SimFQT

1.00.0

Generated by Doxygen 1.8.1.1

Tue Feb 12 2013 12:10:39

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

```
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.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

1. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

```
a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices
stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no
charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a
table of data to be supplied by an application program that uses
the facility, other than as an argument passed when the facility
is invoked, then you must make a good faith effort to ensure that,
in the event an application does not supply such function or
table, the facility still operates, and performs whatever part of
its purpose remains meaningful.

(For example, a function in a library to compute square roots has
```

```
a purpose that is entirely well-defined independent of the
application.  Therefore, Subsection 2d requires that any
application-supplied function or table used by this function must
be optional: if the application does not supply it, the square
root function must still compute square roots.)
```

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

1. 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.

1. 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.

1. 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.

1. 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.)

b) Use a suitable shared library mechanism for linking with the
Library.  A suitable mechanism is one that (1) uses at run time a
copy of the library already present on the user's computer system,
rather than copying library functions into the executable, and (2)
will operate properly with a modified version of the library, if
the user installs one, as long as the modified version is
interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at
least three years, to give the same user the materials
specified in Subsection 6a, above, for a charge no more
than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy
from a designated place, offer equivalent access to copy the above
specified materials from the same place.

e) Verify that the user has already received a copy of these
materials or that you have already sent this user a copy.
```

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

1. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

   ```
   a) Accompany the combined library with a copy of the same work
   based on the Library, uncombined with any other library
   facilities.  This must be distributed under the terms of the
   Sections above.

   b) Give prominent notice with the combined library of the fact
   that part of it is a work based on the Library, and explaining
   where to find the accompanying uncombined form of the same work.
   ```

1. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and

will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

1. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

1. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

1. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

1. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

1. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

1. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

4.3.1 NO WARRANTY

1. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

1. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

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 <tt>'CMakeLists.txt'</tt> is used to create the \c '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=P-
REFIX', 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=P-
REFIX' 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=P-
REFIX' 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/samples
-- 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_test_f
--   - Boost required libraries ... : optimized;/usr/lib/libboost_iostreams-mt.so;debug;/usr/lib/libboost_iost
--
-- * 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 Introdution

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/aclocal'. 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:

```
// Fares: fare ID; OriginCity; DestinationCity; TripType; DateRangeStart;
       DateRangeEnd; DepartureTimeRangeStart; DepartureTimeRangeEnd; POS; CabinCode;
       Channel; AdvancePurchase; SaturdayNight; ChangeFees; NonRefundable; MinimumStay; Price;
       nb Segments
// 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:

```
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:

```
    void parseAndLoad (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(const bool isForCRS =
  false);
```

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:

```
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:

```
void quotePrices (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 pair:
  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-xxx-
  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) and
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 request
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), to
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 Introdution

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;
const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);
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,
   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;
       LHR, SYD; 21:45. A corresponding fare option for the 'BA Y' class is: Class
      path: Y; 450 EUR; conditions: 1 1 1
[D]../../../simfqt/service/SIMFQT_Service.cpp:352: Fare Quote retrieving: 0.001
      403 - 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 :

```
// Fares: fare ID; OriginCity; DestinationCity; TripType; DateRangeStart;
       DateRangeEnd; DepartureTimeRangeStart; DepartureTimeRangeEnd; POS; CabinCode;
       Channel; AdvancePurchase; SaturdayNight; ChangeFees; NonRefundable; MinimumStay; Price;
       nb Segments
// 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:

```
// Fares: fare ID; OriginCity; DestinationCity; TripType; DateRangeStart;
       DateRangeEnd; DepartureTimeRangeStart; DepartureTimeRangeEnd; POS; CabinCode;
       Channel; AdvancePurchase; SaturdayNight; ChangeFees; NonRefundable; MinimumStay; Price;
       nb Segments
// 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:

```
// Fares: fare ID; OriginCity; DestinationCity; TripType; DateRangeStart;
       DateRangeEnd; DepartureTimeRangeStart; DepartureTimeRangeEnd; POS; CabinCode;
       Channel; AdvancePurchase; SaturdayNight; ChangeFees; NonRefundable; MinimumStay; Price;
       nb Segments
// 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_tests);
  }

  ~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.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()

/*!
```

17 Namespace Index

17.1 Namespace List

Here is a list of all namespaces with brief descriptions:

18 Class Index

18.1 Class Hierarchy

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

19 Class Index

19.1 Class List

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

20 File Index

20.1 File List

Here is a list of all files with brief descriptions:

21 Namespace Documentation

21.1 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"

21.1.1 Typedef Documentation

21.1.1.1 typedef unsigned int SIMFQT::FareQuoteID_T

ID for the Fare Quote system.

Definition at line 143 of file SIMFQT_Types.hpp.

21.1.1.2 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.

21.1.2 Variable Documentation

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

Default ID for the SIMFQT_Service.

Definition at line 10 of file BasConst.cpp.

21.2 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

21.2.1 Variable Documentation

21.2.1.1 stdair::int1_p_t SIMFQT::FareParserHelper::int1_p

Namespaces. 1-digit-integer parser

Definition at line 447 of file FareParserHelper.cpp.

21.2.1.2 stdair::uint2_p_t SIMFQT::FareParserHelper::uint2_p

2-digit-integer parser

Definition at line 450 of file FareParserHelper.cpp.

21.2.1.3 stdair::uint4_p_t SIMFQT::FareParserHelper::uint4_p

4-digit-integer parser

Definition at line 453 of file FareParserHelper.cpp.

21.2.1.4 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().

21.2.1.5 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().

21.2.1.6 stdair::minute_p_t SIMFQT::FareParserHelper::minute_p

Definition at line 460 of file FareParserHelper.cpp.

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

21.2.1.7 stdair::second_p_t SIMFQT::FareParserHelper::second_p

Definition at line 461 of file FareParserHelper.cpp.

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

21.2.1.8 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().

21.2.1.9 stdair::month_p_t SIMFQT::FareParserHelper::month_p

Definition at line 465 of file FareParserHelper.cpp.

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

21.2.1.10 stdair::day_p_t SIMFQT::FareParserHelper::day_p

Definition at line 466 of file FareParserHelper.cpp.

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

21.3 stdair Namespace Reference

Forward declarations.

21.3.1 Detailed Description

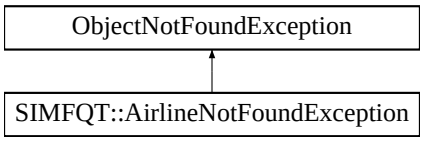
Forward declarations.

22 Class Documentation

22.1 SIMFQT::AirlineNotFoundException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`

Inheritance diagram for SIMFQT::AirlineNotFoundException:



Public Member Functions

- AirlineNotFoundException (const std::string &iWhat)

22.1.1 Detailed Description

The airline can not be found.

Definition at line 99 of file SIMFQT_Types.hpp.

22.1.2 Constructor & Destructor Documentation

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

Constructor.

Definition at line 104 of file SIMFQT_Types.hpp.

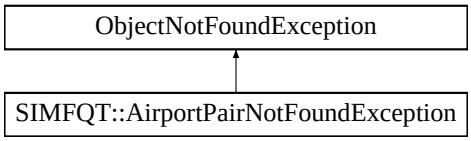
The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

22.2 SIMFQT::AirportPairNotFoundException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`

Inheritance diagram for SIMFQT::AirportPairNotFoundException:



Public Member Functions

- AirportPairNotFoundException (const std::string &iWhat)

22.2.1 Detailed Description

The given airport pair can not be found.

Definition at line 39 of file SIMFQT_Types.hpp.

22.2.2 Constructor & Destructor Documentation

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

Constructor.

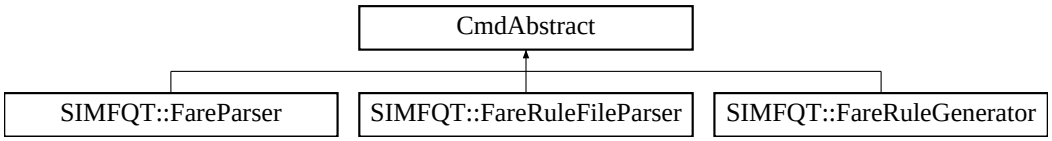
Definition at line 44 of file SIMFQT_Types.hpp.

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

- simfqt/SIMFQT_Types.hpp

22.3 CmdAbstract Class Reference

Inheritance diagram for CmdAbstract:



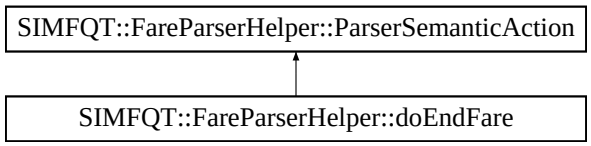
The documentation for this class was generated from the following file:

- simfqt/command/FareRuleGenerator.hpp

22.4 SIMFQT::FareParserHelper::doEndFare Struct Reference

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

Inheritance diagram for 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

22.4.1 Detailed Description

Mark the end of the fare-rule parsing.

Definition at line 230 of file FareParserHelper.hpp.

22.4.2 Constructor & Destructor Documentation

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

Actor Constructor.

Definition at line 420 of file FareParserHelper.cpp.

22.4.3 Member Function Documentation

22.4.3.1 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 SIMFQT::FareRuleStruct::describe().

22.4.4 Member Data Documentation

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

Actor Specific Context.

Definition at line 238 of file FareParserHelper.hpp.

Referenced by operator()().

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

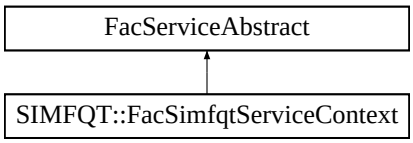
Referenced by SIMFQT::FareParserHelper::storeFareId::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()(),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

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

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

22.5 FacServiceAbstract Class Reference

Inheritance diagram for FacServiceAbstract:



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

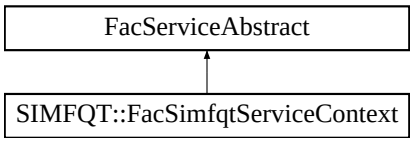
- simfqt/factory/FacSimfqtServiceContext.hpp

22.6 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 ()

22.6.1 Detailed Description

Factory for the service context.

Definition at line 22 of file FacSimfqtServiceContext.hpp.

22.6.2 Constructor & Destructor Documentation

22.6.2.1 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.

22.6.2.2 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().

22.6.3 Member Function Documentation

22.6.3.1 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().

22.6.3.2 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.

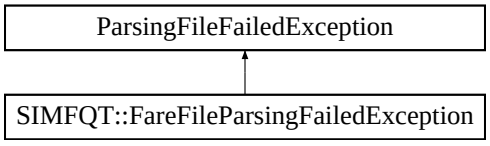
The documentation for this class was generated from the following files:

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

22.7 SIMFQT::FareFileParsingFailedException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`

Inheritance diagram for SIMFQT::FareFileParsingFailedException:



Public Member Functions

- FareFileParsingFailedException (const std::string &iWhat)

22.7.1 Detailed Description

The fare input file can not be parsed.

Definition at line 26 of file SIMFQT_Types.hpp.

22.7.2 Constructor & Destructor Documentation

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

Constructor.

Definition at line 32 of file SIMFQT_Types.hpp.

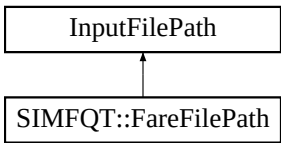
The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

22.8 SIMFQT::FareFilePath Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`

Inheritance diagram for SIMFQT::FareFilePath:



Public Member Functions

- FareFilePath (const stdair::Filename_T &iFilename)

22.8.1 Detailed Description

Fare input file.

Definition at line 130 of file SIMFQT_Types.hpp.

22.8.2 Constructor & Destructor Documentation

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

Constructor.

Definition at line 135 of file SIMFQT_Types.hpp.

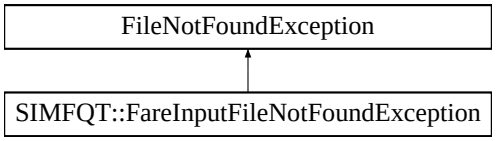
The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

22.9 SIMFQT::FareInputFileNotFoundException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`

Inheritance diagram for SIMFQT::FareInputFileNotFoundException:



Public Member Functions

- FareInputFileNotFoundException (const std::string &iWhat)

22.9.1 Detailed Description

The fare input file can not be found.

Definition at line 111 of file SIMFQT_Types.hpp.

22.9.2 Constructor & Destructor Documentation

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

Constructor.

Definition at line 116 of file SIMFQT_Types.hpp.

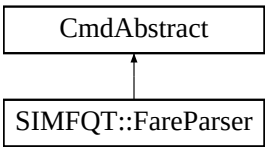
The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

22.10 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 &)

22.10.1 Detailed Description

Class wrapping the parser entry point.

Definition at line 23 of file FareParser.hpp.

22.10.2 Member Function Documentation

22.10.2.1 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

22.11 SIMFQT::FareQuoter Class Reference

Command wrapping the pricing request process.

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

Friends

- class SIMFQT_Service

22.11.1 Detailed Description

Command wrapping the pricing request process.

Definition at line 29 of file FareQuoter.hpp.

22.11.2 Friends And Related Function Documentation

22.11.2.1 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.

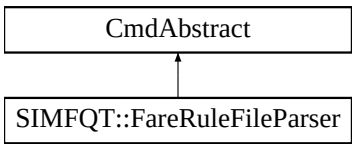
The documentation for this class was generated from the following files:

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

22.12 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 ()

22.12.1 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.

22.12.2 Constructor & Destructor Documentation

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

Constructor.

Definition at line 645 of file FareParserHelper.cpp.

22.12.3 Member Function Documentation

22.12.3.1 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().

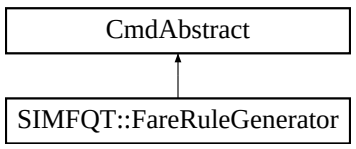
The documentation for this class was generated from the following files:

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

22.13 SIMFQT::FareRuleGenerator Class Reference

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

Inheritance diagram for SIMFQT::FareRuleGenerator:



Friends

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

22.13.1 Detailed Description

Class handling the generation / instantiation of the Fare BOM.

Definition at line 33 of file FareRuleGenerator.hpp.

22.13.2 Friends And Related Function Documentation

22.13.2.1 friend class FareFileParser [friend]

Definition at line 38 of file FareRuleGenerator.hpp.

22.13.2.2 friend struct FareParserHelper::doEndFare [friend]

Definition at line 39 of file FareRuleGenerator.hpp.

22.13.2.3 friend class FareParser [friend]

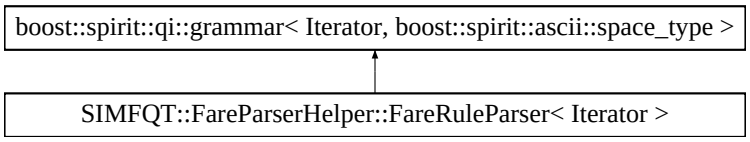
Definition at line 40 of file FareRuleGenerator.hpp.

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

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

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

Inheritance diagram for 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

22.14.1 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.

22.14.2 Constructor & Destructor Documentation

22.14.2.1 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.

22.14.3 Member Data Documentation

22.14.3.1 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().

22.14.3.2 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().

22.14.3.3 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().

22.14.3.4 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().

22.14.3.5 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().

22.14.3.6 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().

22.14.3.7 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().

22.14.3.8 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().

22.14.3.9 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().

22.14.3.10 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().

22.14.3.11 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().

22.14.3.12 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().

22.14.3.13 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().

22.14.3.14 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().

22.14.3.15 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().

22.14.3.16 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().

22.14.3.17 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().

22.14.3.18 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().

22.14.3.19 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().

22.14.3.20 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().

22.14.3.21 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().

22.14.3.22 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().

22.14.3.23 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().

22.14.3.24 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().

22.14.3.25 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().

22.14.3.26 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().

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

Definition at line 631 of file FareParserHelper.cpp.

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

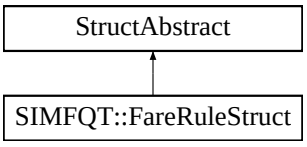
The documentation for this struct was generated from the following file:

- simfqt/command/FareParserHelper.cpp

22.15 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

22.15.1 Detailed Description

Utility Structure for the parsing of fare-rule structures.

Definition at line 21 of file FareRuleStruct.hpp.

22.15.2 Constructor & Destructor Documentation

22.15.2.1 SIMFQT::FareRuleStruct::FareRuleStruct ( )

Default constructor.

Definition at line 17 of file FareRuleStruct.cpp.

22.15.3 Member Function Documentation

22.15.3.1 SIMFQT::FareQuoteID_T SIMFQT::FareRuleStruct::getFareID ( ) const [inline]

Get the fare ID.

Definition at line 30 of file FareRuleStruct.hpp.

22.15.3.2 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()().

22.15.3.3 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()().

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

Get the trip type.

Definition at line 45 of file FareRuleStruct.hpp.

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

Get the date range start.

Definition at line 50 of file FareRuleStruct.hpp.

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

Get the date range end.

Definition at line 55 of file FareRuleStruct.hpp.

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

Get the time range start.

Definition at line 60 of file FareRuleStruct.hpp.

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

Get the time range end.

Definition at line 65 of file FareRuleStruct.hpp.

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

Get the cabin code.

Definition at line 70 of file FareRuleStruct.hpp.

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

Get the point-of-sale.

Definition at line 75 of file FareRuleStruct.hpp.

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

Get the channel.

Definition at line 80 of file FareRuleStruct.hpp.

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

Get the advance purchase.

Definition at line 85 of file FareRuleStruct.hpp.

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

Get the saturday stay option.

Definition at line 90 of file FareRuleStruct.hpp.

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

Get the change fees.

Definition at line 95 of file FareRuleStruct.hpp.

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

Get the refundable option.

Definition at line 100 of file FareRuleStruct.hpp.

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

Get the minimum stay.

Definition at line 105 of file FareRuleStruct.hpp.

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

Get the fare.

Definition at line 110 of file FareRuleStruct.hpp.

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

Get the airline code.

Definition at line 115 of file FareRuleStruct.hpp.

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

Get the class code.

Definition at line 120 of file FareRuleStruct.hpp.

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

Get the size of the airline code list.

Definition at line 125 of file FareRuleStruct.hpp.

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

Get the size of the class code list.

Definition at line 130 of file FareRuleStruct.hpp.

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

Get the airline code list.

Definition at line 135 of file FareRuleStruct.hpp.

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

Get the class code list.

Definition at line 140 of file FareRuleStruct.hpp.

22.15.3.24 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::storeDateRangeStart::operator()(), and SIMFQT::FareParserHelper::storeDateRangeEnd::operator()().

22.15.3.25 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::storeStartRangeTime::operator()(), and SIMFQT::FareParserHelper::storeEndRangeTime::operator()().

22.15.3.26 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()().

22.15.3.27 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()().

22.15.3.28 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()().

22.15.3.29 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()().

22.15.3.30 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()().

22.15.3.31 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()().

22.15.3.32 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()().

22.15.3.33 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()().

22.15.3.34 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()().

22.15.3.35 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()().

22.15.3.36 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()().

22.15.3.37 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()().

22.15.3.38 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()().

22.15.3.39 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()().

22.15.3.40 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()().

22.15.3.41 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()().

22.15.3.42 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()().

22.15.3.43 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()().

22.15.3.44 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()().

22.15.3.45 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()().

22.15.3.46 void SIMFQT::FareRuleStruct::clearAirlineCodeList ( ) [inline]

Empty the airline code list.

Definition at line 253 of file FareRuleStruct.hpp.

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

22.15.3.47 void SIMFQT::FareRuleStruct::clearClassCodeList ( ) [inline]

Empty the class code list.

Definition at line 258 of file FareRuleStruct.hpp.

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

22.15.3.48 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()().

22.15.3.49 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()().

22.15.4 Member Data Documentation

22.15.4.1 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().

22.15.4.2 stdair::month_t SIMFQT::FareRuleStruct::_itMonth

Definition at line 276 of file FareRuleStruct.hpp.

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

22.15.4.3 stdair::day_t SIMFQT::FareRuleStruct::_itDay

Definition at line 277 of file FareRuleStruct.hpp.

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

22.15.4.4 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().

22.15.4.5 stdair::minute_t SIMFQT::FareRuleStruct::_itMinutes

Definition at line 281 of file FareRuleStruct.hpp.

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

22.15.4.6 stdair::second_t SIMFQT::FareRuleStruct::_itSeconds

Definition at line 282 of file FareRuleStruct.hpp.

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

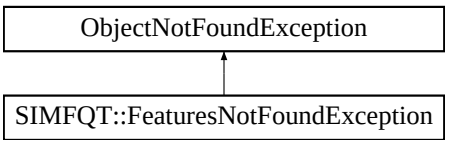
The documentation for this struct was generated from the following files:

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

22.16 SIMFQT::FeaturesNotFoundException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`

Inheritance diagram for SIMFQT::FeaturesNotFoundException:



Public Member Functions

- FeaturesNotFoundException (const std::string &iWhat)

22.16.1 Detailed Description

The fare features can not be found.

Definition at line 87 of file SIMFQT_Types.hpp.

22.16.2 Constructor & Destructor Documentation

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

Constructor.

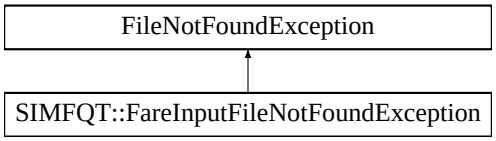
Definition at line 92 of file SIMFQT_Types.hpp.

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

- simfqt/SIMFQT_Types.hpp

22.17 FileNotFoundException Class Reference

Inheritance diagram for FileNotFoundException:



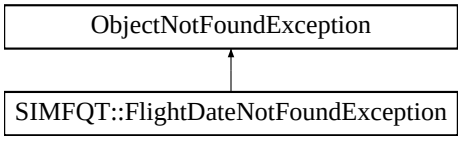
The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

22.18 SIMFQT::FlightDateNotFoundException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`

Inheritance diagram for SIMFQT::FlightDateNotFoundException:



Public Member Functions

- FlightDateNotFoundException (const std::string &iWhat)

22.18.1 Detailed Description

The departure date of the flight can not be found.

Definition at line 63 of file SIMFQT_Types.hpp.

22.18.2 Constructor & Destructor Documentation

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

Constructor.

Definition at line 68 of file SIMFQT_Types.hpp.

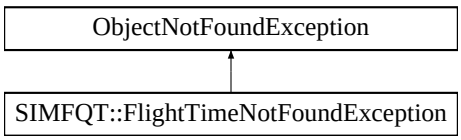
The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

22.19 SIMFQT::FlightTimeNotFoundException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`

Inheritance diagram for SIMFQT::FlightTimeNotFoundException:



Public Member Functions

- FlightTimeNotFoundException (const std::string &iWhat)

22.19.1 Detailed Description

The departure time of the flight can not be found.

Definition at line 75 of file SIMFQT_Types.hpp.

22.19.2 Constructor & Destructor Documentation

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

Constructor.

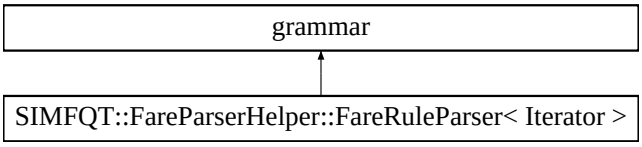
Definition at line 80 of file SIMFQT_Types.hpp.

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

- simfqt/SIMFQT_Types.hpp

22.20 grammar Class Reference

Inheritance diagram for grammar:

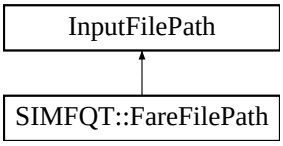


The documentation for this class was generated from the following file:

- simfqt/command/FareParserHelper.cpp

22.21 InputFilePath Class Reference

Inheritance diagram for InputFilePath:



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

- simfqt/SIMFQT_Types.hpp

22.22 ObjectNotFoundException Class Reference

Inheritance diagram for ObjectNotFoundException:

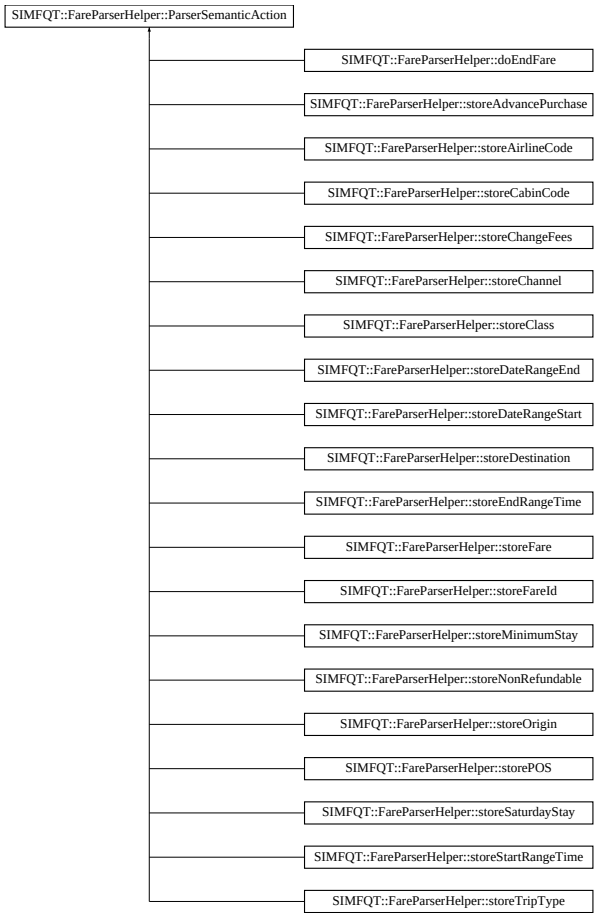
The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

22.23 SIMFQT::FareParserHelper::ParserSemanticAction Struct Reference

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

Inheritance diagram for SIMFQT::FareParserHelper::ParserSemanticAction:



Public Member Functions

- ParserSemanticAction (FareRuleStruct &)

Public Attributes

- FareRuleStruct & _fareRule

22.23.1 Detailed Description

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

Definition at line 31 of file FareParserHelper.hpp.

22.23.2 Constructor & Destructor Documentation

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

Actor Constructor.

Definition at line 30 of file FareParserHelper.cpp.

22.23.3 Member Data Documentation

22.23.3.1 FareRuleStruct& SIMFQT::FareParserHelper::ParserSemanticAction::_fareRule

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

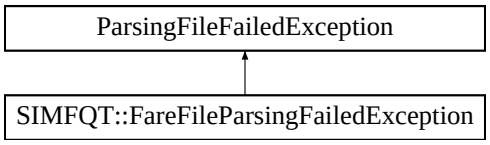
Referenced by SIMFQT::FareParserHelper::storeFareId::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()(),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

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

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

22.24 ParsingFileFailedException Class Reference

Inheritance diagram for ParsingFileFailedException:



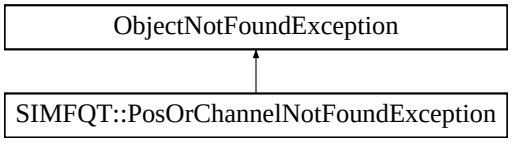
The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

22.25 SIMFQT::PosOrChannelNotFoundException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`

Inheritance diagram for SIMFQT::PosOrChannelNotFoundException:



Public Member Functions

- PosOrChannelNotFoundException (const std::string &iWhat)

22.25.1 Detailed Description

The given POS/channel can not be found.

Definition at line 51 of file SIMFQT_Types.hpp.

