      dvancePurchase; SaturdayNight; ChangeFees; NonRefundable; MinimumStay; Price; nb
      Segments}
\textcolor{comment}{// Segment: AirlineCode; Class; }
1; LHR; SYD; ??; 2011-01-01; 2011-12-31; 00:00; 23:59; LHR; ?; IN; 0; ?; ?; ?; 0;
       ????; BA; ?;
2; LHR; SYD; ??; 2011-01-01; 2011-12-31; 00:00; 23:59; LHR; ?; IF; 0; ?; ?; ?; 0;
       ????; BA; ?;
3; LHR; SYD; ??; 2011-01-01; 2011-12-31; 00:00; 23:59; LHR; ?; DN; 0; ?; ?; ?; 0;
       ????; BA; ?;
4; LHR; SYD; ??; 2011-01-01; 2011-12-31; 00:00; 23:59; LHR; ?; DF; 0; ?; ?; ?; 0;
       ????; BA; ?;
5; LHR; SYD; ??; 2011-01-01; 2011-12-31; 00:00; 23:59; SYD; ?; IN; 0; ?; ?; ?; 0;
       ????; BA; ?;
6; LHR; SYD; ??; 2011-01-01; 2011-12-31; 00:00; 23:59; SYD; ?; IF; 0; ?; ?; ?; 0;
       ????; BA; ?;
7; LHR; SYD; ??; 2011-01-01; 2011-12-31; 00:00; 23:59; SYD; ?; DN; 0; ?; ?; ?; 0;
       ????; BA; ?;
8; LHR; SYD; ??; 2011-01-01; 2011-12-31; 00:00; 23:59; SYD; ?; DF; 0; ?; ?; ?; 0;
       ????; BA; ?;
9; LHR; SYD; ??; 2011-01-01; 2011-12-31; 00:00; 23:59; ROW; ?; IN; 0; ?; ?; ?; 0;
       ????; BA; ?;
10; LHR; SYD; ??; 2011-01-01; 2011-12-31; 00:00; 23:59; ROW; ?; IF; 0; ?; ?; ?; 0
      ; ????; BA; ?;
11; LHR; SYD; ??; 2011-01-01; 2011-12-31; 00:00; 23:59; ROW; ?; DN; 0; ?; ?; ?; 0
      ; ????; BA; ?;
12; LHR; SYD; ??; 2011-01-01; 2011-12-31; 00:00; 23:59; ROW; ?; DF; 0; ?; ?; ?; 0
      ; ????; BA; ?;
```

Let us say we have just the Economy cabin "Y" and Bristish Airways prices ticket for class "Y".

No information about the trip type, so we duplicate all the fare rules for both type: one-way "OW" and round-trip "RT" (to access this information, we could look to the default booking request).

The fare options are all set to a default value "T" (meaning true) and the fare values are chosen to be all distinct.

We obtain:

```
\textcolor{comment}{// Fares: fare ID; OriginCity; DestinationCity; TripType; DateRangeStart; DateRan
      geEnd; DepartureTimeRangeStart; DepartureTimeRangeEnd; POS; CabinCode; Channel; A
      dvancePurchase; SaturdayNight; ChangeFees; NonRefundable; MinimumStay; Price; nb
      Segments}
\textcolor{comment}{// Segment: AirlineCode; Class; }
1; LHR; SYD; OW; 2011-01-01; 2011-12-31; 00:00; 23:59; LHR; Y; IN; 0; T; T; T; 0;
       50; BA; Y;
```

```
2; LHR; SYD; OW; 2011-01-01; 2011-12-31; 00:00; 23:59; LHR; Y; IF; 0; T; T; T; 0;
       150; BA; Y;
3; LHR; SYD; OW; 2011-01-01; 2011-12-31; 00:00; 23:59; LHR; Y; DN; 0; T; T; T; 0;
       250; BA; Y;
4; LHR; SYD; OW; 2011-01-01; 2011-12-31; 00:00; 23:59; LHR; Y; DF; 0; T; T; T; 0;
       350; BA; Y;
5; LHR; SYD; OW; 2011-01-01; 2011-12-31; 00:00; 23:59; SYD; Y; IN; 0; T; T; T; 0;
       450; BA; Y;
6; LHR; SYD; OW; 2011-01-01; 2011-12-31; 00:00; 23:59; SYD; Y; IF; 0; T; T; T; 0;
       550; BA; Y;
7; LHR; SYD; OW; 2011-01-01; 2011-12-31; 00:00; 23:59; SYD; Y; DN; 0; T; T; T; 0;
       650; BA; Y;
8; LHR; SYD; OW; 2011-01-01; 2011-12-31; 00:00; 23:59; SYD; Y; DF; 0; T; T; T; 0;
       750; BA; Y;
9; LHR; SYD; OW; 2011-01-01; 2011-12-31; 00:00; 23:59; ROW; Y; IN; 0; T; T; T; 0;
       850; BA; Y;
10; LHR; SYD; OW; 2011-01-01; 2011-12-31; 00:00; 23:59; ROW; Y; IF; 0; T; T; T; 0
      ; 950; BA; Y;
11; LHR; SYD; OW; 2011-01-01; 2011-12-31; 00:00; 23:59; ROW; Y; DN; 0; T; T; T; 0
      ; 1050; BA; Y;
12; LHR; SYD; OW; 2011-01-01; 2011-12-31; 00:00; 23:59; ROW; Y; DF; 0; T; T; T; 0
      ; 1150; BA; Y;
13; LHR; SYD; RT; 2011-01-01; 2011-12-31; 00:00; 23:59; LHR; Y; IN; 0; T; T; T; 0
      ; 90; BA; Y;
14; LHR; SYD; RT; 2011-01-01; 2011-12-31; 00:00; 23:59; LHR; Y; IF; 0; T; T; T; 0
      ; 190; BA; Y;
15; LHR; SYD; RT; 2011-01-01; 2011-12-31; 00:00; 23:59; LHR; Y; DN; 0; T; T; T; 0
      ; 290; BA; Y;
16; LHR; SYD; RT; 2011-01-01; 2011-12-31; 00:00; 23:59; LHR; Y; DF; 0; T; T; T; 0
      ; 390; BA; Y;
17; LHR; SYD; RT; 2011-01-01; 2011-12-31; 00:00; 23:59; SYD; Y; IN; 0; T; T; T; 0
      ; 490; BA; Y;
18; LHR; SYD; RT; 2011-01-01; 2011-12-31; 00:00; 23:59; SYD; Y; IF; 0; T; T; T; 0
      ; 590; BA; Y;
19; LHR; SYD; RT; 2011-01-01; 2011-12-31; 00:00; 23:59; SYD; Y; DN; 0; T; T; T; 0
      ; 690; BA; Y;
20; LHR; SYD; RT; 2011-01-01; 2011-12-31; 00:00; 23:59; SYD; Y; DF; 0; T; T; T; 0
      ; 790; BA; Y;
21; LHR; SYD; RT; 2011-01-01; 2011-12-31; 00:00; 23:59; ROW; Y; IN; 0; T; T; T; 0
      ; 890; BA; Y;
22; LHR; SYD; RT; 2011-01-01; 2011-12-31; 00:00; 23:59; ROW; Y; IF; 0; T; T; T; 0
      ; 990; BA; Y;
23; LHR; SYD; RT; 2011-01-01; 2011-12-31; 00:00; 23:59; ROW; Y; DN; 0; T; T; T; 0
      ; 1090; BA; Y;
24; LHR; SYD; RT; 2011-01-01; 2011-12-31; 00:00; 23:59; ROW; Y; DF; 0; T; T; T; 0
      ; 1190; BA; Y;
```

15.4.2 Building the BOM tree with an input file

The steps are the same as before Summary of the different steps except the bom tree must be built using the fare input file :

15.4.3 Result of the Batch Program

When the `simfqt.cpp` program is run with the `-f` option linking with the file built just above:

```
~/simfqt -f ~/<YourFileName>.csv
```

the last lines of the log output should look like:

```
[D]~/simfqtgit/simfqt/batches/simfqt.cpp:223: Travel solutions:
    [0] [0] BA, 9, 2011-06-10, LHR, SYD, 21:45 --- Y, 145, 1 1 1 ---
```

We have just one fare option added to the travel solution. We can deduce from the price value 145 that the fare quoter used the fare rule number 15 to price the travel solution. We have an inbound or outbound trip of a round trip: the total price 290 has been divided by 2.

16 Command-Line Test to Demonstrate How To Test the SimFQT Project

```
 */
// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <sstream>
#include <fstream>
#include <string>
// Boost Unit Test Framework (UTF)
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MAIN
#define BOOST_TEST_MODULE FQTTestSuite
#include <boost/test/unit_test.hpp>
// StdAir
#include <stdair/basic/BasLogParams.hpp>
#include <stdair/basic/BasDBParams.hpp>
#include <stdair/basic/BasFileMgr.hpp>
#include <stdair/service/Logger.hpp>
#include <stdair/bom/TravelSolutionStruct.hpp>
#include <stdair/bom/BookingRequestStruct.hpp>
// SimFQT
#include <simfqt/SIMFQT_Service.hpp>
#include <simfqt/config/simfqt-paths.hpp>

namespace boost_utf = boost::unit_test;

struct UnitTestConfig {
  UnitTestConfig() {
    static std::ofstream _test_log ("FQTTestSuite_utfresults.xml");
    boost_utf::unit_test_log.set_stream (_test_log);
    boost_utf::unit_test_log.set_format (boost_utf::XML);
    boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
    //boost_utf::unit_test_log.set_threshold_level (boost_utf::log_successful_tes
      ts);
  }

  ~UnitTestConfig() {
  }
};

// //////////////////////////////////////////////////////////////////////
void testFareQuoterHelper (const unsigned short iTestFlag,
                           const stdair::Filename_T iFareInputFilename,
                           const bool isBuiltin) {

  // Output log File
  std::ostringstream oStr;
  oStr << "FQTTestSuite_" << iTestFlag << ".log";
  const stdair::Filename_T lLogFilename (oStr.str());

  // Set the log parameters
  std::ofstream logOutputFile;
  // Open and clean the log outputfile
  logOutputFile.open (lLogFilename.c_str());
  logOutputFile.clear();

  // Initialise the SimFQT service object
  const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG,
                                         logOutputFile);

  // Initialise the Simfqt service object
  SIMFQT::SIMFQT_Service simfqtService (lLogParams);

  // Check wether or not a (CSV) input file should be read
  if (isBuiltin == true) {

    // Build the default sample BOM tree (filled with fares) for Simfqt
    simfqtService.buildSampleBom();

  } else {

    // Build the BOM tree from parsing the fare input file
    SIMFQT::FareFilePath lFareFilePath (iFareInputFilename);
    simfqtService.parseAndLoad (lFareFilePath);
  }

  // Build a sample list of travel solutions and a booking request.
  stdair::TravelSolutionList_T lTravelSolutionList;
  simfqtService.buildSampleTravelSolutions (lTravelSolutionList);
  stdair::BookingRequestStruct lBookingRequest =
    simfqtService.buildBookingRequest();

  // Try to fareQuote the sample list of travel solutions
  simfqtService.quotePrices (lBookingRequest, lTravelSolutionList);

  // Close the log file
  logOutputFile.close();

}

// /////////////// Main: Unit Test Suite //////////////

// Set the UTF configuration (re-direct the output to a specific file)
BOOST_GLOBAL_FIXTURE (UnitTestConfig);

// Start the test suite
BOOST_AUTO_TEST_SUITE (master_test_suite)


BOOST_AUTO_TEST_CASE (simfqt_simple_pricing_test) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fare01.csv");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_NO_THROW (testFareQuoterHelper (0, lFareInputFilename, isBuiltin));


}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_01) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fareError01.cs
      v");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (1, lFareInputFilename, isBuiltin),
                     SIMFQT::AirportPairNotFoundException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_02) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fareError02.cs
      v");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (2, lFareInputFilename, isBuiltin),
                     SIMFQT::PosOrChannelNotFoundException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_03) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fareError03.cs
      v");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (3, lFareInputFilename, isBuiltin),
                     SIMFQT::FlightDateNotFoundException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_04) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fareError04.cs
      v");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (4, lFareInputFilename, isBuiltin),
                     SIMFQT::FlightTimeNotFoundException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_05) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fareError05.cs
      v");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (5, lFareInputFilename, isBuiltin),
                     SIMFQT::FeaturesNotFoundException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_06) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fareError06.cs
    v");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (6, lFareInputFilename, isBuiltin),
                     SIMFQT::AirlineNotFoundException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_07) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fareError07.cs
    v");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (7, lFareInputFilename, isBuiltin),
                     SIMFQT::FareFileParsingFailedException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_08) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/missingFile.cs
    v");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (8, lFareInputFilename, isBuiltin),
                     SIMFQT::FareInputFileNotFoundException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_09) {

  // Input file name
  const stdair::Filename_T lEmptyInputFilename (STDAIR_SAMPLE_DIR "/ ");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = true;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_NO_THROW(testFareQuoterHelper (9, lEmptyInputFilename, isBuiltin));

}


// End the test suite
BOOST_AUTO_TEST_SUITE_END()