22.25.2 Constructor & Destructor Documentation

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

Constructor.

Definition at line 56 of file SIMFQT_Types.hpp.

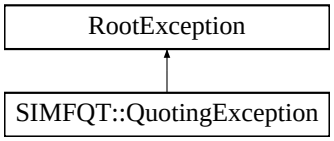
The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

22.26 SIMFQT::QuotingException Class Reference

`#include <simfqt/SIMFQT_Types.hpp>`

Inheritance diagram for SIMFQT::QuotingException:



22.26.1 Detailed Description

The pricing operation fails.

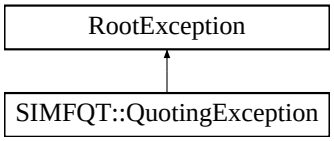
Definition at line 123 of file SIMFQT_Types.hpp.

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

- simfqt/SIMFQT_Types.hpp

22.27 RootException Class Reference

Inheritance diagram for RootException:

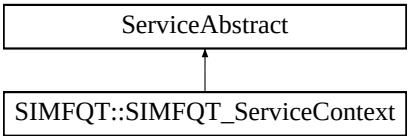


The documentation for this class was generated from the following file:

- simfqt/SIMFQT_Types.hpp

22.28 ServiceAbstract Class Reference

Inheritance diagram for ServiceAbstract:



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

- simfqt/service/SIMFQT_ServiceContext.hpp

22.29 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

22.29.1 Detailed Description

Interface for the SIMFQT Services.

Definition at line 32 of file SIMFQT_Service.hpp.

22.29.2 Constructor & Destructor Documentation

22.29.2.1 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.

22.29.2.2 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.

22.29.2.3 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.

22.29.2.4 SIMFQT::SIMFQT_Service::~SIMFQT_Service ( )

Destructor.

Definition at line 94 of file SIMFQT_Service.cpp.

22.29.3 Member Function Documentation

22.29.3.1 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().

22.29.3.2 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().

22.29.3.3 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().

22.29.3.4 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().

22.29.3.5 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().

22.29.3.6 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

TravelSolution-List_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().

22.29.3.7 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::Booking-RequestStruct&	Booking request.
stdair::Travel-SolutionList_T&	List of travel solution.

Definition at line 480 of file SIMFQT_Service.cpp.

Referenced by main().

22.29.3.8 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().

22.29.3.9 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.

22.29.3.10 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.

22.29.3.11 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.

22.29.3.12 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:

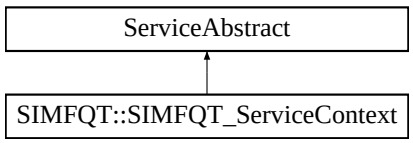
- simfqt/SIMFQT_Service.hpp
- simfqt/service/SIMFQT_Service.cpp

22.30 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

22.30.1 Detailed Description

Class holding the context of the SimFQT services.

Definition at line 25 of file SIMFQT_ServiceContext.hpp.

22.30.2 Friends And Related Function Documentation

22.30.2.1 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.

22.30.2.2 friend class FacSimfqtServiceContext [friend]

Definition at line 32 of file SIMFQT_ServiceContext.hpp.

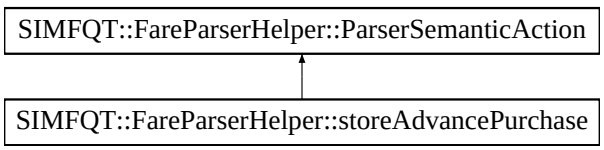
The documentation for this class was generated from the following files:

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

22.31 SIMFQT::FareParserHelper::storeAdvancePurchase Struct Reference

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

Inheritance diagram for 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

22.31.1 Detailed Description

Store the parsed advance purchase days.

Definition at line 150 of file FareParserHelper.hpp.

22.31.2 Constructor & Destructor Documentation

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

Actor Constructor.

Definition at line 254 of file FareParserHelper.cpp.

22.31.3 Member Function Documentation

22.31.3.1 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().

22.31.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

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

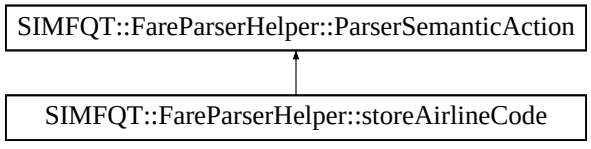
The documentation for this struct was generated from the following files:

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

22.32 SIMFQT::FareParserHelper::storeAirlineCode Struct Reference

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

Inheritance diagram for 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

22.32.1 Detailed Description

Store the parsed airline code.

Definition at line 210 of file FareParserHelper.hpp.

22.32.2 Constructor & Destructor Documentation

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

Actor Constructor.

Definition at line 378 of file FareParserHelper.cpp.

22.32.3 Member Function Documentation

22.32.3.1 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().

22.32.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

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

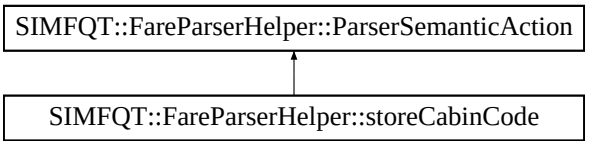
The documentation for this struct was generated from the following files:

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

22.33 SIMFQT::FareParserHelper::storeCabinCode Struct Reference

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

Inheritance diagram for 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

22.33.1 Detailed Description

Store the cabin code.

Definition at line 130 of file FareParserHelper.hpp.

22.33.2 Constructor & Destructor Documentation

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

Actor Constructor.

Definition at line 212 of file FareParserHelper.cpp.

22.33.3 Member Function Documentation

22.33.3.1 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().

22.33.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

Referenced by SIMFQT::FareParserHelper::storeFareId::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), operator()(), SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeAdvance-

Purchase::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), and SIMFQT::FareParserHelper::doEndFare::operator()().

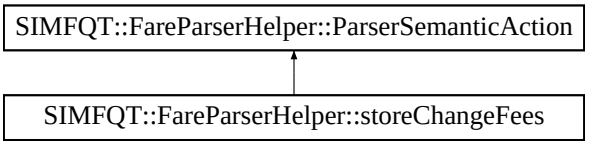
The documentation for this struct was generated from the following files:

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

22.34 SIMFQT::FareParserHelper::storeChangeFees Struct Reference

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

Inheritance diagram for 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

22.34.1 Detailed Description

Store the parsed change fees.

Definition at line 170 of file FareParserHelper.hpp.

22.34.2 Constructor & Destructor Documentation

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

Actor Constructor.

Definition at line 295 of file FareParserHelper.cpp.

22.34.3 Member Function Documentation

22.34.3.1 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().

22.34.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

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

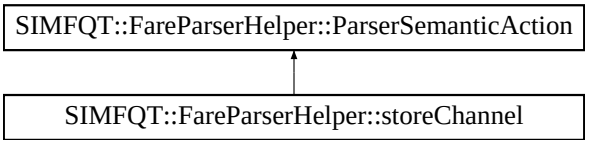
The documentation for this struct was generated from the following files:

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

22.35 SIMFQT::FareParserHelper::storeChannel Struct Reference

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

Inheritance diagram for 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

22.35.1 Detailed Description

Store the channel distribution.

Definition at line 140 of file FareParserHelper.hpp.

22.35.2 Constructor & Destructor Documentation

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

Actor Constructor.

Definition at line 233 of file FareParserHelper.cpp.

22.35.3 Member Function Documentation

22.35.3.1 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().

22.35.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

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

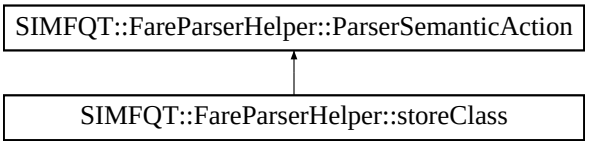
The documentation for this struct was generated from the following files:

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

22.36 SIMFQT::FareParserHelper::storeClass Struct Reference

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

Inheritance diagram for 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

22.36.1 Detailed Description

Store the parsed class code.

Definition at line 220 of file FareParserHelper.hpp.

22.36.2 Constructor & Destructor Documentation

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

Actor Constructor.

Definition at line 396 of file FareParserHelper.cpp.

22.36.3 Member Function Documentation

22.36.3.1 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().

22.36.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

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

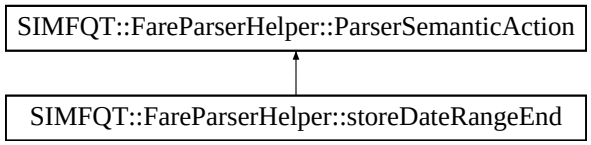
The documentation for this struct was generated from the following files:

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

22.37 SIMFQT::FareParserHelper::storeDateRangeEnd Struct Reference

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

Inheritance diagram for 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

22.37.1 Detailed Description

Store the parsed end of the date range.

Definition at line 90 of file FareParserHelper.hpp.

22.37.2 Constructor & Destructor Documentation

22.37.2.1 SIMFQT::FareParserHelper::storeDateRangeEnd::storeDateRangeEnd (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 129 of file FareParserHelper.cpp.

22.37.3 Member Function Documentation

22.37.3.1 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().

22.37.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

Referenced by SIMFQT::FareParserHelper::storeFareId::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(),

SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), and SIMFQT::FareParserHelper::doEndFare::operator()().

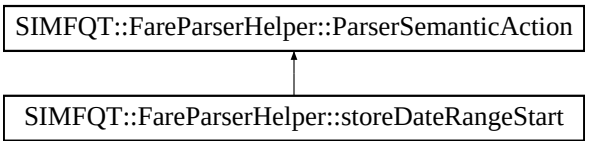
The documentation for this struct was generated from the following files:

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

22.38 SIMFQT::FareParserHelper::storeDateRangeStart Struct Reference

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

Inheritance diagram for 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

22.38.1 Detailed Description

Store the parsed start of the date range.

Definition at line 80 of file FareParserHelper.hpp.

22.38.2 Constructor & Destructor Documentation

22.38.2.1 SIMFQT::FareParserHelper::storeDateRangeStart::storeDateRangeStart (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 113 of file FareParserHelper.cpp.

22.38.3 Member Function Documentation

22.38.3.1 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().

22.38.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

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

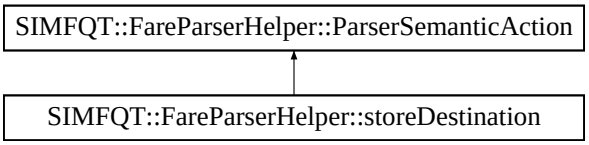
The documentation for this struct was generated from the following files:

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

22.39 SIMFQT::FareParserHelper::storeDestination Struct Reference

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

Inheritance diagram for 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

22.39.1 Detailed Description

Store the parsed destination.

Definition at line 59 of file FareParserHelper.hpp.

22.39.2 Constructor & Destructor Documentation

22.39.2.1 SIMFQT::FareParserHelper::storeDestination::storeDestination (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 75 of file FareParserHelper.cpp.

22.39.3 Member Function Documentation

22.39.3.1 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().

22.39.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

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

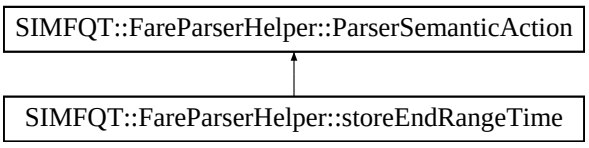
The documentation for this struct was generated from the following files:

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

22.40 SIMFQT::FareParserHelper::storeEndRangeTime Struct Reference

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

Inheritance diagram for 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

22.40.1 Detailed Description

Store the parsed end range time.

Definition at line 110 of file FareParserHelper.hpp.

22.40.2 Constructor & Destructor Documentation

22.40.2.1 SIMFQT::FareParserHelper::storeEndRangeTime::storeEndRangeTime (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 168 of file FareParserHelper.cpp.

22.40.3 Member Function Documentation

22.40.3.1 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().

22.40.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

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

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

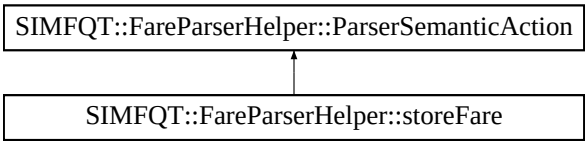
- simfqt/command/FareParserHelper.hpp

- simfqt/command/FareParserHelper.cpp

22.41 SIMFQT::FareParserHelper::storeFare Struct Reference

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

Inheritance diagram for 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

22.41.1 Detailed Description

Store the parsed fare value.

Definition at line 200 of file FareParserHelper.hpp.

22.41.2 Constructor & Destructor Documentation

22.41.2.1 SIMFQT::FareParserHelper::storeFare::storeFare (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 362 of file FareParserHelper.cpp.

22.41.3 Member Function Documentation

22.41.3.1 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().

22.41.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

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

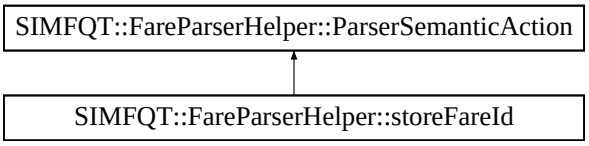
The documentation for this struct was generated from the following files:

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

22.42 SIMFQT::FareParserHelper::storeFareId Struct Reference

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

Inheritance diagram for 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

22.42.1 Detailed Description

Store the parsed fare Id.

Definition at line 39 of file FareParserHelper.hpp.

22.42.2 Constructor & Destructor Documentation

22.42.2.1 SIMFQT::FareParserHelper::storeFareId::storeFareId (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 36 of file FareParserHelper.cpp.

22.42.3 Member Function Documentation

22.42.3.1 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().

22.42.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

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

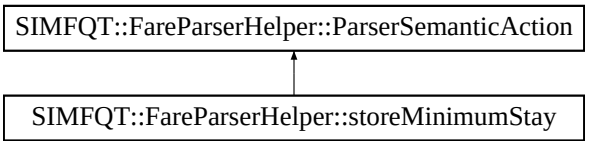
The documentation for this struct was generated from the following files:

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

22.43 SIMFQT::FareParserHelper::storeMinimumStay Struct Reference

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

Inheritance diagram for 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

22.43.1 Detailed Description

Store the parsed minimum stay.

Definition at line 190 of file FareParserHelper.hpp.

22.43.2 Constructor & Destructor Documentation

22.43.2.1 SIMFQT::FareParserHelper::storeMinimumStay::storeMinimumStay (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 346 of file FareParserHelper.cpp.

22.43.3 Member Function Documentation

22.43.3.1 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().

22.43.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

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

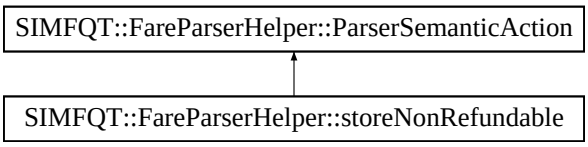
The documentation for this struct was generated from the following files:

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

22.44 SIMFQT::FareParserHelper::storeNonRefundable Struct Reference

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

Inheritance diagram for 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

22.44.1 Detailed Description

Store the parsed refundable option

Definition at line 180 of file FareParserHelper.hpp.

22.44.2 Constructor & Destructor Documentation

22.44.2.1 SIMFQT::FareParserHelper::storeNonRefundable::storeNonRefundable (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 321 of file FareParserHelper.cpp.

22.44.3 Member Function Documentation

22.44.3.1 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().

22.44.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

Referenced by SIMFQT::FareParserHelper::storeFareId::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()(), SIMFQT::FareParserHelper::storeStartRangeTime::operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(), SIMFQT::FareParserHelper::storeChannel::operator()(),

SIMFQT::FareParserHelper::storeAdvancePurchase::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::storeChangeFees::operator()(), operator()(), SIMFQT::FareParserHelper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParserHelper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), and SIMFQT::FareParserHelper::doEndFare::operator()().

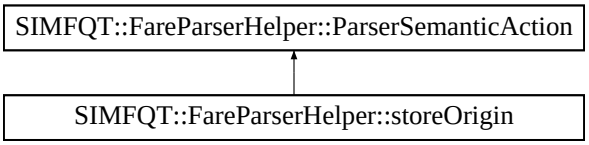
The documentation for this struct was generated from the following files:

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

22.45 SIMFQT::FareParserHelper::storeOrigin Struct Reference

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

Inheritance diagram for 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

22.45.1 Detailed Description

Store the parsed origin.

Definition at line 49 of file FareParserHelper.hpp.

22.45.2 Constructor & Destructor Documentation

22.45.2.1 SIMFQT::FareParserHelper::storeOrigin::storeOrigin (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 59 of file FareParserHelper.cpp.

22.45.3 Member Function Documentation

22.45.3.1 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().

22.45.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

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

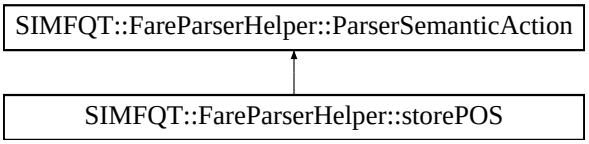
The documentation for this struct was generated from the following files:

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

22.46 SIMFQT::FareParserHelper::storePOS Struct Reference

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

Inheritance diagram for 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

22.46.1 Detailed Description

Store the parsed customer point_of_sale.

Definition at line 120 of file FareParserHelper.hpp.

22.46.2 Constructor & Destructor Documentation

22.46.2.1 SIMFQT::FareParserHelper::storePOS::storePOS (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 186 of file FareParserHelper.cpp.

22.46.3 Member Function Documentation

22.46.3.1 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().

22.46.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

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

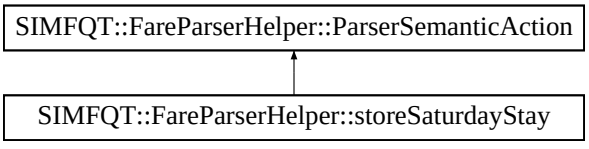
The documentation for this struct was generated from the following files:

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

22.47 SIMFQT::FareParserHelper::storeSaturdayStay Struct Reference

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

Inheritance diagram for 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

22.47.1 Detailed Description

Store the parsed saturday night.

Definition at line 160 of file FareParserHelper.hpp.

22.47.2 Constructor & Destructor Documentation

22.47.2.1 SIMFQT::FareParserHelper::storeSaturdayStay::storeSaturdayStay (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 270 of file FareParserHelper.cpp.

22.47.3 Member Function Documentation

22.47.3.1 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().

22.47.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

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

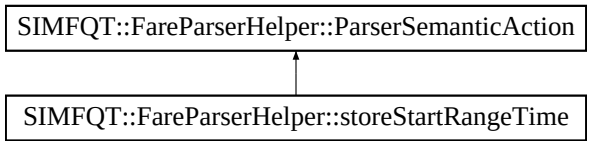
The documentation for this struct was generated from the following files:

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

22.48 SIMFQT::FareParserHelper::storeStartRangeTime Struct Reference

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

Inheritance diagram for 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

22.48.1 Detailed Description

Store the parsed start range time.

Definition at line 100 of file FareParserHelper.hpp.

22.48.2 Constructor & Destructor Documentation

22.48.2.1 SIMFQT::FareParserHelper::storeStartRangeTime::storeStartRangeTime (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 150 of file FareParserHelper.cpp.

22.48.3 Member Function Documentation

22.48.3.1 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().

22.48.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

Referenced by SIMFQT::FareParserHelper::storeFareId::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), SIMFQT::FareParserHelper::storeTripType::operator()(), SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()(), operator()(), SIMFQT::FareParserHelper::storeEndRangeTime::operator()(), SIMFQT::FareParserHelper::storePOS::operator()(), SIMFQT::FareParserHelper::storeCabinCode::operator()(),

SIMFQT::FareParserHelper::storeChannel::operator()(), SIMFQT::FareParserHelper::storeAdvancePurchase-::operator()(), SIMFQT::FareParserHelper::storeSaturdayStay::operator()(), SIMFQT::FareParserHelper::store-ChangeFees::operator()(), SIMFQT::FareParserHelper::storeNonRefundable::operator()(), SIMFQT::FareParser-Helper::storeMinimumStay::operator()(), SIMFQT::FareParserHelper::storeFare::operator()(), SIMFQT::FareParser-Helper::storeAirlineCode::operator()(), SIMFQT::FareParserHelper::storeClass::operator()(), and SIMFQT::Fare-ParserHelper::doEndFare::operator()().

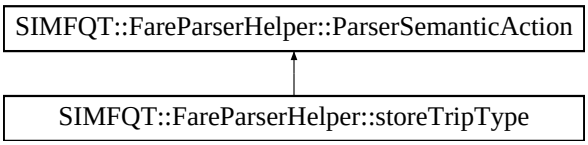
The documentation for this struct was generated from the following files:

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

22.49 SIMFQT::FareParserHelper::storeTripType Struct Reference

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

Inheritance diagram for 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

22.49.1 Detailed Description

Store the parsed customer trip type.

Definition at line 69 of file FareParserHelper.hpp.

22.49.2 Constructor & Destructor Documentation

22.49.2.1 SIMFQT::FareParserHelper::storeTripType::storeTripType (FareRuleStruct & *ioFareRule*)

Actor Constructor.

Definition at line 91 of file FareParserHelper.cpp.

22.49.3 Member Function Documentation

22.49.3.1 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().

22.49.4 Member Data Documentation

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

Actor Context.

Definition at line 35 of file FareParserHelper.hpp.

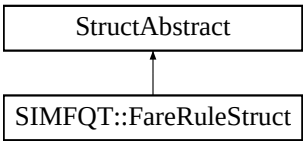
Referenced by SIMFQT::FareParserHelper::storeFareId::operator()(), SIMFQT::FareParserHelper::storeOrigin::operator()(), SIMFQT::FareParserHelper::storeDestination::operator()(), operator()(), SIMFQT::FareParserHelper::storeDateRangeStart::operator()(), SIMFQT::FareParserHelper::storeDateRangeEnd::operator()(),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

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

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

22.50 StructAbstract Class Reference

Inheritance diagram for StructAbstract:



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

- simfqt/bom/FareRuleStruct.hpp

23 File Documentation

23.1 doc/local/authors.doc File Reference

23.2 doc/local/codingrules.doc File Reference

23.3 doc/local/copyright.doc File Reference

23.4 doc/local/documentation.doc File Reference

23.5 doc/local/features.doc File Reference

23.6 doc/local/help_wanted.doc File Reference

23.7 doc/local/howto_release.doc File Reference

23.8 doc/local/index.doc File Reference

23.9 doc/local/installation.doc File Reference

23.10 doc/local/linking.doc File Reference

23.11 doc/local/test.doc File Reference

23.12 doc/local/users_guide.doc File Reference

23.13 doc/local/verification.doc File Reference

23.14 doc/tutorial/tutorial.doc File Reference

23.15 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"

23.16 BasConst.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 #include <simfqt/basic/BasConst_General.hpp>
00005 #include <simfqt/basic/BasConst_SIMFQT_Service.hpp
      >
00006
00007 namespace SIMFQT {
00008
00010   const std::string DEFAULT_FARE_QUOTER_ID = "IATA";
00011
00012 }
```

23.17 simfqt/basic/BasConst_General.hpp File Reference

Namespaces

- namespace SIMFQT

23.18 BasConst_General.hpp

```
00001 #ifndef __SIMFQT_BAS_BASCONST_GENERAL_HPP
00002 #define __SIMFQT_BAS_BASCONST_GENERAL_HPP
```

```
00003
00004 // ////////////////////////////////////////////////////////////////////
00005 // Import section
00006 // ////////////////////////////////////////////////////////////////////
00007
00008 namespace SIMFQT {
00009
00010 }
00011 #endif // __SIMFQT_BAS_BASCONST_GENERAL_HPP
```

23.19 simfqt/basic/BasConst_SIMFQT_Service.hpp File Reference

```
#include <string>
```

Namespaces

- namespace SIMFQT

23.20 BasConst_SIMFQT_Service.hpp

```
00001 #ifndef __SIMFQT_BAS_BASCONST_SIMFQT_SERVICE_HPP
00002 #define __SIMFQT_BAS_BASCONST_SIMFQT_SERVICE_HPP
00003
00004 // ////////////////////////////////////////////////////////////////////
00005 // Import section
00006 // ////////////////////////////////////////////////////////////////////
00007 #include <string>
00008
00009 namespace SIMFQT {
00010
00012   extern const std::string DEFAULT_FARE_QUOTER_ID;
00013
00014 }
00015 #endif // __SIMFQT_BAS_BASCONST_SIMFQT_SERVICE_HPP
```

23.21 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 &ioFareInputFilename, 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

23.21.1 Typedef Documentation

23.21.1.1 typedef std::vector<std::string> WordList_T

Definition at line 24 of file simfqt_parseFareRules.cpp.

23.21.2 Function Documentation

23.21.2.1 const std::string K_SIMFQT_DEFAULT_LOG_FILENAME ("simfqt_parseFareRules.log")

Default name and location for the log file.

Referenced by readConfiguration().

23.21.2.2 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().

23.21.2.3 template<class T > std::ostream& operator<< (std::ostream & *os*, const std::vector< T > & *v*)

Definition at line 44 of file simfqt_parseFareRules.cpp.

23.21.2.4 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(), K_SIMFQT_EARLY_RETURN_STATUS, PACKAGE_NAME, PACKAGE_VERSION, and PREFIXDIR.

Referenced by main().

23.21.2.5 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().

23.21.3 Variable Documentation

23.21.3.1 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().

23.21.3.2 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().

23.22 simfqt_parseFareRules.cpp

```
00001 // STL
00002 #include <cassert>
00003 #include <iostream>
00004 #include <sstream>
00005 #include <fstream>
00006 #include <vector>
00007 #include <list>
00008 #include <string>
00009 // Boost (Extended STL)
00010 #include <boost/date_time/posix_time/posix_time.hpp>
00011 #include <boost/date_time/gregorian/gregorian.hpp>
00012 #include <boost/tokenizer.hpp>
00013 #include <boost/program_options.hpp>
00014 // StdAir
00015 #include <stdair/STDAIR_Service.hpp>
00016 #include <stdair/bom/TravelSolutionStruct.hpp>
00017 #include <stdair/bom/BookingRequestStruct.hpp>
00018 #include <stdair/service/Logger.hpp>
00019 // Simfqt
00020 #include <simfqt/SIMFQT_Service.hpp>
00021 #include <simfqt/config/simfqt-paths.hpp>
00022
00023 // //////// Type definitions ///////
00024 typedef std::vector<std::string> WordList_T;
00025
00026
00027 // //////// Constants //////
00029 const std::string K_SIMFQT_DEFAULT_LOG_FILENAME ("
      simfqt_parseFareRules.log");
00030
00032 const std::string K_SIMFQT_DEFAULT_FARE_INPUT_FILENAME
       (STDAIR_SAMPLE_DIR
00033                                                          "/fare01.csv");
00034
00037 const bool K_SIMFQT_DEFAULT_BUILT_IN_INPUT =
      false;
00038
00040 const int K_SIMFQT_EARLY_RETURN_STATUS = 99;
00041
00042 // ///////// Parsing of Options & Configuration /////////
00043 // A helper function to simplify the main part.
00044 template<class T> std::ostream& operator<< (std::ostream& os,
00045                                             const std::vector<T>& v) {
00046   std::copy (v.begin(), v.end(), std::ostream_iterator<T> (std::cout, " "));
00047   return os;
00048 }
00049
00051 int readConfiguration (int argc, char* argv[], bool&
      ioIsBuiltin,
00052                        stdair::Filename_T& ioFareInputFilename,
00053                        std::string& ioLogFilename) {
00054
00055   // Default for the built-in input
00056   ioIsBuiltin = K_SIMFQT_DEFAULT_BUILT_IN_INPUT;
00057
00058   // Declare a group of options that will be allowed only on command line
00059   boost::program_options::options_description generic ("Generic options");
00060   generic.add_options()
```