/*!
```

17 Directory Hierarchy

17.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

18 Namespace Index

18.1 Namespace List

Here is a list of all namespaces with brief descriptions:

19 Class Index

19.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

std::basic_fstream< char >
std::basic_fstream< wchar_t >
std::basic_ifstream< char >
std::basic_ifstream< wchar_t >
std::basic_ios< char >
std::basic_ios< wchar_t >
std::basic_iostream< char >
std::basic_iostream< wchar_t >
std::basic_istream< char >
std::basic_istream< wchar_t >
std::basic_istringstream< char >
std::basic_istringstream< wchar_t >
std::basic_ofstream< char >

20 Class Index

20.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

21 File Index

21.1 File List

Here is a list of all files with brief descriptions:

22 Directory Documentation

22.1 simfqt/basic/ Directory Reference

Files

- file BasConst.cpp
- file BasConst_General.hpp
- file BasConst_SIMFQT_Service.hpp

22.2 simfqt/batches/ Directory Reference

Files

- file simfqt_parseFareRules.cpp

22.3 simfqt/bom/ Directory Reference

Files

- file FareRuleStruct.cpp
- file FareRuleStruct.hpp

22.4 simfqt/ui/cmdline/ Directory Reference

Files

- file simfqt.cpp

22.5 simfqt/command/ Directory Reference

Files

- file FareParser.cpp
- file FareParser.hpp
- file FareParserHelper.cpp
- file FareParserHelper.hpp
- file FareQuoter.cpp
- file FareQuoter.hpp
- file FareRuleGenerator.cpp
- file FareRuleGenerator.hpp

22.6 simfqt/factory/ Directory Reference

Files

- file FacSimfqtServiceContext.cpp
- file FacSimfqtServiceContext.hpp

22.7 simfqt/service/ Directory Reference

Files

- file SIMFQT_Service.cpp
- file SIMFQT_ServiceContext.cpp
- file SIMFQT_ServiceContext.hpp

22.8 test/simfqt/ Directory Reference

Files

- file FQTTestSuite.cpp

22.9 simfqt/ Directory Reference

Directories

- directory basic
- directory batches
- directory bom
- directory command
- directory factory
- directory service
- directory ui

Files

- file SIMFQT_Service.hpp
- file SIMFQT_Types.hpp

22.10 test/ Directory Reference

Directories

- directory simfqt

22.11 simfqt/ui/ Directory Reference

Directories

- directory cmdline

23 Namespace Documentation

23.1 SIMFQT Namespace Reference

Namespaces

- namespace FareParserHelper

Classes

- struct FareRuleStruct
- class FareParser
- class FareRuleFileParser
- class FareQuoter

  Command wrapping the pricing request process.

- class FareRuleGenerator
- class FacSimfqtServiceContext

  Factory for the service context.

- class SIMFQT_ServiceContext

  Class holding the context of the SimFQT services.

- class SIMFQT_Service

  Interface for the SIMFQT Services.

- class FareFileParsingFailedException
- class AirportPairNotFoundException
- class PosOrChannelNotFoundException
- class FlightDateNotFoundException
- class FlightTimeNotFoundException
- class FeaturesNotFoundException
- class AirlineNotFoundException
- class FareInputFileNotFoundException
- class QuotingException
- class FareFilePath

Typedefs

- typedef unsigned int FareQuoteID_T
- typedef boost::shared_ptr< SIMFQT_Service > SIMFQT_ServicePtr_T

Variables

- const std::string DEFAULT_FARE_QUOTER_ID = "IATA"

23.1.1 Typedef Documentation

23.1.1.1 typedef unsigned int SIMFQT::FareQuoteID_T

ID for the Fare Quote system.

Definition at line 143 of file SIMFQT_Types.hpp.

23.1.1.2 typedef boost::shared_ptr<SIMFQT_Service> SIMFQT::SIMFQT_ServicePtr_T

(Smart) Pointer on the SimFQT service handler.

Definition at line 148 of file SIMFQT_Types.hpp.

23.1.2 Variable Documentation

23.1.2.1 const std::string SIMFQT::DEFAULT_FARE_QUOTER_ID = "IATA"

Default ID for the SIMFQT_Service.

Definition at line 10 of file BasConst.cpp.

23.2 SIMFQT::FareParserHelper Namespace Reference

Classes

- struct FareRuleParser
- struct ParserSemanticAction
- struct storeFareId
- struct storeOrigin
- struct storeDestination
- struct storeTripType
- struct storeDateRangeStart
- struct storeDateRangeEnd
- struct storeStartRangeTime
- struct storeEndRangeTime
- struct storePOS
- struct storeCabinCode
- struct storeChannel
- struct storeAdvancePurchase
- struct storeSaturdayStay
- struct storeChangeFees
- struct storeNonRefundable
- struct storeMinimumStay
- struct storeFare
- struct storeAirlineCode
- struct storeClass
- struct doEndFare

Variables

- stdair::int1_p_t int1_p
- stdair::uint2_p_t uint2_p
- stdair::uint4_p_t uint4_p
- stdair::uint1_4_p_t uint1_4_p
- stdair::hour_p_t hour_p
- stdair::minute_p_t minute_p
- stdair::second_p_t second_p
- stdair::year_p_t year_p
- stdair::month_p_t month_p
- stdair::day_p_t day_p

23.2.1 Variable Documentation

23.2.1.1 stdair::int1_p_t SIMFQT::FareParserHelper::int1_p

Namespaces. 1-digit-integer parser

Definition at line 447 of file FareParserHelper.cpp.

23.2.1.2 stdair::uint2_p_t SIMFQT::FareParserHelper::uint2_p

2-digit-integer parser

Definition at line 450 of file FareParserHelper.cpp.

23.2.1.3 stdair::uint4_p_t SIMFQT::FareParserHelper::uint4_p

4-digit-integer parser

Definition at line 453 of file FareParserHelper.cpp.

23.2.1.4 stdair::uint1_4_p_t SIMFQT::FareParserHelper::uint1_4_p

Up-to-4-digit-integer parser

Definition at line 456 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

23.2.1.5 stdair::hour_p_t SIMFQT::FareParserHelper::hour_p

Time element parsers.

Definition at line 459 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

23.2.1.6 stdair::minute_p_t SIMFQT::FareParserHelper::minute_p

Definition at line 460 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

23.2.1.7 stdair::second_p_t SIMFQT::FareParserHelper::second_p

Definition at line 461 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

23.2.1.8 stdair::year_p_t SIMFQT::FareParserHelper::year_p

Date element parsers.

Definition at line 464 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

23.2.1.9 stdair::month_p_t SIMFQT::FareParserHelper::month_p

Definition at line 465 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

23.2.1.10 stdair::day_p_t SIMFQT::FareParserHelper::day_p

Definition at line 466 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

23.3 stdair Namespace Reference

Forward declarations.

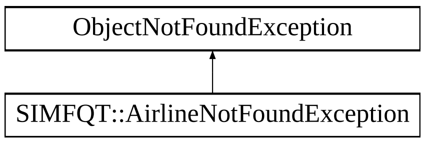
23.3.1 Detailed Description

Forward declarations.

24 Class Documentation

24.1 SIMFQT::AirlineNotFoundException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`Inheritance diagram for SIMFQT::AirlineNotFoundException::



Public Member Functions

- AirlineNotFoundException (const std::string &iWhat)

24.1.1 Detailed Description

The airline can not be found.

Definition at line 99 of file SIMFQT_Types.hpp.

24.1.2 Constructor & Destructor Documentation

24.1.2.1 SIMFQT::AirlineNotFoundException::AirlineNotFoundException (const std::string & *iWhat*) `[inline]`

Constructor.

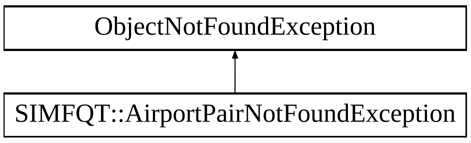
Definition at line 104 of file SIMFQT_Types.hpp.

The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.2 SIMFQT::AirportPairNotFoundException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`Inheritance diagram for SIMFQT::AirportPairNotFoundException::



Public Member Functions

- AirportPairNotFoundException (const std::string &iWhat)

24.2.1 Detailed Description

The given airport pair can not be found.

Definition at line 39 of file SIMFQT_Types.hpp.

24.2.2 Constructor & Destructor Documentation

24.2.2.1 SIMFQT::AirportPairNotFoundException::AirportPairNotFoundException (const std::string & *iWhat*) `[inline]`

Constructor.

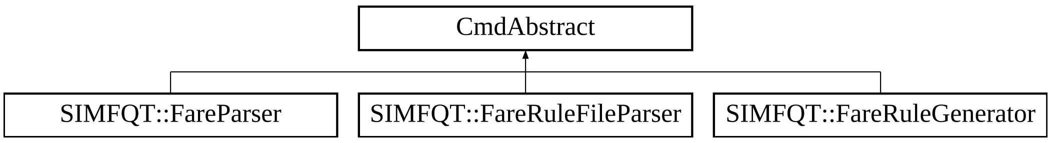
Definition at line 44 of file SIMFQT_Types.hpp.

The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.3 CmdAbstract Class Reference

Inheritance diagram for CmdAbstract::

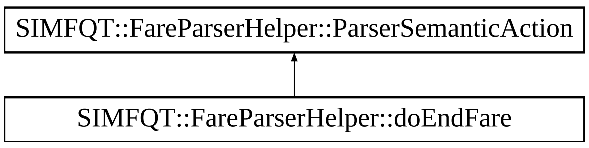


The documentation for this class was generated from the following files:

- simfqt/command/FareRuleGenerator.hpp
- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParser.hpp

24.4 SIMFQT::FareParserHelper::doEndFare Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::doEndFare::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::doEndFare

Public Member Functions

- doEndFare (stdair::BomRoot &, FareRuleStruct &)
- void operator() (boost::spirit::qi::unused_type, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- stdair::BomRoot & _bomRoot
- FareRuleStruct & _fareRule

24.4.1 Detailed Description

Mark the end of the fare-rule parsing.

Definition at line 230 of file FareParserHelper.hpp.

24.4.2 Constructor & Destructor Documentation

24.4.2.1 SIMFQT::FareParserHelper::doEndFare::doEndFare (stdair::BomRoot & *ioBomRoot*, FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 420 of file FareParserHelper.cpp.

24.4.3 Member Function Documentation

24.4.3.1 void SIMFQT::FareParserHelper::doEndFare::operator() (boost::spirit::qi::unused_-type, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 427 of file FareParserHelper.cpp.

References _bomRoot, SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, and SIM-FQT::FareRuleStruct::describe().

24.4.4 Member Data Documentation

24.4.4.1 stdair::BomRoot& SIMFQT::FareParserHelper::doEndFare::_bomRoot

Actor Specific Context.

Definition at line 238 of file FareParserHelper.hpp.

Referenced by operator()().

24.4.4.2 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule [inherited]

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

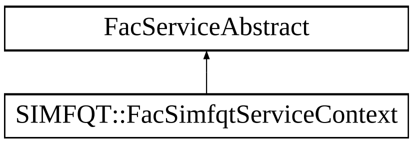
Referenced by operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIM-FQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()(), SIM-FQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()() SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.5 FacServiceAbstract Class Reference

Inheritance diagram for FacServiceAbstract::



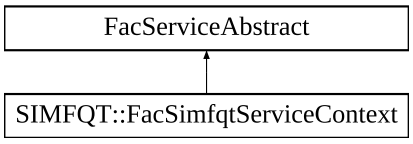
The documentation for this class was generated from the following file:

- simfqt/factory/FacSimfqtServiceContext.hpp

24.6 SIMFQT::FacSimfqtServiceContext Class Reference

Factory for the service context.

`#include <simfqt/factory/FacSimfqtServiceContext.hpp>`Inheritance diagram for SIMFQT::FacSimfqtServiceContext::



Public Member Functions

- ∼FacSimfqtServiceContext ()
- SIMFQT_ServiceContext & create ()

Static Public Member Functions

- static FacSimfqtServiceContext & instance ()

Protected Member Functions

- FacSimfqtServiceContext ()

24.6.1 Detailed Description

Factory for the service context.

Definition at line 22 of file FacSimfqtServiceContext.hpp.

24.6.2 Constructor & Destructor Documentation

24.6.2.1 SIMFQT::FacSimfqtServiceContext::∼FacSimfqtServiceContext ()

Destructor.

The Destruction put the _instance to NULL in order to be clean for the next FacSimfqtServiceContext::instance().

Definition at line 17 of file FacSimfqtServiceContext.cpp.

24.6.2.2 SIMFQT::FacSimfqtServiceContext::FacSimfqtServiceContext () `[inline, protected]`

Default Constructor.

This constructor is protected in order to ensure the singleton pattern.

Definition at line 57 of file FacSimfqtServiceContext.hpp.

Referenced by instance().

24.6.3 Member Function Documentation

24.6.3.1 FacSimfqtServiceContext & SIMFQT::FacSimfqtServiceContext::instance () `[static]`

Provide the unique instance.

The singleton is instantiated when first used.

Returns:

FacServiceContext&

Definition at line 22 of file FacSimfqtServiceContext.cpp.

References FacSimfqtServiceContext().

24.6.3.2 SIMFQT_ServiceContext & SIMFQT::FacSimfqtServiceContext::create ()

Create a new ServiceContext object.

This new object is added to the list of instantiated objects.

Returns:

ServiceContext& The newly created object.

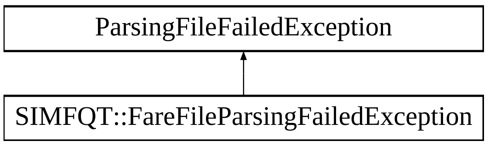
Definition at line 34 of file FacSimfqtServiceContext.cpp.

The documentation for this class was generated from the following files:

- simfqt/factory/FacSimfqtServiceContext.hpp
- simfqt/factory/FacSimfqtServiceContext.cpp

24.7 SIMFQT::FareFileParsingFailedException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`Inheritance diagram for SIMFQT::FareFileParsingFailedException::


ParsingFileFailedException
SIMFQT::FareFileParsingFailedException

Public Member Functions

- FareFileParsingFailedException (const std::string &iWhat)

24.7.1 Detailed Description

The fare input file can not be parsed.

Definition at line 26 of file SIMFQT_Types.hpp.

24.7.2 Constructor & Destructor Documentation

24.7.2.1 SIMFQT::FareFileParsingFailedException::FareFileParsingFailedException (const std::string & *iWhat*) `[inline]`

Constructor.

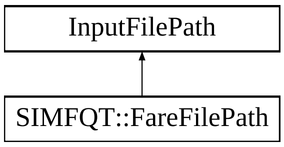
Definition at line 32 of file SIMFQT_Types.hpp.

The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.8 SIMFQT::FareFilePath Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`Inheritance diagram for SIMFQT::FareFilePath::



Public Member Functions

- FareFilePath (const stdair::Filename_T &iFilename)

24.8.1 Detailed Description

Fare input file.

Definition at line 130 of file SIMFQT_Types.hpp.

24.8.2 Constructor & Destructor Documentation

24.8.2.1 SIMFQT::FareFilePath::FareFilePath (const stdair::Filename_T & *iFilename*) `[inline, explicit]`

Constructor.

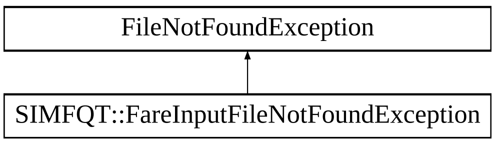
Definition at line 135 of file SIMFQT_Types.hpp.

The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.9 SIMFQT::FareInputFileNotFoundException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`Inheritance diagram for SIMFQT::FareInputFileNotFoundException::



Public Member Functions

- FareInputFileNotFoundException (const std::string &iWhat)

24.9.1 Detailed Description

The fare input file can not be found.

Definition at line 111 of file SIMFQT_Types.hpp.

24.9.2 Constructor & Destructor Documentation

24.9.2.1 SIMFQT::FareInputFileNotFoundException::FareInputFileNotFoundException (const std::string & *iWhat*) `[inline]`

Constructor.

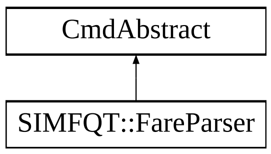
Definition at line 116 of file SIMFQT_Types.hpp.

The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.10 SIMFQT::FareParser Class Reference

`#include <simfqt/command/FareParser.hpp>`Inheritance diagram for SIMFQT::FareParser::



Static Public Member Functions

- static void fareRuleGeneration (const FareFilePath &, stdair::BomRoot &)

24.10.1 Detailed Description

Class wrapping the parser entry point.

Definition at line 23 of file FareParser.hpp.

24.10.2 Member Function Documentation

24.10.2.1 void SIMFQT::FareParser::fareRuleGeneration (const FareFilePath & *iFareFilename*, stdair::BomRoot & *ioBomRoot*) `[static]`

Parses the CSV file describing the fares for the simulator, and generates the fare bom tree accordingly.

Parameters:

const FareFilePath&	The file-name of the CSV-formatted fare input file.
stdair::BomRoot&	Root of the BOM tree.

Definition at line 17 of file FareParser.cpp.

References SIMFQT::FareRuleFileParser::generateFareRules().

Referenced by SIMFQT::SIMFQT_Service::parseAndLoad().

The documentation for this class was generated from the following files:

- simfqt/command/FareParser.hpp
- simfqt/command/FareParser.cpp

24.11 SIMFQT::FareQuoter Class Reference

Command wrapping the pricing request process.

`#include <simfqt/command/FareQuoter.hpp>`

Friends

- class SIMFQT_Service

24.11.1 Detailed Description

Command wrapping the pricing request process.

Definition at line 29 of file FareQuoter.hpp.

24.11.2 Friends And Related Function Documentation

24.11.2.1 friend class SIMFQT_Service [friend]

Friend classes: only the SimFQT service may access to the methods of that command class.

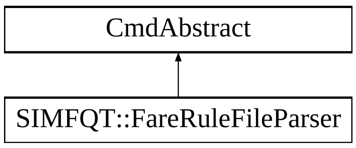
Definition at line 32 of file FareQuoter.hpp.

The documentation for this class was generated from the following files:

- simfqt/command/FareQuoter.hpp
- simfqt/command/FareQuoter.cpp

24.12 SIMFQT::FareRuleFileParser Class Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareRuleFileParser::



Public Member Functions

- FareRuleFileParser (stdair::BomRoot &ioBomRoot, const stdair::Filename_T &iFilename)
- void generateFareRules ()

24.12.1 Detailed Description

Class wrapping the initialisation and entry point of the parser.

The seemingly redundancy is used to force the instantiation of the actual parser, which is a templatised Boost Spirit grammar. Hence, the actual parser is instantiated within that class object code.

Definition at line 254 of file FareParserHelper.hpp.

24.12.2 Constructor & Destructor Documentation

24.12.2.1 SIMFQT::FareRuleFileParser::FareRuleFileParser (stdair::BomRoot & *ioBomRoot*, const stdair::Filename_T & *iFilename*)

Constructor.

Definition at line 645 of file FareParserHelper.cpp.

24.12.3 Member Function Documentation

24.12.3.1 void SIMFQT::FareRuleFileParser::generateFareRules ()

Parse the input file and generate the fare rules.

Definition at line 667 of file FareParserHelper.cpp.

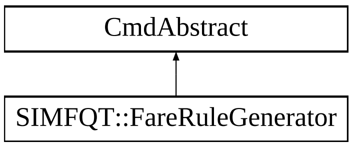
Referenced by SIMFQT::FareParser::fareRuleGeneration().

The documentation for this class was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.13 SIMFQT::FareRuleGenerator Class Reference

`#include <simfqt/command/FareRuleGenerator.hpp>`Inheritance diagram for SIMFQT::FareRuleGenerator::



Friends

- class FareFileParser
- struct FareParserHelper::doEndFare
- class FareParser

24.13.1 Detailed Description

Class handling the generation / instantiation of the Fare BOM.

Definition at line 33 of file FareRuleGenerator.hpp.

24.13.2 Friends And Related Function Documentation

24.13.2.1 friend class FareFileParser `[friend]`

Definition at line 38 of file FareRuleGenerator.hpp.

24.13.2.2 friend struct FareParserHelper::doEndFare `[friend]`

Definition at line 39 of file FareRuleGenerator.hpp.

24.13.2.3 friend class FareParser [friend]

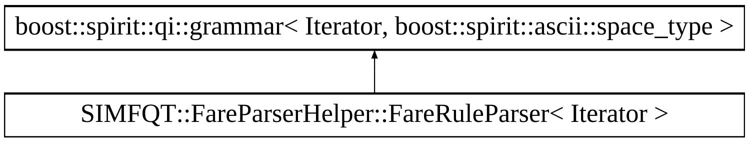
Definition at line 40 of file FareRuleGenerator.hpp.

The documentation for this class was generated from the following files:

- simfqt/command/FareRuleGenerator.hpp
- simfqt/command/FareRuleGenerator.cpp

24.14 SIMFQT::FareParserHelper::FareRuleParser< Iterator > Struct Template Reference

Inheritance diagram for SIMFQT::FareParserHelper::FareRuleParser< Iterator >::


boost::spirit::qi::grammar< Iterator, boost::spirit::ascii::space_type >
SIMFQT::FareParserHelper::FareRuleParser< Iterator >

Public Member Functions

- FareRuleParser (stdair::BomRoot &ioBomRoot, FareRuleStruct &iofareRule)

Public Attributes

- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > start
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > comments
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > fare_rule
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > fare_rule_end
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > fare_key
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > fare_id
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > origin
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > destination
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > tripType
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > dateRangeStart
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > dateRangeEnd
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > date
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > timeRangeStart
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > timeRangeEnd
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > time
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > point_of_sale
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > cabinCode
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > channel
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > advancePurchase
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > saturdayStay
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > changeFees
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > nonRefundable

- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > minimumStay
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > fare
- boost::spirit::qi::rule< Iterator, boost::spirit::ascii::space_type > segment
- stdair::BomRoot & _bomRoot
- FareRuleStruct & _fareRule

24.14.1 Detailed Description

template<typename Iterator> struct SIMFQT::FareParserHelper::FareRuleParser< Iterator >

Fare: fareID; OriginCity; DestinationCity; DateRangeStart; DateRangeEnd; DepartureTimeRangeStart; DepartureTimeRangeEnd; POS; AdvancePurchase; SaturdayNight; ChangeFees; NonRefundable; MinimumStay; Price; AirlineCode; Class;

fareID OriginCity (3-char airport code) DestinationCity (3-char airport code) DateRangeStart (yyyy-mm-dd) DateRangeEnd (yyyy-mm-dd) DepartureTimeRangeStart (hh:mm) DepartureTimeRangeEnd (hh:mm) POS (3-char point_of_sale city) Cabin Code (1-char cabin code) Channel (D=direct, I=indirect, N=oNline, F=oFfline) AdvancePurchase SaturdayNight (T=True, F=False) ChangeFees (T=True, F=False) NonRefundable (T=True, F=False) MinimumStay Price AirlineCode (2-char airline code) ClassList (List of 1-char class code) Grammar for the Fare-Rule parser.

Definition at line 503 of file FareParserHelper.cpp.

24.14.2 Constructor & Destructor Documentation

24.14.2.1 template<typename Iterator> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser (stdair::BomRoot & *ioBomRoot*, FareRuleStruct & *iofareRule*) `[inline]`

Definition at line 507 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::FareRuleParser< Iterator >::_bomRoot, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::_fareRule, SIMFQT::FareRuleStruct::_itDay, SIMFQT::FareRuleStruct::_itHours, SIMFQT::FareRuleStruct::_itMinutes, SIMFQT::FareRuleStruct::_itMonth, SIMFQT::FareRuleStruct::_itSeconds, SIMFQT::FareRuleStruct::_itYear, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::advancePurchase, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::cabinCode, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::changeFees, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::channel, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::comments, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::date, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::dateRangeEnd, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::dateRangeStart, SIMFQT::FareParserHelper::day_p, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::destination, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::fare, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::fare_id, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::fare_key, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::fare_rule, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::fare_rule_end, SIMFQT::FareParserHelper::hour_p, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::minimumStay, SIMFQT::FareParserHelper::minute_p, SIMFQT::FareParserHelper::month_p, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::nonRefundable, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::origin, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::point_of_sale, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::saturdayStay, SIMFQT::FareParserHelper::second_p, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::segment, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::start, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::time, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::timeRangeEnd, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::timeRangeStart, SIMFQT::FareParserHelper::FareRuleParser< Iterator >::tripType, SIMFQT::FareParserHelper::uint1_4_p, and SIMFQT::FareParserHelper::year_p.

24.14.3 Member Data Documentation

24.14.3.1 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::start

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.2 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::comments

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.3 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::fare_rule

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.4 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::fare_rule_end

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.5 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::fare_key

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.6 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::fare_id

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.7 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::origin

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.8 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::destination

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.9 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::tripType

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.10 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::dateRangeStart

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.11 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::dateRangeEnd

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.12 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::date

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.13 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::timeRangeStart

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.14 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::timeRangeEnd

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.15 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::time

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.16 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::point_of_sale

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.17 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::cabinCode

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.18 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::channel

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.19 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::advancePurchase

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.20 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::saturdayStay

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.21 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::changeFees

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.22 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::nonRefundable

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.23 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::minimumStay

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.24 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::fare

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.25 template<typename Iterator> boost::spirit::qi::rule<Iterator, boost::spirit::ascii::space_type> SIMFQT::FareParserHelper::FareRuleParser< Iterator >::segment

Definition at line 623 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.26 template<typename Iterator> stdair::BomRoot& SIMFQT::FareParserHelper::FareRuleParser< Iterator >::_bomRoot

Definition at line 630 of file FareParserHelper.cpp.

Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.14.3.27 template<typename Iterator> FareRuleStruct& SIMFQT::FareParserHelper::FareRuleParser< Iterator >::_fareRule

Definition at line 631 of file FareParserHelper.cpp.

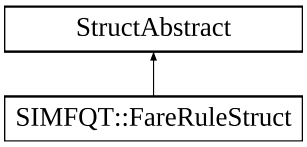
Referenced by SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

The documentation for this struct was generated from the following file:

- simfqt/command/FareParserHelper.cpp

24.15 SIMFQT::FareRuleStruct Struct Reference

`#include <simfqt/bom/FareRuleStruct.hpp>`Inheritance diagram for SIMFQT::FareRuleStruct::



Public Member Functions

- FareRuleStruct ()
- SIMFQT::FareQuoteID_T getFareID () const
- stdair::AirportCode_T getOrigin () const
- stdair::AirportCode_T getDestination () const
- stdair::TripType_T getTripType () const
- stdair::Date_T getDateRangeStart () const
- stdair::Date_T getDateRangeEnd () const
- stdair::Duration_T getTimeRangeStart () const
- stdair::Duration_T getTimeRangeEnd () const
- stdair::CabinCode_T getCabinCode () const
- const stdair::CityCode_T getPOS () const
- stdair::ChannelLabel_T getChannel () const
- stdair::DayDuration_T getAdvancePurchase () const
- stdair::SaturdayStay_T getSaturdayStay () const
- stdair::ChangeFees_T getChangeFees () const
- stdair::NonRefundable_T getNonRefundable () const
- stdair::DayDuration_T getMinimumStay () const
- stdair::PriceValue_T getFare () const
- stdair::AirlineCode_T getAirlineCode () const
- stdair::ClassCode_T getClassCode () const
- const unsigned int getAirlineListSize () const
- const unsigned int getClassCodeListSize () const
- stdair::AirlineCodeList_T getAirlineList () const
- stdair::ClassList_StringList_T getClassCodeList () const

- stdair::Date_T calculateDate () const
- stdair::Duration_T calculateTime () const
- const std::string describe () const
- void setFareID (const SIMFQT::FareQuoteID_T &iFareQuoteID)
- void setOrigin (const stdair::AirportCode_T &iOrigin)
- void setDestination (const stdair::AirportCode_T &iDestination)
- void setTripType (const stdair::TripType_T &iTripType)
- void setDateRangeStart (const stdair::Date_T &iDateRangeStart)
- void setDateRangeEnd (const stdair::Date_T &iDateRangeEnd)
- void setTimeRangeStart (const stdair::Duration_T &iTimeRangeStart)
- void setTimeRangeEnd (const stdair::Duration_T &iTimeRangeEnd)
- void setCabinCode (const stdair::CabinCode_T &iCabinCode)
- void setPOS (const stdair::CityCode_T &iPOS)
- void setChannel (const stdair::ChannelLabel_T &iChannel)
- void setAdvancePurchase (const stdair::DayDuration_T &iAdvancePurchase)
- void setSaturdayStay (const stdair::SaturdayStay_T &iSaturdayStay)
- void setChangeFees (const stdair::ChangeFees_T &iChangeFees)
- void setNonRefundable (const stdair::NonRefundable_T &iNonRefundable)
- void setMinimumStay (const stdair::DayDuration_T &iMinimumStay)
- void setFare (const stdair::PriceValue_T &iFare)
- void setAirlineCode (const stdair::AirlineCode_T &iAirlineCode)
- void setClassCode (const stdair::ClassCode_T &iClassCode)
- void clearAirlineCodeList ()
- void clearClassCodeList ()
- void addAirlineCode (const stdair::AirlineCode_T &iAirlineCode)
- void addClassCode (const stdair::ClassCode_T &iClassCode)

Public Attributes

- stdair::year_t _itYear
- stdair::month_t _itMonth
- stdair::day_t _itDay
- stdair::hour_t _itHours
- stdair::minute_t _itMinutes
- stdair::second_t _itSeconds

24.15.1 Detailed Description

Utility Structure for the parsing of fare-rule structures.

Definition at line 21 of file FareRuleStruct.hpp.

24.15.2 Constructor & Destructor Documentation

24.15.2.1 SIMFQT::FareRuleStruct::FareRuleStruct ()

Default constructor.

Definition at line 17 of file FareRuleStruct.cpp.

24.15.3 Member Function Documentation

24.15.3.1 SIMFQT::FareQuoteID_T SIMFQT::FareRuleStruct::getFareID () const `[inline]`

Get the fare ID.

Definition at line 30 of file FareRuleStruct.hpp.

24.15.3.2 stdair::AirportCode_T SIMFQT::FareRuleStruct::getOrigin () const `[inline]`

Get the origin.

Definition at line 35 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storePOS::operator()().

24.15.3.3 stdair::AirportCode_T SIMFQT::FareRuleStruct::getDestination () const `[inline]`

Get the destination.

Definition at line 40 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storePOS::operator()().

24.15.3.4 stdair::TripType_T SIMFQT::FareRuleStruct::getTripType () const `[inline]`

Get the trip type.

Definition at line 45 of file FareRuleStruct.hpp.

24.15.3.5 stdair::Date_T SIMFQT::FareRuleStruct::getDateRangeStart () const `[inline]`

Get the date range start.

Definition at line 50 of file FareRuleStruct.hpp.

24.15.3.6 stdair::Date_T SIMFQT::FareRuleStruct::getDateRangeEnd () const `[inline]`

Get the date range end.

Definition at line 55 of file FareRuleStruct.hpp.

24.15.3.7 stdair::Duration_T SIMFQT::FareRuleStruct::getTimeRangeStart () const `[inline]`

Get the time range start.

Definition at line 60 of file FareRuleStruct.hpp.

24.15.3.8 stdair::Duration_T SIMFQT::FareRuleStruct::getTimeRangeEnd () const `[inline]`

Get the time range end.

Definition at line 65 of file FareRuleStruct.hpp.

24.15.3.9 stdair::CabinCode_T SIMFQT::FareRuleStruct::getCabinCode () const `[inline]`

Get the cabin code.

Definition at line 70 of file FareRuleStruct.hpp.

24.15.3.10 const stdair::CityCode_T SIMFQT::FareRuleStruct::getPOS () const `[inline]`

Get the point-of-sale.

Definition at line 75 of file FareRuleStruct.hpp.

24.15.3.11 stdair::ChannelLabel_T SIMFQT::FareRuleStruct::getChannel () const `[inline]`

Get the channel.

Definition at line 80 of file FareRuleStruct.hpp.

24.15.3.12 stdair::DayDuration_T SIMFQT::FareRuleStruct::getAdvancePurchase () const `[inline]`

Get the advance purchase.

Definition at line 85 of file FareRuleStruct.hpp.

24.15.3.13 stdair::SaturdayStay_T SIMFQT::FareRuleStruct::getSaturdayStay () const `[inline]`

Get the saturday stay option.

Definition at line 90 of file FareRuleStruct.hpp.

24.15.3.14 stdair::ChangeFees_T SIMFQT::FareRuleStruct::getChangeFees () const `[inline]`

Get the change fees.

Definition at line 95 of file FareRuleStruct.hpp.

24.15.3.15 stdair::NonRefundable_T SIMFQT::FareRuleStruct::getNonRefundable () const `[inline]`

Get the refundable option.

Definition at line 100 of file FareRuleStruct.hpp.

24.15.3.16 stdair::DayDuration_T SIMFQT::FareRuleStruct::getMinimumStay () const `[inline]`

Get the minimum stay.

Definition at line 105 of file FareRuleStruct.hpp.

24.15.3.17 stdair::PriceValue_T SIMFQT::FareRuleStruct::getFare () const `[inline]`

Get the fare.

Definition at line 110 of file FareRuleStruct.hpp.

24.15.3.18 stdair::AirlineCode_T SIMFQT::FareRuleStruct::getAirlineCode () const `[inline]`

Get the airline code.

Definition at line 115 of file FareRuleStruct.hpp.

24.15.3.19 stdair::ClassCode_T SIMFQT::FareRuleStruct::getClassCode () const `[inline]`

Get the class code.

Definition at line 120 of file FareRuleStruct.hpp.

24.15.3.20 const unsigned int SIMFQT::FareRuleStruct::getAirlineListSize () const `[inline]`

Get the size of the airline code list.

Definition at line 125 of file FareRuleStruct.hpp.

24.15.3.21 const unsigned int SIMFQT::FareRuleStruct::getClassCodeListSize () const `[inline]`

Get the size of the class code list.

Definition at line 130 of file FareRuleStruct.hpp.

24.15.3.22 stdair::AirlineCodeList_T SIMFQT::FareRuleStruct::getAirlineList () const `[inline]`

Get the airline code list.

Definition at line 135 of file FareRuleStruct.hpp.

24.15.3.23 stdair::ClassList_StringList_T SIMFQT::FareRuleStruct::getClassCodeList () const `[inline]`

Get the class code list.

Definition at line 140 of file FareRuleStruct.hpp.

24.15.3.24 stdair::Date_T SIMFQT::FareRuleStruct::calculateDate () const

Calculate the date from the staging details.

Definition at line 39 of file FareRuleStruct.cpp.

References _itDay, _itMonth, and _itYear.

Referenced by SIMFQT::FareParserHelper::storeDateRangeEnd::operator()(), and SIMFQT::FareParserHelper::storeDateRangeStart::operator()().

24.15.3.25 stdair::Duration_T SIMFQT::FareRuleStruct::calculateTime () const

Calculate the time from the staging details.

Definition at line 45 of file FareRuleStruct.cpp.

References _itHours, _itMinutes, and _itSeconds.

Referenced by SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), and SIMFQT::FareParserHelper::storeStartRangeTime::operator()().

24.15.3.26 const std::string SIMFQT::FareRuleStruct::describe () const

Display of the structure.

Definition at line 54 of file FareRuleStruct.cpp.

Referenced by SIMFQT::FareParserHelper::doEndFare::operator()().

24.15.3.27 void SIMFQT::FareRuleStruct::setFareID (const SIMFQT::FareQuoteID_T & *iFareQuoteID*) `[inline]`

Set the fare ID.

Definition at line 158 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeFareId::operator()().

24.15.3.28 void SIMFQT::FareRuleStruct::setOrigin (const stdair::AirportCode_T & *iOrigin*) `[inline]`

Set the origin.

Definition at line 163 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeOrigin::operator()().

24.15.3.29 void SIMFQT::FareRuleStruct::setDestination (const stdair::AirportCode_T & *iDestination*) `[inline]`

Set the destination.

Definition at line 168 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeDestination::operator()().

24.15.3.30 void SIMFQT::FareRuleStruct::setTripType (const stdair::TripType_T & *iTripType*) `[inline]`

Set the trip type.

Definition at line 173 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeTripType::operator()().

24.15.3.31 void SIMFQT::FareRuleStruct::setDateRangeStart (const stdair::Date_T & *iDateRangeStart*) `[inline]`

Set the date range start.

Definition at line 178 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeDateRangeStart::operator()().

24.15.3.32 void SIMFQT::FareRuleStruct::setDateRangeEnd (const stdair::Date_T & *iDateRangeEnd*) `[inline]`

Set the date range end.

Definition at line 183 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeDateRangeEnd::operator()().

24.15.3.33 void SIMFQT::FareRuleStruct::setTimeRangeStart (const stdair::Duration_T & *iTimeRangeStart*) `[inline]`

Set the time range start.

Definition at line 188 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeStartRangeTime::operator()().

24.15.3.34 void SIMFQT::FareRuleStruct::setTimeRangeEnd (const stdair::Duration_T & *iTimeRangeEnd*) `[inline]`

Set the time range end.

Definition at line 193 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeEndRangeTime::operator()().

24.15.3.35 void SIMFQT::FareRuleStruct::setCabinCode (const stdair::CabinCode_T & *iCabinCode*) `[inline]`

Set the cabin code.

Definition at line 198 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeCabinCode::operator()().

24.15.3.36 void SIMFQT::FareRuleStruct::setPOS (const stdair::CityCode_T & *iPOS*) `[inline]`

Set the point-of-sale.

Definition at line 203 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storePOS::operator()().

24.15.3.37 void SIMFQT::FareRuleStruct::setChannel (const stdair::ChannelLabel_T & *iChannel*) `[inline]`

Set the channel.

Definition at line 208 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeChannel::operator()().

24.15.3.38 void SIMFQT::FareRuleStruct::setAdvancePurchase (const stdair::DayDuration_T & *iAdvancePurchase*) `[inline]`

Set the advance purchase.

Definition at line 213 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeAdvancePurchase::operator()().

24.15.3.39 void SIMFQT::FareRuleStruct::setSaturdayStay (const stdair::SaturdayStay_T & *iSaturdayStay*) `[inline]`

Set the saturday stay option.

Definition at line 218 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeSaturdayStay::operator()().

24.15.3.40 void SIMFQT::FareRuleStruct::setChangeFees (const stdair::ChangeFees_T & *iChangeFees*) `[inline]`

Set the change fees.

Definition at line 223 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeChangeFees::operator()().

24.15.3.41 void SIMFQT::FareRuleStruct::setNonRefundable (const stdair::NonRefundable_T & *iNonRefundable*) `[inline]`

Set the refundable option.

Definition at line 228 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeNonRefundable::operator()().

24.15.3.42 void SIMFQT::FareRuleStruct::setMinimumStay (const stdair::DayDuration_T & *iMinimumStay*) `[inline]`

Set the minimum stay.

Definition at line 233 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeMinimumStay::operator()().

24.15.3.43 void SIMFQT::FareRuleStruct::setFare (const stdair::PriceValue_T & *iFare*) `[inline]`

Set the fare.

Definition at line 238 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeFare::operator()().

24.15.3.44 void SIMFQT::FareRuleStruct::setAirlineCode (const stdair::AirlineCode_T & *iAirlineCode*) `[inline]`

Set the airline code.

Definition at line 243 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeFareId::operator()().

24.15.3.45 void SIMFQT::FareRuleStruct::setClassCode (const stdair::ClassCode_T & *iClassCode*) `[inline]`

Set the class code.

Definition at line 248 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeFareId::operator()().

24.15.3.46 void SIMFQT::FareRuleStruct::clearAirlineCodeList () `[inline]`

Empty the airline code list.

Definition at line 253 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeFareId::operator()().

24.15.3.47 void SIMFQT::FareRuleStruct::clearClassCodeList () `[inline]`

Empty the class code list.

Definition at line 258 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeFareId::operator()().

24.15.3.48 void SIMFQT::FareRuleStruct::addAirlineCode (const stdair::AirlineCode_T & *iAirlineCode*) `[inline]`

Add an airline code to the list.

Definition at line 263 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeAirlineCode::operator()().

24.15.3.49 void SIMFQT::FareRuleStruct::addClassCode (const stdair::ClassCode_T & *iClassCode*) `[inline]`

Add a class code to the list.

Definition at line 268 of file FareRuleStruct.hpp.

Referenced by SIMFQT::FareParserHelper::storeClass::operator()().

24.15.4 Member Data Documentation

24.15.4.1 stdair::year_t SIMFQT::FareRuleStruct::_itYear

Staging Date.

Definition at line 275 of file FareRuleStruct.hpp.

Referenced by calculateDate(), and SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.15.4.2 stdair::month_t SIMFQT::FareRuleStruct::_itMonth

Definition at line 276 of file FareRuleStruct.hpp.

Referenced by calculateDate(), and SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.15.4.3 stdair::day_t SIMFQT::FareRuleStruct::_itDay

Definition at line 277 of file FareRuleStruct.hpp.

Referenced by calculateDate(), and SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.15.4.4 stdair::hour_t SIMFQT::FareRuleStruct::_itHours

Staging Time.

Definition at line 280 of file FareRuleStruct.hpp.

Referenced by calculateTime(), and SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.15.4.5 stdair::minute_t SIMFQT::FareRuleStruct::_itMinutes

Definition at line 281 of file FareRuleStruct.hpp.

Referenced by calculateTime(), and SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser().

24.15.4.6 stdair::second_t SIMFQT::FareRuleStruct::_itSeconds

Definition at line 282 of file FareRuleStruct.hpp.

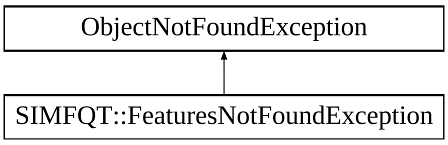
Referenced by calculateTime(), SIMFQT::FareParserHelper::FareRuleParser< Iterator >::FareRuleParser(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/bom/FareRuleStruct.hpp
- simfqt/bom/FareRuleStruct.cpp

24.16 SIMFQT::FeaturesNotFoundException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`Inheritance diagram for SIMFQT::FeaturesNotFoundException::



Public Member Functions

- FeaturesNotFoundException (const std::string &iWhat)

24.16.1 Detailed Description

The fare features can not be found.

Definition at line 87 of file SIMFQT_Types.hpp.

24.16.2 Constructor & Destructor Documentation

24.16.2.1 SIMFQT::FeaturesNotFoundException::FeaturesNotFoundException (const std::string & *iWhat*) `[inline]`

Constructor.

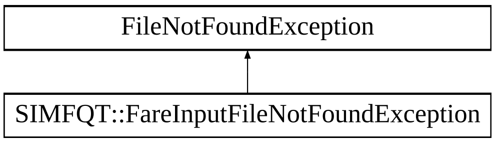
Definition at line 92 of file SIMFQT_Types.hpp.

The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.17 FileNotFoundException Class Reference

Inheritance diagram for FileNotFoundException::

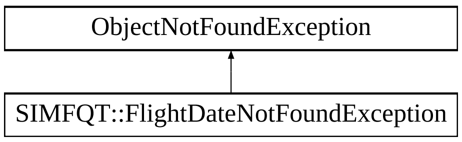


The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.18 SIMFQT::FlightDateNotFoundException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`Inheritance diagram for SIMFQT::FlightDateNotFoundException::



Public Member Functions

- FlightDateNotFoundException (const std::string &iWhat)

24.18.1 Detailed Description

The departure date of the flight can not be found.

Definition at line 63 of file SIMFQT_Types.hpp.

24.18.2 Constructor & Destructor Documentation

24.18.2.1 SIMFQT::FlightDateNotFoundException::FlightDateNotFoundException (const std::string & *iWhat*) `[inline]`

Constructor.

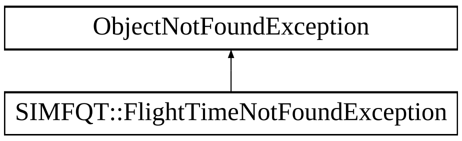
Definition at line 68 of file SIMFQT_Types.hpp.

The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.19 SIMFQT::FlightTimeNotFoundException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`Inheritance diagram for SIMFQT::FlightTimeNotFoundException::



Public Member Functions

- FlightTimeNotFoundException (const std::string &iWhat)

24.19.1 Detailed Description

The departure time of the flight can not be found.

Definition at line 75 of file SIMFQT_Types.hpp.

24.19.2 Constructor & Destructor Documentation

24.19.2.1 SIMFQT::FlightTimeNotFoundException::FlightTimeNotFoundException (const std::string & *iWhat*) `[inline]`

Constructor.

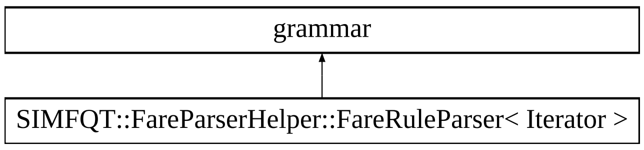
Definition at line 80 of file SIMFQT_Types.hpp.

The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.20 grammar Class Reference

Inheritance diagram for grammar::

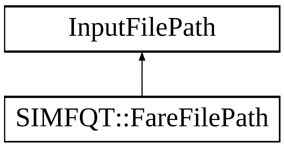


The documentation for this class was generated from the following file:

- simfqt/command/FareParserHelper.cpp

24.21 InputFilePath Class Reference

Inheritance diagram for InputFilePath::

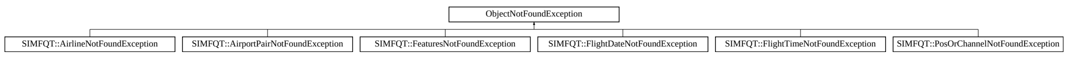


The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.22 ObjectNotFoundException Class Reference

Inheritance diagram for ObjectNotFoundException::

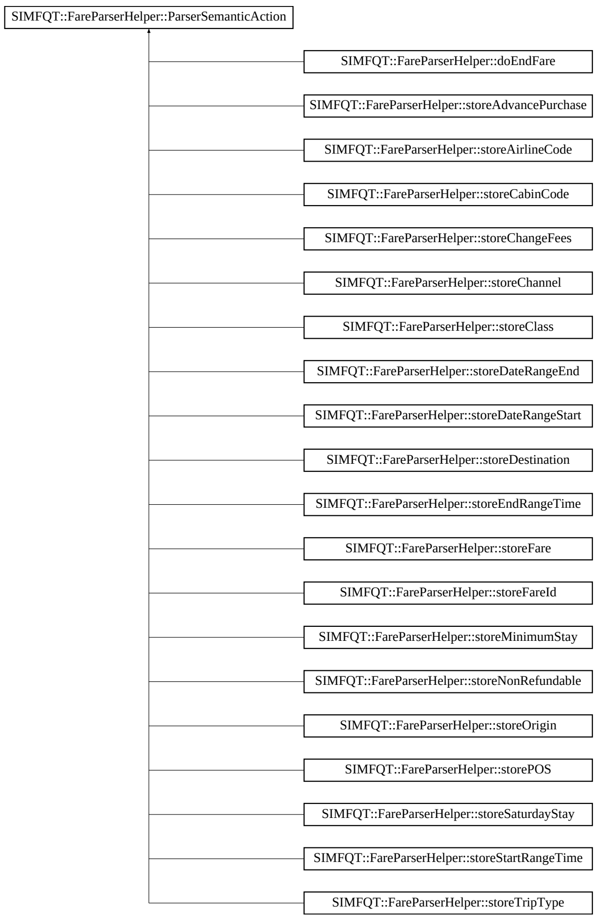

ObjectNotFoundException
SIMFQT::AirlineNotFoundException
SIMFQT::AirportPairNotFoundException
SIMFQT::FeaturesNotFoundException
SIMFQT::FlightDateNotFoundException
SIMFQT::FlightTimeNotFoundException
SIMFQT::PosOrChannelNotFoundException

The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.23 SIMFQT::FareParserHelper::ParserSemanticAction Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::ParserSemanticAction::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::doEndFare
SIMFQT::FareParserHelper::storeAdvancePurchase
SIMFQT::FareParserHelper::storeAirlineCode
SIMFQT::FareParserHelper::storeCabinCode
SIMFQT::FareParserHelper::storeChangeFees
SIMFQT::FareParserHelper::storeChannel
SIMFQT::FareParserHelper::storeClass
SIMFQT::FareParserHelper::storeDateRangeEnd
SIMFQT::FareParserHelper::storeDateRangeStart
SIMFQT::FareParserHelper::storeDestination
SIMFQT::FareParserHelper::storeEndRangeTime
SIMFQT::FareParserHelper::storeFare
SIMFQT::FareParserHelper::storeFareId
SIMFQT::FareParserHelper::storeMinimumStay
SIMFQT::FareParserHelper::storeNonRefundable
SIMFQT::FareParserHelper::storeOrigin
SIMFQT::FareParserHelper::storePOS
SIMFQT::FareParserHelper::storeSaturdayStay
SIMFQT::FareParserHelper::storeStartRangeTime
SIMFQT::FareParserHelper::storeTripType

Public Member Functions

- ParserSemanticAction (FareRuleStruct &)

Public Attributes

- FareRuleStruct & _fareRule

24.23.1 Detailed Description

Generic Semantic Action (Actor / Functor) for the Fare Parser.

Definition at line 31 of file FareParserHelper.hpp.

24.23.2 Constructor & Destructor Documentation

24.23.2.1 SIMFQT::FareParserHelper::ParserSemanticAction::ParserSemanticAction (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 30 of file FareParserHelper.cpp.

24.23.3 Member Data Documentation

24.23.3.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

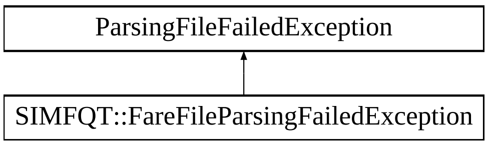
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()() SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()() SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.24 ParsingFileFailedException Class Reference

Inheritance diagram for ParsingFileFailedException::

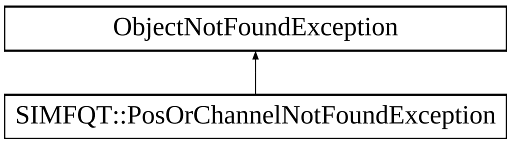

ParsingFileFailedException
SIMFQT::FareFileParsingFailedException

The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.25 SIMFQT::PosOrChannelNotFoundException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`Inheritance diagram for SIMFQT::PosOrChannelNotFoundException::



Public Member Functions

- PosOrChannelNotFoundException (const std::string &iWhat)

24.25.1 Detailed Description

The given POS/channel can not be found.

Definition at line 51 of file SIMFQT_Types.hpp.

24.25.2 Constructor & Destructor Documentation

24.25.2.1 SIMFQT::PosOrChannelNotFoundException::PosOrChannelNotFoundException (const std::string & *iWhat*) `[inline]`

Constructor.

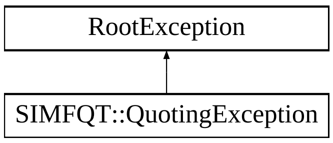
Definition at line 56 of file SIMFQT_Types.hpp.

The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.26 SIMFQT::QuotingException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`Inheritance diagram for SIMFQT::QuotingException::



24.26.1 Detailed Description

The pricing operation fails.

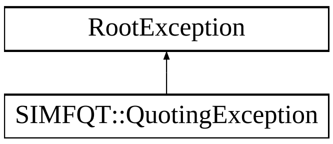
Definition at line 123 of file SIMFQT_Types.hpp.

The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.27 RootException Class Reference

Inheritance diagram for RootException::

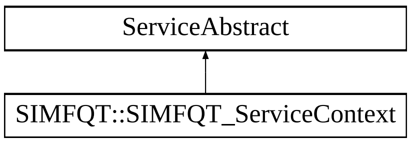


The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

24.28 ServiceAbstract Class Reference

Inheritance diagram for ServiceAbstract::



The documentation for this class was generated from the following file:

- simfqt/service/SIMFQT_ServiceContext.hpp

24.29 SIMFQT::SIMFQT_Service Class Reference

Interface for the SIMFQT Services.

```
#include <simfqt/SIMFQT_Service.hpp>
```

Public Member Functions

- SIMFQT_Service (const stdair::BasLogParams &)
- SIMFQT_Service (const stdair::BasLogParams &, const stdair::BasDBParams &)
- SIMFQT_Service (stdair::STDAIR_ServicePtr_T ioSTDAIR_ServicePtr)
- void parseAndLoad (const FareFilePath &iFareFilename)
- ∼SIMFQT_Service ()
- void buildSampleBom ()
- void clonePersistentBom ()
- void buildComplementaryLinks (stdair::BomRoot &)
- stdair::BookingRequestStruct buildBookingRequest (const bool isForCRS=false)
- void buildSampleTravelSolutions (stdair::TravelSolutionList_T &)
- void quotePrices (const stdair::BookingRequestStruct &, stdair::TravelSolutionList_T &)
- std::string csvDisplay () const
- std::string csvDisplay (const stdair::TravelSolutionList_T &) const
- std::string csvDisplay (const stdair::AirportCode_T &ioOrigin, const stdair::AirportCode_T &ioDestination, const stdair::Date_T &ioDepartureDate) const
- std::string list () const
- bool check (const stdair::AirportCode_T &ioOrigin, const stdair::AirportCode_T &ioDestination, const stdair::Date_T &ioDepartureDate) const

24.29.1 Detailed Description

Interface for the SIMFQT Services.

Definition at line 32 of file SIMFQT_Service.hpp.

24.29.2 Constructor & Destructor Documentation

24.29.2.1 SIMFQT::SIMFQT_Service::SIMFQT_Service (const stdair::BasLogParams & *iLogParams*)

Constructor.

The initSimfqtService() method is called; see the corresponding documentation for more details.

A reference on an output stream is given, so that log outputs can be directed onto that stream.

Parameters:

const	stdair::BasLogParams& Parameters for the output log stream.

Definition at line 36 of file SIMFQT_Service.cpp.

24.29.2.2 SIMFQT::SIMFQT_Service::SIMFQT_Service (const stdair::BasLogParams & *iLogParams*, const stdair::BasDBParams & *iDBParams*)

Constructor.

The initSimfqtService() method is called; see the corresponding documentation for more details.

A reference on an output stream is given, so that log outputs can be directed onto that stream.

Parameters:

const	stdair::BasLogParams& Parameters for the output log stream.
const	stdair::BasDBParams& Parameters for the database access.

Definition at line 56 of file SIMFQT_Service.cpp.

24.29.2.3 SIMFQT::SIMFQT_Service::SIMFQT_Service (stdair::STDAIR_ServicePtr_T *ioSTDAIR_ServicePtr*)

Constructor.

The initSimfqtService() method is called; see the corresponding documentation for more details.

Moreover, as no reference on any output stream is given, it is assumed that the StdAir log service has already been initialised with the proper log output stream by some other methods in the calling chain (for instance, when the SIMFQT_Service is itself being initialised by another library service such as SIMCRS_-Service).

Parameters:

stdair::STDAIR_ServicePtr_T	Reference on the STDAIR service.

Definition at line 78 of file SIMFQT_Service.cpp.

24.29.2.4 SIMFQT::SIMFQT_Service::∼SIMFQT_Service ()

Destructor.

Definition at line 94 of file SIMFQT_Service.cpp.

24.29.3 Member Function Documentation

24.29.3.1 void SIMFQT::SIMFQT_Service::parseAndLoad (const FareFilePath & *iFareFilename*)

Parse the fare dump and load it into memory.

The CSV file, describing the fare rule for the simulator, is parsed and instantiated in memory accordingly.

Parameters:

const FareFilePath& Filename of the input fare file.

Definition at line 171 of file SIMFQT_Service.cpp.

References buildComplementaryLinks(), clonePersistentBom(), and SIMFQT::FareParser::fareRuleGeneration().

Referenced by main().

24.29.3.2 void SIMFQT::SIMFQT_Service::buildSampleBom ()

Build a sample BOM tree, and attach it to the BomRoot instance.

As for now, two sample BOM trees can be built.

- One BOM tree is based on two actual inventories (one for BA, another for AF). Each inventory contains one flight. One of those flights has two legs (and therefore three segments).
- The other BOM tree is fake, as a hook for RMOL to work.

Definition at line 223 of file SIMFQT_Service.cpp.

References buildComplementaryLinks(), and clonePersistentBom().

Referenced by main().

24.29.3.3 void SIMFQT::SIMFQT_Service::clonePersistentBom ()

Clone the persistent BOM object.

Definition at line 279 of file SIMFQT_Service.cpp.

References buildComplementaryLinks().

Referenced by buildSampleBom(), and parseAndLoad().

24.29.3.4 void SIMFQT::SIMFQT_Service::buildComplementaryLinks (stdair::BomRoot & *ioBomRoot*)

Build all the complementary links in the given bom root object.

Note:

Do nothing for now.

Definition at line 315 of file SIMFQT_Service.cpp.

Referenced by buildSampleBom(), clonePersistentBom(), and parseAndLoad().

24.29.3.5 stdair::BookingRequestStruct SIMFQT::SIMFQT_Service::buildBookingRequest (const bool *isForCRS* = `false`)

Build a BookingRequest structure (for test purposes).

Returns:

stdair::BookingRequestStruct The created BookingRequest structure.

Definition at line 320 of file SIMFQT_Service.cpp.

Referenced by main().

24.29.3.6 void SIMFQT::SIMFQT_Service::buildSampleTravelSolutions (stdair::TravelSolutionList_T & *ioTravelSolutionList*)

Build a sample list of travel solutions.

As of now (March 2011), that list is made of the following travel solutions:

- BA9
- LHR-SYD
- 2011-06-10
- Q
- WTP: 900
- Change fee: 20; Non refundable; Saturday night stay

Parameters:

TravelSolutionList_T& Sample list of travel solution structures. It should be given empty. It is altered with the returned sample.

Definition at line 344 of file SIMFQT_Service.cpp.

Referenced by main().

24.29.3.7 void SIMFQT::SIMFQT_Service::quotePrices (const stdair::BookingRequestStruct & *iBookingRequest*, stdair::TravelSolutionList_T & *ioTravelSolutionList*)

Calculate the prices for a given list of travel solutions.

A stdair::Fare_T attribute is calculated for every travel solution of the list.

Parameters:

stdair::BookingRequestStruct& Booking request.

stdair::TravelSolutionList_T& List of travel solution.

Definition at line 480 of file SIMFQT_Service.cpp.

Referenced by main().

24.29.3.8 std::string SIMFQT::SIMFQT_Service::csvDisplay () const

Recursively display (dump in the returned string) the objects of the BOM tree.

Returns:

std::string Output string in which the BOM tree is logged/dumped.

Definition at line 365 of file SIMFQT_Service.cpp.

Referenced by main().

24.29.3.9 std::string SIMFQT::SIMFQT_Service::csvDisplay (const stdair::TravelSolutionList_T & *ioTravelSolutionList*) const

Display (dump in the returned string) the full list of travel solution structures.

Returns:

std::string Output string in which the list of travel solutions is logged/dumped.

Definition at line 392 of file SIMFQT_Service.cpp.

24.29.3.10 std::string SIMFQT::SIMFQT_Service::csvDisplay (const stdair::AirportCode_T & *ioOrigin*, const stdair::AirportCode_T & *ioDestination*, const stdair::Date_T & *ioDepartureDate*) const

Recursively display (dump in the returned string) the fare-rules corresponding to the parameters given as input.

Parameters:

const stdair::AirportCode_T& Origin airport of the fare-rules to display

const stdair::AirportCode_T& Destination airport of the fare- rules to display.

const stdair::Date_T& Departure date of the fare-rules to display.

Returns:

std::string Output string in which the BOM tree is logged/dumped.

Definition at line 414 of file SIMFQT_Service.cpp.

24.29.3.11 std::string SIMFQT::SIMFQT_Service::list () const

Display (dump in the returned string) the airport pairs and the corresponding departure dates of the fare rules stored in the BOM tree.

Returns:

std::string Output string in which the airport pairs and departure dates are logged/dumped.

Definition at line 437 of file SIMFQT_Service.cpp.

24.29.3.12 bool SIMFQT::SIMFQT_Service::check (const stdair::AirportCode_T & *ioOrigin*, const stdair::AirportCode_T & *ioDestination*, const stdair::Date_T & *ioDepartureDate*) const

Check whether the given couple airportpair-date is a valid one.

Parameters:

const	stdair::AirportCode_T& Origin airport of the fare rule to check.
const	stdair::AirportCode_T& Destination airport of the fare rule to check.
const	stdair::Date_T& Departure date of the fare rule to check.

Returns:

bool Whether or not the given airportpair-date couple is a valid one.

Definition at line 458 of file SIMFQT_Service.cpp.

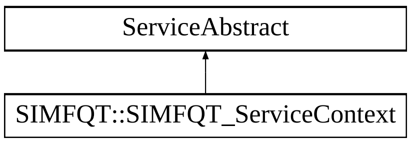
The documentation for this class was generated from the following files:

- simfqt/SIMFQT_Service.hpp
- simfqt/service/SIMFQT_Service.cpp

24.30 SIMFQT::SIMFQT_ServiceContext Class Reference

Class holding the context of the SimFQT services.

`#include <simfqt/service/SIMFQT_ServiceContext.hpp>`Inheritance diagram for SIMFQT::SIMFQT_ServiceContext::



Friends

- class SIMFQT_Service
- class FacSimfqtServiceContext

24.30.1 Detailed Description

Class holding the context of the SimFQT services.

Definition at line 25 of file SIMFQT_ServiceContext.hpp.

24.30.2 Friends And Related Function Documentation

24.30.2.1 friend class SIMFQT_Service `[friend]`

The SIMFQT_Service class should be the sole class to get access to ServiceContext content: general users do not want to bother with a context interface.

Definition at line 31 of file SIMFQT_ServiceContext.hpp.

24.30.2.2 friend class FacSimfqtServiceContext `[friend]`

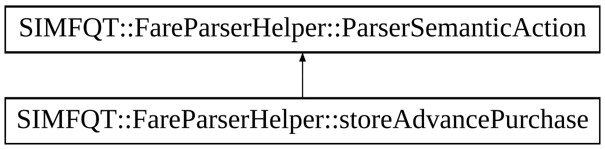
Definition at line 32 of file SIMFQT_ServiceContext.hpp.

The documentation for this class was generated from the following files:

- simfqt/service/SIMFQT_ServiceContext.hpp
- simfqt/service/SIMFQT_ServiceContext.cpp

24.31 SIMFQT::FareParserHelper::storeAdvancePurchase Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeAdvancePurchase::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeAdvancePurchase

Public Member Functions

- storeAdvancePurchase (FareRuleStruct &)
- void operator() (unsigned int, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.31.1 Detailed Description

Store the parsed advance purchase days.

Definition at line 150 of file FareParserHelper.hpp.

24.31.2 Constructor & Destructor Documentation

24.31.2.1 SIMFQT::FareParserHelper::storeAdvancePurchase::storeAdvancePurchase (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 254 of file FareParserHelper.cpp.

24.31.3 Member Function Documentation

24.31.3.1 void SIMFQT::FareParserHelper::storeAdvancePurchase::operator() (unsigned int *iAdancePurchase*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 259 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, and SIMFQT::FareRuleStruct::setAdvancePurchase().

24.31.4 Member Data Documentation

24.31.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

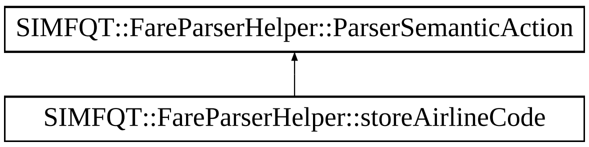
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), operator()(), SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator() SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.32 SIMFQT::FareParserHelper::storeAirlineCode Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeAirlineCode::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeAirlineCode

Public Member Functions

- storeAirlineCode (FareRuleStruct &)
- void operator() (std::vector< char >, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.32.1 Detailed Description

Store the parsed airline code.

Definition at line 210 of file FareParserHelper.hpp.

24.32.2 Constructor & Destructor Documentation

24.32.2.1 SIMFQT::FareParserHelper::storeAirlineCode::storeAirlineCode (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 378 of file FareParserHelper.cpp.

24.32.3 Member Function Documentation

24.32.3.1 void SIMFQT::FareParserHelper::storeAirlineCode::operator() (std::vector< char > *iChar*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 383 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, and SIMFQT::FareRuleStruct::addAirlineCode().

24.32.4 Member Data Documentation

24.32.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()(), SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()() SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

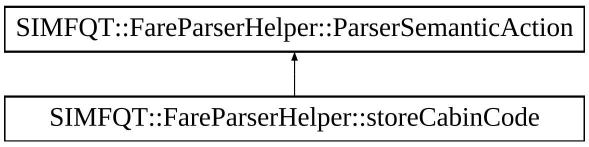
The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp

- simfqt/command/FareParserHelper.cpp

24.33 SIMFQT::FareParserHelper::storeCabinCode Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeCabinCode::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeCabinCode

Public Member Functions

- storeCabinCode (FareRuleStruct &)
- void operator() (char, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.33.1 Detailed Description

Store the cabin code.

Definition at line 130 of file FareParserHelper.hpp.

24.33.2 Constructor & Destructor Documentation

24.33.2.1 SIMFQT::FareParserHelper::storeCabinCode::storeCabinCode (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 212 of file FareParserHelper.cpp.

24.33.3 Member Function Documentation

24.33.3.1 void SIMFQT::FareParserHelper::storeCabinCode::operator() (char *iChar*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 217 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, and SIMFQT::FareRuleStruct::setCabinCode().

24.33.4 Member Data Documentation

24.33.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule [inherited]

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

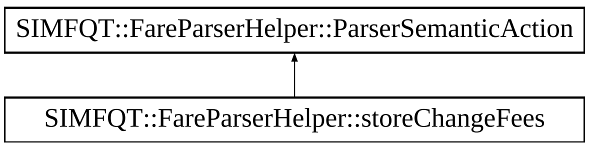
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()() SIMFQT::FareParserHelper::storeChannel::operator()(), operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()() SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.34 SIMFQT::FareParserHelper::storeChangeFees Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeChangeFees::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeChangeFees

Public Member Functions

- storeChangeFees (FareRuleStruct &)
- void operator() (char, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.34.1 Detailed Description

Store the parsed change fees.

Definition at line 170 of file FareParserHelper.hpp.

24.34.2 Constructor & Destructor Documentation

24.34.2.1 SIMFQT::FareParserHelper::storeChangeFees::storeChangeFees (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 295 of file FareParserHelper.cpp.

24.34.3 Member Function Documentation

24.34.3.1 void SIMFQT::FareParserHelper::storeChangeFees::operator() (char *iChangefees*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 300 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, and SIMFQT::FareRuleStruct::setChangeFees().

24.34.4 Member Data Documentation

24.34.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

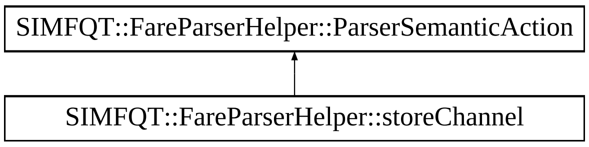
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()(), SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()() SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.35 SIMFQT::FareParserHelper::storeChannel Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeChannel::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeChannel

Public Member Functions

- storeChannel (FareRuleStruct &)
- void operator() (std::vector< char >, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.35.1 Detailed Description

Store the channel distribution.

Definition at line 140 of file FareParserHelper.hpp.

24.35.2 Constructor & Destructor Documentation

24.35.2.1 SIMFQT::FareParserHelper::storeChannel::storeChannel (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 233 of file FareParserHelper.cpp.

24.35.3 Member Function Documentation

24.35.3.1 void SIMFQT::FareParserHelper::storeChannel::operator() (std::vector< char > *iChar*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 238 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, and SIMFQT::FareRuleStruct::setChannel().

24.35.4 Member Data Documentation

24.35.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

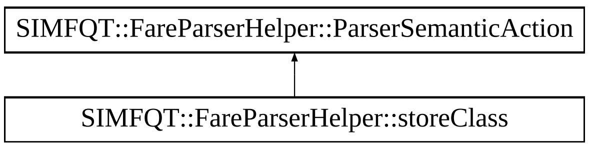
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()(), operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()() SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.36 SIMFQT::FareParserHelper::storeClass Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeClass::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeClass

Public Member Functions

- storeClass (FareRuleStruct &)
- void operator() (std::vector< char >, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.36.1 Detailed Description

Store the parsed class code.

Definition at line 220 of file FareParserHelper.hpp.

24.36.2 Constructor & Destructor Documentation

24.36.2.1 SIMFQT::FareParserHelper::storeClass::storeClass (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 396 of file FareParserHelper.cpp.

24.36.3 Member Function Documentation

24.36.3.1 void SIMFQT::FareParserHelper::storeClass::operator() (std::vector$<$ char $>$ *iChar*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 401 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, and SIMFQT::FareRuleStruct::addClassCode().

24.36.4 Member Data Documentation

24.36.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

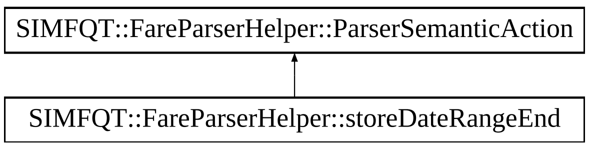
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()() SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()() SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.37 SIMFQT::FareParserHelper::storeDateRangeEnd Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeDateRangeEnd::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeDateRangeEnd

Public Member Functions

- storeDateRangeEnd (FareRuleStruct &)

- void operator() (boost::spirit::qi::unused_type, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.37.1 Detailed Description

Store the parsed end of the date range.

Definition at line 90 of file FareParserHelper.hpp.

24.37.2 Constructor & Destructor Documentation

24.37.2.1 SIMFQT::FareParserHelper::storeDateRangeEnd::storeDateRangeEnd (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 129 of file FareParserHelper.cpp.

24.37.3 Member Function Documentation

24.37.3.1 void SIMFQT::FareParserHelper::storeDateRangeEnd::operator() (boost::spirit::qi::unused_type, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 134 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, SIMFQT::FareRuleStruct::calculateDate(), and SIMFQT::FareRuleStruct::setDateRangeEnd().

24.37.4 Member Data Documentation

24.37.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

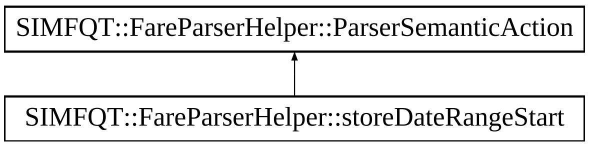
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()() SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), operator()(), SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.38 SIMFQT::FareParserHelper::storeDateRangeStart Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeDateRangeStart::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeDateRangeStart

Public Member Functions

- storeDateRangeStart (FareRuleStruct &)
- void operator() (boost::spirit::qi::unused_type, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.38.1 Detailed Description

Store the parsed start of the date range.

Definition at line 80 of file FareParserHelper.hpp.

24.38.2 Constructor & Destructor Documentation

24.38.2.1 SIMFQT::FareParserHelper::storeDateRangeStart::storeDateRangeStart (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 113 of file FareParserHelper.cpp.

24.38.3 Member Function Documentation

24.38.3.1 void SIMFQT::FareParserHelper::storeDateRangeStart::operator() (boost::spirit::qi::unused_type, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 118 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, SIMFQT::FareRuleStruct::calculateDate(), and SIMFQT::FareRuleStruct::setDateRangeStart().

24.38.4 Member Data Documentation

24.38.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule [inherited]

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

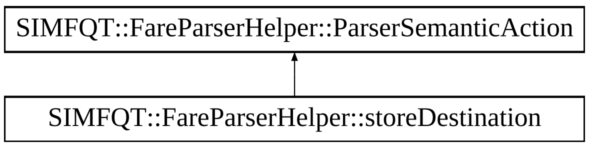
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()() SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()() operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.39 SIMFQT::FareParserHelper::storeDestination Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeDestination::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeDestination

Public Member Functions

- storeDestination (FareRuleStruct &)
- void operator() (std::vector< char >, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.39.1 Detailed Description

Store the parsed destination.

Definition at line 59 of file FareParserHelper.hpp.

24.39.2 Constructor & Destructor Documentation

24.39.2.1 SIMFQT::FareParserHelper::storeDestination::storeDestination (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 75 of file FareParserHelper.cpp.

24.39.3 Member Function Documentation

24.39.3.1 void SIMFQT::FareParserHelper::storeDestination::operator() (std::vector< char > *iChar*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 80 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, and SIMFQT::FareRuleStruct::setDestination().

24.39.4 Member Data Documentation

24.39.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

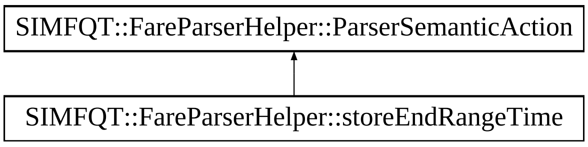
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()() SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()() SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.40 SIMFQT::FareParserHelper::storeEndRangeTime Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeEndRangeTime::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeEndRangeTime

Public Member Functions

- storeEndRangeTime (FareRuleStruct &)
- void operator() (boost::spirit::qi::unused_type, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.40.1 Detailed Description

Store the parsed end range time.

Definition at line 110 of file FareParserHelper.hpp.

24.40.2 Constructor & Destructor Documentation

24.40.2.1 SIMFQT::FareParserHelper::storeEndRangeTime::storeEndRangeTime (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 168 of file FareParserHelper.cpp.

24.40.3 Member Function Documentation

24.40.3.1 void SIMFQT::FareParserHelper::storeEndRangeTime::operator() (boost::spirit::qi::unused_type, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 173 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, SIMFQT::FareRuleStruct::_itSeconds, SIMFQT::FareRuleStruct::calculateTime(), and SIMFQT::FareRuleStruct::setTimeRangeEnd().

24.40.4 Member Data Documentation

24.40.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

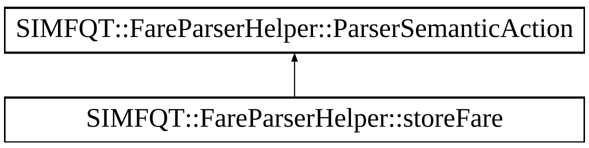
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()(), SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()(), SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.41 SIMFQT::FareParserHelper::storeFare Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeFare::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeFare

Public Member Functions

- storeFare (FareRuleStruct &)
- void operator() (double, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.41.1 Detailed Description

Store the parsed fare value.

Definition at line 200 of file FareParserHelper.hpp.

24.41.2 Constructor & Destructor Documentation

24.41.2.1 SIMFQT::FareParserHelper::storeFare::storeFare (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 362 of file FareParserHelper.cpp.

24.41.3 Member Function Documentation

24.41.3.1 void SIMFQT::FareParserHelper::storeFare::operator() (double *iFare*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 367 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, and SIMFQT::FareRuleStruct::setFare().

24.41.4 Member Data Documentation

24.41.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

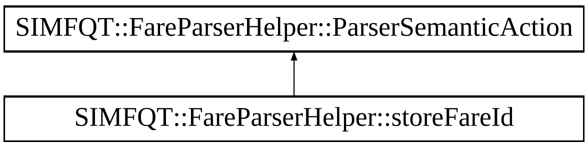
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()() SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()() SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.42 SIMFQT::FareParserHelper::storeFareId Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeFareId::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeFareId

Public Member Functions

- storeFareId (FareRuleStruct &)
- void operator() (unsigned int, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.42.1 Detailed Description

Store the parsed fare Id.

Definition at line 39 of file FareParserHelper.hpp.

24.42.2 Constructor & Destructor Documentation

24.42.2.1 SIMFQT::FareParserHelper::storeFareId::storeFareId (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 36 of file FareParserHelper.cpp.

24.42.3 Member Function Documentation

24.42.3.1 void SIMFQT::FareParserHelper::storeFareId::operator() (unsigned int *iFareId*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 41 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, SIMFQT::FareRuleStruct::_itSeconds, SIMFQT::FareRuleStruct::clearAirlineCodeList(), SIMFQT::FareRuleStruct::clearClassCodeList(), SIMFQT::FareRuleStruct::setAirlineCode(), SIMFQT::FareRuleStruct::setClassCode(), and SIMFQT::FareRuleStruct::setFareID().

24.42.4 Member Data Documentation

24.42.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

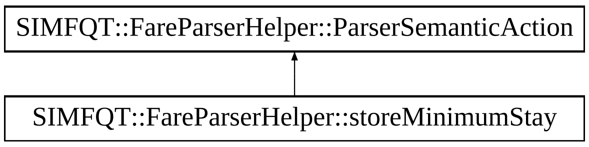
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()() SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()() SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.43 SIMFQT::FareParserHelper::storeMinimumStay Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeMinimumStay::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeMinimumStay

Public Member Functions

- storeMinimumStay (FareRuleStruct &)
- void operator() (unsigned int, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.43.1 Detailed Description

Store the parsed minimum stay.

Definition at line 190 of file FareParserHelper.hpp.

24.43.2 Constructor & Destructor Documentation

24.43.2.1 SIMFQT::FareParserHelper::storeMinimumStay::storeMinimumStay (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 346 of file FareParserHelper.cpp.

24.43.3 Member Function Documentation

24.43.3.1 void SIMFQT::FareParserHelper::storeMinimumStay::operator() (unsigned int *iMinStay*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 351 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, and SIMFQT::FareRuleStruct::setMinimumStay().

24.43.4 Member Data Documentation

24.43.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule [inherited]

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

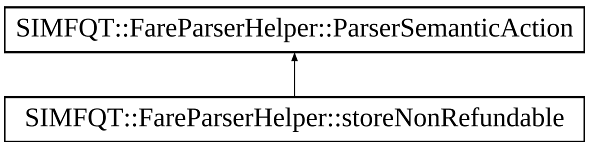
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()(), SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()(), SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.44 SIMFQT::FareParserHelper::storeNonRefundable Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeNonRefundable::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeNonRefundable

Public Member Functions

- storeNonRefundable (FareRuleStruct &)
- void operator() (char, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.44.1 Detailed Description

Store the parsed refundable option

Definition at line 180 of file FareParserHelper.hpp.

24.44.2 Constructor & Destructor Documentation

24.44.2.1 SIMFQT::FareParserHelper::storeNonRefundable::storeNonRefundable (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 321 of file FareParserHelper.cpp.

24.44.3 Member Function Documentation

24.44.3.1 void SIMFQT::FareParserHelper::storeNonRefundable::operator() (char *iNonRefundable*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 326 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, and SIMFQT::FareRuleStruct::setNonRefundable().

24.44.4 Member Data Documentation

24.44.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()(), SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()() SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

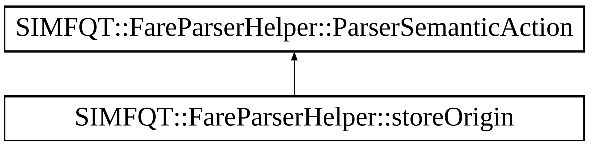
The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp

- simfqt/command/FareParserHelper.cpp

24.45 SIMFQT::FareParserHelper::storeOrigin Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeOrigin::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeOrigin

Public Member Functions

- storeOrigin (FareRuleStruct &)
- void operator() (std::vector< char >, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.45.1 Detailed Description

Store the parsed origin.

Definition at line 49 of file FareParserHelper.hpp.

24.45.2 Constructor & Destructor Documentation

24.45.2.1 SIMFQT::FareParserHelper::storeOrigin::storeOrigin (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 59 of file FareParserHelper.cpp.

24.45.3 Member Function Documentation

24.45.3.1 void SIMFQT::FareParserHelper::storeOrigin::operator() (std::vector< char > *iChar*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 64 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, and SIMFQT::FareRuleStruct::setOrigin().

24.45.4 Member Data Documentation

24.45.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

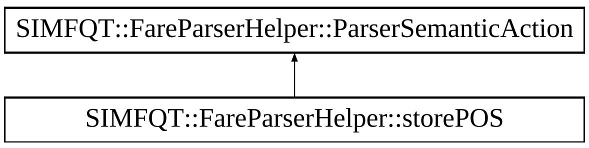
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()(), SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()() SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.46 SIMFQT::FareParserHelper::storePOS Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storePOS::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storePOS

Public Member Functions

- storePOS (FareRuleStruct &)
- void operator() (std::vector< char >, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.46.1 Detailed Description

Store the parsed customer point_of_sale.

Definition at line 120 of file FareParserHelper.hpp.

24.46.2 Constructor & Destructor Documentation

24.46.2.1 SIMFQT::FareParserHelper::storePOS::storePOS (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 186 of file FareParserHelper.cpp.

24.46.3 Member Function Documentation

24.46.3.1 void SIMFQT::FareParserHelper::storePOS::operator() (std::vector< char > *iChar*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 191 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, SIMFQT::FareRuleStruct::getDestination(), SIMFQT::FareRuleStruct::getOrigin(), and SIMFQT::FareRuleStruct::setPOS().

24.46.4 Member Data Documentation

24.46.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule [inherited]

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

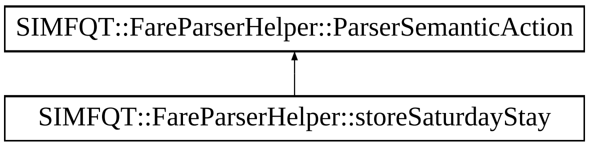
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()() SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()(), SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.47 SIMFQT::FareParserHelper::storeSaturdayStay Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeSaturdayStay::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeSaturdayStay

Public Member Functions

- storeSaturdayStay (FareRuleStruct &)
- void operator() (char, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.47.1 Detailed Description

Store the parsed saturday night.

Definition at line 160 of file FareParserHelper.hpp.

24.47.2 Constructor & Destructor Documentation

24.47.2.1 SIMFQT::FareParserHelper::storeSaturdayStay::storeSaturdayStay (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 270 of file FareParserHelper.cpp.

24.47.3 Member Function Documentation

24.47.3.1 void SIMFQT::FareParserHelper::storeSaturdayStay::operator() (char *iSaturdayStay*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 275 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, and SIMFQT::FareRuleStruct::setSaturdayStay().

24.47.4 Member Data Documentation

24.47.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

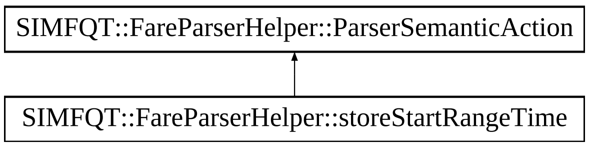
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()(), SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()() SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.48 SIMFQT::FareParserHelper::storeStartRangeTime Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeStartRangeTime::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeStartRangeTime

Public Member Functions

- storeStartRangeTime (FareRuleStruct &)
- void operator() (boost::spirit::qi::unused_type, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.48.1 Detailed Description

Store the parsed start range time.

Definition at line 100 of file FareParserHelper.hpp.

24.48.2 Constructor & Destructor Documentation

24.48.2.1 SIMFQT::FareParserHelper::storeStartRangeTime::storeStartRangeTime (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 150 of file FareParserHelper.cpp.

24.48.3 Member Function Documentation

24.48.3.1 void SIMFQT::FareParserHelper::storeStartRangeTime::operator() (boost::spirit::qi::unused_type, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 155 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, SIMFQT::FareRuleStruct::_itSeconds, SIMFQT::FareRuleStruct::calculateTime(), and SIMFQT::FareRuleStruct::setTimeRangeStart().

24.48.4 Member Data Documentation

24.48.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

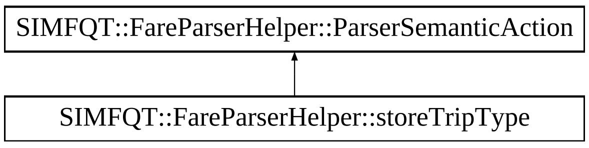
Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()() SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()(), SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.49 SIMFQT::FareParserHelper::storeTripType Struct Reference

`#include <simfqt/command/FareParserHelper.hpp>`Inheritance diagram for SIMFQT::FareParserHelper::storeTripType::


SIMFQT::FareParserHelper::ParserSemanticAction
SIMFQT::FareParserHelper::storeTripType

Public Member Functions

- storeTripType (FareRuleStruct &)

- void operator() (std::vector< char >, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Public Attributes

- FareRuleStruct & _fareRule

24.49.1 Detailed Description

Store the parsed customer trip type.

Definition at line 69 of file FareParserHelper.hpp.

24.49.2 Constructor & Destructor Documentation

24.49.2.1 SIMFQT::FareParserHelper::storeTripType::storeTripType (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 91 of file FareParserHelper.cpp.

24.49.3 Member Function Documentation

24.49.3.1 void SIMFQT::FareParserHelper::storeTripType::operator() (std::vector< char > *iChar*, boost::spirit::qi::unused_type, boost::spirit::qi::unused_type) const

Actor Function (functor).

Definition at line 96 of file FareParserHelper.cpp.

References SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule, and SIMFQT::FareRuleStruct::setTripType().

24.49.4 Member Data Documentation

24.49.4.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule `[inherited]`

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

Referenced by SIMFQT::FareParserHelper::doEndFare::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()(), SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()()

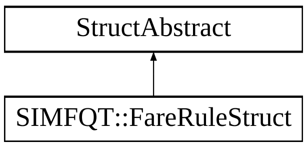
SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), and SIMFQT::FareParserHelper::storeFareId::operator()().

The documentation for this struct was generated from the following files:

- simfqt/command/FareParserHelper.hpp
- simfqt/command/FareParserHelper.cpp

24.50 StructAbstract Class Reference

Inheritance diagram for StructAbstract::



The documentation for this class was generated from the following file:

- simfqt/bom/FareRuleStruct.hpp

25 File Documentation

25.1 doc/local/authors.doc File Reference

25.2 doc/local/codingrules.doc File Reference

25.3 doc/local/copyright.doc File Reference

25.4 doc/local/documentation.doc File Reference

25.5 doc/local/features.doc File Reference

25.6 doc/local/help_wanted.doc File Reference

25.7 doc/local/howto_release.doc File Reference

25.8 doc/local/index.doc File Reference

25.9 doc/local/installation.doc File Reference

25.10 doc/local/linking.doc File Reference

25.11 doc/local/test.doc File Reference

25.12 doc/local/users_guide.doc File Reference

25.13 doc/local/verification.doc File Reference

25.14 doc/tutorial/tutorial.doc File Reference

25.15 simfqt/basic/BasConst.cpp File Reference

```
#include <simfqt/basic/BasConst_General.hpp>
#include <simfqt/basic/BasConst_SIMFQT_Service.hpp>
```

Namespaces

- namespace SIMFQT

Variables

- const std::string SIMFQT::DEFAULT_FARE_QUOTER_ID = "IATA"

25.16 BasConst.cpp

```
// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
#include <simfqt/basic/BasConst_General.hpp>
#include <simfqt/basic/BasConst_SIMFQT_Service.hpp>

namespace SIMFQT {

  const std::string DEFAULT_FARE_QUOTER_ID = "IATA";

}
```

25.17 simfqt/basic/BasConst_General.hpp File Reference

Namespaces

- namespace SIMFQT

25.18 BasConst_General.hpp

```
#ifndef __SIMFQT_BAS_BASCONST_GENERAL_HPP
#define __SIMFQT_BAS_BASCONST_GENERAL_HPP

// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////

namespace SIMFQT {

}
#endif // __SIMFQT_BAS_BASCONST_GENERAL_HPP
```

25.19 simfqt/basic/BasConst_SIMFQT_Service.hpp File Reference

```
#include <string>
```

Namespaces

- namespace SIMFQT

25.20 BasConst_SIMFQT_Service.hpp

```
#ifndef __SIMFQT_BAS_BASCONST_SIMFQT_SERVICE_HPP
#define __SIMFQT_BAS_BASCONST_SIMFQT_SERVICE_HPP

// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
#include <string>

namespace SIMFQT {

  extern const std::string DEFAULT_FARE_QUOTER_ID;

}
#endif // __SIMFQT_BAS_BASCONST_SIMFQT_SERVICE_HPP
```

25.21 simfqt/batches/simfqt_parseFareRules.cpp File Reference

```
#include <cassert>
#include <iostream>
#include <sstream>
#include <fstream>
#include <vector>
#include <list>
#include <string>
#include <boost/date_time/posix_time/posix_time.hpp>
#include <boost/date_time/gregorian/gregorian.hpp>
#include <boost/tokenizer.hpp>
#include <boost/program_options.hpp>
#include <stdair/STDAIR_Service.hpp>
#include <stdair/bom/TravelSolutionStruct.hpp>
#include <stdair/bom/BookingRequestStruct.hpp>
#include <stdair/service/Logger.hpp>
#include <simfqt/SIMFQT_Service.hpp>
#include <simfqt/config/simfqt-paths.hpp>
```

Typedefs

- typedef std::vector< std::string > WordList_T

Functions

- const std::string K_SIMFQT_DEFAULT_LOG_FILENAME ("simfqt_parseFareRules.log")
- const std::string K_SIMFQT_DEFAULT_FARE_INPUT_FILENAME (STDAIR_SAMPLE_-DIR"/fare01.csv")
- template<class T >
  std::ostream & operator<< (std::ostream &os, const std::vector< T > &v)
- int readConfiguration (int argc, char ∗argv[ ], bool &ioIsBuiltin, stdair::Filename_T &ioFareInput-Filename, std::string &ioLogFilename)
- int main (int argc, char ∗argv[ ])

Variables

- const bool K_SIMFQT_DEFAULT_BUILT_IN_INPUT = false
- const int K_SIMFQT_EARLY_RETURN_STATUS = 99

25.21.1 Typedef Documentation

25.21.1.1 typedef std::vector<std::string> WordList_T

Definition at line 24 of file simfqt_parseFareRules.cpp.

25.21.2 Function Documentation

25.21.2.1 const std::string K_SIMFQT_DEFAULT_LOG_FILENAME ("simfqt_parseFareRules.log")

Default name and location for the log file.

Referenced by readConfiguration().

25.21.2.2 const std::string K_SIMFQT_DEFAULT_FARE_INPUT_FILENAME (STDAIR_SAMPLE_DIR"/fare01.csv")

Default name and location for the (CSV) input file.

Referenced by readConfiguration().

25.21.2.3 template<class T > std::ostream& operator<< (std::ostream & *os*, const std::vector< T > & *v*) `[inline]`

Definition at line 44 of file simfqt_parseFareRules.cpp.

25.21.2.4 int readConfiguration (int *argc*, char * *argv*[ ], bool & *ioIsBuiltin*, stdair::Filename_T & *ioFareInputFilename*, std::string & *ioLogFilename*)

Read and parse the command line options.

Definition at line 51 of file simfqt_parseFareRules.cpp.

References K_SIMFQT_DEFAULT_BUILT_IN_INPUT, K_SIMFQT_DEFAULT_FARE_INPUT_FILENAME(), K_SIMFQT_DEFAULT_LOG_FILENAME(), and K_SIMFQT_EARLY_RETURN_STATUS.

Referenced by main().

25.21.2.5 int main (int *argc*, char * *argv*[ ])

Definition at line 154 of file simfqt_parseFareRules.cpp.

References SIMFQT::SIMFQT_Service::buildBookingRequest(), SIMFQT::SIMFQT_Service::buildSampleBom(), SIMFQT::SIMFQT_Service::buildSampleTravelSolutions(), SIMFQT::SIMFQT_Service::csvDisplay(), K_SIMFQT_EARLY_RETURN_STATUS, SIMFQT::SIMFQT_Service::parseAndLoad(), SIMFQT::SIMFQT_Service::quotePrices(), and readConfiguration().

25.21.3 Variable Documentation

25.21.3.1 const bool K_SIMFQT_DEFAULT_BUILT_IN_INPUT = false

Default for the input type. It can be either built-in or provided by an input file. That latter must then be given with the -i option.

Definition at line 37 of file simfqt_parseFareRules.cpp.

Referenced by readConfiguration().

25.21.3.2 const int K_SIMFQT_EARLY_RETURN_STATUS = 99

Early return status (so that it can be differentiated from an error).

Definition at line 40 of file simfqt_parseFareRules.cpp.

Referenced by main(), and readConfiguration().

25.22 simfqt_parseFareRules.cpp