```
00061     ("prefix", "print installation prefix")
00062     ("version,v", "print version string")
00063     ("help,h", "produce help message");
00064
00065   // Declare a group of options that will be allowed both on command
00066   // line and in config file
00067   boost::program_options::options_description config ("Configuration");
00068   config.add_options()
00069     ("builtin,b",
00070      "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")
00071     ("fare,f",
00072      boost::program_options::value< std::string >(&ioFareInputFilename)->
      default_value(K_SIMFQT_DEFAULT_FARE_INPUT_FILENAME
      ),
00073      "(CSV) input file for the fare rules")
00074     ("log,l",
00075      boost::program_options::value< std::string >(&ioLogFilename)->
      default_value(K_SIMFQT_DEFAULT_LOG_FILENAME),
00076      "Filename for the logs")
00077     ;
00078
00079   // Hidden options, will be allowed both on command line and
00080   // in config file, but will not be shown to the user.
00081   boost::program_options::options_description hidden ("Hidden options");
00082   hidden.add_options()
00083     ("copyright",
00084      boost::program_options::value< std::vector<std::string> >(),
00085      "Show the copyright (license)");
00086
00087   boost::program_options::options_description cmdline_options;
00088   cmdline_options.add(generic).add(config).add(hidden);
00089
00090   boost::program_options::options_description config_file_options;
00091   config_file_options.add(config).add(hidden);
00092
00093   boost::program_options::options_description visible ("Allowed options");
00094   visible.add(generic).add(config);
00095
00096   boost::program_options::positional_options_description p;
00097   p.add ("copyright", -1);
00098
00099   boost::program_options::variables_map vm;
00100   boost::program_options::
00101     store (boost::program_options::command_line_parser (argc, argv).
00102            options (cmdline_options).positional(p).run(), vm);
00103
00104   std::ifstream ifs ("simfqt.cfg");
00105   boost::program_options::store (parse_config_file (ifs, config_file_options),
00106                                  vm);
00107   boost::program_options::notify (vm); if (vm.count ("help")) {
00108     std::cout << visible << std::endl;
00109     return K_SIMFQT_EARLY_RETURN_STATUS;
00110   }
00111
00112   if (vm.count ("version")) {
00113     std::cout << PACKAGE_NAME << ", version " << PACKAGE_VERSION
       << std::endl;
00114     return K_SIMFQT_EARLY_RETURN_STATUS;
00115   }
00116
00117   if (vm.count ("prefix")) {
00118     std::cout << "Installation prefix: " << PREFIXDIR << std::endl;
00119     return K_SIMFQT_EARLY_RETURN_STATUS;
00120   }
00121
00122   if (vm.count ("builtin")) {
00123     ioIsBuiltin = true;
00124   }
00125   const std::string isBuiltinStr = (ioIsBuiltin == true)?"yes":"no";
00126   std::cout << "The BOM should be built-in? " << isBuiltinStr << std::endl;
00127
00128   if (ioIsBuiltin == false) {
00129
00130     // The BOM tree should be built from parsing a fare (and O&D) file
00131     if (vm.count ("fare")) {
00132       ioFareInputFilename = vm["fare"].as< std::string >();
00133       std::cout << "Input fare filename is: " << ioFareInputFilename
00134                 << std::endl;
00135
00136     } else {
00137       // The built-in option is not selected. However, no fare file
00138       // is specified
00139       std::cerr << "Either one among the -b/--builtin and -f/--fare "
00140                 << "options must be specified" << std::endl;
00141     }
00142   }
```

```
00143
00144   if (vm.count ("log")) {
00145     ioLogFilename = vm["log"].as< std::string >();
00146     std::cout << "Log filename is: " << ioLogFilename << std::endl;
00147   }
00148
00149   return 0;
00150 }
00151
00152
00153 // ///////////////// M A I N //////////////////
00154 int main (int argc, char* argv[]) {
00155
00156   // State whether the BOM tree should be built-in or parsed from an input file
00157   bool isBuiltin;
00158
00159   // Fare input filename
00160   stdair::Filename_T lFareInputFilename;
00161
00162   // Output log File
00163   stdair::Filename_T lLogFilename;
00164
00165   // Call the command-line option parser
00166   const int lOptionParserStatus =
00167     readConfiguration (argc, argv, isBuiltin,
      lFareInputFilename, lLogFilename);
00168
00169   if (lOptionParserStatus == K_SIMFQT_EARLY_RETURN_STATUS
      ) {
00170     return 0;
00171   }
00172
00173   // Set the log parameters
00174   std::ofstream logOutputFile;
00175   // Open and clean the log outputfile
00176   logOutputFile.open (lLogFilename.c_str());
00177   logOutputFile.clear();
00178
00179   // Initialise the Simfqt service object
00180   const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);
00181
00182   SIMFQT::SIMFQT_Service simfqtService (lLogParams);
00183
00184   // DEBUG
00185   STDAIR_LOG_DEBUG ("Welcome to Simfqt");
00186
00187   // Build a default sample list of travel solutions
00188   stdair::TravelSolutionList_T lTravelSolutionList;
00189   simfqtService.buildSampleTravelSolutions (
      lTravelSolutionList);
00190
00191   // Build a default booking request
00192   stdair::BookingRequestStruct lBookingRequest =
00193     simfqtService.buildBookingRequest();
00194
00195   // Check wether or not a (CSV) input file should be read
00196   if (isBuiltin == true) {
00197
00198     // Build the default sample BOM tree (filled with fares) for Simfqt
00199     simfqtService.buildSampleBom();
00200
00201   } else {
00202
00203     // Build the BOM tree from parsing a fare file
00204     SIMFQT::FareFilePath lFareFilePath (lFareInputFilename)
      ;
00205     simfqtService.parseAndLoad (lFareFilePath);
00206
00207   }
00208
00209   // DEBUG: Display the travel solutions
00210   const std::string& lTSCSVDump =
00211     simfqtService.csvDisplay (lTravelSolutionList);
00212   STDAIR_LOG_DEBUG (lTSCSVDump);
00213
00214   // FareQuote the sample list of travel solutions
00215   simfqtService.quotePrices (lBookingRequest, lTravelSolutionList);
00216
00217   // DEBUG: Display the whole BOM tree
00218   const std::string& lBOMCSVDump = simfqtService.csvDisplay();
00219   STDAIR_LOG_DEBUG ("BOM tree: " << lBOMCSVDump);
00220
00221   // DEBUG: Display the travel solutions
00222   const std::string& lTSCSVDumpEnd
00223     = simfqtService.csvDisplay (lTravelSolutionList);
00224   STDAIR_LOG_DEBUG (lTSCSVDumpEnd);
00225
```

```
00226   // Close the Log outputFile
00227   logOutputFile.close();
00228
00229   /*
00230     Note: as that program is not intended to be run on a server in
00231     production, it is better not to catch the exceptions. When it
00232     happens (that an exception is thrownded), that way we get the
00233     call stack.
00234   */
00235
00236   return 0;
00237 }
00238
```

23.23 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

23.24 FareRuleStruct.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 #include <vector>
00008 // StdAir
00009 #include <stdair/basic/BasConst_General.hpp>
00010 #include <stdair/service/Logger.hpp>
00011 // SIMFQT
00012 #include <simfqt/bom/FareRuleStruct.hpp>
00013
00014 namespace SIMFQT {
00015
00016   // ////////////////////////////////////////////////////////////////////
00017   FareRuleStruct::FareRuleStruct ()
00018     :_fareId(0),
00019      _origin(""),
00020      _destination(""),
00021      _dateRangeStart(stdair::DEFAULT_DATE),
00022      _dateRangeEnd(stdair::DEFAULT_DATE),
00023      _timeRangeStart(stdair::DEFAULT_EPSILON_DURATION),
00024      _timeRangeEnd(stdair::DEFAULT_EPSILON_DURATION),
00025      _cabinCode (""),
00026      _pos ("),
00027      _advancePurchase(0),
00028      _saturdayStay("T"),
00029      _changeFees("T"),
00030      _nonRefundable("T"),
00031      _minimumStay(0),
00032      _fare(0),
00033      _airlineCode(""),
00034      _classCode("") {
00035
00036   }
00037
00038   // ////////////////////////////////////////////////////////////////////
00039   stdair::Date_T FareRuleStruct::calculateDate()
       const {
00040     _itYear.check(); _itMonth.check(); _itDay.check();
00041     return stdair::Date_T (_itYear._value, _itMonth._value,
      _itDay._value);
00042   }
00043
```

```
00044   // ////////////////////////////////////////////////////////////////////
00045   stdair::Duration_T FareRuleStruct::calculateTime
    () const {
00046     _itHours.check(); _itMinutes.check(); _itSeconds
    .check();
00047     return boost::posix_time::hours (_itHours._value)
00048       + boost::posix_time::minutes (_itMinutes._value)
00049       + boost::posix_time::seconds (_itSeconds._value);
00050   }
00051
00052
00053   // ////////////////////////////////////////////////////////////////////
00054   const std::string FareRuleStruct::describe () const {
00055
00056     std::ostringstream oStr;
00057     oStr << "FareRule: " << _fareId << ", ";
00058
00059     oStr << _origin << "-" << _destination << " ("
00060          << _pos << "), " << _channel << ", [";
00061     oStr << _dateRangeStart << "/" << _dateRangeEnd << "] - ["
00062          << boost::posix_time::to_simple_string (_timeRangeStart) << "/"
00063          << boost::posix_time::to_simple_string (_timeRangeEnd) << "], ";
00064
00065     oStr << _cabinCode << ", " << _fare  << " EUR, ";
00066     oStr << _tripType << ", " << _saturdayStay << ", "
00067          <<  _changeFees << ", " << _nonRefundable << ", "
00068          << _advancePurchase << ", " << _minimumStay << ", ";
00069
00070     // Sanity check
00071     assert (_airlineCodeList.size() == _classCodeList.size());
00072
00073     // Browse the airline and class pathes
00074     unsigned short idx = 0;
00075     stdair::ClassList_StringList_T::const_iterator itClass =
00076       _classCodeList.begin();
00077     for (stdair::AirlineCodeList_T::const_iterator itAirline =
00078            _airlineCodeList.begin();
00079          itAirline != _airlineCodeList.end(); ++itAirline, ++itClass, ++idx) {
00080       if (idx != 0) {
00081         oStr << " - ";
00082       }
00083       const stdair::AirlineCode_T lAirlineCode = *itAirline;
00084       const stdair::ClassCode_T lClassCode = *itClass;
00085       oStr << lAirlineCode << " / " << lClassCode;
00086     }
00087
00088     return oStr.str();
00089   }
00090
00091 }
00092
```

23.25 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

23.26 FareRuleStruct.hpp

```
00001 #ifndef __SIMFQT_BOM_FARERULESTRUCT_HPP
00002 #define __SIMFQT_BOM_FARERULESTRUCT_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <string>
00009 #include <vector>
00010 // StdAir
00011 #include <stdair/stdair_demand_types.hpp>
00012 #include <stdair/stdair_inventory_types.hpp>
00013 #include <stdair/basic/StructAbstract.hpp>
00014 #include <stdair/basic/BasParserHelperTypes.hpp>
00015 // SIMFQT
00016 #include <simfqt/SIMFQT_Types.hpp>
00017
00018 namespace SIMFQT {
00019
00021   struct FareRuleStruct : public stdair::StructAbstract {
00022   public:
00023
00025     FareRuleStruct ();
00026
00027   public:
00028     // ///////// Getters ///////////
00030     SIMFQT::FareQuoteID_T getFareID () const {
00031       return _fareId;
00032     }
00033
00035     stdair::AirportCode_T getOrigin () const {
00036       return _origin;
00037     }
00038
00040     stdair::AirportCode_T getDestination () const {
00041       return _destination;
00042     }
00043
00045     stdair::TripType_T getTripType () const {
00046       return _tripType;
00047     }
00048
00050     stdair::Date_T getDateRangeStart () const {
00051       return _dateRangeStart;
00052     }
00053
00055     stdair::Date_T getDateRangeEnd () const {
00056       return _dateRangeEnd;
00057     }
00058
00060     stdair::Duration_T getTimeRangeStart () const {
00061       return _timeRangeStart;
00062     }
00063
00065     stdair::Duration_T getTimeRangeEnd () const {
00066       return _timeRangeEnd;
00067     }
00068
00070     stdair::CabinCode_T getCabinCode () const {
00071       return _cabinCode;
00072     }
00073
00075     const stdair::CityCode_T getPOS () const {
00076       return _pos;
00077     }
00078
00080     stdair::ChannelLabel_T getChannel () const {
00081       return _channel;
00082     }
00083
00085     stdair::DayDuration_T getAdvancePurchase () const {
00086       return _advancePurchase;
00087     }
00088
00090     stdair::SaturdayStay_T getSaturdayStay () const {
00091       return _saturdayStay;
00092     }
00093
00095     stdair::ChangeFees_T getChangeFees () const {
00096       return _changeFees;
00097     }
00098
00100     stdair::NonRefundable_T getNonRefundable () const {
00101       return _nonRefundable;
00102     }
```

```
00103
00105     stdair::DayDuration_T getMinimumStay () const {
00106       return _minimumStay;
00107     }
00108
00110     stdair::PriceValue_T getFare () const {
00111       return _fare;
00112     }
00113
00115     stdair::AirlineCode_T getAirlineCode () const {
00116       return _airlineCode;
00117     }
00118
00120     stdair::ClassCode_T getClassCode () const {
00121       return _classCode;
00122     }
00123
00125     const unsigned int getAirlineListSize () const {
00126       return _airlineCodeList.size();
00127     }
00128
00130     const unsigned int getClassCodeListSize () const {
00131       return _classCodeList.size();
00132     }
00133
00135     stdair::AirlineCodeList_T getAirlineList () const {
00136       return _airlineCodeList;
00137     }
00138
00140     stdair::ClassList_StringList_T getClassCodeList () const {
00141       return _classCodeList;
00142     }
00143
00144   public:
00145     // ///////// Display support methods //////////
00147     stdair::Date_T calculateDate() const;
00148
00150     stdair::Duration_T calculateTime() const;
00151
00153     const std::string describe() const;
00154
00155   public:
00156     // ///////// Setters //////////
00158     void setFareID (const SIMFQT::FareQuoteID_T&
      iFareQuoteID) {
00159       _fareId = iFareQuoteID;
00160     }
00161
00163     void setOrigin (const stdair::AirportCode_T& iOrigin) {
00164       _origin = iOrigin;
00165     }
00166
00168     void setDestination (const stdair::AirportCode_T&
      iDestination) {
00169       _destination = iDestination;
00170     }
00171
00173     void setTripType (const stdair::TripType_T& iTripType) {
00174       _tripType = iTripType;
00175     }
00176
00178     void setDateRangeStart (const stdair::Date_T&
      iDateRangeStart) {
00179       _dateRangeStart = iDateRangeStart;
00180     }
00181
00183     void setDateRangeEnd (const stdair::Date_T& iDateRangeEnd) {
00184       _dateRangeEnd = iDateRangeEnd;
00185     }
00186
00188     void setTimeRangeStart (const stdair::Duration_T&
      iTimeRangeStart) {
00189       _timeRangeStart = iTimeRangeStart;
00190     }
00191
00193     void setTimeRangeEnd (const stdair::Duration_T&
      iTimeRangeEnd) {
00194       _timeRangeEnd = iTimeRangeEnd;
00195     }
00196
00198     void setCabinCode (const stdair::CabinCode_T& iCabinCode) {
00199       _cabinCode = iCabinCode;
00200     }
00201
00203     void setPOS (const stdair::CityCode_T& iPOS) {
00204       _pos = iPOS;
00205     }
```

```
00206
00208     void setChannel (const stdair::ChannelLabel_T& iChannel) {
00209       _channel = iChannel;
00210     }
00211
00213     void setAdvancePurchase (const stdair::DayDuration_T&
      iAdvancePurchase) {
00214       _advancePurchase = iAdvancePurchase;
00215     }
00216
00218     void setSaturdayStay (const stdair::SaturdayStay_T&
      iSaturdayStay) {
00219       _saturdayStay = iSaturdayStay;
00220     }
00221
00223     void setChangeFees (const stdair::ChangeFees_T& iChangeFees) {
00224       _changeFees = iChangeFees;
00225     }
00226
00228     void setNonRefundable (const stdair::NonRefundable_T&
      iNonRefundable) {
00229       _nonRefundable = iNonRefundable;
00230     }
00231
00233     void setMinimumStay (const stdair::DayDuration_T&
      iMinimumStay) {
00234       _minimumStay = iMinimumStay;
00235     }
00236
00238     void setFare (const stdair::PriceValue_T& iFare) {
00239       _fare = iFare;
00240     }
00241
00243     void setAirlineCode (const stdair::AirlineCode_T&
      iAirlineCode) {
00244       _airlineCode = iAirlineCode;
00245     }
00246
00248     void setClassCode (const stdair::ClassCode_T& iClassCode) {
00249       _classCode = iClassCode;
00250     }
00251
00253     void clearAirlineCodeList () {
00254       _airlineCodeList.clear();
00255     }
00256
00258     void clearClassCodeList () {
00259       _classCodeList.clear();
00260     }
00261
00263     void addAirlineCode (const stdair::AirlineCode_T&
      iAirlineCode)  {
00264       _airlineCodeList.push_back (iAirlineCode);
00265     }
00266
00268     void addClassCode (const stdair::ClassCode_T& iClassCode) {
00269       _classCodeList.push_back (iClassCode);
00270     }
00271
00272   public:
00273     // ////////////////// Attributes /////////////////
00275     stdair::year_t _itYear;
00276     stdair::month_t _itMonth;
00277     stdair::day_t _itDay;
00278
00280     stdair::hour_t _itHours;
00281     stdair::minute_t _itMinutes;
00282     stdair::second_t _itSeconds;
00283
00284   private:
00285     // ////////////////// Attributes /////////////////
00287     SIMFQT::FareQuoteID_T _fareId;
00288
00290     stdair::AirportCode_T _origin;
00291
00293     stdair::AirportCode_T _destination;
00294
00296     stdair::TripType_T _tripType;
00297
00299     stdair::Date_T _dateRangeStart;
00300
00302     stdair::Date_T _dateRangeEnd;
00303
00305     stdair::Duration_T _timeRangeStart;
00306
00308     stdair::Duration_T _timeRangeEnd;
00309
```

```
00311     stdair::CabinCode_T _cabinCode;
00312
00314     stdair::CityCode_T _pos;
00315
00317     stdair::ChannelLabel_T _channel;
00318
00320     stdair::DayDuration_T _advancePurchase;
00321
00323     stdair::SaturdayStay_T _saturdayStay;
00324
00326     stdair::ChangeFees_T _changeFees;
00327
00329     stdair::NonRefundable_T _nonRefundable;
00330
00332     stdair::DayDuration_T _minimumStay;
00333
00335     stdair::PriceValue_T _fare;
00336
00338     stdair::AirlineCode_T _airlineCode;
00339
00341     stdair::ClassCode_T _classCode;
00342
00345     stdair::AirlineCodeList_T _airlineCodeList;
00346
00349     stdair::ClassList_StringList_T _classCodeList;
00350
00351   };
00352
00353 }
00354 #endif // __SIMFQT_BOM_FARERULESTRUCT_HPP
```

23.27 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

23.28 FareParser.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <string>
00007 // StdAir
00008 #include <stdair/basic/BasFileMgr.hpp>
00009 #include <stdair/service/Logger.hpp>
00010 // AirSched
00011 #include <simfqt/command/FareParserHelper.hpp
      >
00012 #include <simfqt/command/FareParser.hpp>
00013
00014 namespace SIMFQT {
00015
00016   // ////////////////////////////////////////////////////////////////////
00017   void FareParser::fareRuleGeneration (const
      FareFilePath& iFareFilename,
00018                                        stdair::BomRoot& ioBomRoot) {
00019
00020     const stdair::Filename_T lFilename = iFareFilename.name();
00021
00022     // Check that the file path given as input corresponds to an actual file
00023     const bool doesExistAndIsReadable =
00024       stdair::BasFileMgr::doesExistAndIsReadable (lFilename);
00025     if (doesExistAndIsReadable == false) {
00026       STDAIR_LOG_ERROR ("The fare input file, '" << lFilename
```

```
00027                         << "', can not be retrieved on the file-system");
00028       throw FareInputFileNotFoundException ("The
     fare input file '" + lFilename
00029                                             + "' does not exist or can not "
00030                                             "be read");
00031     }
00032
00033     // Initialise the fare file parser.
00034     FareRuleFileParser lFareRuleFileParser (ioBomRoot,
    lFilename);
00035
00036     // Parse the CSV-formatted fare input file and generate the
00037     // corresponding fare rules.
00038     lFareRuleFileParser.generateFareRules ();
00039
00040   }
00041
00042 }
```

23.29 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

23.30 FareParser.hpp

```
00001 #ifndef __SIMFQT_CMD_FAREPARSER_HPP
00002 #define __SIMFQT_CMD_FAREPARSER_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <string>
00009 // StdAir
00010 #include <stdair/stdair_basic_types.hpp>
00011 #include <stdair/command/CmdAbstract.hpp>
00012 // SimFQT
00013 #include <simfqt/SIMFQT_Types.hpp>
00014
00015 // Forward declarations.
00016 namespace stdair {
00017   class BomRoot;
00018 }
00019
00020 namespace SIMFQT {
00021
00023   class FareParser : public stdair::CmdAbstract {
00024   public:
00030     static void fareRuleGeneration (const FareFilePath
    &, stdair::BomRoot&);
00031   };
00032 }
00033 #endif // __SIMFQT_CMD_FAREPARSER_HPP
```

23.31 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

23.32 FareParserHelper.cpp

```
00001  // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <vector>
00007 #include <fstream>
00008 // StdAir
00009 #include <stdair/basic/BasFileMgr.hpp>
00010 #include <stdair/basic/BasConst_Request.hpp>
00011 #include <stdair/bom/BomRoot.hpp>
00012 #include <stdair/service/Logger.hpp>
00013 //#define BOOST_SPIRIT_DEBUG
00014 #include <stdair/basic/BasParserTypes.hpp>
00015 // SIMFQT
00016 #include <simfqt/command/FareParserHelper.hpp
     >
00017 #include <simfqt/command/FareRuleGenerator.hpp
     >
00018
00019
00020
00021 namespace SIMFQT {
00022
00023   namespace FareParserHelper {
```

```
00024
00025     // ////////////////////////////////////////////////////////////////////
00026     //  Semantic actions
00027     // ////////////////////////////////////////////////////////////////////
00028
00029     ParserSemanticAction::
00030     ParserSemanticAction (FareRuleStruct&
      ioFareRule)
00031       : _fareRule (ioFareRule) {
00032     }
00033
00034     // ////////////////////////////////////////////////////////////////////
00035     storeFareId::
00036     storeFareId (FareRuleStruct& ioFareRule)
00037       : ParserSemanticAction (ioFareRule) {
00038     }
00039
00040     // ////////////////////////////////////////////////////////////////////
00041     void storeFareId::operator() (unsigned int iFareId,
00042                                   boost::spirit::qi::unused_type,
00043                                   boost::spirit::qi::unused_type) const {
00044       _fareRule.setFareID (iFareId);
00045
00046       // DEBUG
00047       //STDAIR_LOG_DEBUG ( "Fare Id: " << _fareRule.getFareID ());
00048       const stdair::AirlineCode_T lEmptyAirlineCode ("");
00049       _fareRule.setAirlineCode(lEmptyAirlineCode);
00050       _fareRule.clearAirlineCodeList();
00051       const stdair::ClassCode_T lEmptyClassCode ("");
00052       _fareRule.setClassCode(lEmptyClassCode);
00053       _fareRule.clearClassCodeList();
00054       _fareRule._itSeconds = 0;
00055     }
00056
00057     // ////////////////////////////////////////////////////////////////////
00058     storeOrigin ::
00059     storeOrigin (FareRuleStruct& ioFareRule)
00060       : ParserSemanticAction (ioFareRule) {
00061     }
00062
00063     // ////////////////////////////////////////////////////////////////////
00064     void storeOrigin::operator() (std::vector<char>
      iChar,
00065                                   boost::spirit::qi::unused_type,
00066                                   boost::spirit::qi::unused_type) const {
00067        const stdair::AirportCode_T lOrigin (iChar.begin(), iChar.end());
00068        _fareRule.setOrigin (lOrigin);
00069        // DEBUG
00070        //STDAIR_LOG_DEBUG ( "Origin: " << _fareRule.getOrigin ());
00071     }
00072
00073     // ////////////////////////////////////////////////////////////////////
00074     storeDestination ::
00075     storeDestination (FareRuleStruct&
      ioFareRule)
00076       : ParserSemanticAction (ioFareRule) {
00077     }
00078
00079     // ////////////////////////////////////////////////////////////////////
00080     void storeDestination::operator() (
      std::vector<char> iChar,
00081                                        boost::spirit::qi::unused_type,
00082                                        boost::spirit::qi::unused_type) const {
00083        const stdair::AirportCode_T lDestination (iChar.begin(), iChar.end());
00084        _fareRule.setDestination (lDestination);
00085        // DEBUG
00086        //STDAIR_LOG_DEBUG ( "Destination: " << _fareRule.getDestination ());
00087     }
00088
00089     // ////////////////////////////////////////////////////////////////////
00090     storeTripType ::
00091     storeTripType (FareRuleStruct& ioFareRule)
00092       : ParserSemanticAction (ioFareRule) {
00093     }
00094
00095     // ////////////////////////////////////////////////////////////////////
00096     void storeTripType::operator() (std::vector<char>
      iChar,
00097                                     boost::spirit::qi::unused_type,
00098                                     boost::spirit::qi::unused_type) const {
00099       const stdair::TripType_T lTripType (iChar.begin(), iChar.end());
00100       if (lTripType == "OW" || lTripType == "RT") {
00101         _fareRule.setTripType (lTripType);
00102       } else {
00103         // ERROR
00104         STDAIR_LOG_ERROR ("Invalid trip type  " << lTripType);
00105       }
```