```
// STL
#include <cassert>
#include <iostream>
#include <sstream>
#include <fstream>
#include <vector>
#include <list>
#include <string>
// Boost (Extended STL)
#include <boost/date_time/posix_time/posix_time.hpp>
#include <boost/date_time/gregorian/gregorian.hpp>
#include <boost/tokenizer.hpp>
#include <boost/program_options.hpp>
// StdAir
#include <stdair/STDAIR_Service.hpp>
#include <stdair/bom/TravelSolutionStruct.hpp>
#include <stdair/bom/BookingRequestStruct.hpp>
#include <stdair/service/Logger.hpp>
// Simfqt
#include <simfqt/SIMFQT_Service.hpp>
#include <simfqt/config/simfqt-paths.hpp>

// //////// Type definitions ///////
typedef std::vector<std::string> WordList_T;


// //////// Constants //////
const std::string K_SIMFQT_DEFAULT_LOG_FILENAME ("simfqt_parseFareRules.log");

const std::string K_SIMFQT_DEFAULT_FARE_INPUT_FILENAME (STDAIR_SAMPLE_DIR
                                                         "/fare01.csv");

const bool K_SIMFQT_DEFAULT_BUILT_IN_INPUT = false;

const int K_SIMFQT_EARLY_RETURN_STATUS = 99;

// ///////// Parsing of Options & Configuration /////////
// A helper function to simplify the main part.
template<class T> std::ostream& operator<< (std::ostream& os,
                                            const std::vector<T>& v) {
  std::copy (v.begin(), v.end(), std::ostream_iterator<T> (std::cout, " "));
  return os;
}

int readConfiguration (int argc, char* argv[], bool& ioIsBuiltin,
                       stdair::Filename_T& ioFareInputFilename,
                       std::string& ioLogFilename) {

  // Default for the built-in input
  ioIsBuiltin = K_SIMFQT_DEFAULT_BUILT_IN_INPUT;

  // Declare a group of options that will be allowed only on command line
  boost::program_options::options_description generic ("Generic options");
  generic.add_options()
    ("prefix", "print installation prefix")
    ("version,v", "print version string")
    ("help,h", "produce help message");

  // Declare a group of options that will be allowed both on command
  // line and in config file
  boost::program_options::options_description config ("Configuration");
  config.add_options()
    ("builtin,b",
     "The sample BOM tree can be either built-in or parsed from an input file. That latter must then be given with the -f/--fare option")
    ("fare,f",
     boost::program_options::value< std::string >(&ioFareInputFilename)->default_
value(K_SIMFQT_DEFAULT_FARE_INPUT_FILENAME),
     "(CSV) input file for the fare rules")
    ("log,l",
     boost::program_options::value< std::string >(&ioLogFilename)->default_value(
K_SIMFQT_DEFAULT_LOG_FILENAME),
     "Filename for the logs")
    ;

  // Hidden options, will be allowed both on command line and
  // in config file, but will not be shown to the user.
  boost::program_options::options_description hidden ("Hidden options");
  hidden.add_options()
    ("copyright",
     boost::program_options::value< std::vector<std::string> >(),
     "Show the copyright (license)");

  boost::program_options::options_description cmdline_options;
  cmdline_options.add(generic).add(config).add(hidden);

  boost::program_options::options_description config_file_options;
  config_file_options.add(config).add(hidden);

  boost::program_options::options_description visible ("Allowed options");
  visible.add(generic).add(config);

  boost::program_options::positional_options_description p;
  p.add ("copyright", -1);

  boost::program_options::variables_map vm;
  boost::program_options::
    store (boost::program_options::command_line_parser (argc, argv).
           options (cmdline_options).positional(p).run(), vm);

  std::ifstream ifs ("simfqt.cfg");
  boost::program_options::store (parse_config_file (ifs, config_file_options),
                                 vm);
  boost::program_options::notify (vm); if (vm.count ("help")) {
    std::cout << visible << std::endl;
    return K_SIMFQT_EARLY_RETURN_STATUS;
  }

  if (vm.count ("version")) {
    std::cout << PACKAGE_NAME << ", version " << PACKAGE_VERSION << std::endl;
    return K_SIMFQT_EARLY_RETURN_STATUS;
  }

  if (vm.count ("prefix")) {
    std::cout << "Installation prefix: " << PREFIXDIR << std::endl;
    return K_SIMFQT_EARLY_RETURN_STATUS;
  }

  if (vm.count ("builtin")) {
    ioIsBuiltin = true;
  }
  const std::string isBuiltinStr = (ioIsBuiltin == true)?"yes":"no";
  std::cout << "The BOM should be built-in? " << isBuiltinStr << std::endl;

  if (ioIsBuiltin == false) {

    // The BOM tree should be built from parsing a fare (and O&D) file
    if (vm.count ("fare")) {
      ioFareInputFilename = vm["fare"].as< std::string >();
      std::cout << "Input fare filename is: " << ioFareInputFilename
                << std::endl;

    } else {
      // The built-in option is not selected. However, no fare file
      // is specified
      std::cerr << "Either one among the -b/--builtin and -f/--fare "
                << "options must be specified" << std::endl;
    }
  }

  if (vm.count ("log")) {
    ioLogFilename = vm["log"].as< std::string >();
    std::cout << "Log filename is: " << ioLogFilename << std::endl;
  }

  return 0;
}


// /////////////// M A I N ////////////////
int main (int argc, char* argv[]) {

  // State whether the BOM tree should be built-in or parsed from an input file
  bool isBuiltin;

  // Fare input filename
  stdair::Filename_T lFareInputFilename;

  // Output log File
  stdair::Filename_T lLogFilename;

  // Call the command-line option parser
  const int lOptionParserStatus =
    readConfiguration (argc, argv, isBuiltin, lFareInputFilename, lLogFilename);

  if (lOptionParserStatus == K_SIMFQT_EARLY_RETURN_STATUS) {
    return 0;
  }

  // Set the log parameters
  std::ofstream logOutputFile;
  // Open and clean the log outputfile
  logOutputFile.open (lLogFilename.c_str());
  logOutputFile.clear();

  // Initialise the Simfqt service object
  const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);

  SIMFQT::SIMFQT_Service simfqtService (lLogParams);

  // DEBUG
  STDAIR_LOG_DEBUG ("Welcome to Simfqt");

  // Build a default sample list of travel solutions
  stdair::TravelSolutionList_T lTravelSolutionList;
  simfqtService.buildSampleTravelSolutions (lTravelSolutionList);

  // Build a default booking request
  stdair::BookingRequestStruct lBookingRequest =
    simfqtService.buildBookingRequest();

  // Check wether or not a (CSV) input file should be read
  if (isBuiltin == true) {

    // Build the default sample BOM tree (filled with fares) for Simfqt
    simfqtService.buildSampleBom();

  } else {

    // Build the BOM tree from parsing a fare file
    SIMFQT::FareFilePath lFareFilePath (lFareInputFilename);
    simfqtService.parseAndLoad (lFareFilePath);

  }

  // DEBUG: Display the travel solutions
  const std::string& lTSCSVDump =
    simfqtService.csvDisplay (lTravelSolutionList);
  STDAIR_LOG_DEBUG (lTSCSVDump);

  // FareQuote the sample list of travel solutions
  simfqtService.quotePrices (lBookingRequest, lTravelSolutionList);

  // DEBUG: Display the whole BOM tree
  const std::string& lBOMCSVDump = simfqtService.csvDisplay();
  STDAIR_LOG_DEBUG ("BOM tree: " << lBOMCSVDump);

  // DEBUG: Display the travel solutions
  const std::string& lTSCSVDumpEnd
    = simfqtService.csvDisplay (lTravelSolutionList);
  STDAIR_LOG_DEBUG (lTSCSVDumpEnd);

  // Close the Log outputFile
  logOutputFile.close();

  /*
    Note: as that program is not intended to be run on a server in
    production, it is better not to catch the exceptions. When it
    happens (that an exception is throwned), that way we get the
    call stack.
  */

  return 0;
}

```

25.23 simfqt/bom/FareRuleStruct.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <vector>
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/service/Logger.hpp>
#include <simfqt/bom/FareRuleStruct.hpp>
```

Namespaces

- namespace SIMFQT

25.24 FareRuleStruct.cpp

```
// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <cassert>
#include <sstream>
#include <vector>
// StdAir
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/service/Logger.hpp>
// SIMFQT
#include <simfqt/bom/FareRuleStruct.hpp>

namespace SIMFQT {

  // ////////////////////////////////////////////////////////////////////
  FareRuleStruct::FareRuleStruct ()
    :_fareId(0),
     _origin(""),
     _destination(""),
     _dateRangeStart(stdair::DEFAULT_DATE),
     _dateRangeEnd(stdair::DEFAULT_DATE),
     _timeRangeStart(stdair::DEFAULT_EPSILON_DURATION),
     _timeRangeEnd(stdair::DEFAULT_EPSILON_DURATION),
     _cabinCode (""),
     _pos (""),
     _advancePurchase(0),
     _saturdayStay("T"),
     _changeFees("T"),
     _nonRefundable("T"),
     _minimumStay(0),
     _fare(0),
     _airlineCode(""),
     _classCode("") {

  }

  // ////////////////////////////////////////////////////////////////////
  stdair::Date_T FareRuleStruct::calculateDate() const {
    _itYear.check(); _itMonth.check(); _itDay.check();
    return stdair::Date_T (_itYear._value, _itMonth._value, _itDay._value);
  }

  // ////////////////////////////////////////////////////////////////////
  stdair::Duration_T FareRuleStruct::calculateTime() const {
    _itHours.check(); _itMinutes.check(); _itSeconds.check();
    return boost::posix_time::hours (_itHours._value)
      + boost::posix_time::minutes (_itMinutes._value)
      + boost::posix_time::seconds (_itSeconds._value);
  }


  // ////////////////////////////////////////////////////////////////////
  const std::string FareRuleStruct::describe () const {

    std::ostringstream oStr;
    oStr << "FareRule: " << _fareId << ", ";

    oStr << _origin << "-" << _destination << " ("
         << _pos << "), " << _channel << ", [";
    oStr << _dateRangeStart << "/" << _dateRangeEnd << "] - ["
         << boost::posix_time::to_simple_string (_timeRangeStart) << "/"
         << boost::posix_time::to_simple_string (_timeRangeEnd) << "], ";

    oStr << _cabinCode << ", " << _fare  << " EUR, ";
    oStr << _tripType << ", " << _saturdayStay << ", "
         <<  _changeFees << ", " << _nonRefundable << ", "
         << _advancePurchase << ", " << _minimumStay << ", ";

    // Sanity check
    assert (_airlineCodeList.size() == _classCodeList.size());

    // Browse the airline and class pathes
    unsigned short idx = 0;
    stdair::ClassList_StringList_T::const_iterator itClass =
      _classCodeList.begin();
    for (stdair::AirlineCodeList_T::const_iterator itAirline =
           _airlineCodeList.begin();
         itAirline != _airlineCodeList.end(); ++itAirline, ++itClass, ++idx) {
      if (idx != 0) {
        oStr << " - ";
      }
      const stdair::AirlineCode_T lAirlineCode = *itAirline;
      const stdair::ClassCode_T lClassCode = *itClass;
      oStr << lAirlineCode << " / " << lClassCode;
    }

    return oStr.str();
  }

}

```

25.25 simfqt/bom/FareRuleStruct.hpp File Reference

```
#include <string>
#include <vector>
#include <stdair/stdair_demand_types.hpp>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/basic/StructAbstract.hpp>
#include <stdair/basic/BasParserHelperTypes.hpp>
#include <simfqt/SIMFQT_Types.hpp>
```

Classes

- struct SIMFQT::FareRuleStruct

Namespaces

- namespace SIMFQT

25.26 FareRuleStruct.hpp

```
#ifndef __SIMFQT_BOM_FARERULESTRUCT_HPP
#define __SIMFQT_BOM_FARERULESTRUCT_HPP

// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <string>
#include <vector>
// StdAir
#include <stdair/stdair_demand_types.hpp>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/basic/StructAbstract.hpp>
#include <stdair/basic/BasParserHelperTypes.hpp>
// SIMFQT
#include <simfqt/SIMFQT_Types.hpp>

namespace SIMFQT {

  struct FareRuleStruct : public stdair::StructAbstract {
  public:

    FareRuleStruct ();

  public:
    // ///////// Getters //////////
    SIMFQT::FareQuoteID_T getFareID () const {
      return _fareId;
    }

    stdair::AirportCode_T getOrigin () const {
      return _origin;
    }

    stdair::AirportCode_T getDestination () const {
      return _destination;
    }

    stdair::TripType_T getTripType () const {
      return _tripType;
    }

    stdair::Date_T getDateRangeStart () const {
      return _dateRangeStart;
    }

    stdair::Date_T getDateRangeEnd () const {
      return _dateRangeEnd;
    }

    stdair::Duration_T getTimeRangeStart () const {
      return _timeRangeStart;
    }

    stdair::Duration_T getTimeRangeEnd () const {
      return _timeRangeEnd;
    }

    stdair::CabinCode_T getCabinCode () const {
      return _cabinCode;
    }

    const stdair::CityCode_T getPOS () const {
      return _pos;
    }
    stdair::ChannelLabel_T getChannel () const {
      return _channel;
    }

    stdair::DayDuration_T getAdvancePurchase () const {
      return _advancePurchase;
    }

    stdair::SaturdayStay_T getSaturdayStay () const {
      return _saturdayStay;
    }

    stdair::ChangeFees_T getChangeFees () const {
      return _changeFees;
    }

    stdair::NonRefundable_T getNonRefundable () const {
      return _nonRefundable;
    }

    stdair::DayDuration_T getMinimumStay () const {
      return _minimumStay;
    }

    stdair::PriceValue_T getFare () const {
      return _fare;
    }

    stdair::AirlineCode_T getAirlineCode () const {
      return _airlineCode;
    }

    stdair::ClassCode_T getClassCode () const {
      return _classCode;
    }

    const unsigned int getAirlineListSize () const {
      return _airlineCodeList.size();
    }

    const unsigned int getClassCodeListSize () const {
      return _classCodeList.size();
    }

    stdair::AirlineCodeList_T getAirlineList () const {
      return _airlineCodeList;
    }

    stdair::ClassList_StringList_T getClassCodeList () const {
      return _classCodeList;
    }

  public:
    // ////////// Display support methods //////////
    stdair::Date_T calculateDate() const;

    stdair::Duration_T calculateTime() const;

    const std::string describe() const;

  public:
    // ////////// Setters //////////
    void setFareID (const SIMFQT::FareQuoteID_T& iFareQuoteID) {
      _fareId = iFareQuoteID;
    }

    void setOrigin (const stdair::AirportCode_T& iOrigin) {
      _origin = iOrigin;
    }

    void setDestination (const stdair::AirportCode_T& iDestination) {
      _destination = iDestination;
    }

    void setTripType (const stdair::TripType_T& iTripType) {
      _tripType = iTripType;
    }

    void setDateRangeStart (const stdair::Date_T& iDateRangeStart) {
      _dateRangeStart = iDateRangeStart;
    }

    void setDateRangeEnd (const stdair::Date_T& iDateRangeEnd) {
      _dateRangeEnd = iDateRangeEnd;
    }

    void setTimeRangeStart (const stdair::Duration_T& iTimeRangeStart) {
      _timeRangeStart = iTimeRangeStart;
    }

    void setTimeRangeEnd (const stdair::Duration_T& iTimeRangeEnd) {
      _timeRangeEnd = iTimeRangeEnd;
    }

    void setCabinCode (const stdair::CabinCode_T& iCabinCode) {
      _cabinCode = iCabinCode;
    }

    void setPOS (const stdair::CityCode_T& iPOS) {
      _pos = iPOS;
    }

    void setChannel (const stdair::ChannelLabel_T& iChannel) {
      _channel = iChannel;
    }

    void setAdvancePurchase (const stdair::DayDuration_T& iAdvancePurchase) {
      _advancePurchase = iAdvancePurchase;
    }

    void setSaturdayStay (const stdair::SaturdayStay_T& iSaturdayStay) {
      _saturdayStay = iSaturdayStay;
    }

    void setChangeFees (const stdair::ChangeFees_T& iChangeFees) {
      _changeFees = iChangeFees;
    }

    void setNonRefundable (const stdair::NonRefundable_T& iNonRefundable) {
      _nonRefundable = iNonRefundable;
    }

    void setMinimumStay (const stdair::DayDuration_T& iMinimumStay) {
      _minimumStay = iMinimumStay;
    }

    void setFare (const stdair::PriceValue_T& iFare) {
      _fare = iFare;
    }

    void setAirlineCode (const stdair::AirlineCode_T&  iAirlineCode) {
      _airlineCode = iAirlineCode;
    }

    void setClassCode (const stdair::ClassCode_T& iClassCode) {
      _classCode = iClassCode;
    }

    void clearAirlineCodeList () {
      _airlineCodeList.clear();
    }

    void clearClassCodeList () {
      _classCodeList.clear();
    }

    void addAirlineCode (const stdair::AirlineCode_T& iAirlineCode)  {
      _airlineCodeList.push_back (iAirlineCode);
    }

    void addClassCode (const stdair::ClassCode_T& iClassCode) {
      _classCodeList.push_back (iClassCode);
    }

  public:
    // ////////////////// Attributes /////////////////
    stdair::year_t _itYear;
    stdair::month_t _itMonth;
    stdair::day_t _itDay;

    stdair::hour_t _itHours;
    stdair::minute_t _itMinutes;
    stdair::second_t _itSeconds;

  private:
    // ////////////////// Attributes /////////////////
    SIMFQT::FareQuoteID_T _fareId;

    stdair::AirportCode_T _origin;

    stdair::AirportCode_T _destination;

    stdair::TripType_T _tripType;

    stdair::Date_T _dateRangeStart;

    stdair::Date_T _dateRangeEnd;

    stdair::Duration_T _timeRangeStart;

    stdair::Duration_T _timeRangeEnd;

    stdair::CabinCode_T _cabinCode;

    stdair::CityCode_T _pos;

    stdair::ChannelLabel_T _channel;

    stdair::DayDuration_T _advancePurchase;

    stdair::SaturdayStay_T _saturdayStay;

    stdair::ChangeFees_T _changeFees;

    stdair::NonRefundable_T _nonRefundable;

    stdair::DayDuration_T _minimumStay;

    stdair::PriceValue_T _fare;

    stdair::AirlineCode_T _airlineCode;

    stdair::ClassCode_T _classCode;

    stdair::AirlineCodeList_T _airlineCodeList;

    stdair::ClassList_StringList_T _classCodeList;

  };

}
#endif // __SIMFQT_BOM_FARERULESTRUCT_HPP
```

25.27 simfqt/command/FareParser.cpp File Reference

```
#include <cassert>
#include <string>
#include <stdair/basic/BasFileMgr.hpp>
#include <stdair/service/Logger.hpp>
#include <simfqt/command/FareParserHelper.hpp>
#include <simfqt/command/FareParser.hpp>
```

Namespaces

- namespace SIMFQT

25.28 FareParser.cpp

```
// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <cassert>
#include <string>
// StdAir
#include <stdair/basic/BasFileMgr.hpp>
#include <stdair/service/Logger.hpp>
// AirSched
#include <simfqt/command/FareParserHelper.hpp>
#include <simfqt/command/FareParser.hpp>

namespace SIMFQT {

  // //////////////////////////////////////////////////////////////////////
  void FareParser::fareRuleGeneration (const FareFilePath& iFareFilename,
                                       stdair::BomRoot& ioBomRoot) {

    const stdair::Filename_T lFilename = iFareFilename.name();

    // Check that the file path given as input corresponds to an actual file
    const bool doesExistAndIsReadable =
      stdair::BasFileMgr::doesExistAndIsReadable (lFilename);
    if (doesExistAndIsReadable == false) {
      STDAIR_LOG_ERROR ("The fare input file, '" << lFilename
                        << "', can not be retrieved on the file-system");
      throw FareInputFileNotFoundException ("The fare input file '" + lFilename
                                            + "' does not exist or can not "
                                            "be read");
    }

    // Initialise the fare file parser.
    FareRuleFileParser lFareRuleFileParser (ioBomRoot, lFilename);

    // Parse the CSV-formatted fare input file and generate the
    // corresponding fare rules.
    lFareRuleFileParser.generateFareRules ();

  }

}
```

25.29 simfqt/command/FareParser.hpp File Reference

```
#include <string>
#include <stdair/stdair_basic_types.hpp>
#include <stdair/command/CmdAbstract.hpp>
#include <simfqt/SIMFQT_Types.hpp>
```

Classes

- class SIMFQT::FareParser

Namespaces

- namespace stdair

  Forward declarations.

- namespace SIMFQT

25.30 FareParser.hpp

```
#ifndef __SIMFQT_CMD_FAREPARSER_HPP
#define __SIMFQT_CMD_FAREPARSER_HPP

// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <string>
// StdAir
#include <stdair/stdair_basic_types.hpp>
#include <stdair/command/CmdAbstract.hpp>
// SimFQT
#include <simfqt/SIMFQT_Types.hpp>

// Forward declarations.
namespace stdair {
  class BomRoot;
}

namespace SIMFQT {

  class FareParser : public stdair::CmdAbstract {
  public:
    static void fareRuleGeneration (const FareFilePath&, stdair::BomRoot&);
  };
}
#endif // __SIMFQT_CMD_FAREPARSER_HPP
```

25.31 simfqt/command/FareParserHelper.cpp File Reference

```
#include <cassert>
#include <vector>
#include <fstream>
#include <stdair/basic/BasFileMgr.hpp>
#include <stdair/basic/BasConst_Request.hpp>
#include <stdair/bom/BomRoot.hpp>
#include <stdair/service/Logger.hpp>
#include <stdair/basic/BasParserTypes.hpp>
#include <simfqt/command/FareParserHelper.hpp>
#include <simfqt/command/FareRuleGenerator.hpp>
```

Classes

- struct SIMFQT::FareParserHelper::FareRuleParser< Iterator >

Namespaces

- namespace SIMFQT
- namespace SIMFQT::FareParserHelper

Variables

- stdair::int1_p_t SIMFQT::FareParserHelper::int1_p
- stdair::uint2_p_t SIMFQT::FareParserHelper::uint2_p
- stdair::uint4_p_t SIMFQT::FareParserHelper::uint4_p
- stdair::uint1_4_p_t SIMFQT::FareParserHelper::uint1_4_p
- stdair::hour_p_t SIMFQT::FareParserHelper::hour_p
- stdair::minute_p_t SIMFQT::FareParserHelper::minute_p
- stdair::second_p_t SIMFQT::FareParserHelper::second_p
- stdair::year_p_t SIMFQT::FareParserHelper::year_p
- stdair::month_p_t SIMFQT::FareParserHelper::month_p
- stdair::day_p_t SIMFQT::FareParserHelper::day_p

25.32 FareParserHelper.cpp

```
 // //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <cassert>
#include <vector>
#include <fstream>
// StdAir
#include <stdair/basic/BasFileMgr.hpp>
#include <stdair/basic/BasConst_Request.hpp>
#include <stdair/bom/BomRoot.hpp>
#include <stdair/service/Logger.hpp>
//#define BOOST_SPIRIT_DEBUG
#include <stdair/basic/BasParserTypes.hpp>
// SIMFQT
#include <simfqt/command/FareParserHelper.hpp>
#include <simfqt/command/FareRuleGenerator.hpp>



namespace SIMFQT {

  namespace FareParserHelper {

    // //////////////////////////////////////////////////////////////////
    //  Semantic actions
    // //////////////////////////////////////////////////////////////////

    ParserSemanticAction::
    ParserSemanticAction (FareRuleStruct& ioFareRule)
      : _fareRule (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    storeFareId::
    storeFareId (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storeFareId::operator() (unsigned int iFareId,
                                  boost::spirit::qi::unused_type,
                                  boost::spirit::qi::unused_type) const {
      _fareRule.setFareID (iFareId);

      // DEBUG
      //STDAIR_LOG_DEBUG ( "Fare Id: " << _fareRule.getFareID ());
      const stdair::AirlineCode_T lEmptyAirlineCode ("");
      _fareRule.setAirlineCode(lEmptyAirlineCode);
      _fareRule.clearAirlineCodeList();
      const stdair::ClassCode_T lEmptyClassCode ("");
      _fareRule.setClassCode(lEmptyClassCode);
      _fareRule.clearClassCodeList();
      _fareRule._itSeconds = 0;
    }

    // //////////////////////////////////////////////////////////////////
    storeOrigin ::
    storeOrigin (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storeOrigin::operator() (std::vector<char> iChar,
                                  boost::spirit::qi::unused_type,
                                         boost::spirit::qi::unused_type) const {
      const stdair::AirportCode_T lOrigin (iChar.begin(), iChar.end());
      _fareRule.setOrigin (lOrigin);
      // DEBUG
      //STDAIR_LOG_DEBUG ( "Origin: " << _fareRule.getOrigin ());
    }

    // //////////////////////////////////////////////////////////////////
    storeDestination ::
    storeDestination (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storeDestination::operator() (std::vector<char> iChar,
                                       boost::spirit::qi::unused_type,
                                       boost::spirit::qi::unused_type) const {
      const stdair::AirportCode_T lDestination (iChar.begin(), iChar.end());
      _fareRule.setDestination (lDestination);
      // DEBUG
      //STDAIR_LOG_DEBUG ( "Destination: " << _fareRule.getDestination ());
    }

    // //////////////////////////////////////////////////////////////////
    storeTripType ::
    storeTripType (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storeTripType::operator() (std::vector<char> iChar,
                                    boost::spirit::qi::unused_type,
                                    boost::spirit::qi::unused_type) const {
      const stdair::TripType_T lTripType (iChar.begin(), iChar.end());
      if (lTripType == "OW" || lTripType == "RT") {
        _fareRule.setTripType (lTripType);
      } else {
        // ERROR
        STDAIR_LOG_ERROR ("Invalid trip type  " << lTripType);
      }
      // DEBUG
      //STDAIR_LOG_DEBUG ("TripType: " << _fareRule.getTripType ());
    }


    // //////////////////////////////////////////////////////////////////
    storeDateRangeStart::
    storeDateRangeStart (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storeDateRangeStart::operator() (boost::spirit::qi::unused_type,
                                          boost::spirit::qi::unused_type,
                                          boost::spirit::qi::unused_type) const {

      const stdair::Date_T& lDateStart = _fareRule.calculateDate ();
      _fareRule.setDateRangeStart (lDateStart);
      // DEBUG
      //STDAIR_LOG_DEBUG ("Date Range Start: " << _fareRule.getDateRangeStart ())
;
    }

    // //////////////////////////////////////////////////////////////////
    storeDateRangeEnd::
    storeDateRangeEnd(FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storeDateRangeEnd::operator() (boost::spirit::qi::unused_type,
                                        boost::spirit::qi::unused_type,
                                        boost::spirit::qi::unused_type) const {
      const stdair::Date_T& lDateEnd = _fareRule.calculateDate ();
      // As a Boost date period (DatePeriod_T) defines the last day of
      // the period to be end-date - one day, we have to add one day to that
      // end date before.
      const stdair::DateOffset_T oneDay (1);
      const stdair::Date_T lBoostDateEnd = lDateEnd + oneDay;
      _fareRule.setDateRangeEnd (lBoostDateEnd);
       // DEBUG
      //STDAIR_LOG_DEBUG ("Date Range End: " << _fareRule.getDateRangeEnd ());
    }

    // //////////////////////////////////////////////////////////////////
    storeStartRangeTime::
    storeStartRangeTime (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storeStartRangeTime::operator() (boost::spirit::qi::unused_type,
                                          boost::spirit::qi::unused_type,
                                          boost::spirit::qi::unused_type) const {

      const stdair::Duration_T& lTimeStart = _fareRule.calculateTime ();
      _fareRule.setTimeRangeStart (lTimeStart);
      // DEBUG
      //STDAIR_LOG_DEBUG ("Time Range Start: " << _fareRule.getTimeRangeStart ())
;
      // Reset the number of seconds
      _fareRule._itSeconds = 0;
    }

    // //////////////////////////////////////////////////////////////////
    storeEndRangeTime::
    storeEndRangeTime (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storeEndRangeTime::operator() (boost::spirit::qi::unused_type,
                                        boost::spirit::qi::unused_type,
                                        boost::spirit::qi::unused_type) const {
      const stdair::Duration_T& lTimeEnd = _fareRule.calculateTime ();
      _fareRule.setTimeRangeEnd (lTimeEnd);
      // DEBUG
      //STDAIR_LOG_DEBUG ("Time Range End: " << _fareRule.getTimeRangeEnd ());
      // Reset the number of seconds
      _fareRule._itSeconds = 0;
    }

    // //////////////////////////////////////////////////////////////////
    storePOS ::
    storePOS (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storePOS::operator() (std::vector<char> iChar,
                               boost::spirit::qi::unused_type,
                               boost::spirit::qi::unused_type) const {
      const stdair::CityCode_T lPOS (iChar.begin(), iChar.end());
      if (lPOS == _fareRule.getOrigin() || lPOS == _fareRule.getDestination()) {
        _fareRule.setPOS (lPOS);
      } else if (lPOS == "ROW") {
        const stdair::CityCode_T lPOSROW ("ROW");
        _fareRule.setPOS (lPOSROW);
      } else if (lPOS == stdair::DEFAULT_POS) {
        _fareRule.setPOS (stdair::DEFAULT_POS);
      } else {
        // ERROR
        STDAIR_LOG_ERROR ("Invalid point of sale " << lPOS);
      }
      // DEBUG
      //STDAIR_LOG_DEBUG ("POS: " << _fareRule.getPOS ());
    }

    // //////////////////////////////////////////////////////////////////
    storeCabinCode ::
    storeCabinCode (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storeCabinCode::operator() (char iChar,
                                     boost::spirit::qi::unused_type,
                                     boost::spirit::qi::unused_type) const {
      std::ostringstream ostr;
      ostr << iChar;
      const std::string cabinCodeStr = ostr.str();
      const stdair::CabinCode_T& lCabinCode (cabinCodeStr);
      _fareRule.setCabinCode (lCabinCode);

      // DEBUG
      //STDAIR_LOG_DEBUG ("Cabin Code: " << _fareRule.getCabinCode ());


    }

    // //////////////////////////////////////////////////////////////////
    storeChannel ::
    storeChannel (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storeChannel::operator() (std::vector<char> iChar,
                                   boost::spirit::qi::unused_type,
                                   boost::spirit::qi::unused_type) const {
      const stdair::ChannelLabel_T lChannel (iChar.begin(), iChar.end());
      if (lChannel != "IN" && lChannel != "IF" && lChannel != "DN"
          && lChannel != "DF" && lChannel != stdair::DEFAULT_CHANNEL) {
        // ERROR
        STDAIR_LOG_ERROR ("Invalid channel " << lChannel);
      }
      _fareRule.setChannel (lChannel);
      // DEBUG
      //STDAIR_LOG_DEBUG ("Channel: " << _fareRule.getChannel ());
    }

    // //////////////////////////////////////////////////////////////////
    storeAdvancePurchase ::
    storeAdvancePurchase (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storeAdvancePurchase::operator() (unsigned int iAdancePurchase,
                                           boost::spirit::qi::unused_type,
                                           boost::spirit::qi::unused_type) const
    {
      const stdair::DayDuration_T& lAdancePurchase = iAdancePurchase;
      _fareRule.setAdvancePurchase (lAdancePurchase);
      // DEBUG
      //STDAIR_LOG_DEBUG ( "Advance Purchase: " << _fareRule.getAdvancePurchase (
));
    }

    // //////////////////////////////////////////////////////////////////////
    storeSaturdayStay ::
    storeSaturdayStay (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////////
    void storeSaturdayStay::operator() (char iSaturdayStay,
                                        boost::spirit::qi::unused_type,
                                        boost::spirit::qi::unused_type) const {
      bool lBool = false;
      if (iSaturdayStay == 'T') {
        lBool = true;
      } else {
        if (iSaturdayStay != 'F') {
          // DEBUG
          STDAIR_LOG_DEBUG ("Invalid saturdayStay char " << iSaturdayStay);
        }
      }
      const stdair::SaturdayStay_T lSaturdayStay (lBool);
      _fareRule.setSaturdayStay (lSaturdayStay);
      // DEBUG
      //STDAIR_LOG_DEBUG ("Saturday Stay: " << _fareRule.getSaturdayStay ());
    }

    // //////////////////////////////////////////////////////////////////////
    storeChangeFees ::
    storeChangeFees (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////////
    void storeChangeFees::operator() (char iChangefees,
                                      boost::spirit::qi::unused_type,
                                      boost::spirit::qi::unused_type) const {

      bool lBool = false;
      if (iChangefees == 'T') {
        lBool = true;
      } else {
        if (iChangefees != 'F') {
          // DEBUG
          STDAIR_LOG_DEBUG ("Invalid change fees char " << iChangefees);
        }
      }
      const stdair::ChangeFees_T lChangefees (lBool);
      _fareRule.setChangeFees (lChangefees);
      // DEBUG
      //STDAIR_LOG_DEBUG ("Change fees: " << _fareRule.getChangeFees ());
    }

    // //////////////////////////////////////////////////////////////////////
    storeNonRefundable ::
    storeNonRefundable (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////////
    void storeNonRefundable::operator() (char iNonRefundable,
                                         boost::spirit::qi::unused_type,
                                         boost::spirit::qi::unused_type) const {
       bool lBool = false;
       if (iNonRefundable == 'T') {
         lBool = true;
       } else {
         if (iNonRefundable != 'F') {
           // DEBUG
           STDAIR_LOG_DEBUG ("Invalid non refundable char " << iNonRefundable);
         }
       }
       const stdair::NonRefundable_T lNonRefundable (lBool);
       _fareRule.setNonRefundable (lNonRefundable);
       // DEBUG
       //STDAIR_LOG_DEBUG ("Non refundable: " << _fareRule.getNonRefundable ());
    }

    // //////////////////////////////////////////////////////////////////
    storeMinimumStay ::
    storeMinimumStay (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storeMinimumStay::operator() (unsigned int iMinStay,
                                       boost::spirit::qi::unused_type,
                                       boost::spirit::qi::unused_type) const {
      const stdair::DayDuration_T lMinStay = iMinStay;
      _fareRule.setMinimumStay (lMinStay);
      // DEBUG
      //STDAIR_LOG_DEBUG ("Minimum Stay: " << _fareRule.getMinimumStay ());
    }

    // //////////////////////////////////////////////////////////////////
    storeFare ::
    storeFare (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storeFare::operator() (double iFare,
                                boost::spirit::qi::unused_type,
                                boost::spirit::qi::unused_type) const {
     const stdair::PriceValue_T lFare = iFare;
      _fareRule.setFare (lFare);
      // DEBUG
      //STDAIR_LOG_DEBUG ("Fare: " << _fareRule.getFare ());
    }

    // //////////////////////////////////////////////////////////////////
    storeAirlineCode ::
    storeAirlineCode (FareRuleStruct& ioFareRule)
      : ParserSemanticAction (ioFareRule) {
    }

    // //////////////////////////////////////////////////////////////////
    void storeAirlineCode::operator() (std::vector<char> iChar,
                                       boost::spirit::qi::unused_type,
                                       boost::spirit::qi::unused_type) const {

      const stdair::AirlineCode_T lAirlineCode (iChar.begin(), iChar.end());
      // Insertion of this airline Code list in the whole AirlineCode name
      _fareRule.addAirlineCode (lAirlineCode);
      // DEBUG
      //STDAIR_LOG_DEBUG ( "Airline code: " << lAirlineCode);
    }

```

```
// //////////////////////////////////////////////////////////////////
storeClass ::
storeClass (FareRuleStruct& ioFareRule)
  : ParserSemanticAction (ioFareRule) {
}

// //////////////////////////////////////////////////////////////////
void storeClass::operator() (std::vector<char> iChar,
                             boost::spirit::qi::unused_type,
                             boost::spirit::qi::unused_type) const {
  std::ostringstream ostr;
  for (std::vector<char>::const_iterator lItVector = iChar.begin();
     lItVector != iChar.end();
     lItVector++) {
    ostr << *lItVector;
  }
  const std::string classCodeStr = ostr.str();
  const stdair::ClassCode_T lClassCode (classCodeStr);
  // Insertion of this class Code list in the whole classCode name
  _fareRule.addClassCode (lClassCode);
  // DEBUG
  //STDAIR_LOG_DEBUG ("Class Code: " << lClassCode);
}

// //////////////////////////////////////////////////////////////////
doEndFare::
doEndFare (stdair::BomRoot& ioBomRoot,
           FareRuleStruct& ioFareRule)
  : ParserSemanticAction (ioFareRule),
    _bomRoot (ioBomRoot) {
}

// //////////////////////////////////////////////////////////////////
void doEndFare::operator() (boost::spirit::qi::unused_type,
                            boost::spirit::qi::unused_type,
                            boost::spirit::qi::unused_type) const {
  // DEBUG
  //STDAIR_LOG_DEBUG ("Do End");
  // Generation of the fare rule object.
  FareRuleGenerator::createAirportPair (_bomRoot, _fareRule);
  STDAIR_LOG_DEBUG(_fareRule.describe());
}

// ///////////////////////////////////////////////////////////////////
//
//  Utility Parsers
//
// ///////////////////////////////////////////////////////////////////
namespace bsq = boost::spirit::qi;
namespace bsa = boost::spirit::ascii;

stdair::int1_p_t int1_p;

stdair::uint2_p_t uint2_p;

stdair::uint4_p_t uint4_p;

stdair::uint1_4_p_t uint1_4_p;

stdair::hour_p_t hour_p;
stdair::minute_p_t minute_p;
stdair::second_p_t second_p;

stdair::year_p_t year_p;
stdair::month_p_t month_p;
stdair::day_p_t day_p;

```

```
    //
    //  (Boost Spirit) Grammar Definition
    //

    template <typename Iterator>
    struct FareRuleParser :
      public boost::spirit::qi::grammar<Iterator,
                                        boost::spirit::ascii::space_type> {

      FareRuleParser (stdair::BomRoot& ioBomRoot,
                      FareRuleStruct& iofareRule) :

        FareRuleParser::base_type(start),
        _bomRoot(ioBomRoot), _fareRule(iofareRule) {


      start = *(comments | fare_rule);

      comments = (bsq::lexeme[bsq::repeat(2)[bsa::char_('/')]
                              >> +(bsa::char_ - bsq::eol)
                              >> bsq::eol]
                  | bsq::lexeme[bsa::char_('/') >>bsa::char_('*')
                                >> +(bsa::char_ - bsa::char_('*'))
                                >> bsa::char_('*') >> bsa::char_('/')]);

      fare_rule = fare_key
        >> +( ';' >> segment )
        >> fare_rule_end[doEndFare(_bomRoot, _fareRule)];

      fare_rule_end = bsa::char_(';');

      fare_key = fare_id
        >> ';' >> origin >> ';' >> destination
        >> ';' >> tripType
        >> ';' >> dateRangeStart >> ';' >> dateRangeEnd
        >> ';' >> timeRangeStart >> ';' >> timeRangeEnd
        >> ';' >> point_of_sale >>  ';' >> cabinCode >> ';' >> channel
        >> ';' >> advancePurchase >> ';' >> saturdayStay
        >> ';' >> changeFees >> ';' >> nonRefundable
        >> ';' >> minimumStay >> ';' >> fare;

      fare_id = uint1_4_p[storeFareId(_fareRule)];

      origin = bsq::repeat(3)[bsa::char_("A-Z")][storeOrigin(_fareRule)];

      destination =
        bsq::repeat(3)[bsa::char_("A-Z")][storeDestination(_fareRule)];

      tripType =
        bsq::repeat(2)[bsa::char_("A-Z")][storeTripType(_fareRule)];

      dateRangeStart = date[storeDateRangeStart(_fareRule)];

      dateRangeEnd = date[storeDateRangeEnd(_fareRule)];

      date = bsq::lexeme
        [year_p[boost::phoenix::ref(_fareRule._itYear) = bsq::labels::_1]
        >> '-'
        >> month_p[boost::phoenix::ref(_fareRule._itMonth) = bsq::labels::_1]
        >> '-'
        >> day_p[boost::phoenix::ref(_fareRule._itDay) = bsq::labels::_1] ];

      timeRangeStart = time[storeStartRangeTime(_fareRule)];

      timeRangeEnd = time[storeEndRangeTime(_fareRule)];

      time =  bsq::lexeme
```

```
          [hour_p[boost::phoenix::ref(_fareRule._itHours) = bsq::labels::_1]
          >> ':'
          >> minute_p[boost::phoenix::ref(_fareRule._itMinutes) = bsq::labels::_1]

          >> - (':' >> second_p[boost::phoenix::ref(_fareRule._itSeconds) = bsq::la
      bels::_1]) ];

        point_of_sale = bsq::repeat(3)[bsa::char_("A-Z")][storePOS(_fareRule)];

        cabinCode = bsa::char_("A-Z")[storeCabinCode(_fareRule)];

        channel = bsq::repeat(2)[bsa::char_("A-Z")][storeChannel(_fareRule)];

        advancePurchase = uint1_4_p[storeAdvancePurchase(_fareRule)];

        saturdayStay = bsa::char_("A-Z")[storeSaturdayStay(_fareRule)];

        changeFees = bsa::char_("A-Z")[storeChangeFees(_fareRule)];

        nonRefundable = bsa::char_("A-Z")[storeNonRefundable(_fareRule)];

        minimumStay = uint1_4_p[storeMinimumStay(_fareRule)];

        fare = bsq::double_[storeFare(_fareRule)];

        segment = bsq::repeat(2)[bsa::char_("A-Z")][storeAirlineCode(_fareRule)]
          >> ';'
          >> bsq::repeat(1,bsq::inf)[bsa::char_("A-Z")][storeClass(_fareRule)];

        //BOOST_SPIRIT_DEBUG_NODE (FareRuleParser);
        BOOST_SPIRIT_DEBUG_NODE (start);
        BOOST_SPIRIT_DEBUG_NODE (comments);
        BOOST_SPIRIT_DEBUG_NODE (fare_rule);
        BOOST_SPIRIT_DEBUG_NODE (fare_rule_end);
        BOOST_SPIRIT_DEBUG_NODE (fare_key);
        BOOST_SPIRIT_DEBUG_NODE (fare_id);
        BOOST_SPIRIT_DEBUG_NODE (origin);
        BOOST_SPIRIT_DEBUG_NODE (destination);
        BOOST_SPIRIT_DEBUG_NODE (tripType);
        BOOST_SPIRIT_DEBUG_NODE (dateRangeStart);
        BOOST_SPIRIT_DEBUG_NODE (dateRangeEnd);
        BOOST_SPIRIT_DEBUG_NODE (date);
        BOOST_SPIRIT_DEBUG_NODE (timeRangeStart);
        BOOST_SPIRIT_DEBUG_NODE (time);
        BOOST_SPIRIT_DEBUG_NODE (point_of_sale);
        BOOST_SPIRIT_DEBUG_NODE (cabinCode);
        BOOST_SPIRIT_DEBUG_NODE (channel);
        BOOST_SPIRIT_DEBUG_NODE (advancePurchase);
        BOOST_SPIRIT_DEBUG_NODE (saturdayStay);
        BOOST_SPIRIT_DEBUG_NODE (changeFees);
        BOOST_SPIRIT_DEBUG_NODE (nonRefundable);
        BOOST_SPIRIT_DEBUG_NODE (minimumStay);
        BOOST_SPIRIT_DEBUG_NODE (fare);
        BOOST_SPIRIT_DEBUG_NODE (segment);

        }

        // Instantiation of rules
        boost::spirit::qi::rule<Iterator,
                                boost::spirit::ascii::space_type>
        start, comments, fare_rule, fare_rule_end, fare_key, fare_id, origin,
          destination, tripType, dateRangeStart, dateRangeEnd, date,
          timeRangeStart, timeRangeEnd, time, point_of_sale, cabinCode, channel,
          advancePurchase, saturdayStay, changeFees, nonRefundable, minimumStay,
          fare, segment;

        // Parser Context
        stdair::BomRoot& _bomRoot;
        FareRuleStruct& _fareRule;
      };

    }


    //
    //  Entry class for the file parser
    //

    // //////////////////////////////////////////////////////////////////////
    FareRuleFileParser::
    FareRuleFileParser (stdair::BomRoot& ioBomRoot,
                        const stdair::Filename_T& iFilename)
      : _filename (iFilename), _bomRoot (ioBomRoot) {
      init();
    }

    // //////////////////////////////////////////////////////////////////////
    void FareRuleFileParser::init() {
      // Check that the file exists and is readable
      const bool doesExistAndIsReadable =
        stdair::BasFileMgr::doesExistAndIsReadable (_filename);

      if (doesExistAndIsReadable == false) {
        STDAIR_LOG_ERROR ("The fare schedule file " << _filename
                          << " does not exist or can not be  read.");

        throw FareInputFileNotFoundException ("The fare file " + _filename
                                              + " does not exist or can not be read
  ");
      }
    }

    // //////////////////////////////////////////////////////////////////////
    void FareRuleFileParser::generateFareRules () {

      STDAIR_LOG_DEBUG ("Parsing fare input file: " << _filename);

      // File to be parsed
      const std::string* lFileName = &_filename;
      const char *lChar = (*lFileName).c_str();
      std::ifstream fileToBeParsed(lChar, std::ios_base::in);

      // Check if the filename exist and can be open
      if (fileToBeParsed.is_open() == false) {
        STDAIR_LOG_ERROR ("The fare file " << _filename << " can not be open."
                            << std::endl);

        throw FareInputFileNotFoundException ("The file " + _filename
                                              + " does not exist or can not be read
  ");
      }

      // Create an input iterator
      stdair::base_iterator_t inputBegin (fileToBeParsed);

      // Convert input iterator to an iterator usable by spirit parser
      stdair::iterator_t
        start (boost::spirit::make_default_multi_pass (inputBegin));
      stdair::iterator_t end;

      // Initialise the parser (grammar) with the helper/staging structure.
      FareParserHelper::FareRuleParser<stdair::iterator_t> lFPParser(_bomRoot, _far
  eRule);

```

```
    // Launch the parsing of the file and, thanks to the doEndFare
    // call-back structure, the building of the whole BomRoot BOM
    const bool hasParsingBeenSuccesful =
       boost::spirit::qi::phrase_parse (start, end, lFPParser,
                                        boost::spirit::ascii::space);

    if (hasParsingBeenSuccesful == false) {
      STDAIR_LOG_ERROR ("Parsing of fare input file: " << _filename
                        << " failed");
      throw FareFileParsingFailedException ("Parsing of fare input file: "
                                            + _filename + " failed");
    }

    if  (start != end) {
      STDAIR_LOG_ERROR ("Parsing of fare input file: " << _filename
                        << " failed");
      throw FareFileParsingFailedException ("Parsing of fare input file: "
                                            + _filename + " failed");
    }

    if (hasParsingBeenSuccesful == true && start == end) {
      STDAIR_LOG_DEBUG ("Parsing of fare input file: " << _filename
      << " succeeded");
    }

  }

}
```

25.33 simfqt/command/FareParserHelper.hpp File Reference

```
#include <string>
#include <boost/spirit/include/qi.hpp>
#include <stdair/command/CmdAbstract.hpp>
#include <simfqt/SIMFQT_Types.hpp>
#include <simfqt/bom/FareRuleStruct.hpp>
```

Classes

- struct SIMFQT::FareParserHelper::ParserSemanticAction
- struct SIMFQT::FareParserHelper::storeFareId
- struct SIMFQT::FareParserHelper::storeOrigin
- struct SIMFQT::FareParserHelper::storeDestination
- struct SIMFQT::FareParserHelper::storeTripType
- struct SIMFQT::FareParserHelper::storeDateRangeStart
- struct SIMFQT::FareParserHelper::storeDateRangeEnd
- struct SIMFQT::FareParserHelper::storeStartRangeTime
- struct SIMFQT::FareParserHelper::storeEndRangeTime
- struct SIMFQT::FareParserHelper::storePOS
- struct SIMFQT::FareParserHelper::storeCabinCode
- struct SIMFQT::FareParserHelper::storeChannel
- struct SIMFQT::FareParserHelper::storeAdvancePurchase
- struct SIMFQT::FareParserHelper::storeSaturdayStay
- struct SIMFQT::FareParserHelper::storeChangeFees
- struct SIMFQT::FareParserHelper::storeNonRefundable
- struct SIMFQT::FareParserHelper::storeMinimumStay
- struct SIMFQT::FareParserHelper::storeFare
- struct SIMFQT::FareParserHelper::storeAirlineCode
- struct SIMFQT::FareParserHelper::storeClass
- struct SIMFQT::FareParserHelper::doEndFare
- class SIMFQT::FareRuleFileParser

Namespaces

- namespace stdair

  Forward declarations.

- namespace SIMFQT
- namespace SIMFQT::FareParserHelper

25.34 FareParserHelper.hpp

```
#ifndef __SIMFQT_CMD_FAREPARSERHELPER_HPP
#define __SIMFQT_CMD_FAREPARSERHELPER_HPP

// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <string>
// Boost
#include <boost/spirit/include/qi.hpp>
// StdAir
#include <stdair/command/CmdAbstract.hpp>
// Simfqt
#include <simfqt/SIMFQT_Types.hpp>
#include <simfqt/bom/FareRuleStruct.hpp>

// Forward declarations
namespace stdair {
  class BomRoot;
}

namespace SIMFQT {

  namespace FareParserHelper {

    // ////////////////////////////////////////////////////////////////////
    //  Semantic actions
    // ///////////////////////////////////////////////////

    struct ParserSemanticAction {
      ParserSemanticAction (FareRuleStruct&);
      FareRuleStruct& _fareRule;
    };

    struct storeFareId : public ParserSemanticAction {
      storeFareId (FareRuleStruct&);
      void operator() (unsigned int,
                       boost::spirit::qi::unused_type,
                       boost::spirit::qi::unused_type) const;
    };

    struct storeOrigin : public ParserSemanticAction {
      storeOrigin (FareRuleStruct&);
      void operator() (std::vector<char>,
                       boost::spirit::qi::unused_type,
                       boost::spirit::qi::unused_type) const;
    };

    struct storeDestination : public ParserSemanticAction {
      storeDestination (FareRuleStruct&);
      void operator() (std::vector<char>,
                       boost::spirit::qi::unused_type,
                       boost::spirit::qi::unused_type) const;
    };

    struct storeTripType : public ParserSemanticAction {
      storeTripType (FareRuleStruct&);
      void operator() (std::vector<char>,
                       boost::spirit::qi::unused_type,
                       boost::spirit::qi::unused_type) const;
    };


    struct storeDateRangeStart : public ParserSemanticAction {
      storeDateRangeStart (FareRuleStruct&);
      void operator() (boost::spirit::qi::unused_type,
                        boost::spirit::qi::unused_type,
                        boost::spirit::qi::unused_type) const;
    };

    struct storeDateRangeEnd : public ParserSemanticAction {
      storeDateRangeEnd (FareRuleStruct&);
      void operator() (boost::spirit::qi::unused_type,
                        boost::spirit::qi::unused_type,
                        boost::spirit::qi::unused_type) const;
    };

    struct storeStartRangeTime : public ParserSemanticAction {
      storeStartRangeTime (FareRuleStruct&);
      void operator() (boost::spirit::qi::unused_type,
                        boost::spirit::qi::unused_type,
                        boost::spirit::qi::unused_type) const;
    };

    struct storeEndRangeTime : public ParserSemanticAction {
      storeEndRangeTime (FareRuleStruct&);
      void operator() (boost::spirit::qi::unused_type,
                        boost::spirit::qi::unused_type,
                        boost::spirit::qi::unused_type) const;
    };

    struct storePOS : public ParserSemanticAction {
      storePOS (FareRuleStruct&);
      void operator() (std::vector<char>,
                        boost::spirit::qi::unused_type,
                        boost::spirit::qi::unused_type) const;
    };

    struct storeCabinCode : public ParserSemanticAction {
      storeCabinCode  (FareRuleStruct&);
      void operator() (char,
                        boost::spirit::qi::unused_type,
                        boost::spirit::qi::unused_type) const;
    };

    struct storeChannel : public ParserSemanticAction {
      storeChannel (FareRuleStruct&);
      void operator() (std::vector<char>,
                        boost::spirit::qi::unused_type,
                        boost::spirit::qi::unused_type) const;
    };

    struct storeAdvancePurchase : public ParserSemanticAction {
      storeAdvancePurchase (FareRuleStruct&);
      void operator() (unsigned int,
                        boost::spirit::qi::unused_type,
                        boost::spirit::qi::unused_type) const;
    };

    struct storeSaturdayStay : public ParserSemanticAction {
      storeSaturdayStay (FareRuleStruct&);
      void operator() (char,
                        boost::spirit::qi::unused_type,
                        boost::spirit::qi::unused_type) const;
    };

    struct storeChangeFees : public ParserSemanticAction {
      storeChangeFees (FareRuleStruct&);
      void operator() (char,
                        boost::spirit::qi::unused_type,
                        boost::spirit::qi::unused_type) const;
    };

    struct storeNonRefundable : public ParserSemanticAction {
      storeNonRefundable (FareRuleStruct&);
      void operator() (char,
                       boost::spirit::qi::unused_type,
                       boost::spirit::qi::unused_type) const;
    };

    struct storeMinimumStay : public ParserSemanticAction {
      storeMinimumStay (FareRuleStruct&);
      void operator() (unsigned int,
                       boost::spirit::qi::unused_type,
                       boost::spirit::qi::unused_type) const;
    };

    struct storeFare : public ParserSemanticAction {
      storeFare (FareRuleStruct&);
      void operator() (double,
                       boost::spirit::qi::unused_type,
                       boost::spirit::qi::unused_type) const;
    };

    struct storeAirlineCode : public ParserSemanticAction {
      storeAirlineCode (FareRuleStruct&);
      void operator() (std::vector<char>,
                       boost::spirit::qi::unused_type,
                       boost::spirit::qi::unused_type) const;
    };

    struct storeClass : public ParserSemanticAction {
      storeClass (FareRuleStruct&);
      void operator() (std::vector<char>,
                       boost::spirit::qi::unused_type,
                       boost::spirit::qi::unused_type) const;
    };

    struct doEndFare : public ParserSemanticAction {
      doEndFare (stdair::BomRoot&, FareRuleStruct&);
      void operator() (boost::spirit::qi::unused_type,
                       boost::spirit::qi::unused_type,
                       boost::spirit::qi::unused_type) const;
      stdair::BomRoot& _bomRoot;
    };

  }

  //
  //  Entry class for the file parser
  //

  class FareRuleFileParser : public stdair::CmdAbstract {
  public:
    FareRuleFileParser (stdair::BomRoot& ioBomRoot,
                        const stdair::Filename_T& iFilename);

    void generateFareRules ();

  private:
    void init();

  private:
    // Attributes
    stdair::Filename_T _filename;

    stdair::BomRoot& _bomRoot;

    FareRuleStruct _fareRule;
  };

}
#endif // __SIMFQT_CMD_FAREPARSERHELPER_HPP
```

25.35 simfqt/command/FareQuoter.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <stdair/basic/BasConst_BomDisplay.hpp>
#include <stdair/basic/BasConst_Request.hpp>
#include <stdair/bom/BomKeyManager.hpp>
#include <stdair/bom/ParsedKey.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/BomRoot.hpp>
#include <stdair/bom/InventoryKey.hpp>
#include <stdair/bom/FlightDateKey.hpp>
#include <stdair/bom/SegmentDateKey.hpp>
#include <stdair/bom/AirlineClassList.hpp>
#include <stdair/bom/AirportPair.hpp>
#include <stdair/bom/PosChannel.hpp>
#include <stdair/bom/DatePeriod.hpp>
#include <stdair/bom/TimePeriod.hpp>
#include <stdair/bom/FareFeatures.hpp>
#include <stdair/bom/BookingRequestStruct.hpp>
#include <stdair/bom/TravelSolutionStruct.hpp>
#include <stdair/service/Logger.hpp>
#include <stdair/bom/key_types.hpp>
#include <simfqt/SIMFQT_Types.hpp>
#include <simfqt/command/FareQuoter.hpp>
```

Namespaces

- namespace SIMFQT

25.36 FareQuoter.cpp

```
// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <cassert>
#include <sstream>
// StdAir
#include <stdair/basic/BasConst_BomDisplay.hpp>
#include <stdair/basic/BasConst_Request.hpp>
#include <stdair/bom/BomKeyManager.hpp>
#include <stdair/bom/ParsedKey.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/BomRoot.hpp>
#include <stdair/bom/InventoryKey.hpp>
#include <stdair/bom/FlightDateKey.hpp>
#include <stdair/bom/SegmentDateKey.hpp>
#include <stdair/bom/AirlineClassList.hpp>
#include <stdair/bom/AirportPair.hpp>
#include <stdair/bom/PosChannel.hpp>
#include <stdair/bom/DatePeriod.hpp>
#include <stdair/bom/TimePeriod.hpp>
#include <stdair/bom/FareFeatures.hpp>
#include <stdair/bom/BookingRequestStruct.hpp>
#include <stdair/bom/TravelSolutionStruct.hpp>
#include <stdair/service/Logger.hpp>
#include <stdair/bom/key_types.hpp>
// SimFQT
#include <simfqt/SIMFQT_Types.hpp>
#include <simfqt/command/FareQuoter.hpp>

namespace SIMFQT {

  bool FareQuoter::_atLeastOneAvailableDateRule = false;
  bool FareQuoter::_atLeastOneAvailablePosChannel = false;
  bool FareQuoter::_atLeastOneAvailableTimeRule = false;
  bool FareQuoter::_atLeastOneAvailableFeaturesRule = false;
  bool FareQuoter::_atLeastOneAvailableAirlineClassRule= false;

  // //////////////////////////////////////////////////////////////////////
  FareQuoter::FareQuoter() {
    assert (false);
  }

  // //////////////////////////////////////////////////////////////////////
  FareQuoter::FareQuoter(const FareQuoter&) {
    assert (false);
  }

  // //////////////////////////////////////////////////////////////////////
  FareQuoter::~FareQuoter() {
  }

  // //////////////////////////////////////////////////////////////////////
  void FareQuoter::reset() {
    _atLeastOneAvailableDateRule = false;
    _atLeastOneAvailablePosChannel = false;
    _atLeastOneAvailableTimeRule = false;
    _atLeastOneAvailableFeaturesRule = false;
    _atLeastOneAvailableAirlineClassRule = false;
  }


  // //////////////////////////////////////////////////////////////////////
  void FareQuoter::
  priceQuote (const stdair::BookingRequestStruct& iBookingRequest,
              stdair::TravelSolutionList_T& ioTravelSolutionList,
              const stdair::BomRoot& iBomRoot) {

    // Do an independent price quote for each travel solution related to the
    // booking request.
    for (stdair::TravelSolutionList_T::iterator itTravelSolution =
           ioTravelSolutionList.begin();
         itTravelSolution != ioTravelSolutionList.end(); ++itTravelSolution) {
      reset();
      // Select a travel solution.
      stdair::TravelSolutionStruct& lTravelSolutionStruct = *itTravelSolution;
      // Price quote the travel solution into question.
      priceQuote (iBookingRequest, lTravelSolutionStruct, iBomRoot);
    }
  }

  // ////////////////////////////////////////////////////////////////////
  void FareQuoter::
  priceQuote (const stdair::BookingRequestStruct& iBookingRequest,
              stdair::TravelSolutionStruct& ioTravelSolution,
              const stdair::BomRoot& iBomRoot) {

    // Get the origin of the first segment in order to get the origin of
    // the solution.
    const stdair::ParsedKey& lFirstSegmentKey =
      getFirstSPParsedKey(ioTravelSolution);
    const stdair::AirportCode_T& lOrigin = lFirstSegmentKey._boardingPoint;

    // Get the destination of the last segment in order to get the
    // destination of the solution.
    const stdair::ParsedKey& lLastSegmentKey =
      getLastSPParsedKey(ioTravelSolution);
    const stdair::AirportCode_T& lDestination = lLastSegmentKey._offPoint;

    // Construct the Airport pair stream of the segment path.
    const stdair::AirportPairKey lAirportPairKey (lOrigin, lDestination);

    // Search for the fare rules having the same origin and destination airports
    // as the travel solution
    const stdair::AirportPair* lAirportPair_ptr = stdair::BomManager::
      getObjectPtr<stdair::AirportPair> (iBomRoot, lAirportPairKey.toString());

    // If no fare rule has the same origin and destination airports, the pricing
    // is not possible, throw an exception.
    if (lAirportPair_ptr == NULL) {
      STDAIR_LOG_ERROR ("No available fare rule for the "
                        << "Origin-Destination pair: "
                        << lAirportPairKey.toString());
      throw AirportPairNotFoundException ("No available fare rule for "
                                          "the Origin-Destination pair: "
                                          + lAirportPairKey.toString());
    }
    // Sanity check.
    assert(lAirportPair_ptr != NULL);

    // Fare rule(s) with the same origin and destination airports exist(s), now
    // the date range need to be checked.
    const stdair::AirportPair& lAirportPair = *lAirportPair_ptr;
    priceQuote(iBookingRequest, ioTravelSolution, lAirportPair);

    if (_atLeastOneAvailableAirlineClassRule == false) {
      displayMissingFareRuleMessage(iBookingRequest, ioTravelSolution);
    }
  }

  // ////////////////////////////////////////////////////////////////////
```

```
void FareQuoter::
priceQuote (const stdair::BookingRequestStruct& iBookingRequest,
            stdair::TravelSolutionStruct& ioTravelSolution,
            const stdair::AirportPair& iAirportPair) {

  // Get the first segment path parsed key.
  const stdair::ParsedKey lFirstSPParsedKey =
    getFirstSPParsedKey(ioTravelSolution);

  // Get the date of the first segment date key.
  const stdair::FlightDateKey& lFlightDateKey =
    lFirstSPParsedKey.getFlightDateKey();
  const stdair::Date_T& lSPDate = lFlightDateKey.getDepartureDate();

  // Get the list of the fare date ranges.
  const stdair::DatePeriodList_T& lFareDatePeriodList =
    stdair::BomManager::getList<stdair::DatePeriod> (iAirportPair);

  // Browse the list of the fare rules date range.
  for (stdair::DatePeriodList_T::const_iterator itDateRange =
         lFareDatePeriodList.begin();
       itDateRange != lFareDatePeriodList.end(); ++itDateRange) {

    const stdair::DatePeriod* lCurrentFareDatePeriod_ptr = *itDateRange ;
    assert (lCurrentFareDatePeriod_ptr != NULL);

    // Select the fare rules having a corresponding date range.
    const bool isDepartureDateValid =
      lCurrentFareDatePeriod_ptr->isDepartureDateValid (lSPDate);

    // If a fare rule has a corresponding date range, its channel and position
    // need to be checked.
    if (isDepartureDateValid == true) {
      _atLeastOneAvailableDateRule = true;
      const stdair::DatePeriod& lCurrentFareDatePeriod =
        *lCurrentFareDatePeriod_ptr;
      priceQuote (iBookingRequest, ioTravelSolution,
                  lCurrentFareDatePeriod, iAirportPair);
    }
  }

}

// //////////////////////////////////////////////////////////////////////
void FareQuoter::
priceQuote (const stdair::BookingRequestStruct& iBookingRequest,
            stdair::TravelSolutionStruct& ioTravelSolution,
            const stdair::DatePeriod& iFareDatePeriod,
            const stdair::AirportPair& iAirportPair) {

  // Get the point-of-sale of the booking request.
  const stdair::CityCode_T& lPointOfSale = iBookingRequest.getPOS();

  // Get the booking request channel.
  const stdair::ChannelLabel_T& lChannel =
    iBookingRequest.getBookingChannel();

  // Construct the corresponding POS-channel primary key.
  const stdair::PosChannelKey lFarePosChannelKey (lPointOfSale, lChannel);

  // Search for the fare rules having the same point-of-sale and channel as
  // the travel solution.
  const stdair::PosChannelList_T lFarePosChannelList =
    stdair::BomManager::getList<stdair::PosChannel> (iFareDatePeriod);

  // Browse the list of the fare rules pos channel.
  for (stdair::PosChannelList_T::const_iterator itPosChannel =
         lFarePosChannelList.begin();
       itPosChannel != lFarePosChannelList.end();
       ++itPosChannel) {
    const stdair::PosChannel* lCurrentFarePosChannel_ptr = *itPosChannel;
    assert (lCurrentFarePosChannel_ptr != NULL);

    // Get the point-of-sale and channel of the current fare rule.
    const stdair::CityCode_T& lCurrentPointOfSale =
      lCurrentFarePosChannel_ptr->getPos();
    const stdair::ChannelLabel_T& lCurrentChannel =
      lCurrentFarePosChannel_ptr->getChannel();

    // Select the fare rules having a corresponding pos channel.
    if (lCurrentPointOfSale == lPointOfSale || lCurrentPointOfSale == stdair::DEFAULT_POS) {
      if (lCurrentChannel == lChannel || lCurrentChannel == stdair::DEFAULT_CHANNEL) {
        _atLeastOneAvailablePosChannel = true;
        // Fare rule(s) with the same point-of-sale and channel exist(s), now
        // the time range need to be checked.
        const stdair::PosChannel& lFarePosChannel= *lCurrentFarePosChannel_ptr;

        STDAIR_LOG_DEBUG (lCurrentPointOfSale + " " + lCurrentChannel);
        priceQuote (iBookingRequest, ioTravelSolution, lFarePosChannel);
      }
    }
  }

}

// //////////////////////////////////////////////////////////////////////
void FareQuoter::
priceQuote (const stdair::BookingRequestStruct& iBookingRequest,
            stdair::TravelSolutionStruct& ioTravelSolution,
            const stdair::PosChannel& iFarePosChannel) {

  // Get the first segment path parsed key.
  const stdair::ParsedKey lFirstSPParsedKey =
    getFirstSPParsedKey(ioTravelSolution);

  // Get the segment boarding time of the segment path.
  const stdair::Duration_T& lSPTime = lFirstSPParsedKey.getBoardingTime();

  // Get the list of the fare rules time period.
  const stdair::TimePeriodList_T& lFareTimePeriodList =
    stdair::BomManager::getList<stdair::TimePeriod> (iFarePosChannel);

  // Browse the list of the fare rules time range.
  for (stdair::TimePeriodList_T::const_iterator itTimeRange =
         lFareTimePeriodList.begin();
       itTimeRange != lFareTimePeriodList.end();
       ++itTimeRange) {
    const stdair::TimePeriod* lCurrentFareTimePeriod_ptr = *itTimeRange ;
    assert (lCurrentFareTimePeriod_ptr != NULL);

    // Select the fare rules having a corresponding time range.
    const bool isDepartureTimeValid =
      lCurrentFareTimePeriod_ptr->isDepartureTimeValid (lSPTime);

    // If a fare rule has a corresponding time range, its advanced purchase,
    // trip type and minimum stay duration need to be checked.
    if (isDepartureTimeValid) {
      _atLeastOneAvailableTimeRule = true;
      const stdair::TimePeriod& lCurrentFareTimePeriod =
        *lCurrentFareTimePeriod_ptr;
      priceQuote (iBookingRequest, ioTravelSolution,
                  lCurrentFareTimePeriod, iFarePosChannel);
    }
  }

}

// //////////////////////////////////////////////////////////////////////
void FareQuoter::
priceQuote (const stdair::BookingRequestStruct& iBookingRequest,
            stdair::TravelSolutionStruct& ioTravelSolution,
            const stdair::TimePeriod& iFareTimePeriod,
            const stdair::PosChannel& iFarePosChannel) {

  // Get the stay duration of the booking request.
  const stdair::DayDuration_T& lStayDuration=
    iBookingRequest.getStayDuration();

  // Get the booking request trip type.
  const stdair::TripType_T& lTripType =
    iBookingRequest.getTripType();

  // Get the booking request date time.
  const stdair::DateTime_T& lRequestDateTime =
    iBookingRequest.getRequestDateTime();

  // Get the referenced departure date of the segment path.
  const stdair::ParsedKey lFirstSPParsedKey =
    getFirstSPParsedKey(ioTravelSolution);
  const stdair::Date_T& lSPDate =
    lFirstSPParsedKey.getFlightDateKey().getDepartureDate();

  // Get the segment boarding time of the segment path.
  const stdair::Duration_T& lSPTime = lFirstSPParsedKey.getBoardingTime();

  // Construct the date-time type correponding to the flight date
  const stdair::DateTime_T lSPDateTime (lSPDate, lSPTime);

  bool isTripTypeValid = false;
  bool isStayDurationValid = false;
  bool isAdvancePurchaseValid = false;

  // Get the list of the fare features (if such list exists: the POS
  // and channel couple can be only present in a yield rule).
  const bool hasFareFeaturesList =
    stdair::BomManager::hasList<stdair::FareFeatures> (iFareTimePeriod);
  if (hasFareFeaturesList == false) {
    return;
  }
  assert (hasFareFeaturesList == true);
  const stdair::FareFeaturesList_T& lFareFeaturesList =
    stdair::BomManager::getList<stdair::FareFeatures> (iFareTimePeriod);

  // Browse the list of the fare rules features.
  for (stdair::FareFeaturesList_T::const_iterator itFareFeatures =
         lFareFeaturesList.begin();
       itFareFeatures != lFareFeaturesList.end();
       ++itFareFeatures) {
    const stdair::FareFeatures* lCurrentFareFeatures_ptr =
      *itFareFeatures;
    assert (lCurrentFareFeatures_ptr != NULL);

    // Does the current fare features correspond to a correct trip
    // type?
    isTripTypeValid =
      lCurrentFareFeatures_ptr->isTripTypeValid (lTripType);
    // Does the current fare features correspond to a correct stay
    // duration?
    isStayDurationValid =
      lCurrentFareFeatures_ptr->isStayDurationValid (lStayDuration);
    // Does the current fare features correspond to a correct advanced
    // purchase?
    isAdvancePurchaseValid = lCurrentFareFeatures_ptr->
      isAdvancePurchaseValid (lRequestDateTime,
                              lSPDateTime);