```
00106       // DEBUG
00107       //STDAIR_LOG_DEBUG ("TripType: " << _fareRule.getTripType ());
00108     }
00109
00110
00111     // ////////////////////////////////////////////////////////////////////
00112     storeDateRangeStart::
00113     storeDateRangeStart (FareRuleStruct&
      ioFareRule)
00114       : ParserSemanticAction (ioFareRule) {
00115     }
00116
00117     // ////////////////////////////////////////////////////////////////////
00118     void storeDateRangeStart::operator() (
      boost::spirit::qi::unused_type,
00119                                           boost::spirit::qi::unused_type,
00120                                           boost::spirit::qi::unused_type) const
       {
00121       const stdair::Date_T& lDateStart = _fareRule.calculateDate
       ();
00122       _fareRule.setDateRangeStart (lDateStart);
00123       // DEBUG
00124       //STDAIR_LOG_DEBUG ("Date Range Start: " << _fareRule.getDateRangeStart
       ());
00125     }
00126
00127     // ////////////////////////////////////////////////////////////////////
00128     storeDateRangeEnd::
00129     storeDateRangeEnd(FareRuleStruct&
      ioFareRule)
00130       : ParserSemanticAction (ioFareRule) {
00131     }
00132
00133     // ////////////////////////////////////////////////////////////////////
00134     void storeDateRangeEnd::operator() (
      boost::spirit::qi::unused_type,
00135                                         boost::spirit::qi::unused_type,
00136                                         boost::spirit::qi::unused_type) const {
00137       const stdair::Date_T& lDateEnd = _fareRule.calculateDate
       ();
00138       // As a Boost date period (DatePeriod_T) defines the last day of
00139       // the period to be end-date - one day, we have to add one day to that
00140       // end date before.
00141       const stdair::DateOffset_T oneDay (1);
00142       const stdair::Date_T lBoostDateEnd = lDateEnd + oneDay;
00143       _fareRule.setDateRangeEnd (lBoostDateEnd);
00144        // DEBUG
00145       //STDAIR_LOG_DEBUG ("Date Range End: " << _fareRule.getDateRangeEnd ());
00146     }
00147
00148     // ////////////////////////////////////////////////////////////////////
00149     storeStartRangeTime::
00150     storeStartRangeTime (FareRuleStruct&
      ioFareRule)
00151       : ParserSemanticAction (ioFareRule) {
00152     }
00153
00154     // ////////////////////////////////////////////////////////////////////
00155     void storeStartRangeTime::operator() (
      boost::spirit::qi::unused_type,
00156                                           boost::spirit::qi::unused_type,
00157                                           boost::spirit::qi::unused_type) const
       {
00158       const stdair::Duration_T& lTimeStart = _fareRule.calculateTime
       ();
00159       _fareRule.setTimeRangeStart (lTimeStart);
00160       // DEBUG
00161       //STDAIR_LOG_DEBUG ("Time Range Start: " << _fareRule.getTimeRangeStart
       ());
00162       // Reset the number of seconds
00163       _fareRule._itSeconds = 0;
00164     }
00165
00166     // ////////////////////////////////////////////////////////////////////
00167     storeEndRangeTime::
00168     storeEndRangeTime (FareRuleStruct&
      ioFareRule)
00169       : ParserSemanticAction (ioFareRule) {
00170     }
00171
00172     // ////////////////////////////////////////////////////////////////////
00173     void storeEndRangeTime::operator() (
      boost::spirit::qi::unused_type,
00174                                         boost::spirit::qi::unused_type,
00175                                         boost::spirit::qi::unused_type) const {
00176       const stdair::Duration_T& lTimeEnd = _fareRule.calculateTime
       ();
```

```
00177       _fareRule.setTimeRangeEnd (lTimeEnd);
00178       // DEBUG
00179       //STDAIR_LOG_DEBUG ("Time Range End: " << _fareRule.getTimeRangeEnd ());
00180       // Reset the number of seconds
00181       _fareRule._itSeconds = 0;
00182     }
00183
00184     // ////////////////////////////////////////////////////////////////////
00185     storePOS ::
00186     storePOS (FareRuleStruct& ioFareRule)
00187       : ParserSemanticAction (ioFareRule) {
00188     }
00189
00190     // ////////////////////////////////////////////////////////////////////
00191     void storePOS::operator() (std::vector<char> iChar,
00192                                boost::spirit::qi::unused_type,
00193                                boost::spirit::qi::unused_type) const {
00194       const stdair::CityCode_T lPOS (iChar.begin(), iChar.end());
00195       if (lPOS == _fareRule.getOrigin() || lPOS == _fareRule
      .getDestination()) {
00196         _fareRule.setPOS (lPOS);
00197       } else if (lPOS == "ROW") {
00198         const stdair::CityCode_T lPOSROW ("ROW");
00199         _fareRule.setPOS (lPOSROW);
00200       } else if (lPOS == stdair::DEFAULT_POS) {
00201         _fareRule.setPOS (stdair::DEFAULT_POS);
00202       } else {
00203         // ERROR
00204         STDAIR_LOG_ERROR ("Invalid point of sale " << lPOS);
00205       }
00206       // DEBUG
00207       //STDAIR_LOG_DEBUG ("POS: " << _fareRule.getPOS ());
00208     }
00209
00210     // ////////////////////////////////////////////////////////////////////
00211     storeCabinCode ::
00212     storeCabinCode (FareRuleStruct& ioFareRule)
00213       : ParserSemanticAction (ioFareRule) {
00214     }
00215
00216     // ////////////////////////////////////////////////////////////////////
00217     void storeCabinCode::operator() (char iChar,
00218                                      boost::spirit::qi::unused_type,
00219                                      boost::spirit::qi::unused_type) const {
00220       std::ostringstream ostr;
00221       ostr << iChar;
00222       const std::string cabinCodeStr = ostr.str();
00223       const stdair::CabinCode_T& lCabinCode (cabinCodeStr);
00224       _fareRule.setCabinCode (lCabinCode);
00225
00226       // DEBUG
00227       //STDAIR_LOG_DEBUG ("Cabin Code: " << _fareRule.getCabinCode ());

00228
00229     }
00230
00231     // ////////////////////////////////////////////////////////////////////
00232     storeChannel ::
00233     storeChannel (FareRuleStruct& ioFareRule)
00234       : ParserSemanticAction (ioFareRule) {
00235     }
00236
00237     // ////////////////////////////////////////////////////////////////////
00238     void storeChannel::operator() (std::vector<char>
      iChar,
00239                                    boost::spirit::qi::unused_type,
00240                                    boost::spirit::qi::unused_type) const {
00241       const stdair::ChannelLabel_T lChannel (iChar.begin(), iChar.end());
00242       if (lChannel != "IN" && lChannel != "IF" && lChannel != "DN"
00243           && lChannel != "DF" && lChannel != stdair::DEFAULT_CHANNEL) {
00244         // ERROR
00245         STDAIR_LOG_ERROR ("Invalid channel " << lChannel);
00246       }
00247       _fareRule.setChannel (lChannel);
00248       // DEBUG
00249       //STDAIR_LOG_DEBUG ("Channel: " << _fareRule.getChannel ());
00250     }
00251
00252     // ////////////////////////////////////////////////////////////////////
00253     storeAdvancePurchase ::
00254     storeAdvancePurchase (FareRuleStruct&
       ioFareRule)
00255       : ParserSemanticAction (ioFareRule) {
00256     }
00257
00258     // ////////////////////////////////////////////////////////////////////
00259     void storeAdvancePurchase::operator() (
```

```
      unsigned int iAdancePurchase,
00260                                          boost::spirit::qi::unused_type,
00261                                          boost::spirit::qi::unused_type)
       const {
00262       const stdair::DayDuration_T& lAdancePurchase = iAdancePurchase;
00263       _fareRule.setAdvancePurchase (lAdancePurchase)
      ;
00264       // DEBUG
00265       //STDAIR_LOG_DEBUG ( "Advance Purchase: " << _fareRule.getAdvancePurchase
       ());
00266     }
00267
00268     // ////////////////////////////////////////////////////////////////////
00269     storeSaturdayStay ::
00270     storeSaturdayStay (FareRuleStruct&
      ioFareRule)
00271       : ParserSemanticAction (ioFareRule) {
00272     }
00273
00274     // ////////////////////////////////////////////////////////////////////
00275     void storeSaturdayStay::operator() (char
      iSaturdayStay,
00276                                         boost::spirit::qi::unused_type,
00277                                         boost::spirit::qi::unused_type) const {
00278       bool lBool = false;
00279       if (iSaturdayStay == 'T') {
00280         lBool = true;
00281       } else {
00282         if (iSaturdayStay != 'F') {
00283           // DEBUG
00284           STDAIR_LOG_DEBUG ("Invalid saturdayStay char " << iSaturdayStay);
00285         }
00286       }
00287       const stdair::SaturdayStay_T lSaturdayStay (lBool);
00288       _fareRule.setSaturdayStay (lSaturdayStay);
00289       // DEBUG
00290       //STDAIR_LOG_DEBUG ("Saturday Stay: " << _fareRule.getSaturdayStay ());
00291     }
00292
00293     // ////////////////////////////////////////////////////////////////////
00294     storeChangeFees ::
00295     storeChangeFees (FareRuleStruct&
      ioFareRule)
00296       : ParserSemanticAction (ioFareRule) {
00297     }
00298
00299     // ////////////////////////////////////////////////////////////////////
00300     void storeChangeFees::operator() (char
      iChangefees,
00301                                       boost::spirit::qi::unused_type,
00302                                       boost::spirit::qi::unused_type) const {
00303
00304       bool lBool = false;
00305       if (iChangefees == 'T') {
00306         lBool = true;
00307       } else {
00308         if (iChangefees != 'F') {
00309           // DEBUG
00310           STDAIR_LOG_DEBUG ("Invalid change fees char " << iChangefees);
00311         }
00312       }
00313       const stdair::ChangeFees_T lChangefees (lBool);
00314       _fareRule.setChangeFees (lChangefees);
00315       // DEBUG
00316       //STDAIR_LOG_DEBUG ("Change fees: " << _fareRule.getChangeFees ());
00317     }
00318
00319     // ////////////////////////////////////////////////////////////////////
00320     storeNonRefundable ::
00321     storeNonRefundable (FareRuleStruct&
      ioFareRule)
00322       : ParserSemanticAction (ioFareRule) {
00323     }
00324
00325     // ////////////////////////////////////////////////////////////////////
00326     void storeNonRefundable::operator() (char
      iNonRefundable,
00327                                          boost::spirit::qi::unused_type,
00328                                          boost::spirit::qi::unused_type) const
      {
00329       bool lBool = false;
00330       if (iNonRefundable == 'T') {
00331         lBool = true;
00332       } else {
00333         if (iNonRefundable != 'F') {
00334           // DEBUG
00335           STDAIR_LOG_DEBUG ("Invalid non refundable char " << iNonRefundable);
```

```
00336         }
00337       }
00338       const stdair::NonRefundable_T lNonRefundable (lBool);
00339       _fareRule.setNonRefundable (lNonRefundable);
00340       // DEBUG
00341       //STDAIR_LOG_DEBUG ("Non refundable: " << _fareRule.getNonRefundable
     ());
00342     }
00343
00344     // ////////////////////////////////////////////////////////////////////
00345     storeMinimumStay ::
00346     storeMinimumStay (FareRuleStruct&
    ioFareRule)
00347       : ParserSemanticAction (ioFareRule) {
00348     }
00349
00350     // ////////////////////////////////////////////////////////////////////
00351     void storeMinimumStay::operator() (unsigned
    int iMinStay,
00352                                        boost::spirit::qi::unused_type,
00353                                        boost::spirit::qi::unused_type) const {
00354       const stdair::DayDuration_T lMinStay = iMinStay;
00355       _fareRule.setMinimumStay (lMinStay);
00356       // DEBUG
00357       //STDAIR_LOG_DEBUG ("Minimum Stay: " << _fareRule.getMinimumStay ());
00358     }
00359
00360     // ////////////////////////////////////////////////////////////////////
00361     storeFare ::
00362     storeFare (FareRuleStruct& ioFareRule)
00363       : ParserSemanticAction (ioFareRule) {
00364     }
00365
00366     // ////////////////////////////////////////////////////////////////////
00367     void storeFare::operator() (double iFare,
00368                                 boost::spirit::qi::unused_type,
00369                                 boost::spirit::qi::unused_type) const {
00370      const stdair::PriceValue_T lFare = iFare;
00371       _fareRule.setFare (lFare);
00372       // DEBUG
00373       //STDAIR_LOG_DEBUG ("Fare: " << _fareRule.getFare ());
00374     }
00375
00376     // ////////////////////////////////////////////////////////////////////
00377     storeAirlineCode ::
00378     storeAirlineCode (FareRuleStruct&
    ioFareRule)
00379       : ParserSemanticAction (ioFareRule) {
00380     }
00381
00382     // ////////////////////////////////////////////////////////////////////
00383     void storeAirlineCode::operator() (
    std::vector<char> iChar,
00384                                        boost::spirit::qi::unused_type,
00385                                        boost::spirit::qi::unused_type) const {
00386
00387       const stdair::AirlineCode_T lAirlineCode (iChar.begin(), iChar.end());
00388       // Insertion of this airline Code list in the whole AirlineCode name
00389       _fareRule.addAirlineCode (lAirlineCode);
00390       // DEBUG
00391       //STDAIR_LOG_DEBUG ( "Airline code: " << lAirlineCode);
00392     }
00393
00394     // ////////////////////////////////////////////////////////////////////
00395     storeClass ::
00396     storeClass (FareRuleStruct& ioFareRule)
00397       : ParserSemanticAction (ioFareRule) {
00398     }
00399
00400     // ////////////////////////////////////////////////////////////////////
00401     void storeClass::operator() (std::vector<char> iChar
    ,
00402                                  boost::spirit::qi::unused_type,
00403                                  boost::spirit::qi::unused_type) const {
00404       std::ostringstream ostr;
00405       for (std::vector<char>::const_iterator lItVector = iChar.begin();
00406          lItVector != iChar.end();
00407          lItVector++) {
00408         ostr << *lItVector;
00409       }
00410       const std::string classCodeStr = ostr.str();
00411       const stdair::ClassCode_T lClassCode (classCodeStr);
00412       // Insertion of this class Code list in the whole classCode name
00413       _fareRule.addClassCode (lClassCode);
00414       // DEBUG
00415       //STDAIR_LOG_DEBUG ("Class Code: " << lClassCode);
00416     }
```

```
00417
00418     // ////////////////////////////////////////////////////////////////////
00419     doEndFare::
00420     doEndFare (stdair::BomRoot& ioBomRoot,
00421                FareRuleStruct& ioFareRule)
00422       : ParserSemanticAction (ioFareRule),
00423         _bomRoot (ioBomRoot) {
00424     }
00425
00426     // ////////////////////////////////////////////////////////////////////
00427     void doEndFare::operator() (
      boost::spirit::qi::unused_type,
00428                                 boost::spirit::qi::unused_type,
00429                                 boost::spirit::qi::unused_type) const {
00430       // DEBUG
00431       //STDAIR_LOG_DEBUG ("Do End");
00432       // Generation of the fare rule object.
00433       FareRuleGenerator::createAirportPair (_bomRoot, _fareRule
      );
00434       STDAIR_LOG_DEBUG(_fareRule.describe());
00435     }
00436
00437     // ////////////////////////////////////////////////////////////////////
00438     //
00439     //  Utility Parsers
00440     //
00441     // ////////////////////////////////////////////////////////////////////
00443     namespace bsq = boost::spirit::qi;
00444     namespace bsa = boost::spirit::ascii;
00445
00447     stdair::int1_p_t int1_p;
00448
00450     stdair::uint2_p_t uint2_p;
00451
00453     stdair::uint4_p_t uint4_p;
00454
00456     stdair::uint1_4_p_t uint1_4_p;
00457
00459     stdair::hour_p_t hour_p;
00460     stdair::minute_p_t minute_p;
00461     stdair::second_p_t second_p;
00462
00464     stdair::year_p_t year_p;
00465     stdair::month_p_t month_p;
00466     stdair::day_p_t day_p;
00467
00469     //
00470     //  (Boost Spirit) Grammar Definition
00471     //
00473
00502     template <typename Iterator>
00503     struct FareRuleParser :
00504       public boost::spirit::qi::grammar<Iterator,
00505                                         boost::spirit::ascii::space_type> {
00506
00507       FareRuleParser (stdair::BomRoot& ioBomRoot,
00508                       FareRuleStruct& iofareRule) :
00509
00510         FareRuleParser::base_type(start),
00511         _bomRoot(ioBomRoot), _fareRule(iofareRule) {
00512
00513
00514       start = *(comments | fare_rule);
00515
00516       comments = (bsq::lexeme[bsq::repeat(2)[bsa::char_('/')]
00517                               >> +(bsa::char_ - bsq::eol)
00518                               >> bsq::eol]
00519                   | bsq::lexeme[bsa::char_('/') >>bsa::char_('*')
00520                               >> +(bsa::char_ - bsa::char_('*'))
00521                               >> bsa::char_('*') >> bsa::char_('/')]);
00522
00523       fare_rule = fare_key
00524         >> +( ';' >> segment )
00525         >> fare_rule_end[doEndFare(_bomRoot,
      _fareRule)];
00526
00527       fare_rule_end = bsa::char_(';');
00528
00529       fare_key = fare_id
00530         >> ';' >> origin >> ';' >> destination
00531         >> ';' >> tripType
00532         >> ';' >> dateRangeStart >> ';' >> dateRangeEnd
00533         >> ';' >> timeRangeStart >> ';' >> timeRangeEnd
00534         >> ';' >> point_of_sale >>  ';' >> cabinCode >>
      ';' >> channel
00535         >> ';' >> advancePurchase >> ';' >> saturdayStay
00536         >> ';' >> changeFees >> ';' >> nonRefundable
```

```
00537         >> ';' >> minimumStay >> ';' >> fare;
00538
00539       fare_id = uint1_4_p[storeFareId(_fareRule
      )];
00540
00541       origin = bsq::repeat(3)[bsa::char_("A-Z")][storeOrigin(
      _fareRule)];
00542
00543       destination =
00544         bsq::repeat(3)[bsa::char_("A-Z")][storeDestination(
      _fareRule)];
00545
00546       tripType =
00547         bsq::repeat(2)[bsa::char_("A-Z")][storeTripType(_fareRule
      )];
00548
00549       dateRangeStart = date[storeDateRangeStart
      (_fareRule)];
00550
00551       dateRangeEnd = date[storeDateRangeEnd(
      _fareRule)];
00552
00553       date = bsq::lexeme
00554         [year_p[boost::phoenix::ref(_fareRule._itYear) =
      bsq::labels::_1]
00555         >> '-'
00556         >> month_p[boost::phoenix::ref(_fareRule._itMonth
      ) = bsq::labels::_1]
00557         >> '-'
00558         >> day_p[boost::phoenix::ref(_fareRule._itDay) =
      bsq::labels::_1] ];
00559
00560       timeRangeStart = time[storeStartRangeTime
      (_fareRule)];
00561
00562       timeRangeEnd = time[storeEndRangeTime(
      _fareRule)];
00563
00564       time =  bsq::lexeme
00565         [hour_p[boost::phoenix::ref(_fareRule._itHours)
      = bsq::labels::_1]
00566         >> ':'
00567         >> minute_p[boost::phoenix::ref(_fareRule._itMinutes
      ) = bsq::labels::_1]
00568         >> - (':' >> second_p[boost::phoenix::ref(_fareRule.
      _itSeconds) = bsq::labels::_1]) ];
00569
00570       point_of_sale = bsq::repeat(3)[bsa::char_("A-Z")][storePOS
      (_fareRule)];
00571
00572       cabinCode = bsa::char_("A-Z")[storeCabinCode(
      _fareRule)];
00573
00574       channel = bsq::repeat(2)[bsa::char_("A-Z")][storeChannel
      (_fareRule)];
00575
00576       advancePurchase = uint1_4_p[storeAdvancePurchase
      (_fareRule)];
00577
00578       saturdayStay = bsa::char_("A-Z")[storeSaturdayStay
      (_fareRule)];
00579
00580       changeFees = bsa::char_("A-Z")[storeChangeFees(
      _fareRule)];
00581
00582       nonRefundable = bsa::char_("A-Z")[storeNonRefundable
      (_fareRule)];
00583
00584       minimumStay = uint1_4_p[storeMinimumStay
      (_fareRule)];
00585
00586       fare = bsq::double_[storeFare(_fareRule)];
00587
00588       segment = bsq::repeat(2)[bsa::char_("A-Z")][storeAirlineCode
      (_fareRule)]
00589         >> ';'
00590         >> bsq::repeat(1,bsq::inf)[bsa::char_("A-Z")][storeClass(
      _fareRule)];
00591
00592       //BOOST_SPIRIT_DEBUG_NODE (FareRuleParser);
00593       BOOST_SPIRIT_DEBUG_NODE (start);
00594       BOOST_SPIRIT_DEBUG_NODE (comments);
00595       BOOST_SPIRIT_DEBUG_NODE (fare_rule);
00596       BOOST_SPIRIT_DEBUG_NODE (fare_rule_end);
00597       BOOST_SPIRIT_DEBUG_NODE (fare_key);
00598       BOOST_SPIRIT_DEBUG_NODE (fare_id);
00599       BOOST_SPIRIT_DEBUG_NODE (origin);
```

```
00600       BOOST_SPIRIT_DEBUG_NODE (destination);
00601       BOOST_SPIRIT_DEBUG_NODE (tripType);
00602       BOOST_SPIRIT_DEBUG_NODE (dateRangeStart);
00603       BOOST_SPIRIT_DEBUG_NODE (dateRangeEnd);
00604       BOOST_SPIRIT_DEBUG_NODE (date);
00605       BOOST_SPIRIT_DEBUG_NODE (timeRangeStart);
00606       BOOST_SPIRIT_DEBUG_NODE (time);
00607       BOOST_SPIRIT_DEBUG_NODE (point_of_sale);
00608       BOOST_SPIRIT_DEBUG_NODE (cabinCode);
00609       BOOST_SPIRIT_DEBUG_NODE (channel);
00610       BOOST_SPIRIT_DEBUG_NODE (advancePurchase);
00611       BOOST_SPIRIT_DEBUG_NODE (saturdayStay);
00612       BOOST_SPIRIT_DEBUG_NODE (changeFees);
00613       BOOST_SPIRIT_DEBUG_NODE (nonRefundable);
00614       BOOST_SPIRIT_DEBUG_NODE (minimumStay);
00615       BOOST_SPIRIT_DEBUG_NODE (fare);
00616       BOOST_SPIRIT_DEBUG_NODE (segment);
00617
00618       }
00619
00620       // Instantiation of rules
00621       boost::spirit::qi::rule<Iterator,
00622                               boost::spirit::ascii::space_type>
00623       start, comments, fare_rule, fare_rule_end
      , fare_key, fare_id, origin,
00624         destination, tripType, dateRangeStart,
       dateRangeEnd, date,
00625         timeRangeStart, timeRangeEnd, time,
      point_of_sale, cabinCode, channel,
00626         advancePurchase, saturdayStay, changeFees
      , nonRefundable, minimumStay,
00627         fare, segment;
00628
00629       // Parser Context
00630       stdair::BomRoot& _bomRoot;
00631       FareRuleStruct& _fareRule;
00632     };
00633
00634   }
00635
00636
00638   //
00639   //  Entry class for the file parser
00640   //
00642
00643   // ////////////////////////////////////////////////////////////////////
00644   FareRuleFileParser::
00645   FareRuleFileParser (stdair::BomRoot& ioBomRoot,
00646                       const stdair::Filename_T& iFilename)
00647     : _filename (iFilename), _bomRoot (ioBomRoot) {
00648     init();
00649   }
00650
00651   // ////////////////////////////////////////////////////////////////////
00652   void FareRuleFileParser::init() {
00653     // Check that the file exists and is readable
00654     const bool doesExistAndIsReadable =
00655       stdair::BasFileMgr::doesExistAndIsReadable (_filename);
00656
00657     if (doesExistAndIsReadable == false) {
00658       STDAIR_LOG_ERROR ("The fare schedule file " << _filename
00659                         << " does not exist or can not be  read.");
00660
00661       throw FareInputFileNotFoundException ("The
       fare file " + _filename
00662                                             + " does not exist or can not be
       read");
00663     }
00664   }
00665
00666   // ////////////////////////////////////////////////////////////////////
00667   void FareRuleFileParser::generateFareRules
      () {
00668
00669     STDAIR_LOG_DEBUG ("Parsing fare input file: " << _filename);
00670
00671     // File to be parsed
00672     const std::string* lFileName = &_filename;
00673     const char *lChar = (*lFileName).c_str();
00674     std::ifstream fileToBeParsed(lChar, std::ios_base::in);
00675
00676     // Check if the filename exist and can be open
00677     if (fileToBeParsed.is_open() == false) {
00678       STDAIR_LOG_ERROR ("The fare file " << _filename << " can not be open."
00679                         << std::endl);
00680
00681       throw FareInputFileNotFoundException ("The
```

```
      file " + _filename
00682                                          + " does not exist or can not be
      read");
00683     }
00684
00685     // Create an input iterator
00686     stdair::base_iterator_t inputBegin (fileToBeParsed);
00687
00688     // Convert input iterator to an iterator usable by spirit parser
00689     stdair::iterator_t
00690       start (boost::spirit::make_default_multi_pass (inputBegin));
00691     stdair::iterator_t end;
00692
00693     // Initialise the parser (grammar) with the helper/staging structure.
00694     FareParserHelper::FareRuleParser<stdair::iterator_t>
      lFPParser(_bomRoot, _fareRule);
00695
00696     // Launch the parsing of the file and, thanks to the doEndFare
00697     // call-back structure, the building of the whole BomRoot BOM
00698     const bool hasParsingBeenSuccesful =
00699        boost::spirit::qi::phrase_parse (start, end, lFPParser,
00700                                         boost::spirit::ascii::space);
00701
00702     if (hasParsingBeenSuccesful == false) {
00703       STDAIR_LOG_ERROR ("Parsing of fare input file: " << _filename
00704                         << " failed");
00705       throw FareFileParsingFailedException ("
      Parsing of fare input file: "
00706                                          + _filename + " failed");
00707     }
00708
00709     if  (start != end) {
00710       STDAIR_LOG_ERROR ("Parsing of fare input file: " << _filename
00711                         << " failed");
00712       throw FareFileParsingFailedException ("
      Parsing of fare input file: "
00713                                          + _filename + " failed");
00714     }
00715
00716     if (hasParsingBeenSuccesful == true && start == end) {
00717       STDAIR_LOG_DEBUG ("Parsing of fare input file: " << _filename
00718       << " succeeded");
00719     }
00720
00721   }
00722
00723 }
```

23.33 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

23.34 FareParserHelper.hpp

```
00001 #ifndef __SIMFQT_CMD_FAREPARSERHELPER_HPP
00002 #define __SIMFQT_CMD_FAREPARSERHELPER_HPP
00003
00004 // ////////////////////////////////////////////////////////////////////
00005 // Import section
00006 // ////////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <string>
00009 // Boost
00010 #include <boost/spirit/include/qi.hpp>
00011 // StdAir
00012 #include <stdair/command/CmdAbstract.hpp>
00013 // Simfqt
00014 #include <simfqt/SIMFQT_Types.hpp>
00015 #include <simfqt/bom/FareRuleStruct.hpp>
00016
00017 // Forward declarations
00018 namespace stdair {
00019   class BomRoot;
00020 }
00021
00022 namespace SIMFQT {
00023
00024   namespace FareParserHelper {
00025
00026     // ////////////////////////////////////////////////////////////////////
00027     //  Semantic actions
00028     // ////////////////////////////////////////////////////
00030
00031     struct ParserSemanticAction {
00033       ParserSemanticAction (FareRuleStruct&);
00035       FareRuleStruct& _fareRule;
00036     };
00037
00039     struct storeFareId : public ParserSemanticAction
      {
00041       storeFareId (FareRuleStruct&);
00043       void operator() (unsigned int,
00044                        boost::spirit::qi::unused_type,
00045                        boost::spirit::qi::unused_type) const;
00046     };
00047
00049     struct storeOrigin : public ParserSemanticAction
      {
00051       storeOrigin (FareRuleStruct&);
00053       void operator() (std::vector<char>,
00054                        boost::spirit::qi::unused_type,
00055                        boost::spirit::qi::unused_type) const;
00056     };
00057
00059     struct storeDestination : public ParserSemanticAction
      {
00061       storeDestination (FareRuleStruct&);
00063       void operator() (std::vector<char>,
```