    // Search for the fare rules having corresponding features.
    if (isStayDurationValid && isAdvancePurchaseValid && isTripTypeValid){
      _atLeastOneAvailableFeaturesRule = true;
      // Create a fare structure for the travel solution.
      stdair::FareOptionStruct lFareOption;
      const stdair::ChangeFees_T& lChangeFees =
        lCurrentFareFeatures_ptr->getChangeFees();
      // Set the fare change fees.
      lFareOption.setChangeFees (lChangeFees);
      const stdair::NonRefundable_T& lNonRefundable =
        lCurrentFareFeatures_ptr->getRefundableOption();
      // Set the fare refundable option.
      lFareOption.setNonRefundable (lNonRefundable);
      const stdair::SaturdayStay_T& lSaturdayStay =
        lCurrentFareFeatures_ptr->getSaturdayStay();
      // Set the fare saturday night stay option.
      lFareOption.setSaturdayStay (lSaturdayStay);
      const stdair::FareFeatures& lCurrentFareFeatures =
        *lCurrentFareFeatures_ptr;
      priceQuote (iBookingRequest, ioTravelSolution,
                  lCurrentFareFeatures, iFarePosChannel,
                  lFareOption);
    }
  }

}


// ////////////////////////////////////////////////////////////////////////
void FareQuoter::
priceQuote (const stdair::BookingRequestStruct& iBookingRequest,
            stdair::TravelSolutionStruct& ioTravelSolution,
            const stdair::FareFeatures& iFareFeatures,
            const stdair::PosChannel& iFarePosChannel,
            stdair::FareOptionStruct& iFareOption) {

  // Get the segment-path of the travel solution.
  const stdair::SegmentPath_T& lSegmentPath =
    ioTravelSolution.getSegmentPath();

  // Get the list of the fare rules.
  const stdair::AirlineClassListList_T& lAirlineClassListList =
    stdair::BomManager::getList<stdair::AirlineClassList> (iFareFeatures);

  bool lCorrectAirlineRule = false;
  bool lAtLeastOneDifferentAirline = false;

  // Browse the list of airline code list and search for the fare rules
  // having a corresponding airline list.
  for (stdair::AirlineClassListList_T::const_iterator itAirlineClassList =
         lAirlineClassListList.begin();
       itAirlineClassList != lAirlineClassListList.end();
       ++itAirlineClassList) {
    const stdair::AirlineClassList* lCurrentAirlineClassList_ptr =
      *itAirlineClassList;
    assert (lCurrentAirlineClassList_ptr != NULL);

    lCorrectAirlineRule = true;
    lAtLeastOneDifferentAirline = false;

```

```
      const stdair::ClassList_StringList_T lClassList_StringList =
        lCurrentAirlineClassList_ptr->getAirlineCodeList();

      // Compare the segment path airline list with the fare rule airline list.
      if (lClassList_StringList.size() == lSegmentPath.size()) {
        // If the two sizes are equal, we need to compare the airline codes.
        stdair::SegmentPath_T::const_iterator itSegmentPath =
          lSegmentPath.begin();

        stdair::ClassList_StringList_T::const_iterator itClassList_String =
          lClassList_StringList.begin();
        // Browse the segment path airline code list (while the segment path
        // airline list is equal to the fare rule airline list).
        while (itSegmentPath != lSegmentPath.end()
               && lAtLeastOneDifferentAirline == false) {

          // Get the segment airline code.
          const std::string lSegmentDateKey = *itSegmentPath;
          const stdair::ParsedKey& lParsedKey =
            stdair::BomKeyManager::extractKeys (lSegmentDateKey);
          const stdair::InventoryKey& lInventoryKey =
            lParsedKey.getInventoryKey();
          const stdair::AirlineCode_T& lSegmentAirlineCode =
            lInventoryKey.getAirlineCode();

          // Get the fare rule airline code.
          const stdair::AirlineCode_T& lFareRuleAirlineCode =
            *itClassList_String;

          if (lSegmentAirlineCode != lFareRuleAirlineCode) {
            lAtLeastOneDifferentAirline = true;
          }
          itSegmentPath++;
          itClassList_String++;
        }

      } else {
        // If the two sizes are different, the fare rule does not match the
        // travel solution into question.
        lCorrectAirlineRule = false;
      }

      // If one segment airline code and one fare rule airline code are
      // different then the fare rule does not match the travel solution.
      if (lAtLeastOneDifferentAirline == true) {
        lCorrectAirlineRule = false;
      }

      // If the current fare rule is a match, add the fare option structure
      // to the travel solution into question.
      if (lCorrectAirlineRule == true) {
        _atLeastOneAvailableAirlineClassRule = true;
        // Get the booking request trip type.
        const stdair::TripType_T& lTripType =
          iBookingRequest.getTripType();

        // Get the travel fare.
        stdair::Fare_T lFare =
          lCurrentAirlineClassList_ptr->getFare();
        // If the trip into question is the inbound or outbound part of a round t
rip,
        // the applicable fare is a half RT fare.
        if (lTripType == "RI" || lTripType == "RO") {
          lFare /= 2;
        }
        // Set the travel fare option.
        iFareOption.setFare (lFare);
        // Copy the class path list into the fare option.
        const stdair::ClassList_StringList_T& lClassCodeList =
          lCurrentAirlineClassList_ptr->getClassCodeList();
        for (stdair::ClassList_StringList_T::const_iterator itClassCodeList =
               lClassCodeList.begin();
             itClassCodeList != lClassCodeList.end(); ++itClassCodeList ) {
          const stdair::ClassList_String_T& lClassCodeList = *itClassCodeList;
          iFareOption.addClassList (lClassCodeList);
        }

        // Add the fare option to the travel solution into question.
        ioTravelSolution.addFareOption (iFareOption);

        // DEBUG
        STDAIR_LOG_DEBUG (ioTravelSolution.describeSegmentPath()
                          << ". A corresponding fare option for the '"
                          << lCurrentAirlineClassList_ptr->describeKey()
                          << "' class is: " << iFareOption);

        iFareOption.emptyClassList();
      }
    }

  }

  // ////////////////////////////////////////////////////////////////////////
  stdair::ParsedKey FareQuoter::
  getFirstSPParsedKey (stdair::TravelSolutionStruct& ioTravelSolution) {

    // Get the segment-path of the travel solution.
    const stdair::SegmentPath_T& lSegmentPath =
      ioTravelSolution.getSegmentPath();

    // Get the number of segments of the travel solution.
    const stdair::NbOfSegments_T& lNbSegments = lSegmentPath.size();

    // Sanity check: there is at least one segment in the travel solution.
    assert (lNbSegments >= 1);

    // Get the first segment of the travel solution.
    const std::string& lFirstSegmentDateKey = lSegmentPath.front();

    // Get the parsed key of the first segment of the travel solution.
    const stdair::ParsedKey& lFirstSegmentParsedKey =
      stdair::BomKeyManager::extractKeys (lFirstSegmentDateKey);

    return lFirstSegmentParsedKey;

  }

  // ////////////////////////////////////////////////////////////////////////
  stdair::ParsedKey FareQuoter::
  getLastSPParsedKey (stdair::TravelSolutionStruct& ioTravelSolution) {

    // Get the segment-path of the travel solution.
    const stdair::SegmentPath_T& lSegmentPath =
      ioTravelSolution.getSegmentPath();

    // Get the number of segments of the travel solution.
    const stdair::NbOfSegments_T& lNbSegments = lSegmentPath.size();

    // Sanity check: there is at least one segment in the travel solution.
    assert (lNbSegments >= 1);

    // Get the last segment of the travel solution.
    const std::string& lLastSegmentDateKey = lSegmentPath.back();

    // Get the parsed key of the last segment of the travel solution.
    const stdair::ParsedKey& lLastSegmentParsedKey =
      stdair::BomKeyManager::extractKeys (lLastSegmentDateKey);

    return lLastSegmentParsedKey;

  }

  // //////////////////////////////////////////////////////////////////////
  void FareQuoter::
  displayMissingFareRuleMessage (const stdair::BookingRequestStruct& iBookingRequ
est,
                                 stdair::TravelSolutionStruct& ioTravelSolution)
{

    // Get the origin of the first segment in order to get the origin of
    // the solution.
    const stdair::ParsedKey lFirstSPParsedKey =
      getFirstSPParsedKey(ioTravelSolution);
    const stdair::AirportCode_T& lOrigin = lFirstSPParsedKey._boardingPoint;

    // Get the destination of the last segment in order to get the
    // destination of the solution.
    const stdair::ParsedKey& lLastSegmentKey =
      getLastSPParsedKey(ioTravelSolution);
    const stdair::AirportCode_T& lDestination = lLastSegmentKey._offPoint;

    // Construct the Airport pair stream of the segment path.
    const stdair::AirportPairKey lAirportPairKey (lOrigin, lDestination);

    // Get the date of the first segment date key.
    const stdair::FlightDateKey& lFlightDateKey =
      lFirstSPParsedKey.getFlightDateKey();

    // Get the point-of-sale of the booking request.
    const stdair::CityCode_T& lPointOfSale = iBookingRequest.getPOS();
    // Get the booking request channel.
    const stdair::ChannelLabel_T& lChannel =
      iBookingRequest.getBookingChannel();
    // Construct the corresponding POS-channel primary key.
    const stdair::PosChannelKey lFarePosChannelKey (lPointOfSale, lChannel);

    // Get the booking request date time.
    const stdair::DateTime_T& lRequestDateTime =
      iBookingRequest.getRequestDateTime();

    // If no fare rule has a corresponding date range, the pricing is not
    // possible, throw an exception.
    if (_atLeastOneAvailableDateRule == false) {
      const stdair::SegmentDateKey lSegmentDateKey =
        lFirstSPParsedKey.getSegmentKey();
      STDAIR_LOG_ERROR ("No available fare rule corresponding to the "
                        "flight date " << lFlightDateKey.toString()
                        << " and the Origin-Destination pair: "
                        << lSegmentDateKey.toString());
      throw FlightDateNotFoundException ("No available fare rule for the "
                                         "flight date "
                                         + lFlightDateKey.toString()
                                         + " and the Origin-Destination pair: "
                                         + lSegmentDateKey.toString());
    }
    // If no fare rule has a corresponding pos channel, the pricing is not possib
le,
    // throw an exception.
    else if (_atLeastOneAvailablePosChannel == false) {
      STDAIR_LOG_ERROR ("No available fare rule corresponding to the "
                        "point of sale " << lPointOfSale
                        << ", to the channel " << lChannel
                        << ", to the flight date "
                        << lFlightDateKey.toString()
                        << " and to the Origin-Destination pair: "
                        << lAirportPairKey.toString());
      throw PosOrChannelNotFoundException ("No available fare rule for the "
                                           "point of sale " + lPointOfSale
                                           + ", the channel " + lChannel
                                           + ", the flight date "
                                           + lFlightDateKey.toString()
                                           + " and the Origin-Destination pair: "
                                           + lAirportPairKey.toString());
    }
    // If no fare rule has a corresponding time range, the pricing is not possible,
    // throw an exception.
    else if (_atLeastOneAvailableTimeRule == false) {
      STDAIR_LOG_ERROR ("No available fare rule corresponding to '"
                        << lFirstSPParsedKey.toString() << "' (parsed key) and to '"
                        << lFarePosChannelKey.toString() << "' (POS and channel)");
      throw FlightTimeNotFoundException ("No available fare rule corresponding "
                                         "to '" + lFirstSPParsedKey.toString()
                                         + "' (parsed key) and to '"
                                         + lFarePosChannelKey.toString()
                                         + "' (POS and channel)");
    }
    // If no fare rule matches the advance purchase, trip type and stay
    // duration criterion, the pricing is not possible, throw an exception.
    else if (_atLeastOneAvailableFeaturesRule == false) {
      // Get the stay duration of the booking request.
      const stdair::DayDuration_T& lStayDuration=
        iBookingRequest.getStayDuration();
      std::ostringstream lStayDurationStream;
      lStayDurationStream << lStayDuration;
      const std::string lStayDurationString (lStayDurationStream.str());

      // Get the booking request trip type.
      const stdair::TripType_T& lTripType =
        iBookingRequest.getTripType();

      STDAIR_LOG_ERROR ("No available fare rule corresponding to a "
                        "trip type " << lTripType
                        << ", to a stay duration of " <<  lStayDurationString
                        << ", to a request date time of " << lRequestDateTime
                        << ", to '" << lFirstSPParsedKey.toString()
                        << "' (parsed key) and to '"
                        << lFarePosChannelKey << "' (POS and channel)");
      throw FeaturesNotFoundException ("No available fare rule corresponding to a "
                                       "trip type " + lTripType
                                       + ", to a stay duration of "
                                       + lStayDurationString
                                       + ", to a request date time of "
                                       + boost::posix_time::to_simple_string(lRequestDateTime)
                                       + ", to '" + lFirstSPParsedKey.toString()
                                       + "' (parsed key) and to '"
                                       + lFarePosChannelKey.toString()
                                       + "' (POS and channel)");
    }
    assert (_atLeastOneAvailableAirlineClassRule == false);
    // If no fare rule matches the airline class path, the pricing is not
    // possible, throw an exception.
    STDAIR_LOG_ERROR ("No available fare rule corresponding to '"
                       << lFirstSPParsedKey .toString() << "' (parsed key), to '"
                       << iBookingRequest.describe()
                       << "' (booking request) and to '"
                       << lFarePosChannelKey.toString() << "' (POS and channel)");

    throw AirlineNotFoundException ("No available fare rule corresponding to '"
                                    + lFirstSPParsedKey .toString()
                                    + "' (parsed key), to '"
                                    + iBookingRequest.describe()
                                    + "' (booking request) and to '"
                                    + lFarePosChannelKey.toString()
                                    + "' (POS and channel)");
  }
}

```

25.37 simfqt/command/FareQuoter.hpp File Reference

```
#include <stdair/stdair_basic_types.hpp>
#include <stdair/bom/TravelSolutionTypes.hpp>
```

Classes

- class SIMFQT::FareQuoter

  Command wrapping the pricing request process.

Namespaces

- namespace stdair

  Forward declarations.

- namespace SIMFQT

25.38 FareQuoter.hpp

```
#ifndef __SIMFQT_CMD_FAREQUOTER_HPP
#define __SIMFQT_CMD_FAREQUOTER_HPP

// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// StdAir
#include <stdair/stdair_basic_types.hpp>
#include <stdair/bom/TravelSolutionTypes.hpp>

namespace stdair {
  class BomRoot;
  struct BookingRequestStruct;
  struct TravelSolutionStruct;
  struct ParsedKey;
  class AirportPair;
  class PosChannel;
  class DatePeriod;
  class TimePeriod;
  class FareFeatures;
}

namespace SIMFQT {

  class FareQuoter {
    friend class SIMFQT_Service;

  private:
    // ////////////////// Business support methods /////////////
    static void priceQuote (const stdair::BookingRequestStruct&,
                            stdair::TravelSolutionList_T&,
                            const stdair::BomRoot&);

    static void priceQuote (const stdair::BookingRequestStruct&,
                            stdair::TravelSolutionStruct&,
                            const stdair::BomRoot&);

    static void priceQuote (const stdair::BookingRequestStruct&,
                            stdair::TravelSolutionStruct&,
                            const stdair::AirportPair&);

    static void priceQuote (const stdair::BookingRequestStruct&,
                            stdair::TravelSolutionStruct&,
                            const stdair::DatePeriod&,
                            const stdair::AirportPair&);

    static void priceQuote (const stdair::BookingRequestStruct&,
                            stdair::TravelSolutionStruct&,
                            const stdair::PosChannel&);

    static void priceQuote (const stdair::BookingRequestStruct&,
                            stdair::TravelSolutionStruct&,
                            const stdair::TimePeriod&,
                            const stdair::PosChannel&);

    static void priceQuote (const stdair::BookingRequestStruct&,
                            stdair::TravelSolutionStruct&,
                            const stdair::FareFeatures&,
                            const stdair::PosChannel&,
                            stdair::FareOptionStruct&);

    static void reset ();

    static void displayMissingFareRuleMessage (const stdair::BookingRequestStruct
      &,
                                                stdair::TravelSolutionStruct&);

    static stdair::ParsedKey getFirstSPParsedKey (stdair::TravelSolutionStruct&);

    static stdair::ParsedKey getLastSPParsedKey (stdair::TravelSolutionStruct&);



  private:
    // ////////////////// Construction and destruction ///////////////
    FareQuoter();

    FareQuoter(const FareQuoter&);

    ~FareQuoter();

  private:

    static bool _atLeastOneAvailableDateRule;

    static bool _atLeastOneAvailablePosChannel;

    static bool _atLeastOneAvailableTimeRule;

    static bool _atLeastOneAvailableFeaturesRule;

    static bool _atLeastOneAvailableAirlineClassRule;

  };

}
#endif // __SIMFQT_CMD_FAREQUOTER_HPP
```

25.39 simfqt/command/FareRuleGenerator.cpp File Reference

```
#include <cassert>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/BomRoot.hpp>
#include <stdair/factory/FacBomManager.hpp>
#include <stdair/service/Logger.hpp>
#include <stdair/bom/AirportPair.hpp>
#include <stdair/bom/PosChannel.hpp>
#include <stdair/bom/DatePeriod.hpp>
#include <stdair/bom/TimePeriod.hpp>
#include <stdair/bom/FareFeatures.hpp>
#include <stdair/bom/AirlineClassList.hpp>
#include <simfqt/bom/FareRuleStruct.hpp>
#include <simfqt/command/FareRuleGenerator.hpp>
```

Namespaces

- namespace SIMFQT

25.40 FareRuleGenerator.cpp

```
// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <cassert>
// StdAir
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/BomRoot.hpp>
#include <stdair/factory/FacBomManager.hpp>
#include <stdair/service/Logger.hpp>
#include <stdair/bom/AirportPair.hpp>
#include <stdair/bom/PosChannel.hpp>
#include <stdair/bom/DatePeriod.hpp>
#include <stdair/bom/TimePeriod.hpp>
#include <stdair/bom/FareFeatures.hpp>
#include <stdair/bom/AirlineClassList.hpp>
// SimFQT
#include <simfqt/bom/FareRuleStruct.hpp>
#include <simfqt/command/FareRuleGenerator.hpp>

namespace SIMFQT {

  // //////////////////////////////////////////////////////////////////////
  void FareRuleGenerator::
  createAirportPair (stdair::BomRoot& ioBomRoot,
                     const FareRuleStruct& iFareRuleStruct) {

    // Create the airport-pair primary key.
    const stdair::AirportCode_T& lBoardPoint = iFareRuleStruct.getOrigin ();
    const stdair::AirportCode_T& lOffPoint =
      iFareRuleStruct.getDestination ();
    const stdair::AirportPairKey lAirportPairKey (lBoardPoint, lOffPoint);

    // Check that the airport-pair object is not already existing. If an
    // airport-pair object with the same key has not already been created,
    // create it and link it to the ioBomRoot object.
    stdair::AirportPair* lAirportPair_ptr = stdair::BomManager::
      getObjectPtr<stdair::AirportPair> (ioBomRoot, lAirportPairKey.toString());
    if (lAirportPair_ptr == NULL) {
      lAirportPair_ptr =
        &stdair::FacBom<stdair::AirportPair>::instance().
        create (lAirportPairKey);
      stdair::FacBomManager::addToListAndMap (ioBomRoot, *lAirportPair_ptr);
      stdair::FacBomManager::linkWithParent (ioBomRoot, *lAirportPair_ptr);
    }
    // Sanity check.
    assert (lAirportPair_ptr != NULL);

    stdair::AirportPair& lAirportPair = *lAirportPair_ptr;
    // Generate the date-period object corresponding to the given
    // fareRule.
    createDateRange (lAirportPair, iFareRuleStruct);

  }

  // //////////////////////////////////////////////////////////////////////
  void FareRuleGenerator::
  createDateRange (stdair::AirportPair& iAirportPair,
                   const FareRuleStruct& iFareRuleStruct) {

    // Create the fare date-period primary key.
    const stdair::Date_T& lDateRangeStart =
      iFareRuleStruct.getDateRangeStart ();
    const stdair::Date_T& lDateRangeEnd =
      iFareRuleStruct.getDateRangeEnd ();
    const stdair::DatePeriod_T lDatePeriod (lDateRangeStart, lDateRangeEnd);
    const stdair::DatePeriodKey lFareDatePeriodKey (lDatePeriod);

    // Check that the date-period object is not already existing.
    // If a date-period object with the same key has not already been
    // created, create it and link it to the airport-pair object.
    stdair::DatePeriod* lFareDatePeriod_ptr = stdair::BomManager::
      getObjectPtr<stdair::DatePeriod> (iAirportPair,
                                        lFareDatePeriodKey.toString());
    if (lFareDatePeriod_ptr == NULL) {
      lFareDatePeriod_ptr = &stdair::FacBom<stdair::DatePeriod>::instance().
        create (lFareDatePeriodKey);
      stdair::FacBomManager::addToListAndMap (iAirportPair,
                                              *lFareDatePeriod_ptr);
      stdair::FacBomManager::linkWithParent (iAirportPair,
                                             *lFareDatePeriod_ptr);
    }
    // Sanity check.
    assert (lFareDatePeriod_ptr != NULL);

    stdair::DatePeriod& lDateRange = *lFareDatePeriod_ptr;
    // Generate the point_of_sale-channel object corresponding to
    // the given fareRule.
    createPOSChannel (lDateRange, iFareRuleStruct);

  }

  // ////////////////////////////////////////////////////////////////////
  void FareRuleGenerator::
  createPOSChannel (stdair::DatePeriod& iDatePeriod,
                    const FareRuleStruct& iFareRuleStruct) {

    // Create the point-of-sale-channel primary key.
    const stdair::CityCode_T& lPosition = iFareRuleStruct.getPOS ();
    const stdair::ChannelLabel_T& lChannel =
        iFareRuleStruct.getChannel ();
    const stdair::PosChannelKey lFarePosChannelKey (lPosition, lChannel);

    // Check that the point_of_sale-channel object is not already existing.
    // If a point_of_sale-channel object with the same key has not already
    // been created, create it and link it to the date-period object.
    stdair::PosChannel* lFarePosChannel_ptr = stdair::BomManager::
      getObjectPtr<stdair::PosChannel> (iDatePeriod,
                                        lFarePosChannelKey.toString());
    if (lFarePosChannel_ptr == NULL) {
      lFarePosChannel_ptr = &stdair::FacBom<stdair::PosChannel>::instance().
        create (lFarePosChannelKey);
      stdair::FacBomManager::addToListAndMap (iDatePeriod,
                                              *lFarePosChannel_ptr);
      stdair::FacBomManager::linkWithParent (iDatePeriod,
                                             *lFarePosChannel_ptr);
    }
    // Sanity check.
    assert (lFarePosChannel_ptr != NULL);

    stdair::PosChannel& lPosChannel = *lFarePosChannel_ptr;
    // Generate the time-period object corresponding to the given
    // fareRule.
    createTimeRange (lPosChannel, iFareRuleStruct);

  }


  // ////////////////////////////////////////////////////////////////////
  void FareRuleGenerator::
  createTimeRange (stdair::PosChannel& iPosChannel,
                   const FareRuleStruct& iFareRuleStruct) {
```

```

    // Create the fare time-period primary key.
    const stdair::Time_T& lTimeRangeStart =
      iFareRuleStruct.getTimeRangeStart ();
    const stdair::Time_T& lTimeRangeEnd =
      iFareRuleStruct.getTimeRangeEnd ();
    const stdair::TimePeriodKey lFareTimePeriodKey (lTimeRangeStart,
                                                    lTimeRangeEnd);

    // Check that the time-period object is not already existing.
    // If a time-period object with the same key has not already been
    // created, create it and link it to the point_of_sale-channel object.

    stdair::TimePeriod* lFareTimePeriod_ptr = stdair::BomManager::
      getObjectPtr<stdair::TimePeriod> (iPosChannel,
                                        lFareTimePeriodKey.toString());
    if (lFareTimePeriod_ptr == NULL) {
      lFareTimePeriod_ptr = &stdair::FacBom<stdair::TimePeriod>::instance().
        create (lFareTimePeriodKey);
      stdair::FacBomManager::addToListAndMap (iPosChannel,
                                              *lFareTimePeriod_ptr);
      stdair::FacBomManager::linkWithParent (iPosChannel,
                                             *lFareTimePeriod_ptr);
    }
    // Sanity check.
    assert (lFareTimePeriod_ptr != NULL);

    stdair::TimePeriod& lTimeRange = *lFareTimePeriod_ptr;
    // Generate the fare-features object corresponding to the given
    // fareRule.
    createFareFeatures (lTimeRange, iFareRuleStruct);

  }

  // //////////////////////////////////////////////////////////////////////
  void FareRuleGenerator::
  createFareFeatures (stdair::TimePeriod& iTimePeriod,
                      const FareRuleStruct& iFareRuleStruct) {

    // Create the fare-features primary key.
    const stdair::TripType_T& lTripType =
      iFareRuleStruct.getTripType ();
    const stdair::DayDuration_T& lAdvancePurchase =
      iFareRuleStruct.getAdvancePurchase ();
    const stdair::SaturdayStay_T& lSaturdayStay =
      iFareRuleStruct.getSaturdayStay ();
    const stdair::ChangeFees_T& lChangeFees =
      iFareRuleStruct.getChangeFees ();
    const stdair::NonRefundable_T& lNonRefundable =
      iFareRuleStruct.getNonRefundable ();
    const stdair::DayDuration_T& lMinimumStay =
      iFareRuleStruct.getMinimumStay ();
    const stdair::FareFeaturesKey
      lFareFeaturesKey (lTripType, lAdvancePurchase, lSaturdayStay,
                        lChangeFees, lNonRefundable, lMinimumStay);

    // Check that the fare features object is not already existing.
    // If a fare features object with the same key has not already been
    // created, create it and link it to the time-period object.
    stdair::FareFeatures* lFareFeatures_ptr = stdair::BomManager::
      getObjectPtr<stdair::FareFeatures> (iTimePeriod,
                                          lFareFeaturesKey.toString());
    if (lFareFeatures_ptr == NULL) {
      lFareFeatures_ptr = &stdair::FacBom<stdair::FareFeatures>::instance().
        create (lFareFeaturesKey);
      assert(lFareFeatures_ptr != NULL);
      stdair::FacBomManager::addToListAndMap (iTimePeriod,
                                               *lFareFeatures_ptr);
      stdair::FacBomManager::linkWithParent (iTimePeriod,
                                             *lFareFeatures_ptr);
    }
    // Sanity check.
    assert(lFareFeatures_ptr != NULL);

    stdair::FareFeatures& lFareFeatures = *lFareFeatures_ptr;
    // Generate the airline-class list object corresponding to the
    // given fareRule
    createAirlineClassList (lFareFeatures, iFareRuleStruct);

  }

  // //////////////////////////////////////////////////////////////////////
  void FareRuleGenerator::
  createAirlineClassList (stdair::FareFeatures& iFareFeatures,
                          const FareRuleStruct& iFareRuleStruct) {

    // Create the AirlineClassList primary key.
    const unsigned int lAirlineListSize =
      iFareRuleStruct.getAirlineListSize();
    const unsigned int lClassCodeListSize =
      iFareRuleStruct.getClassCodeListSize();
    assert (lAirlineListSize == lClassCodeListSize);
    const stdair::AirlineClassListKey
      lAirlineClassListKey (iFareRuleStruct.getAirlineList(),
                            iFareRuleStruct.getClassCodeList());
    const stdair::Fare_T& lFare = iFareRuleStruct.getFare ();

    // Create the airline class list object and link it to the fare features
    // object.
    stdair::AirlineClassList* lAirlineClassList_ptr =
      &stdair::FacBom<stdair::AirlineClassList>::instance().
      create (lAirlineClassListKey);
    lAirlineClassList_ptr->setFare(lFare);
    stdair::FacBomManager::addToListAndMap (iFareFeatures,
                                            *lAirlineClassList_ptr);
    stdair::FacBomManager::linkWithParent(iFareFeatures,
                                          *lAirlineClassList_ptr);
  }

}

```

25.41 simfqt/command/FareRuleGenerator.hpp File Reference

```
#include <stdair/command/CmdAbstract.hpp>
#include <simfqt/SIMFQT_Types.hpp>
```

Classes

- class SIMFQT::FareRuleGenerator

Namespaces

- namespace stdair

  Forward declarations.

- namespace SIMFQT
- namespace SIMFQT::FareParserHelper

25.42 FareRuleGenerator.hpp

```
#ifndef __SIMFQT_CMD_FARERULEGENERATOR_HPP
#define __SIMFQT_CMD_FARERULEGENERATOR_HPP

// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// StdAir
#include <stdair/command/CmdAbstract.hpp>
// Simfqt
#include <simfqt/SIMFQT_Types.hpp>

// Forward declarations
namespace stdair {
  class BomRoot;
  class FareRule;
  class AirportPair;
  class DatePeriod;
  class PosChannel;
  class TimePeriod;
  class FareFeatures;
  class AirlineClassList;
}

namespace SIMFQT {

  // Forward declarations
  struct FareRuleStruct;
  namespace FareParserHelper {
    struct doEndFare;
  }

  class FareRuleGenerator : public stdair::CmdAbstract {

    // Only the following class may use methods of FareGenerator.
    // Indeed, as those methods build the BOM, it is not good to expose
    // them public.
    friend class FareFileParser;
    friend struct FareParserHelper::doEndFare;
    friend class FareParser;

  private:

    static void createAirportPair (stdair::BomRoot&,
                                   const FareRuleStruct&);

    static void createDateRange (stdair::AirportPair&,
                                 const FareRuleStruct&);

    static void createPOSChannel (stdair::DatePeriod&,
                                  const FareRuleStruct&);

    static void createTimeRange (stdair::PosChannel&,
                                 const FareRuleStruct&);

    static void createFareFeatures (stdair::TimePeriod&,
                                    const FareRuleStruct&);

    static void createAirlineClassList (stdair::FareFeatures&,
                                        const FareRuleStruct&);



  };

}
```

```
#endif // __SIMFQT_CMD_FARERULEGENERATOR_HPP
```

25.43 simfqt/factory/FacSimfqtServiceContext.cpp File Reference

```
#include <cassert>
#include <stdair/service/FacSupervisor.hpp>
#include <simfqt/factory/FacSimfqtServiceContext.hpp>
#include <simfqt/service/SIMFQT_ServiceContext.hpp>
```

Namespaces

- namespace SIMFQT

25.44 FacSimfqtServiceContext.cpp

```
// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <cassert>
// StdAir
#include <stdair/service/FacSupervisor.hpp>
// SimFQT
#include <simfqt/factory/FacSimfqtServiceContext.hpp>
#include <simfqt/service/SIMFQT_ServiceContext.hpp>

namespace SIMFQT {

  FacSimfqtServiceContext* FacSimfqtServiceContext::_instance = NULL;

  // //////////////////////////////////////////////////////////////////////
  FacSimfqtServiceContext::~FacSimfqtServiceContext() {
    _instance = NULL;
  }

  // //////////////////////////////////////////////////////////////////////
  FacSimfqtServiceContext& FacSimfqtServiceContext::instance() {

    if (_instance == NULL) {
      _instance = new FacSimfqtServiceContext();
      assert (_instance != NULL);

      stdair::FacSupervisor::instance().registerServiceFactory (_instance);
    }
    return *_instance;
  }

  // //////////////////////////////////////////////////////////////////////
  SIMFQT_ServiceContext& FacSimfqtServiceContext::create() {
    SIMFQT_ServiceContext* aServiceContext_ptr = NULL;

    aServiceContext_ptr = new SIMFQT_ServiceContext();
    assert (aServiceContext_ptr != NULL);

    // The new object is added to the Bom pool
    _pool.push_back (aServiceContext_ptr);

    return *aServiceContext_ptr;
  }

}
```

25.45 simfqt/factory/FacSimfqtServiceContext.hpp File Reference

```
#include <string>
#include <stdair/stdair_basic_types.hpp>
#include <stdair/service/FacServiceAbstract.hpp>
```

Classes

- class SIMFQT::FacSimfqtServiceContext

  Factory for the service context.

Namespaces

- namespace SIMFQT

25.46 FacSimfqtServiceContext.hpp

```
#ifndef __SIMFQT_FAC_FACSIMFQTSERVICECONTEXT_HPP
#define __SIMFQT_FAC_FACSIMFQTSERVICECONTEXT_HPP

// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <string>
// StdAir
#include <stdair/stdair_basic_types.hpp>
#include <stdair/service/FacServiceAbstract.hpp>

namespace SIMFQT {

  class SIMFQT_ServiceContext;


  class FacSimfqtServiceContext : public stdair::FacServiceAbstract {
  public:

    static FacSimfqtServiceContext& instance();

    ~FacSimfqtServiceContext();

    SIMFQT_ServiceContext& create();


  protected:
    FacSimfqtServiceContext() {}


  private:
    static FacSimfqtServiceContext* _instance;
  };

}
#endif // __SIMFQT_FAC_FACSIMFQTSERVICECONTEXT_HPP
```

25.47 simfqt/service/SIMFQT_Service.cpp File Reference

```
#include <cassert>
#include <boost/make_shared.hpp>
#include <stdair/basic/BasChronometer.hpp>
#include <stdair/bom/BomDisplay.hpp>
#include <stdair/bom/TravelSolutionStruct.hpp>
#include <stdair/bom/BookingRequestStruct.hpp>
#include <stdair/service/Logger.hpp>
#include <stdair/STDAIR_Service.hpp>
#include <simfqt/basic/BasConst_SIMFQT_Service.hpp>
#include <simfqt/factory/FacSimfqtServiceContext.hpp>
#include <simfqt/command/FareParser.hpp>
#include <simfqt/command/FareQuoter.hpp>
#include <simfqt/service/SIMFQT_ServiceContext.hpp>
#include <simfqt/SIMFQT_Service.hpp>
```

Namespaces

- namespace SIMFQT

25.48 SIMFQT_Service.cpp

```
// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <cassert>
// Boost
#include <boost/make_shared.hpp>
// StdAir
#include <stdair/basic/BasChronometer.hpp>
#include <stdair/bom/BomDisplay.hpp>
#include <stdair/bom/TravelSolutionStruct.hpp>
#include <stdair/bom/BookingRequestStruct.hpp>
#include <stdair/service/Logger.hpp>
#include <stdair/STDAIR_Service.hpp>
// Simfqt
#include <simfqt/basic/BasConst_SIMFQT_Service.hpp>
#include <simfqt/factory/FacSimfqtServiceContext.hpp>
#include <simfqt/command/FareParser.hpp>
#include <simfqt/command/FareQuoter.hpp>
#include <simfqt/service/SIMFQT_ServiceContext.hpp>
#include <simfqt/SIMFQT_Service.hpp>

namespace SIMFQT {

  // //////////////////////////////////////////////////////////////////////
  SIMFQT_Service::SIMFQT_Service() : _simfqtServiceContext (NULL) {
    assert (false);
  }

  // //////////////////////////////////////////////////////////////////////
  SIMFQT_Service::SIMFQT_Service (const SIMFQT_Service& iService) {
    assert (false);
  }

  // ////////////////////////////////////////////////////////////////////
  SIMFQT_Service::SIMFQT_Service (const stdair::BasLogParams& iLogParams)
    : _simfqtServiceContext (NULL) {

    // Initialise the STDAIR service handler
    stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
      initStdAirService (iLogParams);

    // Initialise the service context
    initServiceContext();