```
00064                       boost::spirit::qi::unused_type,
00065                       boost::spirit::qi::unused_type) const;
00066     };
00067
00069     struct storeTripType : public ParserSemanticAction
      {
00071       storeTripType (FareRuleStruct&);
00073       void operator() (std::vector<char>,
00074                       boost::spirit::qi::unused_type,
00075                       boost::spirit::qi::unused_type) const;
00076     };
00077
00078
00080     struct storeDateRangeStart : public ParserSemanticAction
      {
00082       storeDateRangeStart (FareRuleStruct&);
00084       void operator() (boost::spirit::qi::unused_type,
00085                       boost::spirit::qi::unused_type,
00086                       boost::spirit::qi::unused_type) const;
00087     };
00088
00090     struct storeDateRangeEnd : public ParserSemanticAction
      {
00092       storeDateRangeEnd (FareRuleStruct&);
00094       void operator() (boost::spirit::qi::unused_type,
00095                       boost::spirit::qi::unused_type,
00096                       boost::spirit::qi::unused_type) const;
00097     };
00098
00100     struct storeStartRangeTime : public ParserSemanticAction
      {
00102       storeStartRangeTime (FareRuleStruct&);
00104       void operator() (boost::spirit::qi::unused_type,
00105                       boost::spirit::qi::unused_type,
00106                       boost::spirit::qi::unused_type) const;
00107     };
00108
00110     struct storeEndRangeTime : public ParserSemanticAction
      {
00112       storeEndRangeTime (FareRuleStruct&);
00114       void operator() (boost::spirit::qi::unused_type,
00115                       boost::spirit::qi::unused_type,
00116                       boost::spirit::qi::unused_type) const;
00117     };
00118
00120     struct storePOS : public ParserSemanticAction {
00122       storePOS (FareRuleStruct&);
00124       void operator() (std::vector<char>,
00125                       boost::spirit::qi::unused_type,
00126                       boost::spirit::qi::unused_type) const;
00127     };
00128
00130     struct storeCabinCode : public ParserSemanticAction
      {
00132       storeCabinCode  (FareRuleStruct&);
00134       void operator() (char,
00135                       boost::spirit::qi::unused_type,
00136                       boost::spirit::qi::unused_type) const;
00137     };
00138
00140     struct storeChannel : public ParserSemanticAction
      {
00142       storeChannel (FareRuleStruct&);
00144       void operator() (std::vector<char>,
00145                       boost::spirit::qi::unused_type,
00146                       boost::spirit::qi::unused_type) const;
00147     };
00148
00150     struct storeAdvancePurchase : public
    ParserSemanticAction {
00152       storeAdvancePurchase (FareRuleStruct&);
00154       void operator() (unsigned int,
00155                       boost::spirit::qi::unused_type,
00156                       boost::spirit::qi::unused_type) const;
00157     };
00158
00160     struct storeSaturdayStay : public ParserSemanticAction
      {
00162       storeSaturdayStay (FareRuleStruct&);
00164       void operator() (char,
00165                       boost::spirit::qi::unused_type,
00166                       boost::spirit::qi::unused_type) const;
00167     };
00168
00170     struct storeChangeFees : public ParserSemanticAction
      {
00172       storeChangeFees (FareRuleStruct&);
```

```
00174       void operator() (char,
00175                        boost::spirit::qi::unused_type,
00176                        boost::spirit::qi::unused_type) const;
00177     };
00178
00180     struct storeNonRefundable : public ParserSemanticAction
          {
00182       storeNonRefundable (FareRuleStruct&);
00184       void operator() (char,
00185                        boost::spirit::qi::unused_type,
00186                        boost::spirit::qi::unused_type) const;
00187     };
00188
00190     struct storeMinimumStay : public ParserSemanticAction
          {
00192       storeMinimumStay (FareRuleStruct&);
00194       void operator() (unsigned int,
00195                        boost::spirit::qi::unused_type,
00196                        boost::spirit::qi::unused_type) const;
00197     };
00198
00200     struct storeFare : public ParserSemanticAction
          {
00202       storeFare (FareRuleStruct&);
00204       void operator() (double,
00205                        boost::spirit::qi::unused_type,
00206                        boost::spirit::qi::unused_type) const;
00207     };
00208
00210     struct storeAirlineCode : public ParserSemanticAction
          {
00212       storeAirlineCode (FareRuleStruct&);
00214       void operator() (std::vector<char>,
00215                        boost::spirit::qi::unused_type,
00216                        boost::spirit::qi::unused_type) const;
00217     };
00218
00220     struct storeClass : public ParserSemanticAction
          {
00222       storeClass (FareRuleStruct&);
00224       void operator() (std::vector<char>,
00225                        boost::spirit::qi::unused_type,
00226                        boost::spirit::qi::unused_type) const;
00227     };
00228
00230     struct doEndFare : public ParserSemanticAction
          {
00232       doEndFare (stdair::BomRoot&, FareRuleStruct&);
00234       void operator() (boost::spirit::qi::unused_type,
00235                        boost::spirit::qi::unused_type,
00236                        boost::spirit::qi::unused_type) const;
00238       stdair::BomRoot& _bomRoot;
00239     };
00240
00241   }
00242
00244   //
00245   //  Entry class for the file parser
00246   //
00248
00254   class FareRuleFileParser : public stdair::CmdAbstract {
00255   public:
00257     FareRuleFileParser (stdair::BomRoot& ioBomRoot,
00258                         const stdair::Filename_T& iFilename);
00259
00261     void generateFareRules ();
00262
00263   private:
00265     void init();
00266
00267   private:
00268     // Attributes
00270     stdair::Filename_T _filename;
00271
00273     stdair::BomRoot& _bomRoot;
00274
00276     FareRuleStruct _fareRule;
00277   };
00278
00279 }
00280 #endif // __SIMFQT_CMD_FAREPARSERHELPER_HPP
```

23.35 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

23.36 FareQuoter.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 // StdAir
00008 #include <stdair/basic/BasConst_BomDisplay.hpp>
00009 #include <stdair/basic/BasConst_Request.hpp>
00010 #include <stdair/bom/BomKeyManager.hpp>
00011 #include <stdair/bom/ParsedKey.hpp>
00012 #include <stdair/bom/BomManager.hpp>
00013 #include <stdair/bom/BomRoot.hpp>
00014 #include <stdair/bom/InventoryKey.hpp>
00015 #include <stdair/bom/FlightDateKey.hpp>
00016 #include <stdair/bom/SegmentDateKey.hpp>
00017 #include <stdair/bom/AirlineClassList.hpp>
00018 #include <stdair/bom/AirportPair.hpp>
00019 #include <stdair/bom/PosChannel.hpp>
00020 #include <stdair/bom/DatePeriod.hpp>
00021 #include <stdair/bom/TimePeriod.hpp>
00022 #include <stdair/bom/FareFeatures.hpp>
00023 #include <stdair/bom/BookingRequestStruct.hpp>
00024 #include <stdair/bom/TravelSolutionStruct.hpp>
00025 #include <stdair/service/Logger.hpp>
00026 #include <stdair/bom/key_types.hpp>
00027 // SimFQT
00028 #include <simfqt/SIMFQT_Types.hpp>
00029 #include <simfqt/command/FareQuoter.hpp>
00030
00031 namespace SIMFQT {
00032
00033   bool FareQuoter::_atLeastOneAvailableDateRule = false;
00034   bool FareQuoter::_atLeastOneAvailablePosChannel = false;
00035   bool FareQuoter::_atLeastOneAvailableTimeRule = false;
00036   bool FareQuoter::_atLeastOneAvailableFeaturesRule = false;
```

```
00037   bool FareQuoter::_atLeastOneAvailableAirlineClassRule= false;
00038
00039   // ////////////////////////////////////////////////////////////////////
00040   FareQuoter::FareQuoter() {
00041     assert (false);
00042   }
00043
00044   // ////////////////////////////////////////////////////////////////////
00045   FareQuoter::FareQuoter(const FareQuoter&) {
00046     assert (false);
00047   }
00048
00049   // ////////////////////////////////////////////////////////////////////
00050   FareQuoter::~FareQuoter() {
00051   }
00052
00053   // ////////////////////////////////////////////////////////////////////
00054   void FareQuoter::reset() {
00055     _atLeastOneAvailableDateRule = false;
00056     _atLeastOneAvailablePosChannel = false;
00057     _atLeastOneAvailableTimeRule = false;
00058     _atLeastOneAvailableFeaturesRule = false;
00059     _atLeastOneAvailableAirlineClassRule = false;
00060   }
00061
00062
00063   // ////////////////////////////////////////////////////////////////////
00064   void FareQuoter::
00065   priceQuote (const stdair::BookingRequestStruct& iBookingRequest,
00066               stdair::TravelSolutionList_T& ioTravelSolutionList,
00067               const stdair::BomRoot& iBomRoot) {
00068
00069     // Do an independent price quote for each travel solution related to the
00070     // booking request.
00071     for (stdair::TravelSolutionList_T::iterator itTravelSolution =
00072            ioTravelSolutionList.begin();
00073          itTravelSolution != ioTravelSolutionList.end(); ++itTravelSolution) {
00074       reset();
00075       // Select a travel solution.
00076       stdair::TravelSolutionStruct& lTravelSolutionStruct = *itTravelSolution;
00077       // Price quote the travel solution into question.
00078       priceQuote (iBookingRequest, lTravelSolutionStruct, iBomRoot);
00079     }
00080   }
00081
00082   // ////////////////////////////////////////////////////////////////////
00083   void FareQuoter::
00084   priceQuote (const stdair::BookingRequestStruct& iBookingRequest,
00085               stdair::TravelSolutionStruct& ioTravelSolution,
00086               const stdair::BomRoot& iBomRoot) {
00087
00088     // Get the origin of the first segment in order to get the origin of
00089     // the solution.
00090     const stdair::ParsedKey& lFirstSegmentKey =
00091       getFirstSPParsedKey(ioTravelSolution);
00092     const stdair::AirportCode_T& lOrigin = lFirstSegmentKey._boardingPoint;
00093
00094     // Get the destination of the last segment in order to get the
00095     // destination of the solution.
00096     const stdair::ParsedKey& lLastSegmentKey =
00097       getLastSPParsedKey(ioTravelSolution);
00098     const stdair::AirportCode_T& lDestination = lLastSegmentKey._offPoint;
00099
00100    // Construct the Airport pair stream of the segment path.
00101    const stdair::AirportPairKey lAirportPairKey (lOrigin, lDestination);
00102
00103    // Search for the fare rules having the same origin and destination
       airports
00104    // as the travel solution
00105    const stdair::AirportPair* lAirportPair_ptr = stdair::BomManager::
00106      getObjectPtr<stdair::AirportPair> (iBomRoot, lAirportPairKey.toString());

00107
00108    // If no fare rule has the same origin and destination airports, the
       pricing
00109    // is not possible, throw an exception.
00110    if (lAirportPair_ptr == NULL) {
00111      STDAIR_LOG_ERROR ("No available fare rule for the "
00112                        << "Origin-Destination pair: "
00113                        << lAirportPairKey.toString());
00114      throw AirportPairNotFoundException ("No available fare rule for "
00115                                          "the Origin-Destination pair: "
00116                                          + lAirportPairKey.toString());
00117    }
00118    // Sanity check.
00119    assert(lAirportPair_ptr != NULL);
00120
```

```
00121     // Fare rule(s) with the same origin and destination airports exist(s), now
00122     // the date range need to be checked.
00123     const stdair::AirportPair& lAirportPair = *lAirportPair_ptr;
00124     priceQuote(iBookingRequest, ioTravelSolution, lAirportPair);
00125
00126     if (_atLeastOneAvailableAirlineClassRule == false) {
00127       displayMissingFareRuleMessage(iBookingRequest, ioTravelSolution);
00128     }
00129   }
00130
00131   // ////////////////////////////////////////////////////////////////////
00132   void FareQuoter::
00133   priceQuote (const stdair::BookingRequestStruct& iBookingRequest,
00134               stdair::TravelSolutionStruct& ioTravelSolution,
00135               const stdair::AirportPair& iAirportPair) {
00136
00137     // Get the first segment path parsed key.
00138     const stdair::ParsedKey lFirstSPParsedKey =
00139       getFirstSPParsedKey(ioTravelSolution);
00140
00141     // Get the date of the first segment date key.
00142     const stdair::FlightDateKey& lFlightDateKey =
00143       lFirstSPParsedKey.getFlightDateKey();
00144     const stdair::Date_T& lSPDate = lFlightDateKey.getDepartureDate();
00145
00146     // Get the list of the fare date ranges.
00147     const stdair::DatePeriodList_T& lFareDatePeriodList =
00148       stdair::BomManager::getList<stdair::DatePeriod> (iAirportPair);
00149
00150     // Browse the list of the fare rules date range.
00151     for (stdair::DatePeriodList_T::const_iterator itDateRange =
00152            lFareDatePeriodList.begin();
00153          itDateRange != lFareDatePeriodList.end(); ++itDateRange) {
00154
00155       const stdair::DatePeriod* lCurrentFareDatePeriod_ptr = *itDateRange ;
00156       assert (lCurrentFareDatePeriod_ptr != NULL);
00157
00158       // Select the fare rules having a corresponding date range.
00159       const bool isDepartureDateValid =
00160         lCurrentFareDatePeriod_ptr->isDepartureDateValid (lSPDate);
00161
00162       // If a fare rule has a corresponding date range, its channel and
      position
00163       // need to be checked.
00164       if (isDepartureDateValid == true) {
00165         _atLeastOneAvailableDateRule = true;
00166         const stdair::DatePeriod& lCurrentFareDatePeriod =
00167           *lCurrentFareDatePeriod_ptr;
00168         priceQuote (iBookingRequest, ioTravelSolution,
00169                     lCurrentFareDatePeriod, iAirportPair);
00170       }
00171     }
00172
00173   }
00174
00175   // ////////////////////////////////////////////////////////////////////
00176   void FareQuoter::
00177   priceQuote (const stdair::BookingRequestStruct& iBookingRequest,
00178               stdair::TravelSolutionStruct& ioTravelSolution,
00179               const stdair::DatePeriod& iFareDatePeriod,
00180               const stdair::AirportPair& iAirportPair) {
00181
00182     // Get the point-of-sale of the booking request.
00183     const stdair::CityCode_T& lPointOfSale = iBookingRequest.getPOS();
00184
00185     // Get the booking request channel.
00186     const stdair::ChannelLabel_T& lChannel =
00187       iBookingRequest.getBookingChannel();
00188
00189     // Construct the corresponding POS-channel primary key.
00190     const stdair::PosChannelKey lFarePosChannelKey (lPointOfSale, lChannel);
00191
00192     // Search for the fare rules having the same point-of-sale and channel as
00193     // the travel solution.
00194     const stdair::PosChannelList_T lFarePosChannelList =
00195       stdair::BomManager::getList<stdair::PosChannel> (iFareDatePeriod);
00196
00197     // Browse the list of the fare rules pos channel.
00198     for (stdair::PosChannelList_T::const_iterator itPosChannel =
00199            lFarePosChannelList.begin();
00200          itPosChannel != lFarePosChannelList.end();
00201          ++itPosChannel) {
00202       const stdair::PosChannel* lCurrentFarePosChannel_ptr = *itPosChannel;
00203       assert (lCurrentFarePosChannel_ptr != NULL);
00204
00205       // Get the point-of-sale and channel of the current fare rule.
00206       const stdair::CityCode_T& lCurrentPointOfSale =
```

```
00207         lCurrentFarePosChannel_ptr->getPos();
00208       const stdair::ChannelLabel_T& lCurrentChannel =
00209         lCurrentFarePosChannel_ptr->getChannel();
00210
00211       // Select the fare rules having a corresponding pos channel.
00212       if (lCurrentPointOfSale == lPointOfSale || lCurrentPointOfSale ==
    stdair::DEFAULT_POS) {
00213         if (lCurrentChannel == lChannel || lCurrentChannel ==
    stdair::DEFAULT_CHANNEL) {
00214           _atLeastOneAvailablePosChannel = true;
00215           // Fare rule(s) with the same point-of-sale and channel exist(s), now
00216           // the time range need to be checked.
00217           const stdair::PosChannel& lFarePosChannel= *
    lCurrentFarePosChannel_ptr;
00218           STDAIR_LOG_DEBUG (lCurrentPointOfSale + " " + lCurrentChannel);
00219           priceQuote (iBookingRequest, ioTravelSolution, lFarePosChannel);
00220         }
00221       }
00222     }
00223
00224   }
00225
00226   // ////////////////////////////////////////////////////////////////////
00227   void FareQuoter::
00228   priceQuote (const stdair::BookingRequestStruct& iBookingRequest,
00229               stdair::TravelSolutionStruct& ioTravelSolution,
00230               const stdair::PosChannel& iFarePosChannel) {
00231
00232     // Get the first segment path parsed key.
00233     const stdair::ParsedKey lFirstSPParsedKey =
00234       getFirstSPParsedKey(ioTravelSolution);
00235
00236     // Get the segment boarding time of the segment path.
00237     const stdair::Duration_T& lSPTime = lFirstSPParsedKey.getBoardingTime();
00238
00239     // Get the list of the fare rules time period.
00240     const stdair::TimePeriodList_T& lFareTimePeriodList =
00241       stdair::BomManager::getList<stdair::TimePeriod> (iFarePosChannel);
00242
00243     // Browse the list of the fare rules time range.
00244     for (stdair::TimePeriodList_T::const_iterator itTimeRange =
00245            lFareTimePeriodList.begin();
00246          itTimeRange != lFareTimePeriodList.end();
00247          ++itTimeRange) {
00248       const stdair::TimePeriod* lCurrentFareTimePeriod_ptr = *itTimeRange ;
00249       assert (lCurrentFareTimePeriod_ptr != NULL);
00250
00251       // Select the fare rules having a corresponding time range.
00252       const bool isDepartureTimeValid =
00253         lCurrentFareTimePeriod_ptr->isDepartureTimeValid (lSPTime);
00254
00255       // If a fare rule has a corresponding time range, its advanced purchase,
00256       // trip type and minimum stay duration need to be checked.
00257       if (isDepartureTimeValid) {
00258         _atLeastOneAvailableTimeRule = true;
00259         const stdair::TimePeriod& lCurrentFareTimePeriod =
00260           *lCurrentFareTimePeriod_ptr;
00261         priceQuote (iBookingRequest, ioTravelSolution,
00262                     lCurrentFareTimePeriod, iFarePosChannel);
00263       }
00264     }
00265
00266   }
00267
00268   // ////////////////////////////////////////////////////////////////////
00269   void FareQuoter::
00270   priceQuote (const stdair::BookingRequestStruct& iBookingRequest,
00271               stdair::TravelSolutionStruct& ioTravelSolution,
00272               const stdair::TimePeriod& iFareTimePeriod,
00273               const stdair::PosChannel& iFarePosChannel) {
00274
00275     // Get the stay duration of the booking request.
00276     const stdair::DayDuration_T& lStayDuration=
00277       iBookingRequest.getStayDuration();
00278
00279     // Get the booking request trip type.
00280     const stdair::TripType_T& lTripType =
00281       iBookingRequest.getTripType();
00282
00283     // Get the booking request date time.
00284     const stdair::DateTime_T& lRequestDateTime =
00285       iBookingRequest.getRequestDateTime();
00286
00287     // Get the referenced departure date of the segment path.
00288     const stdair::ParsedKey lFirstSPParsedKey =
00289       getFirstSPParsedKey(ioTravelSolution);
00290     const stdair::Date_T& lSPDate =
```

```
00291       lFirstSPParsedKey.getFlightDateKey().getDepartureDate();
00292
00293     // Get the segment boarding time of the segment path.
00294     const stdair::Duration_T& lSPTime = lFirstSPParsedKey.getBoardingTime();
00295
00296     // Construct the date-time type correponding to the flight date
00297     const stdair::DateTime_T lSPDateTime (lSPDate, lSPTime);
00298
00299     bool isTripTypeValid = false;
00300     bool isStayDurationValid = false;
00301     bool isAdvancePurchaseValid = false;
00302
00303     // Get the list of the fare features (if such list exists: the POS
00304     // and channel couple can be only present in a yield rule).
00305     const bool hasFareFeaturesList =
00306       stdair::BomManager::hasList<stdair::FareFeatures> (iFareTimePeriod);
00307     if (hasFareFeaturesList == false) {
00308       return;
00309     }
00310     assert (hasFareFeaturesList == true);
00311     const stdair::FareFeaturesList_T& lFareFeaturesList =
00312       stdair::BomManager::getList<stdair::FareFeatures> (iFareTimePeriod);
00313
00314     // Browse the list of the fare rules features.
00315     for (stdair::FareFeaturesList_T::const_iterator itFareFeatures =
00316            lFareFeaturesList.begin();
00317          itFareFeatures != lFareFeaturesList.end();
00318          ++itFareFeatures) {
00319       const stdair::FareFeatures* lCurrentFareFeatures_ptr =
00320         *itFareFeatures;
00321       assert (lCurrentFareFeatures_ptr != NULL);
00322
00323       // Does the current fare features correspond to a correct trip
00324       // type?
00325       isTripTypeValid =
00326         lCurrentFareFeatures_ptr->isTripTypeValid (lTripType);
00327       // Does the current fare features correspond to a correct stay
00328       // duration?
00329       isStayDurationValid =
00330         lCurrentFareFeatures_ptr->isStayDurationValid (lStayDuration);
00331       // Does the current fare features correspond to a correct advanced
00332       // purchase?
00333       isAdvancePurchaseValid = lCurrentFareFeatures_ptr->
00334         isAdvancePurchaseValid (lRequestDateTime,
00335                                 lSPDateTime);
00336
00337       // Search for the fare rules having corresponding features.
00338       if (isStayDurationValid && isAdvancePurchaseValid && isTripTypeValid){
00339         _atLeastOneAvailableFeaturesRule = true;
00340         // Create a fare structure for the travel solution.
00341         stdair::FareOptionStruct lFareOption;
00342         const stdair::ChangeFees_T& lChangeFees =
00343           lCurrentFareFeatures_ptr->getChangeFees();
00344         // Set the fare change fees.
00345         lFareOption.setChangeFees (lChangeFees);
00346         const stdair::NonRefundable_T& lNonRefundable =
00347           lCurrentFareFeatures_ptr->getRefundableOption();
00348         // Set the fare refundable option.
00349         lFareOption.setNonRefundable (lNonRefundable);
00350         const stdair::SaturdayStay_T& lSaturdayStay =
00351           lCurrentFareFeatures_ptr->getSaturdayStay();
00352         // Set the fare saturday night stay option.
00353         lFareOption.setSaturdayStay (lSaturdayStay);
00354         const stdair::FareFeatures& lCurrentFareFeatures =
00355           *lCurrentFareFeatures_ptr;
00356         priceQuote (iBookingRequest, ioTravelSolution,
00357                     lCurrentFareFeatures, iFarePosChannel,
00358                     lFareOption);
00359       }
00360     }
00361
00362   }
00363
00364
00365   // ////////////////////////////////////////////////////////////////////
00366   void FareQuoter::
00367   priceQuote (const stdair::BookingRequestStruct& iBookingRequest,
00368               stdair::TravelSolutionStruct& ioTravelSolution,
00369               const stdair::FareFeatures& iFareFeatures,
00370               const stdair::PosChannel& iFarePosChannel,
00371               stdair::FareOptionStruct& iFareOption) {
00372
00373     // Get the segment-path of the travel solution.
00374     const stdair::SegmentPath_T& lSegmentPath =
00375       ioTravelSolution.getSegmentPath();
00376
00377     // Get the list of the fare rules.
```

```
00378     const stdair::AirlineClassListList_T& lAirlineClassListList =
00379       stdair::BomManager::getList<stdair::AirlineClassList> (iFareFeatures);
00380
00381     bool lCorrectAirlineRule = false;
00382     bool lAtLeastOneDifferentAirline = false;
00383
00384     // Browse the list of airline code list and search for the fare rules
00385     // having a corresponding airline list.
00386     for (stdair::AirlineClassListList_T::const_iterator itAirlineClassList =
00387            lAirlineClassListList.begin();
00388          itAirlineClassList != lAirlineClassListList.end();
00389          ++itAirlineClassList) {
00390       const stdair::AirlineClassList* lCurrentAirlineClassList_ptr =
00391         *itAirlineClassList;
00392       assert (lCurrentAirlineClassList_ptr != NULL);
00393
00394       lCorrectAirlineRule = true;
00395       lAtLeastOneDifferentAirline = false;
00396
00397       const stdair::ClassList_StringList_T lClassList_StringList =
00398         lCurrentAirlineClassList_ptr->getAirlineCodeList();
00399
00400       // Compare the segment path airline list with the fare rule airline list.
00401       if (lClassList_StringList.size() == lSegmentPath.size()) {
00402         // If the two sizes are equal, we need to compare the airline codes.
00403         stdair::SegmentPath_T::const_iterator itSegmentPath =
00404           lSegmentPath.begin();
00405
00406         stdair::ClassList_StringList_T::const_iterator itClassList_String =
00407           lClassList_StringList.begin();
00408         // Browse the segment path airline code list (while the segment path
00409         // airline list is equal to the fare rule airline list).
00410         while (itSegmentPath != lSegmentPath.end()
00411                && lAtLeastOneDifferentAirline == false) {
00412
00413           // Get the segment airline code.
00414           const std::string lSegmentDateKey = *itSegmentPath;
00415           const stdair::ParsedKey& lParsedKey =
00416             stdair::BomKeyManager::extractKeys (lSegmentDateKey);
00417           const stdair::InventoryKey& lInventoryKey =
00418             lParsedKey.getInventoryKey();
00419           const stdair::AirlineCode_T& lSegmentAirlineCode =
00420             lInventoryKey.getAirlineCode();
00421
00422           // Get the fare rule airline code.
00423           const stdair::AirlineCode_T& lFareRuleAirlineCode =
00424             *itClassList_String;
00425
00426           if (lSegmentAirlineCode != lFareRuleAirlineCode) {
00427             lAtLeastOneDifferentAirline = true;
00428           }
00429           itSegmentPath++;
00430           itClassList_String++;
00431         }
00432
00433       } else {
00434         // If the two sizes are different, the fare rule does not match the
00435         // travel solution into question.
00436         lCorrectAirlineRule = false;
00437       }
00438
00439       // If one segment airline code and one fare rule airline code are
00440       // different then the fare rule does not match the travel solution.
00441       if (lAtLeastOneDifferentAirline == true) {
00442         lCorrectAirlineRule = false;
00443       }
00444
00445       // If the current fare rule is a match, add the fare option structure
00446       // to the travel solution into question.
00447       if (lCorrectAirlineRule == true) {
00448         _atLeastOneAvailableAirlineClassRule = true;
00449         // Get the booking request trip type.
00450         const stdair::TripType_T& lTripType =
00451           iBookingRequest.getTripType();
00452
00453         // Get the travel fare.
00454         stdair::Fare_T lFare =
00455           lCurrentAirlineClassList_ptr->getFare();
00456         // If the trip into question is the inbound or outbound part of a round
    trip,
00457         // the applicable fare is a half RT fare.
00458         if (lTripType == "RI" || lTripType == "RO") {
00459           lFare /= 2;
00460         }
00461         // Set the travel fare option.
00462         iFareOption.setFare (lFare);
00463         // Copy the class path list into the fare option.
```

```
00464         const stdair::ClassList_StringList_T& lClassCodeList =
00465           lCurrentAirlineClassList_ptr->getClassCodeList();
00466         for (stdair::ClassList_StringList_T::const_iterator itClassCodeList =
00467                lClassCodeList.begin();
00468              itClassCodeList != lClassCodeList.end(); ++itClassCodeList ) {
00469           const stdair::ClassList_String_T& lClassCodeList = *itClassCodeList;
00470           iFareOption.addClassList (lClassCodeList);
00471         }
00472
00473         // Add the fare option to the travel solution into question.
00474         ioTravelSolution.addFareOption (iFareOption);
00475
00476         // DEBUG
00477         STDAIR_LOG_DEBUG (ioTravelSolution.describeSegmentPath()
00478                           << ". A corresponding fare option for the '"
00479                           << lCurrentAirlineClassList_ptr->describeKey()
00480                           << "' class is: " << iFareOption);
00481
00482         iFareOption.emptyClassList();
00483       }
00484     }
00485
00486   }
00487
00488   // ////////////////////////////////////////////////////////////////////
00489   stdair::ParsedKey FareQuoter::
00490   getFirstSPParsedKey (stdair::TravelSolutionStruct& ioTravelSolution) {
00491
00492     // Get the segment-path of the travel solution.
00493     const stdair::SegmentPath_T& lSegmentPath =
00494       ioTravelSolution.getSegmentPath();
00495
00496     // Get the number of segments of the travel solution.
00497     const stdair::NbOfSegments_T& lNbSegments = lSegmentPath.size();
00498
00499     // Sanity check: there is at least one segment in the travel solution.
00500     assert (lNbSegments >= 1);
00501
00502     // Get the first segment of the travel solution.
00503     const std::string& lFirstSegmentDateKey = lSegmentPath.front();
00504
00505     // Get the parsed key of the first segment of the travel solution.
00506     const stdair::ParsedKey& lFirstSegmentParsedKey =
00507       stdair::BomKeyManager::extractKeys (lFirstSegmentDateKey);
00508
00509     return lFirstSegmentParsedKey;
00510
00511   }
00512
00513   // ////////////////////////////////////////////////////////////////////
00514   stdair::ParsedKey FareQuoter::
00515   getLastSPParsedKey (stdair::TravelSolutionStruct& ioTravelSolution) {
00516
00517     // Get the segment-path of the travel solution.
00518     const stdair::SegmentPath_T& lSegmentPath =
00519       ioTravelSolution.getSegmentPath();
00520
00521     // Get the number of segments of the travel solution.
00522     const stdair::NbOfSegments_T& lNbSegments = lSegmentPath.size();
00523
00524     // Sanity check: there is at least one segment in the travel solution.
00525     assert (lNbSegments >= 1);
00526
00527     // Get the last segment of the travel solution.
00528     const std::string& lLastSegmentDateKey = lSegmentPath.back();
00529
00530     // Get the parsed key of the last segment of the travel solution.
00531     const stdair::ParsedKey& lLastSegmentParsedKey =
00532       stdair::BomKeyManager::extractKeys (lLastSegmentDateKey);
00533
00534     return lLastSegmentParsedKey;
00535
00536   }
00537
00538   // ////////////////////////////////////////////////////////////////////
00539   void FareQuoter::
00540   displayMissingFareRuleMessage (const stdair::BookingRequestStruct&
    iBookingRequest,
00541                                  stdair::TravelSolutionStruct& ioTravelSolution
    ) {
00542
00543     // Get the origin of the first segment in order to get the origin of
00544     // the solution.
00545     const stdair::ParsedKey lFirstSPParsedKey =
00546       getFirstSPParsedKey(ioTravelSolution);
00547     const stdair::AirportCode_T& lOrigin = lFirstSPParsedKey._boardingPoint;
00548
```

```
00549     // Get the destination of the last segment in order to get the
00550     // destination of the solution.
00551     const stdair::ParsedKey& lLastSegmentKey =
00552       getLastSPParsedKey(ioTravelSolution);
00553     const stdair::AirportCode_T& lDestination = lLastSegmentKey._offPoint;
00554
00555     // Construct the Airport pair stream of the segment path.
00556     const stdair::AirportPairKey lAirportPairKey (lOrigin, lDestination);
00557
00558     // Get the date of the first segment date key.
00559     const stdair::FlightDateKey& lFlightDateKey =
00560       lFirstSPParsedKey.getFlightDateKey();
00561
00562     // Get the point-of-sale of the booking request.
00563     const stdair::CityCode_T& lPointOfSale = iBookingRequest.getPOS();
00564     // Get the booking request channel.
00565     const stdair::ChannelLabel_T& lChannel =
00566       iBookingRequest.getBookingChannel();
00567     // Construct the corresponding POS-channel primary key.
00568     const stdair::PosChannelKey lFarePosChannelKey (lPointOfSale, lChannel);
00569
00570     // Get the booking request date time.
00571     const stdair::DateTime_T& lRequestDateTime =
00572       iBookingRequest.getRequestDateTime();
00573
00574     // If no fare rule has a corresponding date range, the pricing is not
00575     // possible, throw an exception.
00576     if (_atLeastOneAvailableDateRule == false) {
00577       const stdair::SegmentDateKey lSegmentDateKey =
00578         lFirstSPParsedKey.getSegmentKey();
00579       STDAIR_LOG_ERROR ("No available fare rule corresponding to the "
00580                         "flight date " << lFlightDateKey.toString()
00581                         << " and the Origin-Destination pair: "
00582                         << lSegmentDateKey.toString());
00583       throw FlightDateNotFoundException ("No available fare rule for the "
00584                                          "flight date "
00585                                          + lFlightDateKey.toString()
00586                                          + " and the Origin-Destination pair: "
00587                                          + lSegmentDateKey.toString());
00588     }
00589     // If no fare rule has a corresponding pos channel, the pricing is not
      possible,
00590     // throw an exception.
00591     else if (_atLeastOneAvailablePosChannel == false) {
00592       STDAIR_LOG_ERROR ("No available fare rule corresponding to the "
00593                         "point of sale " << lPointOfSale
00594                         << ", to the channel " << lChannel
00595                         << ", to the flight date "
00596                         << lFlightDateKey.toString()
00597                         << " and to the Origin-Destination pair: "
00598                         << lAirportPairKey.toString());
00599       throw PosOrChannelNotFoundException ("No available fare rule for the "
00600                                            "point of sale " + lPointOfSale
00601                                            + ", the channel " + lChannel
00602                                            + ", the flight date "
00603                                            + lFlightDateKey.toString()
00604                                            + " and the Origin-Destination pair:
      "
00605                                            + lAirportPairKey.toString());
00606     }
00607     // If no fare rule has a corresponding time range, the pricing is not
      possible,
00608     // throw an exception.
00609     else if (_atLeastOneAvailableTimeRule == false) {
00610       STDAIR_LOG_ERROR ("No available fare rule corresponding to '"
00611                         << lFirstSPParsedKey.toString() << "' (parsed key) and
      to '"
00612                         << lFarePosChannelKey.toString() << "' (POS and
      channel)");
00613       throw FlightTimeNotFoundException ("No available fare rule corresponding
      "
00614                                          "to '" + lFirstSPParsedKey.toString()
00615                                          + "' (parsed key) and to '"
00616                                          + lFarePosChannelKey.toString()
00617                                          + "' (POS and channel)");
00618     }
00619     // If no fare rule matches the advance purchase, trip type and stay
00620     // duration criterion, the pricing is not possible, throw an exception.
00621     else if (_atLeastOneAvailableFeaturesRule == false) {
00622       // Get the stay duration of the booking request.
00623       const stdair::DayDuration_T& lStayDuration=
00624         iBookingRequest.getStayDuration();
00625       std::ostringstream lStayDurationStream;
00626       lStayDurationStream << lStayDuration;
00627       const std::string lStayDurationString (lStayDurationStream.str());
00628
00629       // Get the booking request trip type.
```

```
00630       const stdair::TripType_T& lTripType =
00631         iBookingRequest.getTripType();
00632
00633       STDAIR_LOG_ERROR ("No available fare rule corresponding to a "
00634                         "trip type " << lTripType
00635                         << ", to a stay duration of " <<  lStayDurationString
00636                         << ", to a request date time of " << lRequestDateTime
00637                         << ", to '" << lFirstSPParsedKey.toString()
00638                         << "' (parsed key) and to '"
00639                         << lFarePosChannelKey << "' (POS and channel)");
00640       throw FeaturesNotFoundException ("No available fare rule corresponding to
      a "
00641                                        "trip type " + lTripType
00642                                        + ", to a stay duration of "
00643                                        + lStayDurationString
00644                                        + ", to a request date time of "
00645                                        + boost::posix_time::to_simple_string(
      lRequestDateTime)
00646                                        + ", to '" + lFirstSPParsedKey.toString(
      )
00647                                        + "' (parsed key) and to '"
00648                                        + lFarePosChannelKey.toString()
00649                                        + "' (POS and channel)");
00650     }
00651     assert (_atLeastOneAvailableAirlineClassRule == false);
00652     // If no fare rule matches the airline class path, the pricing is not
00653     // possible, throw an exception.
00654     STDAIR_LOG_ERROR ("No available fare rule corresponding to '"
00655                       << lFirstSPParsedKey .toString() << "' (parsed key), to '
      "
00656                       << iBookingRequest.describe()
00657                       << "' (booking request) and to '"
00658                       << lFarePosChannelKey.toString() << "' (POS and channel)"
      );
00659     throw AirlineNotFoundException ("No available fare rule corresponding to '"
00660                                     + lFirstSPParsedKey .toString()
00661                                     + "' (parsed key), to '"
00662                                     + iBookingRequest.describe()
00663                                     + "' (booking request) and to '"
00664                                     + lFarePosChannelKey.toString()
00665                                     + "' (POS and channel)");
00666   }
00667 }
00668
```

23.37 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

23.38 FareQuoter.hpp

```
00001 #ifndef __SIMFQT_CMD_FAREQUOTER_HPP
00002 #define __SIMFQT_CMD_FAREQUOTER_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // StdAir
00008 #include <stdair/stdair_basic_types.hpp>
```

```
00009 #include <stdair/bom/TravelSolutionTypes.hpp>
00010
00012 namespace stdair {
00013   class BomRoot;
00014   struct BookingRequestStruct;
00015   struct TravelSolutionStruct;
00016   struct ParsedKey;
00017   class AirportPair;
00018   class PosChannel;
00019   class DatePeriod;
00020   class TimePeriod;
00021   class FareFeatures;
00022 }
00023
00024 namespace SIMFQT {
00025
00029   class FareQuoter {
00032     friend class SIMFQT_Service;
00033
00034   private:
00035     // ////////////////// Business support methods ///////////////
00045     static void priceQuote (const stdair::BookingRequestStruct&,
00046                             stdair::TravelSolutionList_T&,
00047                             const stdair::BomRoot&);
00048
00060     static void priceQuote (const stdair::BookingRequestStruct&,
00061                             stdair::TravelSolutionStruct&,
00062                             const stdair::BomRoot&);
00063
00074     static void priceQuote (const stdair::BookingRequestStruct&,
00075                             stdair::TravelSolutionStruct&,
00076                             const stdair::AirportPair&);
00077
00092     static void priceQuote (const stdair::BookingRequestStruct&,
00093                             stdair::TravelSolutionStruct&,
00094                             const stdair::DatePeriod&,
00095                             const stdair::AirportPair&);
00096
00108     static void priceQuote (const stdair::BookingRequestStruct&,
00109                             stdair::TravelSolutionStruct&,
00110                             const stdair::PosChannel&);
00111
00126     static void priceQuote (const stdair::BookingRequestStruct&,
00127                             stdair::TravelSolutionStruct&,
00128                             const stdair::TimePeriod&,
00129                             const stdair::PosChannel&);
00130
00148     static void priceQuote (const stdair::BookingRequestStruct&,
00149                             stdair::TravelSolutionStruct&,
00150                             const stdair::FareFeatures&,
00151                             const stdair::PosChannel&,
00152                             stdair::FareOptionStruct&);
00153
00157     static void reset ();
00158
00168     static void displayMissingFareRuleMessage (const
    stdair::BookingRequestStruct&,
00169                                                stdair::TravelSolutionStruct&);
00170
00178     static stdair::ParsedKey getFirstSPParsedKey (stdair::TravelSolutionStruct&
    );
00179
00187     static stdair::ParsedKey getLastSPParsedKey (stdair::TravelSolutionStruct&)
    ;
00188
00189
00190
00191   private:
00192     // ////////////////////// Construction and destruction ////////////////
00196     FareQuoter();
00197
00201     FareQuoter(const FareQuoter&);
00202
00206     ~FareQuoter();
00207
00208   private:
00209
00212     static bool _atLeastOneAvailableDateRule;
00213
00216     static bool _atLeastOneAvailablePosChannel;
00217
00221     static bool _atLeastOneAvailableTimeRule;
00222
00226     static bool _atLeastOneAvailableFeaturesRule;
00227
00231     static bool _atLeastOneAvailableAirlineClassRule;
00232
```

```
00233   };
00234
00235 }
00236 #endif // __SIMFQT_CMD_FAREQUOTER_HPP
00237
```

23.39 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

23.40 FareRuleGenerator.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 // StdAir
00007 #include <stdair/bom/BomManager.hpp>
00008 #include <stdair/bom/BomRoot.hpp>
00009 #include <stdair/factory/FacBomManager.hpp>
00010 #include <stdair/service/Logger.hpp>
00011 #include <stdair/bom/AirportPair.hpp>
00012 #include <stdair/bom/PosChannel.hpp>
00013 #include <stdair/bom/DatePeriod.hpp>
00014 #include <stdair/bom/TimePeriod.hpp>
00015 #include <stdair/bom/FareFeatures.hpp>
00016 #include <stdair/bom/AirlineClassList.hpp>
00017 // SimFQT
00018 #include <simfqt/bom/FareRuleStruct.hpp>
00019 #include <simfqt/command/FareRuleGenerator.hpp
    >
00020
00021 namespace SIMFQT {
00022
00023   // ////////////////////////////////////////////////////////////////////
00024   void FareRuleGenerator::
00025   createAirportPair (stdair::BomRoot& ioBomRoot,
00026                      const FareRuleStruct& iFareRuleStruct) {
00027
00028     // Create the airport-pair primary key.
00029     const stdair::AirportCode_T& lBoardPoint = iFareRuleStruct.getOrigin ();
00030     const stdair::AirportCode_T& lOffPoint =
00031       iFareRuleStruct.getDestination ();
00032     const stdair::AirportPairKey lAirportPairKey (lBoardPoint, lOffPoint);
00033
00034     // Check that the airport-pair object is not already existing. If an
00035     // airport-pair object with the same key has not already been created,
00036     // create it and link it to the ioBomRoot object.
00037     stdair::AirportPair* lAirportPair_ptr = stdair::BomManager::
00038       getObjectPtr<stdair::AirportPair> (ioBomRoot, lAirportPairKey.toString())
    ;
00039     if (lAirportPair_ptr == NULL) {
00040       lAirportPair_ptr =
```

```
00041         &stdair::FacBom<stdair::AirportPair>::instance().
00042         create (lAirportPairKey);
00043       stdair::FacBomManager::addToListAndMap (ioBomRoot, *lAirportPair_ptr);
00044       stdair::FacBomManager::linkWithParent (ioBomRoot, *lAirportPair_ptr);
00045     }
00046     // Sanity check.
00047     assert (lAirportPair_ptr != NULL);
00048
00049     stdair::AirportPair& lAirportPair = *lAirportPair_ptr;
00050     // Generate the date-period object corresponding to the given
00051     // fareRule.
00052     createDateRange (lAirportPair, iFareRuleStruct);
00053
00054   }
00055
00056   // ////////////////////////////////////////////////////////////////////
00057   void FareRuleGenerator::
00058   createDateRange (stdair::AirportPair& iAirportPair,
00059                    const FareRuleStruct& iFareRuleStruct) {
00060
00061     // Create the fare date-period primary key.
00062     const stdair::Date_T& lDateRangeStart =
00063       iFareRuleStruct.getDateRangeStart ();
00064     const stdair::Date_T& lDateRangeEnd =
00065       iFareRuleStruct.getDateRangeEnd ();
00066     const stdair::DatePeriod_T lDatePeriod (lDateRangeStart, lDateRangeEnd);
00067     const stdair::DatePeriodKey lFareDatePeriodKey (lDatePeriod);
00068
00069     // Check that the date-period object is not already existing.
00070     // If a date-period object with the same key has not already been
00071     // created, create it and link it to the airport-pair object.
00072     stdair::DatePeriod* lFareDatePeriod_ptr = stdair::BomManager::
00073       getObjectPtr<stdair::DatePeriod> (iAirportPair,
00074                                         lFareDatePeriodKey.toString());
00075     if (lFareDatePeriod_ptr == NULL) {
00076       lFareDatePeriod_ptr = &stdair::FacBom<stdair::DatePeriod>::instance().
00077         create (lFareDatePeriodKey);
00078       stdair::FacBomManager::addToListAndMap (iAirportPair,
00079                                               *lFareDatePeriod_ptr);
00080       stdair::FacBomManager::linkWithParent (iAirportPair,
00081                                              *lFareDatePeriod_ptr);
00082     }
00083     // Sanity check.
00084     assert (lFareDatePeriod_ptr != NULL);
00085
00086     stdair::DatePeriod& lDateRange = *lFareDatePeriod_ptr;
00087     // Generate the point_of_sale-channel object corresponding to
00088     // the given fareRule.
00089     createPOSChannel (lDateRange, iFareRuleStruct);
00090
00091   }
00092
00093   // ////////////////////////////////////////////////////////////////////
00094   void FareRuleGenerator::
00095   createPOSChannel (stdair::DatePeriod& iDatePeriod,
00096                     const FareRuleStruct& iFareRuleStruct) {
00097
00098     // Create the point-of-sale-channel primary key.
00099     const stdair::CityCode_T& lPosition = iFareRuleStruct.getPOS ();
00100     const stdair::ChannelLabel_T& lChannel =
00101         iFareRuleStruct.getChannel ();
00102     const stdair::PosChannelKey lFarePosChannelKey (lPosition, lChannel);
00103
00104     // Check that the point_of_sale-channel object is not already existing.
00105     // If a point_of_sale-channel object with the same key has not already
00106     // been created, create it and link it to the date-period object.
00107     stdair::PosChannel* lFarePosChannel_ptr = stdair::BomManager::
00108       getObjectPtr<stdair::PosChannel> (iDatePeriod,
00109                                         lFarePosChannelKey.toString());
00110     if (lFarePosChannel_ptr == NULL) {
00111       lFarePosChannel_ptr = &stdair::FacBom<stdair::PosChannel>::instance().
00112         create (lFarePosChannelKey);
00113       stdair::FacBomManager::addToListAndMap (iDatePeriod,
00114                                               *lFarePosChannel_ptr);
00115       stdair::FacBomManager::linkWithParent (iDatePeriod,
00116                                              *lFarePosChannel_ptr);
00117     }
00118     // Sanity check.
00119     assert (lFarePosChannel_ptr != NULL);
00120
00121     stdair::PosChannel& lPosChannel = *lFarePosChannel_ptr;
00122     // Generate the time-period object corresponding to the given
00123     // fareRule.
00124     createTimeRange (lPosChannel, iFareRuleStruct);
00125
00126   }
00127
```

```
00128
00129   // ////////////////////////////////////////////////////////////////////
00130   void FareRuleGenerator::
00131   createTimeRange (stdair::PosChannel& iPosChannel,
00132                    const FareRuleStruct& iFareRuleStruct) {
00133
00134     // Create the fare time-period primary key.
00135     const stdair::Time_T& lTimeRangeStart =
00136       iFareRuleStruct.getTimeRangeStart ();
00137     const stdair::Time_T& lTimeRangeEnd =
00138       iFareRuleStruct.getTimeRangeEnd ();
00139     const stdair::TimePeriodKey lFareTimePeriodKey (lTimeRangeStart,
00140                                                     lTimeRangeEnd);
00141
00142     // Check that the time-period object is not already existing.
00143     // If a time-period object with the same key has not already been
00144     // created, create it and link it to the point_of_sale-channel object.

00145     stdair::TimePeriod* lFareTimePeriod_ptr = stdair::BomManager::
00146       getObjectPtr<stdair::TimePeriod> (iPosChannel,
00147                                         lFareTimePeriodKey.toString());
00148     if (lFareTimePeriod_ptr == NULL) {
00149       lFareTimePeriod_ptr = &stdair::FacBom<stdair::TimePeriod>::instance().
00150         create (lFareTimePeriodKey);
00151       stdair::FacBomManager::addToListAndMap (iPosChannel,
00152                                               *lFareTimePeriod_ptr);
00153       stdair::FacBomManager::linkWithParent (iPosChannel,
00154                                              *lFareTimePeriod_ptr);
00155     }
00156     // Sanity check.
00157     assert (lFareTimePeriod_ptr != NULL);
00158
00159     stdair::TimePeriod& lTimeRange = *lFareTimePeriod_ptr;
00160     // Generate the fare-features object corresponding to the given
00161     // fareRule.
00162     createFareFeatures (lTimeRange, iFareRuleStruct);
00163
00164   }
00165
00166   // ////////////////////////////////////////////////////////////////////
00167   void FareRuleGenerator::
00168   createFareFeatures (stdair::TimePeriod& iTimePeriod,
00169                       const FareRuleStruct& iFareRuleStruct) {
00170
00171     // Create the fare-features primary key.
00172     const stdair::TripType_T& lTripType =
00173       iFareRuleStruct.getTripType ();
00174     const stdair::DayDuration_T& lAdvancePurchase =
00175       iFareRuleStruct.getAdvancePurchase ();
00176     const stdair::SaturdayStay_T& lSaturdayStay =
00177       iFareRuleStruct.getSaturdayStay ();
00178     const stdair::ChangeFees_T& lChangeFees =
00179       iFareRuleStruct.getChangeFees ();
00180     const stdair::NonRefundable_T& lNonRefundable =
00181       iFareRuleStruct.getNonRefundable ();
00182     const stdair::DayDuration_T& lMinimumStay =
00183       iFareRuleStruct.getMinimumStay ();
00184     const stdair::FareFeaturesKey
00185       lFareFeaturesKey (lTripType, lAdvancePurchase, lSaturdayStay,
00186                         lChangeFees, lNonRefundable, lMinimumStay);
00187
00188     // Check that the fare features object is not already existing.
00189     // If a fare features object with the same key has not already been
00190     // created, create it and link it to the time-period object.
00191     stdair::FareFeatures* lFareFeatures_ptr = stdair::BomManager::
00192       getObjectPtr<stdair::FareFeatures> (iTimePeriod,
00193                                           lFareFeaturesKey.toString());
00194     if (lFareFeatures_ptr == NULL) {
00195       lFareFeatures_ptr = &stdair::FacBom<stdair::FareFeatures>::instance().
00196         create (lFareFeaturesKey);
00197       assert(lFareFeatures_ptr != NULL);
00198       stdair::FacBomManager::addToListAndMap (iTimePeriod,
00199                                               *lFareFeatures_ptr);
00200       stdair::FacBomManager::linkWithParent (iTimePeriod,
00201                                              *lFareFeatures_ptr);
00202     }
00203     // Sanity check.
00204     assert(lFareFeatures_ptr != NULL);
00205
00206     stdair::FareFeatures& lFareFeatures = *lFareFeatures_ptr;
00207     // Generate the airline-class list object corresponding to the
00208     // given fareRule
00209     createAirlineClassList (lFareFeatures, iFareRuleStruct);
00210
00211   }
00212
00213   // ////////////////////////////////////////////////////////////////////
```

```
00214   void FareRuleGenerator::
00215   createAirlineClassList (stdair::FareFeatures& iFareFeatures,
00216                           const FareRuleStruct& iFareRuleStruct) {
00217
00218     // Create the AirlineClassList primary key.
00219     const unsigned int lAirlineListSize =
00220       iFareRuleStruct.getAirlineListSize();
00221     const unsigned int lClassCodeListSize =
00222       iFareRuleStruct.getClassCodeListSize();
00223     assert (lAirlineListSize == lClassCodeListSize);
00224     const stdair::AirlineClassListKey
00225       lAirlineClassListKey (iFareRuleStruct.getAirlineList(),
00226                             iFareRuleStruct.getClassCodeList());
00227     const stdair::Fare_T& lFare = iFareRuleStruct.getFare ();
00228
00229     // Create the airline class list object and link it to the fare features
00230     // object.
00231     stdair::AirlineClassList* lAirlineClassList_ptr =
00232       &stdair::FacBom<stdair::AirlineClassList>::instance().
00233       create (lAirlineClassListKey);
00234     lAirlineClassList_ptr->setFare(lFare);
00235     stdair::FacBomManager::addToListAndMap (iFareFeatures,
00236                                             *lAirlineClassList_ptr);
00237     stdair::FacBomManager::linkWithParent(iFareFeatures,
00238                                           *lAirlineClassList_ptr);
00239   }
00240
00241 }
00242
```

23.41 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

23.42 FareRuleGenerator.hpp

```
00001 #ifndef __SIMFQT_CMD_FARERULEGENERATOR_HPP
00002 #define __SIMFQT_CMD_FARERULEGENERATOR_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // StdAir
00008 #include <stdair/command/CmdAbstract.hpp>
00009 // Simfqt
00010 #include <simfqt/SIMFQT_Types.hpp>
00011
00012 // Forward declarations
00013 namespace stdair {
00014   class BomRoot;
00015   class FareRule;
00016   class AirportPair;
00017   class DatePeriod;
00018   class PosChannel;
00019   class TimePeriod;
00020   class FareFeatures;
00021   class AirlineClassList;
00022 }
00023
```

```
00024 namespace SIMFQT {
00025
00026   // Forward declarations
00027   struct FareRuleStruct;
00028   namespace FareParserHelper {
00029     struct doEndFare;
00030   }
00031
00033   class FareRuleGenerator : public stdair::CmdAbstract {
00034
00035     // Only the following class may use methods of FareGenerator.
00036     // Indeed, as those methods build the BOM, it is not good to expose
00037     // them public.
00038     friend class FareFileParser;
00039     friend struct FareParserHelper::doEndFare;
00040     friend class FareParser;
00041
00042   private:
00043
00052     static void createAirportPair (stdair::BomRoot&,
00053                                    const FareRuleStruct&);
00054
00063     static void createDateRange (stdair::AirportPair&,
00064                                  const FareRuleStruct&);
00065
00074     static void createPOSChannel (stdair::DatePeriod&,
00075                                   const FareRuleStruct&);
00076
00085     static void createTimeRange (stdair::PosChannel&,
00086                                  const FareRuleStruct&);
00087
00096     static void createFareFeatures (stdair::TimePeriod&,
00097                                     const FareRuleStruct&);
00098
00107     static void createAirlineClassList (stdair::FareFeatures&,
00108                                         const FareRuleStruct&);
00109
00110
00111
00112   };
00113
00114 }
00115 #endif // __SIMFQT_CMD_FARERULEGENERATOR_HPP
```

23.43 simfqt/config/simfqt-paths.hpp File Reference

Macros

- #define PACKAGE "simfqt"
- #define PACKAGE_NAME "SIMFQT"
- #define PACKAGE_VERSION "1.00.0"
- #define PREFIXDIR "/usr"
- #define EXEC_PREFIX "/usr"
- #define BINDIR "/usr/bin"
- #define LIBDIR "/usr/lib"
- #define LIBEXECDIR "/usr/libexec"
- #define SBINDIR "/usr/sbin"
- #define SYSCONFDIR "/usr/etc"
- #define INCLUDEDIR "/usr/include"
- #define DATAROOTDIR "/usr/share"
- #define DATADIR "/usr/share"
- #define DOCDIR "/usr/share/doc/simfqt-1.00.0"
- #define MANDIR "/usr/share/man"
- #define INFODIR "/usr/share/info"
- #define HTMLDIR "/usr/share/doc/simfqt-1.00.0/html"
- #define PDFDIR "/usr/share/doc/simfqt-1.00.0/html"
- #define STDAIR_SAMPLE_DIR "/usr/share/stdair/samples"

23.43.1 Macro Definition Documentation

23.43.1.1 #define PACKAGE "simfqt"

Definition at line 4 of file simfqt-paths.hpp.

23.43.1.2 #define PACKAGE_NAME "SIMFQT"

Definition at line 5 of file simfqt-paths.hpp.

Referenced by readConfiguration().

23.43.1.3 #define PACKAGE_VERSION "1.00.0"

Definition at line 6 of file simfqt-paths.hpp.

Referenced by readConfiguration().

23.43.1.4 #define PREFIXDIR "/usr"

Definition at line 7 of file simfqt-paths.hpp.

Referenced by readConfiguration().

23.43.1.5 #define EXEC_PREFIX "/usr"

Definition at line 8 of file simfqt-paths.hpp.

23.43.1.6 #define BINDIR "/usr/bin"

Definition at line 9 of file simfqt-paths.hpp.

23.43.1.7 #define LIBDIR "/usr/lib"

Definition at line 10 of file simfqt-paths.hpp.

23.43.1.8 #define LIBEXECDIR "/usr/libexec"

Definition at line 11 of file simfqt-paths.hpp.

23.43.1.9 #define SBINDIR "/usr/sbin"

Definition at line 12 of file simfqt-paths.hpp.

23.43.1.10 #define SYSCONFDIR "/usr/etc"

Definition at line 13 of file simfqt-paths.hpp.

23.43.1.11 #define INCLUDEDIR "/usr/include"

Definition at line 14 of file simfqt-paths.hpp.

23.43.1.12 #define DATAROOTDIR "/usr/share"

Definition at line 15 of file simfqt-paths.hpp.

23.43.1.13 #define DATADIR "/usr/share"

Definition at line 16 of file simfqt-paths.hpp.

23.43.1.14 #define DOCDIR "/usr/share/doc/simfqt-1.00.0"

Definition at line 17 of file simfqt-paths.hpp.

23.43.1.15 #define MANDIR "/usr/share/man"

Definition at line 18 of file simfqt-paths.hpp.

23.43.1.16 #define INFODIR "/usr/share/info"

Definition at line 19 of file simfqt-paths.hpp.

23.43.1.17 #define HTMLDIR "/usr/share/doc/simfqt-1.00.0/html"

Definition at line 20 of file simfqt-paths.hpp.

23.43.1.18 #define PDFDIR "/usr/share/doc/simfqt-1.00.0/html"

Definition at line 21 of file simfqt-paths.hpp.

23.43.1.19 #define STDAIR_SAMPLE_DIR "/usr/share/stdair/samples"

Definition at line 22 of file simfqt-paths.hpp.

23.44 simfqt-paths.hpp