    // Add the StdAir service context to the SIMFQT service context
    // \note SIMFQT owns the STDAIR service resources here.
    const bool ownStdairService = true;
    addStdAirService (lSTDAIR_Service_ptr, ownStdairService);

    // Initialise the (remaining of the) context
    initSimfqtService();
  }

  // ////////////////////////////////////////////////////////////////////
  SIMFQT_Service::SIMFQT_Service (const stdair::BasLogParams& iLogParams,
                                  const stdair::BasDBParams& iDBParams)
    : _simfqtServiceContext (NULL) {

    // Initialise the STDAIR service handler
    stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
      initStdAirService (iLogParams, iDBParams);

    // Initialise the service context
    initServiceContext();

    // Add the StdAir service context to the SIMFQT service context
    // \note SIMFQT owns the STDAIR service resources here.
    const bool ownStdairService = true;
    addStdAirService (lSTDAIR_Service_ptr, ownStdairService);

    // Initialise the (remaining of the) context
    initSimfqtService();
  }

  // //////////////////////////////////////////////////////////////////////
  SIMFQT_Service::
  SIMFQT_Service (stdair::STDAIR_ServicePtr_T ioSTDAIR_Service_ptr)
    : _simfqtServiceContext (NULL) {

    // Initialise the service context
    initServiceContext();

    // Store the STDAIR service object within the (SIMFQT) service context
    // \note Simfqt does not own the STDAIR service resources here.
    const bool doesNotOwnStdairService = false;
    addStdAirService (ioSTDAIR_Service_ptr, doesNotOwnStdairService);

    // Initialise the context
    initSimfqtService();
  }

  // ////////////////////////////////////////////////////////////////////////
  SIMFQT_Service::~SIMFQT_Service() {
    // Delete/Clean all the objects from memory
    finalise();
  }

  // ////////////////////////////////////////////////////////////////////////
  void SIMFQT_Service::finalise() {
    assert (_simfqtServiceContext != NULL);
    // Reset the (Boost.)Smart pointer pointing on the STDAIR_Service object.
    _simfqtServiceContext->reset();
  }

  // ////////////////////////////////////////////////////////////////////////
  void SIMFQT_Service::initServiceContext() {
    // Initialise the service context
    SIMFQT_ServiceContext& lSIMFQT_ServiceContext =
      FacSimfqtServiceContext::instance().create();
    _simfqtServiceContext = &lSIMFQT_ServiceContext;
  }

  // //////////////////////////////////////////////////////////////////////
  void SIMFQT_Service::
  addStdAirService (stdair::STDAIR_ServicePtr_T ioSTDAIR_Service_ptr,
                    const bool iOwnStdairService) {

    // Retrieve the SimFQT service context
    assert (_simfqtServiceContext != NULL);
    SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *_simfqtServiceContext;

    // Store the STDAIR service object within the (SimFQT) service context
    lSIMFQT_ServiceContext.setSTDAIR_Service (ioSTDAIR_Service_ptr,
                                              iOwnStdairService);
  }

  // ////////////////////////////////////////////////////////////////////////
  stdair::STDAIR_ServicePtr_T SIMFQT_Service::
  initStdAirService (const stdair::BasLogParams& iLogParams,
                     const stdair::BasDBParams& iDBParams) {

```

```
    stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
      boost::make_shared<stdair::STDAIR_Service> (iLogParams, iDBParams);
    assert (lSTDAIR_Service_ptr != NULL);

    return lSTDAIR_Service_ptr;
  }

  // ////////////////////////////////////////////////////////////////////
  stdair::STDAIR_ServicePtr_T SIMFQT_Service::
  initStdAirService (const stdair::BasLogParams& iLogParams) {

    stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
      boost::make_shared<stdair::STDAIR_Service> (iLogParams);
    assert (lSTDAIR_Service_ptr != NULL);

    return lSTDAIR_Service_ptr;
  }

  // //////////////////////////////////////////////////////////////////////
  void SIMFQT_Service::initSimfqtService() {
    // Do nothing at this stage. A sample BOM tree may be built by
    // calling the buildSampleBom() method
  }

  // //////////////////////////////////////////////////////////////////////
  void SIMFQT_Service::
  parseAndLoad (const FareFilePath& iFareFilename) {

    // Retrieve the SimFQT service context
    if (_simfqtServiceContext == NULL) {
      throw stdair::NonInitialisedServiceException ("The SimFQT service "
                                                    "has not been initialised");
    }
    assert (_simfqtServiceContext != NULL);

    // Retrieve the SimFQT service context and whether it owns the Stdair
    // service
    SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *_simfqtServiceContext;
    const bool doesOwnStdairService =
      lSIMFQT_ServiceContext.getOwnStdairServiceFlag();

    // Retrieve the StdAir service object from the (SimFQT) service context
    stdair::STDAIR_Service& lSTDAIR_Service =
      lSIMFQT_ServiceContext.getSTDAIR_Service();

    // Retrieve the persistent BOM root object.
    stdair::BomRoot& lPersistentBomRoot =
      lSTDAIR_Service.getPersistentBomRoot();

    FareParser::fareRuleGeneration (iFareFilename, lPersistentBomRoot);

    buildComplementaryLinks (lPersistentBomRoot);

    if (doesOwnStdairService == true) {
      //
      clonePersistentBom ();
    }
  }

  // //////////////////////////////////////////////////////////////////////
  void SIMFQT_Service::buildSampleBom() {

    // Retrieve the SimFQT service context
    if (_simfqtServiceContext == NULL) {
      throw stdair::NonInitialisedServiceException ("The SimFQT service "
                                                    "has not been initialised");
    }
    assert (_simfqtServiceContext != NULL);

    // Retrieve the SimFQT service context and whether it owns the Stdair
    // service
    SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *_simfqtServiceContext;
    const bool doesOwnStdairService =
      lSIMFQT_ServiceContext.getOwnStdairServiceFlag();

    // Retrieve the StdAir service object from the (SimFQT) service context
    stdair::STDAIR_Service& lSTDAIR_Service =
      lSIMFQT_ServiceContext.getSTDAIR_Service();

    // Retrieve the persistent BOM root object.
    stdair::BomRoot& lPersistentBomRoot =
      lSTDAIR_Service.getPersistentBomRoot();

    if (doesOwnStdairService == true) {
      //
      lSTDAIR_Service.buildSampleBom();
    }

    buildComplementaryLinks (lPersistentBomRoot);

    if (doesOwnStdairService == true) {
      //
      clonePersistentBom ();
    }
  }

  // //////////////////////////////////////////////////////////////////////
  void SIMFQT_Service::clonePersistentBom () {

    // Retrieve the SimFQT service context
    if (_simfqtServiceContext == NULL) {
      throw stdair::NonInitialisedServiceException ("The SimFQT service "
                                                    "has not been initialised");
    }
    assert (_simfqtServiceContext != NULL);

    // Retrieve the SimFQT service context and whether it owns the Stdair
    // service
    SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *_simfqtServiceContext;
    const bool doesOwnStdairService =
      lSIMFQT_ServiceContext.getOwnStdairServiceFlag();

    // Retrieve the StdAir service object from the (SimFQT) service context
    stdair::STDAIR_Service& lSTDAIR_Service =
      lSIMFQT_ServiceContext.getSTDAIR_Service();

    if (doesOwnStdairService == true) {
      //
      lSTDAIR_Service.clonePersistentBom ();
    }

    stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
    buildComplementaryLinks (lBomRoot);
  }

  // //////////////////////////////////////////////////////////////////////
  void SIMFQT_Service::buildComplementaryLinks (stdair::BomRoot& ioBomRoot) {
    // Currently, no more things to do by SimFQT at that stage.
  }

  // //////////////////////////////////////////////////////////////////////
  stdair::BookingRequestStruct SIMFQT_Service::buildBookingRequest(const bool isF
orCRS) {

  // Retrieve the SIMFQT service context
  if (_simfqtServiceContext == NULL) {
    throw stdair::NonInitialisedServiceException ("The Simfqt service has not "

                                                  "been initialised");
  }
  assert (_simfqtServiceContext != NULL);

  SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *_simfqtServiceContext;

  // Retrieve the STDAIR service object from the (Simfqt) service context
  stdair::STDAIR_Service& lSTDAIR_Service =
    lSIMFQT_ServiceContext.getSTDAIR_Service();

  // Delegate the BOM building to the dedicated service
  stdair::BookingRequestStruct oBookingRequest =
    lSTDAIR_Service.buildSampleBookingRequest (isForCRS);

  return oBookingRequest;
}

// //////////////////////////////////////////////////////////////////////
void SIMFQT_Service::
buildSampleTravelSolutions(stdair::TravelSolutionList_T& ioTravelSolutionList){


  // Retrieve the SIMFQT service context
  if (_simfqtServiceContext == NULL) {
    throw stdair::NonInitialisedServiceException ("The Simfqt service has not "

                                                  "been initialised");
  }
  assert (_simfqtServiceContext != NULL);

  SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *_simfqtServiceContext;

  // Retrieve the STDAIR service object from the (Simfqt) service context
  stdair::STDAIR_Service& lSTDAIR_Service =
    lSIMFQT_ServiceContext.getSTDAIR_Service();

  // Delegate the BOM building to the dedicated service
  lSTDAIR_Service.buildSampleTravelSolutionForPricing (ioTravelSolutionList);
}


// //////////////////////////////////////////////////////////////////////
std::string SIMFQT_Service::csvDisplay() const {

  // Retrieve the SIMFQT service context
  if (_simfqtServiceContext == NULL) {
    throw stdair::NonInitialisedServiceException ("The SimFQT service "
                                                  "has not been initialised");
  }
  assert (_simfqtServiceContext != NULL);

  SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *_simfqtServiceContext;

  // Retrieve the STDAIR service object from the (SimFQT) service context
  stdair::STDAIR_Service& lSTDAIR_Service =
    lSIMFQT_ServiceContext.getSTDAIR_Service();

  // Get the root of the BOM tree, on which all of the other BOM objects
  // are attached
  stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();

  // Delegate the BOM display to the dedicated service
  std::ostringstream oCSVStr;
```

```
    stdair::BomDisplay::csvSimFQTAirRACDisplay (oCSVStr, lBomRoot);
    return oCSVStr.str();
  }

  // //////////////////////////////////////////////////////////////////////
  std::string SIMFQT_Service::
  csvDisplay (const stdair::TravelSolutionList_T& ioTravelSolutionList) const {

    // Retrieve the Simfqt service context
    if (_simfqtServiceContext == NULL) {
      throw stdair::NonInitialisedServiceException ("The Simfqt service has not "

                                                    "been initialised");
    }
    assert (_simfqtServiceContext != NULL);

    // Retrieve the Simfqt service context
    SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *_simfqtServiceContext;

    // Retrieve the STDAIR service object from the (Simfqt) service context
    stdair::STDAIR_Service& lSTDAIR_Service =
      lSIMFQT_ServiceContext.getSTDAIR_Service();

    // Delegate the BOM building to the dedicated service
    return lSTDAIR_Service.csvDisplay (ioTravelSolutionList);
  }

  // //////////////////////////////////////////////////////////////////////
  std::string SIMFQT_Service::
  csvDisplay (const stdair::AirportCode_T& iOrigin,
              const stdair::AirportCode_T& iDestination,
              const stdair::Date_T& iDepartureDate) const {

    // Retrieve the SIMFQT service context
    if (_simfqtServiceContext == NULL) {
      throw stdair::NonInitialisedServiceException ("The Simfqt service "
                                                    "has not been initialised");
    }
    assert (_simfqtServiceContext != NULL);

    SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *_simfqtServiceContext;

    // Retrieve the STDAIR service object from the (SIMFQT) service context
    stdair::STDAIR_Service& lSTDAIR_Service =
      lSIMFQT_ServiceContext.getSTDAIR_Service();

    // Delegate the BOM display to the dedicated service
    return lSTDAIR_Service.csvDisplay (iOrigin, iDestination,
                                       iDepartureDate);
  }

  // //////////////////////////////////////////////////////////////////////
  std::string SIMFQT_Service::list() const {

    // Retrieve the SIMFQT service context
    if (_simfqtServiceContext == NULL) {
      throw stdair::NonInitialisedServiceException ("The Simfqt service "
                                                    "has not been initialised");
    }
    assert (_simfqtServiceContext != NULL);

    SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *_simfqtServiceContext;

    // Retrieve the STDAIR service object from the (SIMFQT) service context
    stdair::STDAIR_Service& lSTDAIR_Service =
      lSIMFQT_ServiceContext.getSTDAIR_Service();

    // Delegate the BOM display to the dedicated service
    return lSTDAIR_Service.listAirportPairDateRange ();
  }

  // //////////////////////////////////////////////////////////////////////
  bool SIMFQT_Service::
  check (const stdair::AirportCode_T& iOrigin,
         const stdair::AirportCode_T& iDestination,
         const stdair::Date_T& iDepartureDate) const {
    std::ostringstream oFlightListStr;

    if (_simfqtServiceContext == NULL) {
      throw stdair::NonInitialisedServiceException ("The Simfqt service "
                                                    "has not been initialised");
    }
    assert (_simfqtServiceContext != NULL);
    SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *_simfqtServiceContext;

    // Retrieve the STDAIR service object from the (SIMFQT) service context
    stdair::STDAIR_Service& lSTDAIR_Service =
      lSIMFQT_ServiceContext.getSTDAIR_Service();

    // Delegate the BOM display to the dedicated service
    return lSTDAIR_Service.check (iOrigin, iDestination, iDepartureDate);
  }

  // //////////////////////////////////////////////////////////////////////
  void SIMFQT_Service::
  quotePrices (const stdair::BookingRequestStruct& iBookingRequest,
               stdair::TravelSolutionList_T& ioTravelSolutionList) {

    // Retrieve the Simfqt service context
    if (_simfqtServiceContext == NULL) {
      throw stdair::NonInitialisedServiceException ("The SimFQT service "
                                                    "has not been initialised");
    }
    assert (_simfqtServiceContext != NULL);

    SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *_simfqtServiceContext;

    // Retrieve the StdAir service context
    stdair::STDAIR_Service& lSTDAIR_Service =
      lSIMFQT_ServiceContext.getSTDAIR_Service();

    // Get the root of the BOM tree, on which all of the other BOM objects
    // will be attached
    stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();

    // Delegate the action to the dedicated command
    stdair::BasChronometer lFareQuoteRetrievalChronometer;
    lFareQuoteRetrievalChronometer.start();
    FareQuoter::priceQuote (iBookingRequest, ioTravelSolutionList, lBomRoot);

    // DEBUG
    const double lFareQuoteRetrievalMeasure =
      lFareQuoteRetrievalChronometer.elapsed();
    STDAIR_LOG_DEBUG ("Fare Quote retrieving: " << lFareQuoteRetrievalMeasure
                      << " - " << lSIMFQT_ServiceContext.display());
  }

  }
```

25.49 simfqt/service/SIMFQT_ServiceContext.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <simfqt/basic/BasConst_SIMFQT_Service.hpp>
#include <simfqt/service/SIMFQT_ServiceContext.hpp>
```

Namespaces

- namespace SIMFQT

25.50 SIMFQT_ServiceContext.cpp

```
// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <cassert>
#include <sstream>
// SimFQT
#include <simfqt/basic/BasConst_SIMFQT_Service.hpp>
#include <simfqt/service/SIMFQT_ServiceContext.hpp>

namespace SIMFQT {

  // //////////////////////////////////////////////////////////////////////
  SIMFQT_ServiceContext::SIMFQT_ServiceContext() : _ownStdairService (false) {
  }

  // //////////////////////////////////////////////////////////////////////
  SIMFQT_ServiceContext::SIMFQT_ServiceContext (const SIMFQT_ServiceContext&) {
    assert (false);
  }

  // //////////////////////////////////////////////////////////////////////
  SIMFQT_ServiceContext::~SIMFQT_ServiceContext() {
  }

  // //////////////////////////////////////////////////////////////////////
  stdair::STDAIR_Service& SIMFQT_ServiceContext::getSTDAIR_Service() const {
    assert (_stdairService != NULL);
    return *_stdairService;
  }

  // //////////////////////////////////////////////////////////////////////
  const std::string SIMFQT_ServiceContext::shortDisplay() const {
    std::ostringstream oStr;
    oStr << "SIMFQT_ServiceContext -- Owns StdAir service: "
         << _ownStdairService;
    return oStr.str();
  }

  // //////////////////////////////////////////////////////////////////////
  const std::string SIMFQT_ServiceContext::display() const {
    std::ostringstream oStr;
    oStr << shortDisplay();
    return oStr.str();
  }

  // //////////////////////////////////////////////////////////////////////
  const std::string SIMFQT_ServiceContext::describe() const {
    return shortDisplay();
  }

  // //////////////////////////////////////////////////////////////////////
  void SIMFQT_ServiceContext::reset() {

    // The shared_ptr<>::reset() method drops the refcount by one.
    // If the count result is dropping to zero, the resource pointed to
    // by the shared_ptr<> will be freed.

    // Reset the stdair shared pointer
    _stdairService.reset();
  }

}
```

25.51 simfqt/service/SIMFQT_ServiceContext.hpp File Reference

```
#include <string>
#include <stdair/stdair_service_types.hpp>
#include <stdair/service/ServiceAbstract.hpp>
#include <simfqt/SIMFQT_Types.hpp>
```

Classes

- class SIMFQT::SIMFQT_ServiceContext

  Class holding the context of the SimFQT services.

Namespaces

- namespace stdair

  Forward declarations.

- namespace SIMFQT

25.52 SIMFQT_ServiceContext.hpp

```
#ifndef __SIMFQT_SVC_SIMFQTSERVICECONTEXT_HPP
#define __SIMFQT_SVC_SIMFQTSERVICECONTEXT_HPP

// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <string>
// StdAir
#include <stdair/stdair_service_types.hpp>
#include <stdair/service/ServiceAbstract.hpp>
// SimFQT
#include <simfqt/SIMFQT_Types.hpp>

namespace stdair {
  class STDAIR_Service;
}

namespace SIMFQT {

  class SIMFQT_ServiceContext : public stdair::ServiceAbstract {
    friend class SIMFQT_Service;
    friend class FacSimfqtServiceContext;

  private:
    // ///////// Getters //////////
    stdair::STDAIR_ServicePtr_T getSTDAIR_ServicePtr() const {
      return _stdairService;
    }

    stdair::STDAIR_Service& getSTDAIR_Service() const;

    const bool getOwnStdairServiceFlag() const {
      return _ownStdairService;
    }


  private:
    // ///////// Setters //////////
    void setSTDAIR_Service (stdair::STDAIR_ServicePtr_T ioSTDAIR_ServicePtr,
                            const bool iOwnStdairService) {
      _stdairService = ioSTDAIR_ServicePtr;
      _ownStdairService = iOwnStdairService;
    }

    void reset();


  private:
    // ///////// Display Methods //////////
    const std::string shortDisplay() const;

    const std::string display() const;

    const std::string describe() const;


  private:
    // /////// Construction / initialisation ////////
    SIMFQT_ServiceContext (const FareQuoteID_T&);

    SIMFQT_ServiceContext();

    SIMFQT_ServiceContext (const SIMFQT_ServiceContext&);

    ~SIMFQT_ServiceContext();


  private:
    // ///////////// Children ////////////
    stdair::STDAIR_ServicePtr_T _stdairService;

    bool _ownStdairService;
  };

}
#endif // __SIMFQT_SVC_SIMFQTSERVICECONTEXT_HPP
```

25.53 simfqt/SIMFQT_Service.hpp File Reference

```
#include <stdair/stdair_basic_types.hpp>
#include <stdair/stdair_service_types.hpp>
#include <stdair/bom/TravelSolutionTypes.hpp>
#include <simfqt/SIMFQT_Types.hpp>
```

Classes

- class SIMFQT::SIMFQT_Service

  Interface for the SIMFQT Services.

Namespaces

- namespace stdair

  Forward declarations.

- namespace SIMFQT

25.54 SIMFQT_Service.hpp

```
#ifndef __SIMFQT_SVC_SIMFQT_SERVICE_HPP
#define __SIMFQT_SVC_SIMFQT_SERVICE_HPP

// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// StdAir
#include <stdair/stdair_basic_types.hpp>
#include <stdair/stdair_service_types.hpp>
#include <stdair/bom/TravelSolutionTypes.hpp>
// SimFQT
#include <simfqt/SIMFQT_Types.hpp>

namespace stdair {
  class STDAIR_Service;
  class BomRoot;
  struct BookingRequestStruct;
  struct BasLogParams;
  struct BasDBParams;
}

namespace SIMFQT {

  class SIMFQT_ServiceContext;


  class SIMFQT_Service {
  public:

    // ////////////////// Constructors and Destructors //////////////////
    SIMFQT_Service (const stdair::BasLogParams&);

    SIMFQT_Service (const stdair::BasLogParams&, const stdair::BasDBParams&);

    SIMFQT_Service (stdair::STDAIR_ServicePtr_T ioSTDAIR_ServicePtr);

    void parseAndLoad (const FareFilePath& iFareFilename);

    ~SIMFQT_Service();


  public:
    // /////////// Business Methods /////////////
    void buildSampleBom();

    void clonePersistentBom ();

    void buildComplementaryLinks (stdair::BomRoot&);

    stdair::BookingRequestStruct buildBookingRequest(const bool isForCRS = false)
;

    void buildSampleTravelSolutions (stdair::TravelSolutionList_T&);

    void quotePrices (const stdair::BookingRequestStruct&,
                      stdair::TravelSolutionList_T&);


  public:
    // //////////////// Display support methods /////////////////
    std::string csvDisplay() const;

    std::string csvDisplay (const stdair::TravelSolutionList_T&) const;

    std::string csvDisplay (const stdair::AirportCode_T& ioOrigin,
                            const stdair::AirportCode_T& ioDestination,
                            const stdair::Date_T& ioDepartureDate) const;

    std::string list() const;

    bool check (const stdair::AirportCode_T& ioOrigin,
                const stdair::AirportCode_T& ioDestination,
                const stdair::Date_T& ioDepartureDate) const;

  private:
    // /////// Construction and Destruction helper methods ///////
    SIMFQT_Service();

    SIMFQT_Service (const SIMFQT_Service&);

    stdair::STDAIR_ServicePtr_T initStdAirService (const stdair::BasLogParams&,
                                                   const stdair::BasDBParams&);

    stdair::STDAIR_ServicePtr_T initStdAirService (const stdair::BasLogParams&);

    void addStdAirService (stdair::STDAIR_ServicePtr_T ioSTDAIR_ServicePtr,
                           const bool iOwnStdairService);

    void initServiceContext();

    void initSimfqtService();

    void initSimfqtService (const FareFilePath& iFareFilename);

    void finalise();


  private:
    // ///////// Service Context /////////
    SIMFQT_ServiceContext* _simfqtServiceContext;
  };
}
#endif // __SIMFQT_SVC_SIMFQT_SERVICE_HPP
```

25.55 simfqt/SIMFQT_Types.hpp File Reference

```
#include <vector>
#include <string>
#include <boost/shared_ptr.hpp>
#include <stdair/stdair_exceptions.hpp>
#include <stdair/stdair_file.hpp>
```

Classes

- class SIMFQT::FareFileParsingFailedException
- class SIMFQT::AirportPairNotFoundException
- class SIMFQT::PosOrChannelNotFoundException
- class SIMFQT::FlightDateNotFoundException
- class SIMFQT::FlightTimeNotFoundException
- class SIMFQT::FeaturesNotFoundException
- class SIMFQT::AirlineNotFoundException
- class SIMFQT::FareInputFileNotFoundException
- class SIMFQT::QuotingException
- class SIMFQT::FareFilePath

Namespaces

- namespace SIMFQT

Typedefs

- typedef unsigned int SIMFQT::FareQuoteID_T
- typedef boost::shared_ptr< SIMFQT_Service > SIMFQT::SIMFQT_ServicePtr_T

25.56 SIMFQT_Types.hpp

```
#ifndef __SIMFQT_SIMFQT_TYPES_HPP
#define __SIMFQT_SIMFQT_TYPES_HPP

// ////////////////////////////////////////////////////////////////////
// Import section
// ////////////////////////////////////////////////////////////////////
// STL
#include <vector>
#include <string>
// Boost
#include <boost/shared_ptr.hpp>
// StdAir
#include <stdair/stdair_exceptions.hpp>
#include <stdair/stdair_file.hpp>

namespace SIMFQT {

  // Forward declarations
  class SIMFQT_Service;


  // ///////// Exceptions ///////////
  class FareFileParsingFailedException
    : public stdair::ParsingFileFailedException {
  public:
    FareFileParsingFailedException (const std::string& iWhat)
      : stdair::ParsingFileFailedException (iWhat) {}
  };

  class AirportPairNotFoundException : public stdair::ObjectNotFoundException {
  public:
    AirportPairNotFoundException (const std::string& iWhat)
      : stdair::ObjectNotFoundException (iWhat) {}
  };

  class PosOrChannelNotFoundException : public stdair::ObjectNotFoundException {
  public:
    PosOrChannelNotFoundException (const std::string& iWhat)
      : stdair::ObjectNotFoundException (iWhat) {}
  };

  class FlightDateNotFoundException : public stdair::ObjectNotFoundException {
  public:
    FlightDateNotFoundException (const std::string& iWhat)
      : stdair::ObjectNotFoundException (iWhat) {}
  };

  class FlightTimeNotFoundException : public stdair::ObjectNotFoundException {
  public:
    FlightTimeNotFoundException (const std::string& iWhat)
      : stdair::ObjectNotFoundException (iWhat) {}
  };

  class FeaturesNotFoundException : public stdair::ObjectNotFoundException {
  public:
    FeaturesNotFoundException (const std::string& iWhat)
      : stdair::ObjectNotFoundException (iWhat) {}
  };

  class AirlineNotFoundException : public stdair::ObjectNotFoundException {
  public:
    AirlineNotFoundException (const std::string& iWhat)
      : stdair::ObjectNotFoundException (iWhat) {}
  };

  class FareInputFileNotFoundException : public stdair::FileNotFoundException {
  public:
    FareInputFileNotFoundException (const std::string& iWhat)
      : stdair::FileNotFoundException (iWhat) {}
  };

  class QuotingException : public stdair::RootException {
  };

  // ///////// Files ///////////
  class FareFilePath : public stdair::InputFilePath {
  public:
    explicit FareFilePath (const stdair::Filename_T& iFilename)
      : stdair::InputFilePath (iFilename) {}
  };

  // //////// Type definitions specific to SimFQT /////////
  typedef unsigned int FareQuoteID_T;

  typedef boost::shared_ptr<SIMFQT_Service> SIMFQT_ServicePtr_T;
}
#endif // __SIMFQT_SIMFQT_TYPES_HPP
```