```
00001 #ifndef __SIMFQT_PATHS_HPP__
00002 #define __SIMFQT_PATHS_HPP__
00003
00004 #define PACKAGE "simfqt"
00005 #define PACKAGE_NAME "SIMFQT"
00006 #define PACKAGE_VERSION "1.00.0"
00007 #define PREFIXDIR "/usr"
00008 #define EXEC_PREFIX "/usr"
00009 #define BINDIR "/usr/bin"
00010 #define LIBDIR "/usr/lib"
00011 #define LIBEXECDIR "/usr/libexec"
00012 #define SBINDIR "/usr/sbin"
00013 #define SYSCONFDIR "/usr/etc"
00014 #define INCLUDEDIR "/usr/include"
00015 #define DATAROOTDIR "/usr/share"
00016 #define DATADIR "/usr/share"
00017 #define DOCDIR "/usr/share/doc/simfqt-1.00.0"
00018 #define MANDIR "/usr/share/man"
00019 #define INFODIR "/usr/share/info"
00020 #define HTMLDIR "/usr/share/doc/simfqt-1.00.0/html"
00021 #define PDFDIR "/usr/share/doc/simfqt-1.00.0/html"
00022 #define STDAIR_SAMPLE_DIR "/usr/share/stdair/samples"
00023
00024 #endif // __SIMFQT_PATHS_HPP__
```

23.45 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

23.46 FacSimfqtServiceContext.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
```

```
00006 // StdAir
00007 #include <stdair/service/FacSupervisor.hpp>
00008 // SimFQT
00009 #include <simfqt/factory/FacSimfqtServiceContext.hpp
      >
00010 #include <simfqt/service/SIMFQT_ServiceContext.hpp
      >
00011
00012 namespace SIMFQT {
00013
00014   FacSimfqtServiceContext* FacSimfqtServiceContext::_instance = NULL;
00015
00016   // ////////////////////////////////////////////////////////////////////
00017   FacSimfqtServiceContext::~FacSimfqtServiceContext
      () {
00018     _instance = NULL;
00019   }
00020
00021   // ////////////////////////////////////////////////////////////////////
00022   FacSimfqtServiceContext&
      FacSimfqtServiceContext::instance() {
00023
00024     if (_instance == NULL) {
00025       _instance = new FacSimfqtServiceContext();
00026       assert (_instance != NULL);
00027
00028       stdair::FacSupervisor::instance().
      registerServiceFactory (_instance);
00029     }
00030     return *_instance;
00031   }
00032
00033   // ////////////////////////////////////////////////////////////////////
00034   SIMFQT_ServiceContext& FacSimfqtServiceContext::create
      () {
00035     SIMFQT_ServiceContext* aServiceContext_ptr = NULL;
00036
00037     aServiceContext_ptr = new SIMFQT_ServiceContext();
00038     assert (aServiceContext_ptr != NULL);
00039
00040     // The new object is added to the Bom pool
00041     _pool.push_back (aServiceContext_ptr);
00042
00043     return *aServiceContext_ptr;
00044   }
00045
00046 }
```

23.47 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

23.48 FacSimfqtServiceContext.hpp

```
00001 #ifndef __SIMFQT_FAC_FACSIMFQTSERVICECONTEXT_HPP
00002 #define __SIMFQT_FAC_FACSIMFQTSERVICECONTEXT_HPP
00003
00004 // ////////////////////////////////////////////////////////////////////
00005 // Import section
00006 // ////////////////////////////////////////////////////////////////////
```

```
00007 // STL
00008 #include <string>
00009 // StdAir
00010 #include <stdair/stdair_basic_types.hpp>
00011 #include <stdair/service/FacServiceAbstract.hpp>
00012
00013 namespace SIMFQT {
00014
00016   class SIMFQT_ServiceContext;
00017
00018
00022   class FacSimfqtServiceContext : public
      stdair::FacServiceAbstract {
00023   public:
00024
00031     static FacSimfqtServiceContext& instance();
00032
00039     ~FacSimfqtServiceContext();
00040
00048     SIMFQT_ServiceContext& create();
00049
00050
00051   protected:
00057     FacSimfqtServiceContext() {}
00058
00059
00060   private:
00064     static FacSimfqtServiceContext* _instance;
00065   };
00066
00067 }
00068 #endif // __SIMFQT_FAC_FACSIMFQTSERVICECONTEXT_HPP
```

23.49 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

23.50 SIMFQT_Service.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 // Boost
00007 #include <boost/make_shared.hpp>
00008 // StdAir
00009 #include <stdair/basic/BasChronometer.hpp>
00010 #include <stdair/bom/BomDisplay.hpp>
00011 #include <stdair/bom/TravelSolutionStruct.hpp>
00012 #include <stdair/bom/BookingRequestStruct.hpp>
00013 #include <stdair/service/Logger.hpp>
00014 #include <stdair/STDAIR_Service.hpp>
```

```
00015 // Simfqt
00016 #include <simfqt/basic/BasConst_SIMFQT_Service.hpp
      >
00017 #include <simfqt/factory/FacSimfqtServiceContext.hpp
      >
00018 #include <simfqt/command/FareParser.hpp>
00019 #include <simfqt/command/FareQuoter.hpp>
00020 #include <simfqt/service/SIMFQT_ServiceContext.hpp
      >
00021 #include <simfqt/SIMFQT_Service.hpp>
00022
00023 namespace SIMFQT {
00024
00025   // //////////////////////////////////////////////////////////////////
00026   SIMFQT_Service::SIMFQT_Service() : _simfqtServiceContext (NULL) {
00027     assert (false);
00028   }
00029
00030   // //////////////////////////////////////////////////////////////////
00031   SIMFQT_Service::SIMFQT_Service (const SIMFQT_Service& iService) {
00032     assert (false);
00033   }
00034
00035   // //////////////////////////////////////////////////////////////////
00036   SIMFQT_Service::SIMFQT_Service (const stdair::BasLogParams& iLogParams)
00037     : _simfqtServiceContext (NULL) {
00038
00039     // Initialise the STDAIR service handler
00040     stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
00041       initStdAirService (iLogParams);
00042
00043     // Initialise the service context
00044     initServiceContext();
00045
00046     // Add the StdAir service context to the SIMFQT service context
00047     // \note SIMFQT owns the STDAIR service resources here.
00048     const bool ownStdairService = true;
00049     addStdAirService (lSTDAIR_Service_ptr, ownStdairService);
00050
00051     // Initialise the (remaining of the) context
00052     initSimfqtService();
00053   }
00054
00055   // //////////////////////////////////////////////////////////////////
00056   SIMFQT_Service::SIMFQT_Service (const stdair::BasLogParams& iLogParams,
00057                                   const stdair::BasDBParams& iDBParams)
00058     : _simfqtServiceContext (NULL) {
00059
00060     // Initialise the STDAIR service handler
00061     stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
00062       initStdAirService (iLogParams, iDBParams);
00063
00064     // Initialise the service context
00065     initServiceContext();
00066
00067     // Add the StdAir service context to the SIMFQT service context
00068     // \note SIMFQT owns the STDAIR service resources here.
00069     const bool ownStdairService = true;
00070     addStdAirService (lSTDAIR_Service_ptr, ownStdairService);
00071
00072     // Initialise the (remaining of the) context
00073     initSimfqtService();
00074   }
00075
00076   // //////////////////////////////////////////////////////////////////
00077   SIMFQT_Service::
00078   SIMFQT_Service (stdair::STDAIR_ServicePtr_T ioSTDAIR_Service_ptr)
00079     : _simfqtServiceContext (NULL) {
00080
00081     // Initialise the service context
00082     initServiceContext();
00083
00084     // Store the STDAIR service object within the (SIMFQT) service context
00085     // \note Simfqt does not own the STDAIR service resources here.
00086     const bool doesNotOwnStdairService = false;
00087     addStdAirService (ioSTDAIR_Service_ptr, doesNotOwnStdairService);
00088
00089     // Initialise the context
00090     initSimfqtService();
00091   }
00092
00093   // //////////////////////////////////////////////////////////////////
00094   SIMFQT_Service::~SIMFQT_Service() {
00095     // Delete/Clean all the objects from memory
00096     finalise();
00097   }
00098
```

```
00099   // ////////////////////////////////////////////////////////////////////
00100   void SIMFQT_Service::finalise() {
00101     assert (_simfqtServiceContext != NULL);
00102     // Reset the (Boost.)Smart pointer pointing on the STDAIR_Service object.
00103     _simfqtServiceContext->reset();
00104   }
00105
00106   // ////////////////////////////////////////////////////////////////////
00107   void SIMFQT_Service::initServiceContext() {
00108     // Initialise the service context
00109     SIMFQT_ServiceContext& lSIMFQT_ServiceContext =
00110       FacSimfqtServiceContext::instance().
      create();
00111     _simfqtServiceContext = &lSIMFQT_ServiceContext;
00112   }
00113
00114   // ////////////////////////////////////////////////////////////////////
00115   void SIMFQT_Service::
00116   addStdAirService (stdair::STDAIR_ServicePtr_T ioSTDAIR_Service_ptr,
00117                     const bool iOwnStdairService) {
00118
00119     // Retrieve the SimFQT service context
00120     assert (_simfqtServiceContext != NULL);
00121     SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *_simfqtServiceContext;
00122
00123     // Store the STDAIR service object within the (SimFQT) service context
00124     lSIMFQT_ServiceContext.setSTDAIR_Service (ioSTDAIR_Service_ptr,
00125                                               iOwnStdairService);
00126   }
00127
00128   // ////////////////////////////////////////////////////////////////////
00129   stdair::STDAIR_ServicePtr_T SIMFQT_Service::
00130   initStdAirService (const stdair::BasLogParams& iLogParams,
00131                      const stdair::BasDBParams& iDBParams) {
00132
00139     stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
00140       boost::make_shared<stdair::STDAIR_Service> (iLogParams, iDBParams);
00141     assert (lSTDAIR_Service_ptr != NULL);
00142
00143     return lSTDAIR_Service_ptr;
00144   }
00145
00146   // ////////////////////////////////////////////////////////////////////
00147   stdair::STDAIR_ServicePtr_T SIMFQT_Service::
00148   initStdAirService (const stdair::BasLogParams& iLogParams) {
00149
00156     stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
00157       boost::make_shared<stdair::STDAIR_Service> (iLogParams);
00158     assert (lSTDAIR_Service_ptr != NULL);
00159
00160     return lSTDAIR_Service_ptr;
00161   }
00162
00163   // ////////////////////////////////////////////////////////////////////
00164   void SIMFQT_Service::initSimfqtService() {
00165     // Do nothing at this stage. A sample BOM tree may be built by
00166     // calling the buildSampleBom() method
00167   }
00168
00169   // ////////////////////////////////////////////////////////////////////
00170   void SIMFQT_Service::
00171   parseAndLoad (const FareFilePath& iFareFilename) {
00172
00173     // Retrieve the SimFQT service context
00174     if (_simfqtServiceContext == NULL) {
00175       throw stdair::NonInitialisedServiceException ("The SimFQT service "
00176                                                     "has not been initialised")
      ;
00177     }
00178     assert (_simfqtServiceContext != NULL);
00179
00180     // Retrieve the SimFQT service context and whether it owns the Stdair
00181     // service
00182     SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *
      _simfqtServiceContext;
00183     const bool doesOwnStdairService =
00184       lSIMFQT_ServiceContext.getOwnStdairServiceFlag();
00185
00186     // Retrieve the StdAir service object from the (SimFQT) service context
00187     stdair::STDAIR_Service& lSTDAIR_Service =
00188       lSIMFQT_ServiceContext.getSTDAIR_Service();
00189
00190     // Retrieve the persistent BOM root object.
00191     stdair::BomRoot& lPersistentBomRoot =
00192       lSTDAIR_Service.getPersistentBomRoot();
00193
00197     FareParser::fareRuleGeneration (iFareFilename
```

```
    , lPersistentBomRoot);
00198
00210     buildComplementaryLinks (lPersistentBomRoot);
00211
00216     if (doesOwnStdairService == true) {
00217       //
00218       clonePersistentBom ();
00219     }
00220   }
00221
00222   // ////////////////////////////////////////////////////////////////////
00223   void SIMFQT_Service::buildSampleBom() {
00224
00225     // Retrieve the SimFQT service context
00226     if (_simfqtServiceContext == NULL) {
00227       throw stdair::NonInitialisedServiceException ("The SimFQT service "
00228                                                     "has not been initialised")
    ;
00229     }
00230     assert (_simfqtServiceContext != NULL);
00231
00232     // Retrieve the SimFQT service context and whether it owns the Stdair
00233     // service
00234     SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *
    _simfqtServiceContext;
00235     const bool doesOwnStdairService =
00236       lSIMFQT_ServiceContext.getOwnStdairServiceFlag();
00237
00238     // Retrieve the StdAir service object from the (SimFQT) service context
00239     stdair::STDAIR_Service& lSTDAIR_Service =
00240       lSIMFQT_ServiceContext.getSTDAIR_Service();
00241
00242     // Retrieve the persistent BOM root object.
00243     stdair::BomRoot& lPersistentBomRoot =
00244       lSTDAIR_Service.getPersistentBomRoot();
00245
00250     if (doesOwnStdairService == true) {
00251       //
00252       lSTDAIR_Service.buildSampleBom();
00253     }
00254
00266     buildComplementaryLinks (lPersistentBomRoot);
00267
00272     if (doesOwnStdairService == true) {
00273       //
00274       clonePersistentBom ();
00275     }
00276   }
00277
00278   // ////////////////////////////////////////////////////////////////////
00279   void SIMFQT_Service::clonePersistentBom ()
    {
00280
00281     // Retrieve the SimFQT service context
00282     if (_simfqtServiceContext == NULL) {
00283       throw stdair::NonInitialisedServiceException ("The SimFQT service "
00284                                                     "has not been initialised")
    ;
00285     }
00286     assert (_simfqtServiceContext != NULL);
00287
00288     // Retrieve the SimFQT service context and whether it owns the Stdair
00289     // service
00290     SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *
    _simfqtServiceContext;
00291     const bool doesOwnStdairService =
00292       lSIMFQT_ServiceContext.getOwnStdairServiceFlag();
00293
00294     // Retrieve the StdAir service object from the (SimFQT) service context
00295     stdair::STDAIR_Service& lSTDAIR_Service =
00296       lSIMFQT_ServiceContext.getSTDAIR_Service();
00297
00302     if (doesOwnStdairService == true) {
00303       //
00304       lSTDAIR_Service.clonePersistentBom ();
00305     }
00306
00310     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
00311     buildComplementaryLinks (lBomRoot);
00312   }
00313
00314   // ////////////////////////////////////////////////////////////////////
00315   void SIMFQT_Service::buildComplementaryLinks
     (stdair::BomRoot& ioBomRoot) {
00316     // Currently, no more things to do by SimFQT at that stage.
00317   }
00318
```

```
00319   // ////////////////////////////////////////////////////////////////////
00320   stdair::BookingRequestStruct SIMFQT_Service::buildBookingRequest
      (const bool isForCRS) {
00321
00322     // Retrieve the SIMFQT service context
00323     if (_simfqtServiceContext == NULL) {
00324       throw stdair::NonInitialisedServiceException ("The Simfqt service has not
       "
00325                                                     "been initialised");
00326     }
00327     assert (_simfqtServiceContext != NULL);
00328
00329     SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *
      _simfqtServiceContext;
00330
00331     // Retrieve the STDAIR service object from the (Simfqt) service context
00332     stdair::STDAIR_Service& lSTDAIR_Service =
00333       lSIMFQT_ServiceContext.getSTDAIR_Service();
00334
00335     // Delegate the BOM building to the dedicated service
00336     stdair::BookingRequestStruct oBookingRequest =
00337       lSTDAIR_Service.buildSampleBookingRequest (isForCRS);
00338
00339     return oBookingRequest;
00340   }
00341
00342   // ////////////////////////////////////////////////////////////////////
00343   void SIMFQT_Service::
00344   buildSampleTravelSolutions(
      stdair::TravelSolutionList_T& ioTravelSolutionList){
00345
00346     // Retrieve the SIMFQT service context
00347     if (_simfqtServiceContext == NULL) {
00348       throw stdair::NonInitialisedServiceException ("The Simfqt service has not
       "
00349                                                     "been initialised");
00350     }
00351     assert (_simfqtServiceContext != NULL);
00352
00353     SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *
      _simfqtServiceContext;
00354
00355     // Retrieve the STDAIR service object from the (Simfqt) service context
00356     stdair::STDAIR_Service& lSTDAIR_Service =
00357       lSIMFQT_ServiceContext.getSTDAIR_Service();
00358
00359     // Delegate the BOM building to the dedicated service
00360     lSTDAIR_Service.buildSampleTravelSolutionForPricing (ioTravelSolutionList);
00361   }
00362
00363
00364   // ////////////////////////////////////////////////////////////////////
00365   std::string SIMFQT_Service::csvDisplay() const {
00366
00367     // Retrieve the SIMFQT service context
00368     if (_simfqtServiceContext == NULL) {
00369       throw stdair::NonInitialisedServiceException ("The SimFQT service "
00370                                                     "has not been initialised")
      ;
00371     }
00372     assert (_simfqtServiceContext != NULL);
00373
00374     SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *
      _simfqtServiceContext;
00375
00376     // Retrieve the STDAIR service object from the (SimFQT) service context
00377     stdair::STDAIR_Service& lSTDAIR_Service =
00378       lSIMFQT_ServiceContext.getSTDAIR_Service();
00379
00380     // Get the root of the BOM tree, on which all of the other BOM objects
00381     // are attached
00382     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
00383
00384     // Delegate the BOM display to the dedicated service
00385     std::ostringstream oCSVStr;
00386     stdair::BomDisplay::csvSimFQTAirRACDisplay (oCSVStr, lBomRoot);
00387     return oCSVStr.str();
00388   }
00389
00390   // ////////////////////////////////////////////////////////////////////
00391   std::string SIMFQT_Service::
00392   csvDisplay (const stdair::TravelSolutionList_T&
      ioTravelSolutionList) const {
00393
00394     // Retrieve the Simfqt service context
00395     if (_simfqtServiceContext == NULL) {
00396       throw stdair::NonInitialisedServiceException ("The Simfqt service has not
```

```
      "
00397                                                 "been initialised");
00398     }
00399     assert (_simfqtServiceContext != NULL);
00400
00401     // Retrieve the Simfqt service context
00402     SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *
      _simfqtServiceContext;
00403
00404     // Retrieve the STDAIR service object from the (Simfqt) service context
00405     stdair::STDAIR_Service& lSTDAIR_Service =
00406       lSIMFQT_ServiceContext.getSTDAIR_Service();
00407
00408     // Delegate the BOM building to the dedicated service
00409     return lSTDAIR_Service.csvDisplay (ioTravelSolutionList);
00410   }
00411
00412   // ////////////////////////////////////////////////////////////////////
00413   std::string SIMFQT_Service::
00414   csvDisplay (const stdair::AirportCode_T& iOrigin,
00415               const stdair::AirportCode_T& iDestination,
00416               const stdair::Date_T& iDepartureDate) const {
00417
00418     // Retrieve the SIMFQT service context
00419     if (_simfqtServiceContext == NULL) {
00420       throw stdair::NonInitialisedServiceException ("The Simfqt service "
00421                                                     "has not been initialised")
      ;
00422     }
00423     assert (_simfqtServiceContext != NULL);
00424
00425     SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *
      _simfqtServiceContext;
00426
00427     // Retrieve the STDAIR service object from the (SIMFQT) service context
00428     stdair::STDAIR_Service& lSTDAIR_Service =
00429       lSIMFQT_ServiceContext.getSTDAIR_Service();
00430
00431     // Delegate the BOM display to the dedicated service
00432     return lSTDAIR_Service.csvDisplay (iOrigin, iDestination,
00433                                        iDepartureDate);
00434   }
00435
00436   // ////////////////////////////////////////////////////////////////////
00437   std::string SIMFQT_Service::list() const {
00438
00439     // Retrieve the SIMFQT service context
00440     if (_simfqtServiceContext == NULL) {
00441       throw stdair::NonInitialisedServiceException ("The Simfqt service "
00442                                                     "has not been initialised")
      ;
00443     }
00444     assert (_simfqtServiceContext != NULL);
00445
00446     SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *
      _simfqtServiceContext;
00447
00448     // Retrieve the STDAIR service object from the (SIMFQT) service context
00449     stdair::STDAIR_Service& lSTDAIR_Service =
00450       lSIMFQT_ServiceContext.getSTDAIR_Service();
00451
00452     // Delegate the BOM display to the dedicated service
00453     return lSTDAIR_Service.listAirportPairDateRange ();
00454   }
00455
00456   // ////////////////////////////////////////////////////////////////////
00457   bool SIMFQT_Service::
00458   check (const stdair::AirportCode_T& iOrigin,
00459          const stdair::AirportCode_T& iDestination,
00460          const stdair::Date_T& iDepartureDate) const {
00461     std::ostringstream oFlightListStr;
00462
00463     if (_simfqtServiceContext == NULL) {
00464       throw stdair::NonInitialisedServiceException ("The Simfqt service "
00465                                                     "has not been initialised")
      ;
00466     }
00467     assert (_simfqtServiceContext != NULL);
00468     SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *
      _simfqtServiceContext;
00469
00470     // Retrieve the STDAIR service object from the (SIMFQT) service context
00471     stdair::STDAIR_Service& lSTDAIR_Service =
00472       lSIMFQT_ServiceContext.getSTDAIR_Service();
00473
00474     // Delegate the BOM display to the dedicated service
00475     return lSTDAIR_Service.check (iOrigin, iDestination, iDepartureDate);
```

```
00476   }
00477
00478   // ////////////////////////////////////////////////////////////////////
00479   void SIMFQT_Service::
00480   quotePrices (const stdair::BookingRequestStruct& iBookingRequest
      ,
00481                stdair::TravelSolutionList_T& ioTravelSolutionList) {
00482
00483     // Retrieve the Simfqt service context
00484     if (_simfqtServiceContext == NULL) {
00485       throw stdair::NonInitialisedServiceException ("The SimFQT service "
00486                                                     "has not been initialised")
      ;
00487     }
00488     assert (_simfqtServiceContext != NULL);
00489
00490     SIMFQT_ServiceContext& lSIMFQT_ServiceContext = *
      _simfqtServiceContext;
00491
00492     // Retrieve the StdAir service context
00493     stdair::STDAIR_Service& lSTDAIR_Service =
00494       lSIMFQT_ServiceContext.getSTDAIR_Service();
00495
00496     // Get the root of the BOM tree, on which all of the other BOM objects
00497     // will be attached
00498     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
00499
00500     // Delegate the action to the dedicated command
00501     stdair::BasChronometer lFareQuoteRetrievalChronometer;
00502     lFareQuoteRetrievalChronometer.start();
00503     FareQuoter::priceQuote (iBookingRequest, ioTravelSolutionList, lBomRoot);
00504
00505     // DEBUG
00506     const double lFareQuoteRetrievalMeasure =
00507       lFareQuoteRetrievalChronometer.elapsed();
00508     STDAIR_LOG_DEBUG ("Fare Quote retrieving: " << lFareQuoteRetrievalMeasure
00509                       << " - " << lSIMFQT_ServiceContext.display());
00510   }
00511
00512 }
```

23.51 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

23.52 SIMFQT_ServiceContext.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 // SimFQT
00008 #include <simfqt/basic/BasConst_SIMFQT_Service.hpp
      >
00009 #include <simfqt/service/SIMFQT_ServiceContext.hpp
      >
00010
00011 namespace SIMFQT {
00012
00013   // ////////////////////////////////////////////////////////////////////
00014   SIMFQT_ServiceContext::SIMFQT_ServiceContext() : _ownStdairService (false) {
00015   }
00016
00017   // ////////////////////////////////////////////////////////////////////
00018   SIMFQT_ServiceContext::SIMFQT_ServiceContext (const SIMFQT_ServiceContext&) {
00019     assert (false);
```

```
00020   }
00021
00022   // ////////////////////////////////////////////////////////////////////
00023   SIMFQT_ServiceContext::~SIMFQT_ServiceContext() {
00024   }
00025
00026   // ////////////////////////////////////////////////////////////////////
00027   stdair::STDAIR_Service& SIMFQT_ServiceContext::getSTDAIR_Service() const {
00028     assert (_stdairService != NULL);
00029     return *_stdairService;
00030   }
00031
00032   // ////////////////////////////////////////////////////////////////////
00033   const std::string SIMFQT_ServiceContext::shortDisplay() const {
00034     std::ostringstream oStr;
00035     oStr << "SIMFQT_ServiceContext -- Owns StdAir service: "
00036          << _ownStdairService;
00037     return oStr.str();
00038   }
00039
00040   // ////////////////////////////////////////////////////////////////////
00041   const std::string SIMFQT_ServiceContext::display() const {
00042     std::ostringstream oStr;
00043     oStr << shortDisplay();
00044     return oStr.str();
00045   }
00046
00047   // ////////////////////////////////////////////////////////////////////
00048   const std::string SIMFQT_ServiceContext::describe() const {
00049     return shortDisplay();
00050   }
00051
00052   // ////////////////////////////////////////////////////////////////////
00053   void SIMFQT_ServiceContext::reset() {
00054
00055     // The shared_ptr<>::reset() method drops the refcount by one.
00056     // If the count result is dropping to zero, the resource pointed to
00057     // by the shared_ptr<> will be freed.
00058
00059     // Reset the stdair shared pointer
00060     _stdairService.reset();
00061   }
00062
00063 }
```