25.57 simfqt/ui/cmdline/simfqt.cpp File Reference

25.58 simfqt.cpp

```
// STL
#include <cassert>
#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
// Boost (Extended STL)
#include <boost/program_options.hpp>
#include <boost/tokenizer.hpp>
#include <boost/regex.hpp>
// StdAir
#include <stdair/basic/BasLogParams.hpp>
#include <stdair/basic/BasConst_BomDisplay.hpp>
#include <stdair/basic/BasDBParams.hpp>
#include <stdair/basic/BasConst_DefaultObject.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/basic/BasConst_Request.hpp>
#include <stdair/service/Logger.hpp>
#include <stdair/stdair_exceptions.hpp>
#include <stdair/stdair_basic_types.hpp>
#include <stdair/stdair_date_time_types.hpp>
#include <stdair/bom/TravelSolutionStruct.hpp>
#include <stdair/bom/BookingRequestStruct.hpp>
#include <stdair/bom/ParsedKey.hpp>
#include <stdair/bom/BomKeyManager.hpp>
#include <stdair/command/CmdBomManager.hpp>
// Stdair GNU Readline Wrapper
#include <stdair/ui/cmdline/SReadline.hpp>
// Simfqt
#include <simfqt/SIMFQT_Service.hpp>
#include <simfqt/config/simfqt-paths.hpp>


// //////// Constants //////
const std::string K_SIMFQT_DEFAULT_LOG_FILENAME ("simfqt.log");

const std::string K_SIMFQT_DEFAULT_FARE_INPUT_FILENAME (STDAIR_SAMPLE_DIR
                                                         "/fare01.csv");

const bool K_SIMFQT_DEFAULT_BUILT_IN_INPUT = false;

const int K_SIMFQT_EARLY_RETURN_STATUS = 99;

typedef std::vector<std::string> TokenList_T;

struct Command_T {
  typedef enum {
    NOP = 0,
    QUIT,
    HELP,
    LIST,
    DISPLAY,
    PRICE,
    LAST_VALUE
  } Type_T;
};

// ///////// Parsing of Options & Configuration /////////
// A helper function to simplify the main part.
template<class T> std::ostream& operator<< (std::ostream& os,
                                            const std::vector<T>& v) {
  std::copy (v.begin(), v.end(), std::ostream_iterator<T> (std::cout, " "));
  return os;
}
```

```

int readConfiguration (int argc, char* argv[], bool& ioIsBuiltin,
                       stdair::Filename_T& ioFareInputFilename,
                       std::string& ioLogFilename) {

  // Default for the built-in input
  ioIsBuiltin = K_SIMFQT_DEFAULT_BUILT_IN_INPUT;

  // Declare a group of options that will be allowed only on command line
  boost::program_options::options_description generic ("Generic options");
  generic.add_options()
    ("prefix", "print installation prefix")
    ("version,v", "print version string")
    ("help,h", "produce help message");

  // Declare a group of options that will be allowed both on command
  // line and in config file
  boost::program_options::options_description config ("Configuration");
  config.add_options()
    ("builtin,b",
     "The sample BOM tree can be either built-in or parsed from an input file. That latter must then be given with the -f/--fare option")
    ("fare,f",
     boost::program_options::value< std::string >(&ioFareInputFilename)->default_value(K_SIMFQT_DEFAULT_FARE_INPUT_FILENAME),
     "(CSV) input file for the fare rules")
    ("log,l",
     boost::program_options::value< std::string >(&ioLogFilename)->default_value(K_SIMFQT_DEFAULT_LOG_FILENAME),
     "Filename for the logs")
    ;

  // Hidden options, will be allowed both on command line and
  // in config file, but will not be shown to the user.
  boost::program_options::options_description hidden ("Hidden options");
  hidden.add_options()
    ("copyright",
     boost::program_options::value< std::vector<std::string> >(),
     "Show the copyright (license)");

  boost::program_options::options_description cmdline_options;
  cmdline_options.add(generic).add(config).add(hidden);

  boost::program_options::options_description config_file_options;
  config_file_options.add(config).add(hidden);

  boost::program_options::options_description visible ("Allowed options");
  visible.add(generic).add(config);

  boost::program_options::positional_options_description p;
  p.add ("copyright", -1);

  boost::program_options::variables_map vm;
  boost::program_options::
    store (boost::program_options::command_line_parser (argc, argv).
           options (cmdline_options).positional(p).run(), vm);

  std::ifstream ifs ("simfqt.cfg");
  boost::program_options::store (parse_config_file (ifs, config_file_options),
                                 vm);
  boost::program_options::notify (vm); if (vm.count ("help")) {
    std::cout << visible << std::endl;
    return K_SIMFQT_EARLY_RETURN_STATUS;
  }

  if (vm.count ("version")) {
    std::cout << PACKAGE_NAME << ", version " << PACKAGE_VERSION << std::endl;
    return K_SIMFQT_EARLY_RETURN_STATUS;
  }

  if (vm.count ("prefix")) {
    std::cout << "Installation prefix: " << PREFIXDIR << std::endl;
    return K_SIMFQT_EARLY_RETURN_STATUS;
  }

  if (vm.count ("builtin")) {
    ioIsBuiltin = true;
  }
  const std::string isBuiltinStr = (ioIsBuiltin == true)?"yes":"no";
  std::cout << "The BOM should be built-in? " << isBuiltinStr << std::endl;

  if (ioIsBuiltin == false) {

    // The BOM tree should be built from parsing a fare (and O&D) file
    if (vm.count ("fare")) {
      ioFareInputFilename = vm["fare"].as< std::string >();
      std::cout << "Input fare filename is: " << ioFareInputFilename
                << std::endl;

    } else {
      // The built-in option is not selected. However, no fare file
      // is specified
      std::cerr << "Either one among the -b/--builtin and -f/--fare "
                << "options must be specified" << std::endl;
    }
  }

  if (vm.count ("log")) {
    ioLogFilename = vm["log"].as< std::string >();
    std::cout << "Log filename is: " << ioLogFilename << std::endl;
  }

  return 0;

}

// //////////////////////////////////////////////////////////////////////
void initReadline (swift::SReadline& ioInputReader) {

  // Prepare the list of my own completers
  std::vector<std::string> Completers;

  // The following is supported:
  // - "identifiers"
  // - special identifier %file - means to perform a file name completion
  Completers.push_back ("help");
  Completers.push_back ("list");
  Completers.push_back ("display %airport_code %airport_code %departure_date");
  Completers.push_back ("price %airline_code %flight_number %departure_date %airp
ort_code %airport_code %departure_time %booking_date %booking_time %POS %channel%
 %trip_type %stay_duration");
  Completers.push_back ("quit");

  // Now register the completers.
  // Actually it is possible to re-register another set at any time
  ioInputReader.RegisterCompletions (Completers);
}

// //////////////////////////////////////////////////////////////////////
Command_T::Type_T extractCommand (TokenList_T& ioTokenList) {
  Command_T::Type_T oCommandType = Command_T::LAST_VALUE;

  // Interpret the user input
  if (ioTokenList.empty() == false) {
    TokenList_T::iterator itTok = ioTokenList.begin();
    std::string& lCommand (*itTok);
    boost::algorithm::to_lower (lCommand);

    if (lCommand == "help") {
      oCommandType = Command_T::HELP;

    } else if (lCommand == "list") {
      oCommandType = Command_T::LIST;

    } else if (lCommand == "display") {
      oCommandType = Command_T::DISPLAY;

    } else if (lCommand == "price") {
      oCommandType = Command_T::PRICE;

    } else if (lCommand == "quit") {
      oCommandType = Command_T::QUIT;

    }

    // Remove the first token (the command), as the corresponding information
    // has been extracted in the form of the returned command type enumeration
    ioTokenList.erase (itTok);

  } else {
    oCommandType = Command_T::NOP;
  }

  return oCommandType;
}

// ///////////////////////////////////////////////////////////////////
// Re-compose a date using three strings: the year, the month and the
// day. Return true if a correct date has been computed, false if not.
bool retrieveDate (std::string iYearString,
                   std::string iMonthString,
                   std::string iDayString,
                   stdair::Date_T& ioDate) {

  const std::string kMonthStr[12] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
                                     "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};

  // Check the year.
  unsigned short lDateYear;
  try {

    lDateYear = boost::lexical_cast<unsigned short> (iYearString);
    if (lDateYear < 100) {
      lDateYear += 2000;
    }

  } catch (boost::bad_lexical_cast& eCast) {
    std::cerr << "The year ('" << iYearString
              << "') cannot be understood." << std::endl;
    return false;
  }

  // Check the month.
  std::string lDateMonthStr;
  try {

    const boost::regex lMonthRegex ("^(\\d{1,2})$");
    const bool isMonthANumber = regex_match (iMonthString, lMonthRegex);

    if (isMonthANumber == true) {
      const unsigned short lMonth =
        boost::lexical_cast<unsigned short> (iMonthString);
      if (lMonth > 12) {
        throw boost::bad_lexical_cast();
      }
      if (lMonth != 0) {
        lDateMonthStr = kMonthStr[lMonth-1];
      } else {
        std::cerr << "The month ('" << iMonthString
              << "') cannot be understood." << std::endl;
        return false;
      }

    } else {
      if (iMonthString.size() < 3) {
        throw boost::bad_lexical_cast();
      }
      std::string lMonthStr1 (iMonthString.substr (0, 1));
      boost::algorithm::to_upper (lMonthStr1);
      std::string lMonthStr23 (iMonthString.substr (1, 2));
      boost::algorithm::to_lower (lMonthStr23);
      lDateMonthStr = lMonthStr1 + lMonthStr23;
    }

  } catch (boost::bad_lexical_cast& eCast) {
    std::cerr << "The month ('" << iMonthString
              << "') cannot be understood." << std::endl;
    return false;
  }

  // Check the day.
  unsigned short lDateDay;
  try {

    lDateDay = boost::lexical_cast<unsigned short> (iDayString);

  } catch (boost::bad_lexical_cast& eCast) {
    std::cerr << "The day ('" << iDayString
              << "') cannot be understood." << std::endl;
    return false;
  }

  // Re-compose the date.
  std::ostringstream lDateStr;
  lDateStr << lDateYear << "-" << lDateMonthStr
           << "-" << lDateDay;
  try {

    ioDate =
      boost::gregorian::from_simple_string (lDateStr.str());

  } catch (boost::gregorian::bad_month& eCast) {
    std::cerr << "The month of the date ('" << lDateStr.str()
              << "') cannot be understood." << std::endl;
    return false;
  } catch (boost::gregorian::bad_day_of_month& eCast) {
    std::cerr << "The date ('" << lDateStr.str()
              << "') is not correct: the day of month does not exist."
              << std::endl;
    return false;
  } catch (boost::gregorian::bad_year& eCast) {
    std::cerr << "The year ('" << lDateStr.str()
              << "') is not correct."
              << std::endl;
    return false;
  }

  return true;
```

```
}

// //////////////////////////////////////////////////////////////////////
// Re-compose a time using two strings: the hour and the minute.
// Return true if a correct time has been computed, false if not.
bool retrieveTime (std::string iHourString,
                   std::string iMinuteString,
                   stdair::Duration_T& oTime) {

  // Check the hour
  unsigned short lTimeHour;
  try {

    lTimeHour = boost::lexical_cast<unsigned short> (iHourString);

  } catch (boost::bad_lexical_cast& eCast) {
    std::cerr << "The hour of the time ('" << iHourString
              << "') cannot be understood." << std::endl;
    return false;
  }

  // Check the minutes
  unsigned short lTimeMinute;
  try {

    lTimeMinute = boost::lexical_cast<unsigned short> (iMinuteString);

  } catch (boost::bad_lexical_cast& eCast) {
    std::cerr << "The minute of the time ('" << iMinuteString
              << "') cannot be understood." << std::endl;
    return false;
  }


  // Re-compose the time
  std::ostringstream lTimeStr;
  lTimeStr << lTimeHour << ":" << lTimeMinute;
  oTime =
    boost::posix_time::duration_from_string (lTimeStr.str());

  return true;
}

// //////////////////////////////////////////////////////////////////////
// Analyze the tokens of the 'price' command in order to construct
// a travel solution list and a booking request.
const stdair::BookingRequestStruct parseTravelSolutionAndBookingRequestKey
(const TokenList_T& iTokenList,
 stdair::TravelSolutionList_T& ioInteractiveTravelSolutionList,
 const stdair::BookingRequestStruct& ioBookingRequestStruct) {

  TokenList_T::const_iterator itTok = iTokenList.begin();

  if (itTok->empty() == true) {

    std::cerr << "Wrong list of parameters. "
              << "The default booking request and travel solution list are kept."

              << std::endl;
    return ioBookingRequestStruct;


  } else {
    // Parameters corresponding to the tokens.
    // Each parameter correponds to one token except the dates
    // (three tokens) and the times (two tokens).
    stdair::AirlineCode_T lAirlineCode;
    stdair::FlightNumber_T lflightNumber;
    stdair::Date_T lDepartureDate;
    stdair::Duration_T lDepartureTime;
    stdair::AirportCode_T lOriginAirport;
    stdair::AirportCode_T lDestinationAirport;
    stdair::Date_T lRequestDate;
    stdair::Duration_T lRequestTime;
    stdair::CityCode_T lPOS;
    stdair::ChannelLabel_T lChannel;
    stdair::TripType_T lTripType;
    unsigned short lStayDuration;

    // Read the airline code.
    lAirlineCode = *itTok;
    boost::algorithm::to_upper (lAirlineCode);

    // Read the flight-number  .
    ++itTok;
    if (itTok->empty() == false) {
      try {

        lflightNumber = boost::lexical_cast<stdair::FlightNumber_T> (*itTok);

      } catch (boost::bad_lexical_cast& eCast) {
        std::cerr << "The flight number ('" << *itTok
                  << "') cannot be understood."
                  << std::endl;
        return ioBookingRequestStruct;
      }
    }

    // Read the departure date.
    ++itTok;
    if (itTok->empty() == true) {
      return ioBookingRequestStruct;
    }
    const std::string lDepartureYearString = *itTok;
    ++itTok;
    if (itTok->empty() == true) {
      return ioBookingRequestStruct;
    }
    const std::string lDepartureMonthString = *itTok;
    ++itTok;
    if (itTok->empty() == true) {
      return ioBookingRequestStruct;
    }
    const std::string lDepartureDayString = *itTok;
    const bool IsDepartureDateReadable =
      retrieveDate (lDepartureYearString, lDepartureMonthString,
                    lDepartureDayString, lDepartureDate);

    if (IsDepartureDateReadable == false) {
      std::cerr << "The default booking request and travel solution list are kept
."
                << std::endl;
      return ioBookingRequestStruct;
    }

    // Read the origin.
    ++itTok;
    if (itTok->empty() == false) {
      lOriginAirport = *itTok;
      boost::algorithm::to_upper (lOriginAirport);
    }

    // Read the destination.
    ++itTok;
    if (itTok->empty() == false) {
      lDestinationAirport = *itTok;
      boost::algorithm::to_upper (lDestinationAirport);
    }

    // Read the departure time.
    ++itTok;
    if (itTok->empty() == true) {
      return ioBookingRequestStruct;
    }
    const std::string lDepartureHourString = *itTok;
    ++itTok;
    if (itTok->empty() == true) {
      return ioBookingRequestStruct;
    }
    const std::string lDepartureMinuteString = *itTok;
    const bool IsDepartureTimeReadable =
      retrieveTime (lDepartureHourString, lDepartureMinuteString,
                    lDepartureTime);

    if (IsDepartureTimeReadable == false) {
      std::cerr << "The default booking request and travel solution list are kept
."
                << std::endl;
      return ioBookingRequestStruct;
    }

    // Read the request date.
    ++itTok;
    if (itTok->empty() == true) {
        return ioBookingRequestStruct;
    }
    const std::string lRequestYearString = *itTok;
    ++itTok;
    if (itTok->empty() == true) {
      return ioBookingRequestStruct;
    }
    const std::string lRequestMonthString = *itTok;
    ++itTok;
    if (itTok->empty() == true) {
      return ioBookingRequestStruct;
    }
    const std::string lRequestDayString = *itTok;
    const bool IsRequestDateReadable =
      retrieveDate (lRequestYearString, lRequestMonthString,
                    lRequestDayString, lRequestDate);

    if (IsRequestDateReadable == false) {
      std::cerr << "The default booking request and travel solution list are kept
."
                << std::endl;
      return ioBookingRequestStruct;
    }

    // Read the request time.
    ++itTok;
    if (itTok->empty() == true) {
      return ioBookingRequestStruct;
    }
    const std::string lRequestHourString = *itTok;
    ++itTok;
    if (itTok->empty() == true) {
      return ioBookingRequestStruct;
    }
    const std::string lRequestMinuteString = *itTok;
    const bool IsRequestTimeReadable =
      retrieveTime (lRequestHourString, lRequestMinuteString,
                    lRequestTime);

    if (IsRequestTimeReadable == false) {
      std::cerr << "The default booking request and travel solution list are kept
."
                << std::endl;
      return ioBookingRequestStruct;
    }

    // Read the POS.
    ++itTok;
    if (itTok->empty() == false) {
      lPOS = *itTok;
      boost::algorithm::to_upper (lPOS);
    }

    // Read the channel.
    ++itTok;
    if (itTok->empty() == false) {
      lChannel = *itTok;
      boost::algorithm::to_upper (lChannel);
    }

    // Read the trip type.
    ++itTok;
    if (itTok->empty() == false) {
      lTripType = *itTok;
      boost::algorithm::to_upper (lTripType);
    }

    // Read the stay duration.
    ++itTok;
    if (itTok->empty() == false) {
      try {

        lStayDuration = boost::lexical_cast<unsigned short> (*itTok);

      } catch (boost::bad_lexical_cast& eCast) {
        std::cerr << "The stay duration ('" << *itTok
                  << "') cannot be understood." << std::endl;
        return ioBookingRequestStruct;
      }
    }

    // At this step we know that all the parameters designed to construct
    // the travel solution and the booking request are correct.

    // Empty the travel solution list to store a new travel solution.
    ioInteractiveTravelSolutionList.pop_front();
    // Construct the new travel solution.
    stdair::TravelSolutionStruct lTravelSolution;
    std::ostringstream oStr;
    oStr << lAirlineCode
         << stdair::DEFAULT_KEY_FLD_DELIMITER
         << lflightNumber
         << stdair::DEFAULT_KEY_SUB_FLD_DELIMITER
         << lDepartureDate
         << stdair::DEFAULT_KEY_FLD_DELIMITER
         << lOriginAirport
         << stdair::DEFAULT_KEY_SUB_FLD_DELIMITER
         << lDestinationAirport
         << stdair::DEFAULT_KEY_FLD_DELIMITER
         << lDepartureTime;
    lTravelSolution.addSegment (oStr.str());
    ioInteractiveTravelSolutionList.push_front(lTravelSolution);

    // Construct the new booking request.
    stdair::DateTime_T lRequestDateTime (lRequestDate, lRequestTime);
    const stdair::BookingRequestStruct &lBookingRequestStruct =
      stdair::BookingRequestStruct(lOriginAirport,
                                   lDestinationAirport,
                                   lPOS,
                                   lDepartureDate,
                                   lRequestDateTime,
                                   stdair::CABIN_ECO,
                                   stdair::DEFAULT_PARTY_SIZE,
                                   lChannel,
                                   lTripType,
                                   lStayDuration,
                                   stdair::FREQUENT_FLYER_MEMBER,
                                   lDepartureTime,
                                   stdair::DEFAULT_WTP,
                                   stdair::DEFAULT_VALUE_OF_TIME,
                                   true, 50, true, 50);

    return lBookingRequestStruct;
  }
}

// //////////////////////////////////////////////////////////////////////
// Analyze the tokens of the 'display' command in order to retrieve
// an airport pair and a departure date.
void parseFlightDateKey (const TokenList_T& iTokenList,
                         stdair::AirportCode_T& ioOrigin,
                         stdair::AirportCode_T& ioDestination,
                         stdair::Date_T& ioDepartureDate) {

  TokenList_T::const_iterator itTok = iTokenList.begin();

  // Interpret the user input.
  if (itTok->empty() == true) {

    std::cerr << "Wrong parameters specified. Default paramaters '"
              << ioOrigin << "-" << ioDestination
              << "/" << ioDepartureDate
              << "' are kept."
             << std::endl;

  } else {

    // Read the origin.
    ioOrigin = *itTok;
    boost::algorithm::to_upper (ioOrigin);

    // Read the destination.
    ++itTok;
    if (itTok->empty() == false) {
      ioDestination  = *itTok;
      boost::algorithm::to_upper (ioDestination);
    }

    // Read the departure date.
    ++itTok;
    if (itTok->empty() == true) {
      return;
    }
    std::string lYearString = *itTok;
    ++itTok;
    if (itTok->empty() == true) {
      return;
    }
    std::string lMonthString = *itTok;
    ++itTok;
    if (itTok->empty() == true) {
      return;
    }
    std::string lDayString = *itTok;
    const bool IsDepartureDateReadable =
      retrieveDate (lYearString, lMonthString, lDayString,
                    ioDepartureDate);
    if (IsDepartureDateReadable == false) {
      std::cerr << "Default paramaters '"
                << ioOrigin << "-" << ioDestination
                << "/" << ioDepartureDate
                << "' are kept."
                << std::endl;
      return;
    }
  }
}

// ////////////////////////////////////////////////////////////
std::string toString (const TokenList_T& iTokenList) {
  std::ostringstream oStr;

  // Re-create the string with all the tokens, trimmed by read-line
  unsigned short idx = 0;
  for (TokenList_T::const_iterator itTok = iTokenList.begin();
       itTok != iTokenList.end(); ++itTok, ++idx) {
    if (idx != 0) {
      oStr << " ";
    }
    oStr << *itTok;
  }

  return oStr.str();
}

// ////////////////////////////////////////////////////////////
TokenList_T extractTokenList (const TokenList_T& iTokenList,
                              const std::string& iRegularExpression) {
  TokenList_T oTokenList;

  // Re-create the string with all the tokens (which had been trimmed
  // by read-line)
  const std::string lFullLine = toString (iTokenList);

  // See the caller for the regular expression
  boost::regex expression (iRegularExpression);

  std::string::const_iterator start = lFullLine.begin();
  std::string::const_iterator end = lFullLine.end();

  boost::match_results<std::string::const_iterator> what;
  boost::match_flag_type flags = boost::match_default | boost::format_sed;
  regex_search (start, end, what, expression, flags);

  // Put the matched strings in the list of tokens to be returned back
  // to the caller
  const unsigned short lMatchSetSize = what.size();
  for (unsigned short matchIdx = 1; matchIdx != lMatchSetSize; ++matchIdx) {
    const std::string lMatchedString (std::string (what[matchIdx].first,
                                                   what[matchIdx].second));
    //if (lMatchedString.empty() == false) {
    oTokenList.push_back (lMatchedString);
    //}
  }

  // DEBUG
  // std::cout << "After (token list): " << oTokenList << std::endl;

  return oTokenList;
}

// ///////////////////////////////////////////////////////////
// Parse the token list of the 'price' command.
TokenList_T extractTokenListForTSAndBR (const TokenList_T& iTokenList) {
  const std::string lRegEx("^([[:alpha:]]{2,3})"
                           "[[:space:]]+([[:digit:]]{1,4})"
                           "[/ ]*"
                           "[[:space:]]+([[:digit:]]{2,4})[/-]?"
                           "[[:space:]]*([[:alpha:]]{3}|[[:digit:]]{1,2})[/-]?"
                           "[[:space:]]*([[:digit:]]{1,2})[[:space:]]*"
                           "[[:space:]]+([[:alpha:]]{3})"
                           "[[:space:]]+([[:alpha:]]{3})"
                           "[[:space:]]+([[:digit:]]{1,2})[:]?([[:digit:]]{1,2})"

                           "[[:space:]]+([[:digit:]]{2,4})[/-]?"
                           "[[:space:]]*([[:alpha:]]{3}|[[:digit:]]{1,2})[/-]?"
                           "[[:space:]]*([[:digit:]]{1,2})"
                           "[[:space:]]+([[:digit:]]{1,2})[:]?([[:digit:]]{1,2})"

                           "[[:space:]]+([[:alpha:]]{3})"
                           "[[:space:]]+([[:alpha:]]{2})"
                           "[[:space:]]+([[:alpha:]]{2})"
                           "[[:space:]]+([[:digit:]]{1})$");

  //
  const TokenList_T& oTokenList = extractTokenList (iTokenList, lRegEx);
  return oTokenList;
}

// ///////////////////////////////////////////////////////////
// Parse the token list of the 'display' command.
TokenList_T extractTokenListForOriDestDate (const TokenList_T& iTokenList) {
  const std::string lRegEx("^([[:alpha:]]{3})"
                           "[[:space:]]*[/-]?"
                           "[[:space:]]*([[:alpha:]]{3})"
                           "[[:space:]]*[/-]?"
                           "[[:space:]]*([[:digit:]]{2,4})"
                           "[[:space:]]*[/-]?"
                           "[[:space:]]*([[:alpha:]]{3}|[[:digit:]]{1,2})"
                           "[[:space:]]*[/-]?"
                           "[[:space:]]*([[:digit:]]{1,2})$");

  //
  const TokenList_T& oTokenList = extractTokenList (iTokenList, lRegEx);
  return oTokenList;
}

// ///////// M A I N ////////////
int main (int argc, char* argv[]) {

  // State whether the BOM tree should be built-in or parsed from an
  // input file
  bool isBuiltin;

  // Fare input file name
  stdair::Filename_T lFareInputFilename;

  // Readline history
  const unsigned int lHistorySize (100);
  const std::string lHistoryFilename ("simfqt.hist");
  const std::string lHistoryBackupFilename ("simfqt.hist.bak");

  // Default parameters for the interactive session
  stdair::AirportCode_T lInteractiveOrigin;
  stdair::AirportCode_T lInteractiveDestination;
  stdair::Date_T lInteractiveDepartureDate;

  // Output log File
  stdair::Filename_T lLogFilename;

  // Call the command-line option parser
  const int lOptionParserStatus =
    readConfiguration (argc, argv, isBuiltin, lFareInputFilename, lLogFilename);

  if (lOptionParserStatus == K_SIMFQT_EARLY_RETURN_STATUS) {
    return 0;
  }

  // Set the log parameters
  std::ofstream logOutputFile;
  // Open and clean the log outputfile
  logOutputFile.open (lLogFilename.c_str());
  logOutputFile.clear();

  // Initialise the fareQuote service
  const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);
  SIMFQT::SIMFQT_Service simfqtService (lLogParams);

  // DEBUG
  STDAIR_LOG_DEBUG ("Welcome to SimFQT display");

  // Check wether or not a (CSV) input file should be read
  if (isBuiltin == true) {
    // Build the sample BOM tree (filled with fares) for Simfqt
    simfqtService.buildSampleBom();
  } else {
    // Build the BOM tree from parsing a fare file
    SIMFQT::FareFilePath lFareFilePath (lFareInputFilename);
    simfqtService.parseAndLoad (lFareFilePath);
  }

  // DEBUG: Display the whole BOM tree
  const std::string& lCSVDump = simfqtService.csvDisplay();
  STDAIR_LOG_DEBUG (lCSVDump);

  // DEBUG
  STDAIR_LOG_DEBUG ("====================================================");
  STDAIR_LOG_DEBUG ("=       Beginning of the interactive session       =");
  STDAIR_LOG_DEBUG ("====================================================");

  // Initialise the GNU readline wrapper
  swift::SReadline lReader (lHistoryFilename, lHistorySize);
  initReadline (lReader);

  // Now we can ask user for a line
  std::string lUserInput;
  bool EndOfInput (false);
  Command_T::Type_T lCommandType (Command_T::NOP);

  while (lCommandType != Command_T::QUIT && EndOfInput == false) {

    stdair::TravelSolutionList_T lInteractiveTravelSolutionList;
    stdair::TravelSolutionStruct lInteractiveTravelSolution;

    // Update the default booking request.
    // If there is an input file, we want the CRS booking request (defined in stdair).
    // If not, we want the default booking request.
    const bool isCRSBookingRequest = !isBuiltin;
    const stdair::BookingRequestStruct& lInteractiveBookingRequest =
      simfqtService.buildBookingRequest (isCRSBookingRequest);

    // Update the default parameters for the following interactive session.
    if (isBuiltin == true) {
      lInteractiveOrigin = "LHR";
      lInteractiveDestination = "SYD";
      lInteractiveDepartureDate = stdair::Date_T(2011,06,10);
      simfqtService.buildSampleTravelSolutions (lInteractiveTravelSolutionList);
    } else {
      lInteractiveOrigin = "SIN";
      lInteractiveDestination = "BKK";
      lInteractiveDepartureDate = stdair::Date_T(2010,01,30);
      //
      const std::string lBA9_SegmentDateKey ("SQ, 970, 2010-01-30, SIN, BKK, 07:1
0");

      // Add the segment date key to the travel solution.
      lInteractiveTravelSolution.addSegment (lBA9_SegmentDateKey);

      // Add the travel solution to the list
      lInteractiveTravelSolutionList.push_back (lInteractiveTravelSolution);
    }

    // Prompt.
    std::ostringstream oPromptStr;
    oPromptStr << "simfqt "
               << "> ";
    // The last parameter could be ommited.
    TokenList_T lTokenListByReadline;
    lUserInput = lReader.GetLine (oPromptStr.str(), lTokenListByReadline,
                                  EndOfInput);

    // The history could be saved to an arbitrary file at any time.
    lReader.SaveHistory (lHistoryBackupFilename);

    if (EndOfInput) {
      std::cout << std::endl;
      break;
    }

    // Interpret the user input.
    lCommandType = extractCommand (lTokenListByReadline);

    switch (lCommandType) {

      // ////////////////////////////// Help ///////////////////////
    case Command_T::HELP: {
      // Search for information to display default parameters lists.
      // Get the first travel solution.
      stdair::TravelSolutionStruct& lTravelSolutionStruct =
        lInteractiveTravelSolutionList.front();
      // Get the segment-path of the first travel solution.
      const stdair::SegmentPath_T& lSegmentPath =
        lTravelSolutionStruct.getSegmentPath();
      // Get the first segment of the first travel solution.
      const std::string& lSegmentDateKey = lSegmentPath.front();
      // Get the parsed key of the first segment of the first travel solution.
      const stdair::ParsedKey& lParsedKey =
        stdair::BomKeyManager::extractKeys (lSegmentDateKey);
      // Get the request date time
      const stdair::DateTime_T& lRequestDateTime =
        lInteractiveBookingRequest.getRequestDateTime();
      const stdair::Time_T lRequestTime =
        lRequestDateTime.time_of_day();
      std::cout << std::endl;
      // Display help.
      std::cout << "Commands: " << std::endl;
      std::cout << " help" << "\t\t" << "Display this help" << std::endl;
      std::cout << " quit" << "\t\t" << "Quit the application" << std::endl;
      std::cout << " list" << "\t\t"
                << "List all the fare rule O&Ds and the corresponding date ranges
" << std::endl;
      std::cout << " display" << "\t"
                << "Display all fare rules for an O&D and a departure date. \n" <
< "\t\t"
                << "If no parameters specified or wrong list of parameters, defau
lt values are used: \n"<< "\t\t"
                << "      display " <<  lInteractiveOrigin << " "
                << lInteractiveDestination << " "
                << lInteractiveDepartureDate << std::endl;
      std::cout << " price" << "\t\t"
                << "Price the travel solution corresponding to a booking request.
 \n" << "\t\t"
                << "If no parameters specified or wrong list of parameters, defau
lt value are used: \n" << "\t\t"
                << "      price "
                << lParsedKey._airlineCode << " "
                << lParsedKey._flightNumber << " "
                << lParsedKey._departureDate << " "
                << lParsedKey._boardingPoint << " "
                << lParsedKey._offPoint << " "
                << lParsedKey._boardingTime << " "
                << lRequestDateTime.date() << " "
                << lRequestTime.hours() << ":" << lRequestTime.minutes() << " "
                << lInteractiveBookingRequest.getPOS() << " "
                << lInteractiveBookingRequest.getBookingChannel() << " "
                << lInteractiveBookingRequest.getTripType() << " "
                << lInteractiveBookingRequest.getStayDuration() << std::endl;
      std::cout << std::endl;
      break;
    }

      // /////////////////////////////// Quit ////////////////////////
    case Command_T::QUIT: {
      break;
    }

      // /////////////////////////////// List /////////////////////////
    case Command_T::LIST: {

      // Get the list of all airport pairs and date ranges for which
      // there are fares available.
      const std::string& lAirportPairDateListStr =
        simfqtService.list ();

      if (lAirportPairDateListStr.empty() == false) {
        std::cout << lAirportPairDateListStr << std::endl;
        STDAIR_LOG_DEBUG (lAirportPairDateListStr);

      } else {
        std::cerr << "There is no result for airport pairs and date ranges."
                  << "Make sure your input file is not empty."
                  << std::endl;
      }

      break;
    }

      // /////////////////////////////// Display /////////////////////////
    case Command_T::DISPLAY: {

      // If no parameters are entered by the user, keep default ones.
      if (lTokenListByReadline.empty() == true) {

        std::cout << "No parameters specified. Default paramaters '"
                  << lInteractiveOrigin << "-" << lInteractiveDestination
```

```
                   << "/" << lInteractiveDepartureDate
                   << "' are kept."
                   << std::endl;

      } else {

        // Find the best match corresponding to the given parameters.
        TokenList_T lTokenList =
          extractTokenListForOriDestDate (lTokenListByReadline);

        // Parse the best match, and give default values in case the
        // user does not specify all the parameters or does not
        // specify some of them correctly.
        parseFlightDateKey (lTokenList, lInteractiveOrigin,
                            lInteractiveDestination, lInteractiveDepartureDate);

      }

      // Check whether the selected airportpair-date is valid:
      // i.e. if there are corresponding fare rules.
      const bool isAirportPairDateValid =
        simfqtService.check (lInteractiveOrigin, lInteractiveDestination,
                             lInteractiveDepartureDate);

      if (isAirportPairDateValid == false) {
        std::ostringstream oFDKStr;
        oFDKStr << "The airport pair/departure date: "
                << lInteractiveOrigin << "-" << lInteractiveDestination
                << "/" << lInteractiveDepartureDate
                << " does not correpond to any fare rule.\n"
                << "Make sure it exists with the 'list' command.";
        std::cout << oFDKStr.str() << std::endl;
        STDAIR_LOG_ERROR (oFDKStr.str());

        break;
      }

      // Display the list of corresponding fare rules.
      std::cout << "List of fare rules for "
                << lInteractiveOrigin << "-"
                << lInteractiveDestination << "/"
                << lInteractiveDepartureDate
                << std::endl;

      const std::string& lFareRuleListStr =
        simfqtService.csvDisplay (lInteractiveOrigin,
                                  lInteractiveDestination,
                                  lInteractiveDepartureDate);

      assert (lFareRuleListStr.empty() == false);
      std::cout << lFareRuleListStr << std::endl;
      STDAIR_LOG_DEBUG (lFareRuleListStr);

      break;
    }

      // ////////////////////////////// Price ////////////////////////
    case Command_T::PRICE: {

      // If no parameters are entered by the user, keep default ones.
      if (lTokenListByReadline.empty() == true) {

        lInteractiveTravelSolution = lInteractiveTravelSolutionList.front();

        std::cout << "No parameters specified. Default booking request and defaul
t travel solution list are kept.\n"
                  << "Booking request: << "
                  << lInteractiveBookingRequest.display()  << " >>"
                  << "\nTravel Solution: << "
                  << lInteractiveTravelSolution.display() << " >>"
                  << "\n********** \n"
                  << "Fare quote"
                  << "\n**********"
                << std::endl;

        // Try to fareQuote the sample list of travel solutions.
        try {
        simfqtService.quotePrices (lInteractiveBookingRequest,
                                   lInteractiveTravelSolutionList);
        } catch (stdair::ObjectNotFoundException& E) {
          std::cerr << "The given travel solution corresponding to the given booking request can not be priced.\n"
                    << E.what()
                    << std::endl;
          break;
        }
      } else {

        // Find the best match corresponding to the given parameters.
        TokenList_T lTokenList =
          extractTokenListForTSAndBR (lTokenListByReadline);

        // Parse the best match, and give default values in case the
        // user does not specify all the parameters or does not
        // specify some of them correctly.
        stdair::BookingRequestStruct lFinalBookingRequest
          = parseTravelSolutionAndBookingRequestKey (lTokenList,
                                                     lInteractiveTravelSolutionList,
                                                     lInteractiveBookingRequest);



        assert (lInteractiveTravelSolutionList.size() >= 1);
        lInteractiveTravelSolution = lInteractiveTravelSolutionList.front();

        // Display the booking request and the first travel solution
        // before pricing.
        std::cout << "Booking request: << "
                  << lFinalBookingRequest.display()  << " >>"
                  << "\nTravel Solution: << "
                  << lInteractiveTravelSolution.display() << " >>"
                  << "\n********** \n"
                  << "Fare quote"
                  << "\n**********"
                << std::endl;

        // Try to fareQuote the sample list of travel solutions.
        try {
          simfqtService.quotePrices (lFinalBookingRequest,
                                     lInteractiveTravelSolutionList);
        } catch (stdair::ObjectNotFoundException& E) {
          std::cerr << "The given travel solution corresponding to the given booking request can not be priced.\n"
                    << E.what()
                    << std::endl;
          break;
        }
      }

      // Display the first travel solution after pricing:
      // one or more fare option have been added.
      lInteractiveTravelSolution = lInteractiveTravelSolutionList.front();
      std::cout << "Travel Solution: << "
```

```
                  << lInteractiveTravelSolution.display() << " >>\n"
                  << std::endl;

        break;
      }

        // ///////////////////////////// Default / No value ///////////////////////
      case Command_T::NOP: {
        break;
      }
      case Command_T::LAST_VALUE:
      default: {
        // DEBUG
        std::ostringstream oStr;
        oStr << "The '" << lUserInput << "' command is not yet understood.\n"
             << "Type help to have more information." << std::endl;

        STDAIR_LOG_DEBUG (oStr.str());
        std::cout << oStr.str() << std::endl;
      }
      }
    }

    // DEBUG
    STDAIR_LOG_DEBUG ("End of the session. Exiting.");
    std::cout << "End of the session. Exiting." << std::endl;

    // Close the Log outputFile
    logOutputFile.close();

    /*
      Note: as that program is not intended to be run on a server in
      production, it is better not to catch the exceptions. When it
      happens (that an exception is throwned), that way we get the
      call stack.
    */

    return 0;
  }
```

25.59 test/simfqt/FQTTestSuite.cpp File Reference

25.60 FQTTestSuite.cpp

```

// //////////////////////////////////////////////////////////////////////
// Import section
// //////////////////////////////////////////////////////////////////////
// STL
#include <sstream>
#include <fstream>
#include <string>
// Boost Unit Test Framework (UTF)
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MAIN
#define BOOST_TEST_MODULE FQTTestSuite
#include <boost/test/unit_test.hpp>
// StdAir
#include <stdair/basic/BasLogParams.hpp>
#include <stdair/basic/BasDBParams.hpp>
#include <stdair/basic/BasFileMgr.hpp>
#include <stdair/service/Logger.hpp>
#include <stdair/bom/TravelSolutionStruct.hpp>
#include <stdair/bom/BookingRequestStruct.hpp>
// SimFQT
#include <simfqt/SIMFQT_Service.hpp>
#include <simfqt/config/simfqt-paths.hpp>

namespace boost_utf = boost::unit_test;

struct UnitTestConfig {
  UnitTestConfig() {
    static std::ofstream _test_log ("FQTTestSuite_utfresults.xml");
    boost_utf::unit_test_log.set_stream (_test_log);
    boost_utf::unit_test_log.set_format (boost_utf::XML);
    boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
    //boost_utf::unit_test_log.set_threshold_level (boost_utf::log_successful_tes
ts);
  }

  ~UnitTestConfig() {
  }
};

// //////////////////////////////////////////////////////////////////////
void testFareQuoterHelper (const unsigned short iTestFlag,
                           const stdair::Filename_T iFareInputFilename,
                           const bool isBuiltin) {

  // Output log File
  std::ostringstream oStr;
  oStr << "FQTTestSuite_" << iTestFlag << ".log";
  const stdair::Filename_T lLogFilename (oStr.str());

  // Set the log parameters
  std::ofstream logOutputFile;
  // Open and clean the log outputfile
  logOutputFile.open (lLogFilename.c_str());
  logOutputFile.clear();

  // Initialise the SimFQT service object
  const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG,
                                         logOutputFile);

  // Initialise the Simfqt service object
  SIMFQT::SIMFQT_Service simfqtService (lLogParams);

  // Check wether or not a (CSV) input file should be read
  if (isBuiltin == true) {
```

```cpp

    // Build the default sample BOM tree (filled with fares) for Simfqt
    simfqtService.buildSampleBom();

  } else {

    // Build the BOM tree from parsing the fare input file
    SIMFQT::FareFilePath lFareFilePath (iFareInputFilename);
    simfqtService.parseAndLoad (lFareFilePath);
  }

  // Build a sample list of travel solutions and a booking request.
  stdair::TravelSolutionList_T lTravelSolutionList;
  simfqtService.buildSampleTravelSolutions (lTravelSolutionList);
  stdair::BookingRequestStruct lBookingRequest =
    simfqtService.buildBookingRequest();

  // Try to fareQuote the sample list of travel solutions
  simfqtService.quotePrices (lBookingRequest, lTravelSolutionList);

  // Close the log file
  logOutputFile.close();

}

// /////////////// Main: Unit Test Suite //////////////

// Set the UTF configuration (re-direct the output to a specific file)
BOOST_GLOBAL_FIXTURE (UnitTestConfig);

// Start the test suite
BOOST_AUTO_TEST_SUITE (master_test_suite)


BOOST_AUTO_TEST_CASE (simfqt_simple_pricing_test) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fare01.csv");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_NO_THROW (testFareQuoterHelper (0, lFareInputFilename, isBuiltin));

}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_01) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fareError01.csv");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (1, lFareInputFilename, isBuiltin),
                     SIMFQT::AirportPairNotFoundException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_02) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fareError02.csv");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (2, lFareInputFilename, isBuiltin),
                     SIMFQT::PosOrChannelNotFoundException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_03) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fareError03.csv");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (3, lFareInputFilename, isBuiltin),
                     SIMFQT::FlightDateNotFoundException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_04) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fareError04.csv");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (4, lFareInputFilename, isBuiltin),
                     SIMFQT::FlightTimeNotFoundException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_05) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fareError05.csv");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (5, lFareInputFilename, isBuiltin),
                     SIMFQT::FeaturesNotFoundException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_06) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fareError06.csv");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (6, lFareInputFilename, isBuiltin),
                     SIMFQT::AirlineNotFoundException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_07) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/fareError07.csv");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (7, lFareInputFilename, isBuiltin),
                     SIMFQT::FareFileParsingFailedException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_08) {

  // Input file name
  const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR "/missingFile.csv");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = false;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_THROW (testFareQuoterHelper (8, lFareInputFilename, isBuiltin),
                     SIMFQT::FareInputFileNotFoundException);
}

BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_09) {

  // Input file name
  const stdair::Filename_T lEmptyInputFilename (STDAIR_SAMPLE_DIR "/ ");

  // State whether the BOM tree should be built-in or parsed from an input file
  const bool isBuiltin = true;

  // Try to fareQuote the sample default list of travel solutions
  BOOST_CHECK_NO_THROW(testFareQuoterHelper (9, lEmptyInputFilename, isBuiltin));

}


// End the test suite
BOOST_AUTO_TEST_SUITE_END()


```