23.53 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

23.54 SIMFQT_ServiceContext.hpp

```
00001 #ifndef __SIMFQT_SVC_SIMFQTSERVICECONTEXT_HPP
00002 #define __SIMFQT_SVC_SIMFQTSERVICECONTEXT_HPP
00003
00004 // ////////////////////////////////////////////////////////////////////
00005 // Import section
```

```
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <string>
00009 // StdAir
00010 #include <stdair/stdair_service_types.hpp>
00011 #include <stdair/service/ServiceAbstract.hpp>
00012 // SimFQT
00013 #include <simfqt/SIMFQT_Types.hpp>
00014
00016 namespace stdair {
00017   class STDAIR_Service;
00018 }
00019
00020 namespace SIMFQT {
00021
00025   class SIMFQT_ServiceContext : public
      stdair::ServiceAbstract {
00031     friend class SIMFQT_Service;
00032     friend class FacSimfqtServiceContext;
00033
00034   private:
00035     // ///////// Getters ///////////
00039     stdair::STDAIR_ServicePtr_T getSTDAIR_ServicePtr() const {
00040       return _stdairService;
00041     }
00042
00046     stdair::STDAIR_Service& getSTDAIR_Service() const;
00047
00051     const bool getOwnStdairServiceFlag() const {
00052       return _ownStdairService;
00053     }
00054
00055
00056   private:
00057     // ///////// Setters ///////////
00061     void setSTDAIR_Service (stdair::STDAIR_ServicePtr_T ioSTDAIR_ServicePtr,
00062                             const bool iOwnStdairService) {
00063       _stdairService = ioSTDAIR_ServicePtr;
00064       _ownStdairService = iOwnStdairService;
00065     }
00066
00070     void reset();
00071
00072
00073   private:
00074     // ///////// Display Methods ///////////
00078     const std::string shortDisplay() const;
00079
00083     const std::string display() const;
00084
00088     const std::string describe() const;
00089
00090
00091   private:
00092     // /////// Construction / initialisation ////////
00096     SIMFQT_ServiceContext (const FareQuoteID_T&);
00097
00101     SIMFQT_ServiceContext();
00102
00106     SIMFQT_ServiceContext (const SIMFQT_ServiceContext&);
00107
00111     ~SIMFQT_ServiceContext();
00112
00113
00114   private:
00115     // ////////////// Children ////////////
00119     stdair::STDAIR_ServicePtr_T _stdairService;
00120
00124     bool _ownStdairService;
00125   };
00126
00127 }
00128 #endif // __SIMFQT_SVC_SIMFQTSERVICECONTEXT_HPP
```

23.55 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

23.56 SIMFQT_Service.hpp

```
00001 #ifndef __SIMFQT_SVC_SIMFQT_SERVICE_HPP
00002 #define __SIMFQT_SVC_SIMFQT_SERVICE_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // StdAir
00008 #include <stdair/stdair_basic_types.hpp>
00009 #include <stdair/stdair_service_types.hpp>
00010 #include <stdair/bom/TravelSolutionTypes.hpp>
00011 // SimFQT
00012 #include <simfqt/SIMFQT_Types.hpp>
00013
00015 namespace stdair {
00016   class STDAIR_Service;
00017   class BomRoot;
00018   struct BookingRequestStruct;
00019   struct BasLogParams;
00020   struct BasDBParams;
00021 }
00022
00023 namespace SIMFQT {
00024
00026   class SIMFQT_ServiceContext;
00027
00028
00032   class SIMFQT_Service {
00033   public:
00034
00035     // ////////////////// Constructors and Destructors /////////////////
00047     SIMFQT_Service (const stdair::BasLogParams&);
00048
00061     SIMFQT_Service (const stdair::BasLogParams&, const
      stdair::BasDBParams&);
00062
00078     SIMFQT_Service (stdair::STDAIR_ServicePtr_T
      ioSTDAIR_ServicePtr);
00079
00088     void parseAndLoad (const FareFilePath&
      iFareFilename);
00089
00093     ~SIMFQT_Service();
00094
00095
00096   public:
00097     // ////////// Business Methods /////////////
00109     void buildSampleBom();
00110
00114     void clonePersistentBom ();
00115
00120     void buildComplementaryLinks (stdair::BomRoot&);
00121
00128     stdair::BookingRequestStruct buildBookingRequest(const
      bool isForCRS = false);
00129
00147     void buildSampleTravelSolutions (
      stdair::TravelSolutionList_T&);
00148
00158     void quotePrices (const stdair::BookingRequestStruct&,
00159                       stdair::TravelSolutionList_T&);
00160
00161
00162   public:
00163     // ///////////////// Display support methods /////////////////
00171     std::string csvDisplay() const;
```

```
00172
00180     std::string csvDisplay (const stdair::TravelSolutionList_T&)
      const;
00181
00194     std::string csvDisplay (const stdair::AirportCode_T& ioOrigin,
00195                             const stdair::AirportCode_T& ioDestination,
00196                             const stdair::Date_T& ioDepartureDate) const;
00197
00206     std::string list() const;
00207
00220     bool check (const stdair::AirportCode_T& ioOrigin,
00221                 const stdair::AirportCode_T& ioDestination,
00222                 const stdair::Date_T& ioDepartureDate) const;
00223
00224   private:
00225     // /////// Construction and Destruction helper methods ///////
00229     SIMFQT_Service();
00230
00234     SIMFQT_Service (const SIMFQT_Service&);
00235
00245     stdair::STDAIR_ServicePtr_T initStdAirService (const stdair::BasLogParams&,
00246                                                    const stdair::BasDBParams&);
00247
00256     stdair::STDAIR_ServicePtr_T initStdAirService (const stdair::BasLogParams&)
      ;
00257
00266     void addStdAirService (stdair::STDAIR_ServicePtr_T ioSTDAIR_ServicePtr,
00267                            const bool iOwnStdairService);
00268
00273     void initServiceContext();
00274
00281     void initSimfqtService();
00282
00291     void initSimfqtService (const FareFilePath& iFareFilename);
00292
00296     void finalise();
00297
00298
00299   private:
00300     // ///////// Service Context /////////
00304     SIMFQT_ServiceContext* _simfqtServiceContext;
00305   };
00306 }
00307 #endif // __SIMFQT_SVC_SIMFQT_SERVICE_HPP
```

23.57 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

23.58 SIMFQT_Types.hpp

```
00001 #ifndef __SIMFQT_SIMFQT_TYPES_HPP
00002 #define __SIMFQT_SIMFQT_TYPES_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <vector>
00009 #include <string>
00010 // Boost
00011 #include <boost/shared_ptr.hpp>
00012 // StdAir
00013 #include <stdair/stdair_exceptions.hpp>
00014 #include <stdair/stdair_file.hpp>
00015
00016 namespace SIMFQT {
00017
00018   // Forward declarations
00019   class SIMFQT_Service;
00020
00021
00022   // ///////// Exceptions //////////
00026   class FareFileParsingFailedException
00027     : public stdair::ParsingFileFailedException {
00028   public:
00032     FareFileParsingFailedException (const
      std::string& iWhat)
00033       : stdair::ParsingFileFailedException (iWhat) {}
00034   };
00035
00039   class AirportPairNotFoundException : public
      stdair::ObjectNotFoundException {
00040   public:
00044     AirportPairNotFoundException (const std::string
      & iWhat)
00045       : stdair::ObjectNotFoundException (iWhat) {}
00046   };
00047
00051   class PosOrChannelNotFoundException : public
      stdair::ObjectNotFoundException {
00052   public:
00056     PosOrChannelNotFoundException (const
      std::string& iWhat)
00057       : stdair::ObjectNotFoundException (iWhat) {}
00058   };
00059
00063   class FlightDateNotFoundException : public
      stdair::ObjectNotFoundException {
00064   public:
00068     FlightDateNotFoundException (const std::string&
      iWhat)
00069       : stdair::ObjectNotFoundException (iWhat) {}
00070   };
00071
00075   class FlightTimeNotFoundException : public
      stdair::ObjectNotFoundException {
00076   public:
00080     FlightTimeNotFoundException (const std::string&
      iWhat)
00081       : stdair::ObjectNotFoundException (iWhat) {}
00082   };
00083
00087   class FeaturesNotFoundException : public
      stdair::ObjectNotFoundException {
00088   public:
00092     FeaturesNotFoundException (const std::string&
      iWhat)
00093       : stdair::ObjectNotFoundException (iWhat) {}
00094   };
00095
00099   class AirlineNotFoundException : public
      stdair::ObjectNotFoundException {
00100   public:
00104     AirlineNotFoundException (const std::string& iWhat)
```

```
00105       : stdair::ObjectNotFoundException (iWhat) {}
00106   };
00107
00111   class FareInputFileNotFoundException : public
      stdair::FileNotFoundException {
00112   public:
00116     FareInputFileNotFoundException (const
      std::string& iWhat)
00117       : stdair::FileNotFoundException (iWhat) {}
00118   };
00119
00123   class QuotingException : public stdair::RootException {
00124   };
00125
00126   // ///////// Files ///////////
00130   class FareFilePath : public stdair::InputFilePath {
00131   public:
00135     explicit FareFilePath (const stdair::Filename_T& iFilename)
00136       : stdair::InputFilePath (iFilename) {}
00137   };
00138
00139   // //////// Type definitions specific to SimFQT /////////
00143   typedef unsigned int FareQuoteID_T;
00144
00148   typedef boost::shared_ptr<SIMFQT_Service> SIMFQT_ServicePtr_T
      ;
00149 }
00150 #endif // __SIMFQT_SIMFQT_TYPES_HPP
```

23.59 simfqt/ui/cmdline/simfqt.cpp File Reference

23.60 simfqt.cpp

```
00001
00005 // STL
00006 #include <cassert>
00007 #include <iostream>
00008 #include <sstream>
00009 #include <fstream>
00010 #include <string>
00011 // Boost (Extended STL)
00012 #include <boost/program_options.hpp>
00013 #include <boost/tokenizer.hpp>
00014 #include <boost/regex.hpp>
00015 // StdAir
00016 #include <stdair/basic/BasLogParams.hpp>
00017 #include <stdair/basic/BasConst_BomDisplay.hpp>
00018 #include <stdair/basic/BasDBParams.hpp>
00019 #include <stdair/basic/BasConst_DefaultObject.hpp>
00020 #include <stdair/basic/BasConst_Inventory.hpp>
00021 #include <stdair/basic/BasConst_Request.hpp>
00022 #include <stdair/service/Logger.hpp>
00023 #include <stdair/stdair_exceptions.hpp>
00024 #include <stdair/stdair_basic_types.hpp>
00025 #include <stdair/stdair_date_time_types.hpp>
00026 #include <stdair/bom/TravelSolutionStruct.hpp>
00027 #include <stdair/bom/BookingRequestStruct.hpp>
00028 #include <stdair/bom/ParsedKey.hpp>
00029 #include <stdair/bom/BomKeyManager.hpp>
00030 #include <stdair/command/CmdBomManager.hpp>
00031 // Stdair GNU Readline Wrapper
00032 #include <stdair/ui/cmdline/SReadline.hpp>
00033 // Simfqt
00034 #include <simfqt/SIMFQT_Service.hpp>
00035 #include <simfqt/config/simfqt-paths.hpp>
00036
00037
00038 // //////// Constants //////
00042 const std::string K_SIMFQT_DEFAULT_LOG_FILENAME ("
      simfqt.log");
00043
00047 const std::string K_SIMFQT_DEFAULT_FARE_INPUT_FILENAME
       (STDAIR_SAMPLE_DIR
00048                                                    "/fare01.csv");
00049
00054 const bool K_SIMFQT_DEFAULT_BUILT_IN_INPUT =
      false;
00055
00059 const int K_SIMFQT_EARLY_RETURN_STATUS = 99;
00060
00065 typedef std::vector<std::string> TokenList_T;
00066
00070 struct Command_T {
```

```
00071   typedef enum {
00072     NOP = 0,
00073     QUIT,
00074     HELP,
00075     LIST,
00076     DISPLAY,
00077     PRICE,
00078     LAST_VALUE
00079   } Type_T;
00080 };
00081
00082 // ///////// Parsing of Options & Configuration /////////
00083 // A helper function to simplify the main part.
00084 template<class T> std::ostream& operator<< (std::ostream& os,
00085                                             const std::vector<T>& v) {
00086   std::copy (v.begin(), v.end(), std::ostream_iterator<T> (std::cout, " "));
00087   return os;
00088 }
00089
00093 int readConfiguration (int argc, char* argv[], bool&
      ioIsBuiltin,
00094                        stdair::Filename_T& ioFareInputFilename,
00095                        std::string& ioLogFilename) {
00096
00097   // Default for the built-in input
00098   ioIsBuiltin = K_SIMFQT_DEFAULT_BUILT_IN_INPUT;
00099
00100   // Declare a group of options that will be allowed only on command line
00101   boost::program_options::options_description generic ("Generic options");
00102   generic.add_options()
00103     ("prefix", "print installation prefix")
00104     ("version,v", "print version string")
00105     ("help,h", "produce help message");
00106
00107   // Declare a group of options that will be allowed both on command
00108   // line and in config file
00109   boost::program_options::options_description config ("Configuration");
00110   config.add_options()
00111     ("builtin,b",
00112      "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")
00113     ("fare,f",
00114      boost::program_options::value< std::string >(&ioFareInputFilename)->
      default_value(K_SIMFQT_DEFAULT_FARE_INPUT_FILENAME
      ),
00115      "(CSV) input file for the fare rules")
00116     ("log,l",
00117      boost::program_options::value< std::string >(&ioLogFilename)->
      default_value(K_SIMFQT_DEFAULT_LOG_FILENAME),
00118      "Filename for the logs")
00119     ;
00120
00121   // Hidden options, will be allowed both on command line and
00122   // in config file, but will not be shown to the user.
00123   boost::program_options::options_description hidden ("Hidden options");
00124   hidden.add_options()
00125     ("copyright",
00126      boost::program_options::value< std::vector<std::string> >(),
00127      "Show the copyright (license)");
00128
00129   boost::program_options::options_description cmdline_options;
00130   cmdline_options.add(generic).add(config).add(hidden);
00131
00132   boost::program_options::options_description config_file_options;
00133   config_file_options.add(config).add(hidden);
00134
00135   boost::program_options::options_description visible ("Allowed options");
00136   visible.add(generic).add(config);
00137
00138   boost::program_options::positional_options_description p;
00139   p.add ("copyright", -1);
00140
00141   boost::program_options::variables_map vm;
00142   boost::program_options::
00143     store (boost::program_options::command_line_parser (argc, argv).
00144            options (cmdline_options).positional(p).run(), vm);
00145
00146   std::ifstream ifs ("simfqt.cfg");
00147   boost::program_options::store (parse_config_file (ifs, config_file_options),
00148                                  vm);
00149   boost::program_options::notify (vm); if (vm.count ("help")) {
00150     std::cout << visible << std::endl;
00151     return K_SIMFQT_EARLY_RETURN_STATUS;
00152   }
00153
00154   if (vm.count ("version")) {
00155     std::cout << PACKAGE_NAME << ", version " << PACKAGE_VERSION
```

```
      << std::endl;
00156     return K_SIMFQT_EARLY_RETURN_STATUS;
00157   }
00158
00159   if (vm.count ("prefix")) {
00160     std::cout << "Installation prefix: " << PREFIXDIR << std::endl;
00161     return K_SIMFQT_EARLY_RETURN_STATUS;
00162   }
00163
00164   if (vm.count ("builtin")) {
00165     ioIsBuiltin = true;
00166   }
00167   const std::string isBuiltinStr = (ioIsBuiltin == true)?"yes":"no";
00168   std::cout << "The BOM should be built-in? " << isBuiltinStr << std::endl;
00169
00170   if (ioIsBuiltin == false) {
00171
00172     // The BOM tree should be built from parsing a fare (and O&D) file
00173     if (vm.count ("fare")) {
00174       ioFareInputFilename = vm["fare"].as< std::string >();
00175       std::cout << "Input fare filename is: " << ioFareInputFilename
00176                 << std::endl;
00177
00178     } else {
00179       // The built-in option is not selected. However, no fare file
00180       // is specified
00181       std::cerr << "Either one among the -b/--builtin and -f/--fare "
00182                 << "options must be specified" << std::endl;
00183     }
00184   }
00185
00186   if (vm.count ("log")) {
00187     ioLogFilename = vm["log"].as< std::string >();
00188     std::cout << "Log filename is: " << ioLogFilename << std::endl;
00189   }
00190
00191   return 0;
00192
00193 }
00194
00195 // //////////////////////////////////////////////////////////////////
00196 void initReadline (swift::SReadline& ioInputReader) {
00197
00198   // Prepare the list of my own completers
00199   std::vector<std::string> Completers;
00200
00201   // The following is supported:
00202   // - "identifiers"
00203   // - special identifier %file - means to perform a file name completion
00204   Completers.push_back ("help");
00205   Completers.push_back ("list");
00206   Completers.push_back ("display %airport_code %airport_code %departure_date");
00207   Completers.push_back ("price %airline_code %flight_number %departure_date
      %airport_code %airport_code %departure_time %booking_date %booking_time %POS
      %channel% %trip_type %stay_duration");
00208   Completers.push_back ("quit");
00209
00210   // Now register the completers.
00211   // Actually it is possible to re-register another set at any time
00212   ioInputReader.RegisterCompletions (Completers);
00213 }
00214
00215 // //////////////////////////////////////////////////////////////////
00216 Command_T::Type_T extractCommand (TokenList_T& ioTokenList) {
00217   Command_T::Type_T oCommandType = Command_T::LAST_VALUE;
00218
00219   // Interpret the user input
00220   if (ioTokenList.empty() == false) {
00221     TokenList_T::iterator itTok = ioTokenList.begin();
00222     std::string& lCommand (*itTok);
00223     boost::algorithm::to_lower (lCommand);
00224
00225     if (lCommand == "help") {
00226       oCommandType = Command_T::HELP;
00227
00228     } else if (lCommand == "list") {
00229       oCommandType = Command_T::LIST;
00230
00231     } else if (lCommand == "display") {
00232       oCommandType = Command_T::DISPLAY;
00233
00234     } else if (lCommand == "price") {
00235       oCommandType = Command_T::PRICE;
00236
00237     } else if (lCommand == "quit") {
00238       oCommandType = Command_T::QUIT;
00239
```

```
00240     }
00241
00242     // Remove the first token (the command), as the corresponding information
00243     // has been extracted in the form of the returned command type enumeration
00244     ioTokenList.erase (itTok);
00245
00246   } else {
00247     oCommandType = Command_T::NOP;
00248   }
00249
00250   return oCommandType;
00251 }
00252
00253 // ////////////////////////////////////////////////////////////////////
00254 // Re-compose a date using three strings: the year, the month and the
00255 // day. Return true if a correct date has been computed, false if not.
00256 bool retrieveDate (std::string iYearString,
00257                    std::string iMonthString,
00258                    std::string iDayString,
00259                    stdair::Date_T& ioDate) {
00260
00261   const std::string kMonthStr[12] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
00262                                      "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
00263
00264   // Check the year.
00265   unsigned short lDateYear;
00266   try {
00267
00268     lDateYear = boost::lexical_cast<unsigned short> (iYearString);
00269     if (lDateYear < 100) {
00270       lDateYear += 2000;
00271     }
00272
00273   } catch (boost::bad_lexical_cast& eCast) {
00274     std::cerr << "The year ('" << iYearString
00275               << "') cannot be understood." << std::endl;
00276     return false;
00277   }
00278
00279   // Check the month.
00280   std::string lDateMonthStr;
00281   try {
00282
00283     const boost::regex lMonthRegex ("^(\\d{1,2})$");
00284     const bool isMonthANumber = regex_match (iMonthString, lMonthRegex);
00285
00286     if (isMonthANumber == true) {
00287       const unsigned short lMonth =
00288         boost::lexical_cast<unsigned short> (iMonthString);
00289       if (lMonth > 12) {
00290         throw boost::bad_lexical_cast();
00291       }
00292       if (lMonth != 0) {
00293         lDateMonthStr = kMonthStr[lMonth-1];
00294       } else {
00295         std::cerr << "The month ('" << iMonthString
00296                   << "') cannot be understood." << std::endl;
00297         return false;
00298       }
00299
00300     } else {
00301       if (iMonthString.size() < 3) {
00302         throw boost::bad_lexical_cast();
00303       }
00304       std::string lMonthStr1 (iMonthString.substr (0, 1));
00305       boost::algorithm::to_upper (lMonthStr1);
00306       std::string lMonthStr23 (iMonthString.substr (1, 2));
00307       boost::algorithm::to_lower (lMonthStr23);
00308       lDateMonthStr = lMonthStr1 + lMonthStr23;
00309     }
00310
00311   } catch (boost::bad_lexical_cast& eCast) {
00312     std::cerr << "The month ('" << iMonthString
00313               << "') cannot be understood." << std::endl;
00314     return false;
00315   }
00316
00317   // Check the day.
00318   unsigned short lDateDay;
00319   try {
00320
00321     lDateDay = boost::lexical_cast<unsigned short> (iDayString);
00322
00323   } catch (boost::bad_lexical_cast& eCast) {
00324     std::cerr << "The day ('" << iDayString
00325               << "') cannot be understood." << std::endl;
00326     return false;
```

```
00327   }
00328
00329   // Re-compose the date.
00330   std::ostringstream lDateStr;
00331   lDateStr << lDateYear << "-" << lDateMonthStr
00332            << "-" << lDateDay;
00333   try {
00334
00335     ioDate =
00336       boost::gregorian::from_simple_string (lDateStr.str());
00337
00338   } catch (boost::gregorian::bad_month& eCast) {
00339     std::cerr << "The month of the date ('" << lDateStr.str()
00340               << "') cannot be understood." << std::endl;
00341     return false;
00342   } catch (boost::gregorian::bad_day_of_month& eCast) {
00343     std::cerr << "The date ('" << lDateStr.str()
00344               << "') is not correct: the day of month does not exist."
00345               << std::endl;
00346     return false;
00347   } catch (boost::gregorian::bad_year& eCast) {
00348     std::cerr << "The year ('" << lDateStr.str()
00349               << "') is not correct."
00350               << std::endl;
00351     return false;
00352   }
00353
00354   return true;
00355 }
00356
00357 // ////////////////////////////////////////////////////////////////////
00358 // Re-compose a time using two strings: the hour and the minute.
00359 // Return true if a correct time has been computed, false if not.
00360 bool retrieveTime (std::string iHourString,
00361                    std::string iMinuteString,
00362                    stdair::Duration_T& oTime) {
00363
00364   // Check the hour
00365   unsigned short lTimeHour;
00366   try {
00367
00368     lTimeHour = boost::lexical_cast<unsigned short> (iHourString);
00369
00370   } catch (boost::bad_lexical_cast& eCast) {
00371     std::cerr << "The hour of the time ('" << iHourString
00372               << "') cannot be understood." << std::endl;
00373     return false;
00374   }
00375
00376   // Check the minutes
00377   unsigned short lTimeMinute;
00378   try {
00379
00380     lTimeMinute = boost::lexical_cast<unsigned short> (iMinuteString);
00381
00382   } catch (boost::bad_lexical_cast& eCast) {
00383     std::cerr << "The minute of the time ('" << iMinuteString
00384               << "') cannot be understood." << std::endl;
00385     return false;
00386   }
00387
00388
00389   // Re-compose the time
00390   std::ostringstream lTimeStr;
00391   lTimeStr << lTimeHour << ":" << lTimeMinute;
00392   oTime =
00393     boost::posix_time::duration_from_string (lTimeStr.str());
00394
00395   return true;
00396 }
00397
00398 // ////////////////////////////////////////////////////////////////////
00399 // Analyze the tokens of the 'price' command in order to construct
00400 // a travel solution list and a booking request.
00401 const stdair::BookingRequestStruct parseTravelSolutionAndBookingRequestKey
00402 (const TokenList_T& iTokenList,
00403  stdair::TravelSolutionList_T& ioInteractiveTravelSolutionList,
00404  const stdair::BookingRequestStruct& ioBookingRequestStruct) {
00405
00406   TokenList_T::const_iterator itTok = iTokenList.begin();
00407
00408   if (itTok->empty() == true) {
00409
00410     std::cerr << "Wrong list of parameters. "
00411               << "The default booking request and travel solution list are
      kept."
00412               << std::endl;
```

```
00413     return ioBookingRequestStruct;
00414
00415
00416   } else {
00417     // Parameters corresponding to the tokens.
00418     // Each parameter correponds to one token except the dates
00419     // (three tokens) and the times (two tokens).
00420     stdair::AirlineCode_T lAirlineCode;
00421     stdair::FlightNumber_T lflightNumber;
00422     stdair::Date_T lDepartureDate;
00423     stdair::Duration_T lDepartureTime;
00424     stdair::AirportCode_T lOriginAirport;
00425     stdair::AirportCode_T lDestinationAirport;
00426     stdair::Date_T lRequestDate;
00427     stdair::Duration_T lRequestTime;
00428     stdair::CityCode_T lPOS;
00429     stdair::ChannelLabel_T lChannel;
00430     stdair::TripType_T lTripType;
00431     unsigned short lStayDuration;
00432
00433     // Read the airline code.
00434     lAirlineCode = *itTok;
00435     boost::algorithm::to_upper (lAirlineCode);
00436
00437     // Read the flight-number  .
00438     ++itTok;
00439     if (itTok->empty() == false) {
00440       try {
00441
00442         lflightNumber = boost::lexical_cast<stdair::FlightNumber_T> (*itTok);
00443
00444       } catch (boost::bad_lexical_cast& eCast) {
00445         std::cerr << "The flight number ('" << *itTok
00446                   << "') cannot be understood."
00447                   << std::endl;
00448         return ioBookingRequestStruct;
00449       }
00450     }
00451
00452     // Read the departure date.
00453     ++itTok;
00454     if (itTok->empty() == true) {
00455       return ioBookingRequestStruct;
00456     }
00457     const std::string lDepartureYearString = *itTok;
00458     ++itTok;
00459     if (itTok->empty() == true) {
00460       return ioBookingRequestStruct;
00461     }
00462     const std::string lDepartureMonthString = *itTok;
00463     ++itTok;
00464     if (itTok->empty() == true) {
00465       return ioBookingRequestStruct;
00466     }
00467     const std::string lDepartureDayString = *itTok;
00468     const bool IsDepartureDateReadable =
00469       retrieveDate (lDepartureYearString, lDepartureMonthString,
00470                     lDepartureDayString, lDepartureDate);
00471
00472     if (IsDepartureDateReadable == false) {
00473       std::cerr << "The default booking request and travel solution list are
      kept."
00474                 << std::endl;
00475       return ioBookingRequestStruct;
00476     }
00477
00478     // Read the origin.
00479     ++itTok;
00480     if (itTok->empty() == false) {
00481       lOriginAirport = *itTok;
00482       boost::algorithm::to_upper (lOriginAirport);
00483     }
00484
00485     // Read the destination.
00486     ++itTok;
00487     if (itTok->empty() == false) {
00488       lDestinationAirport = *itTok;
00489       boost::algorithm::to_upper (lDestinationAirport);
00490     }
00491
00492     // Read the departure time.
00493     ++itTok;
00494     if (itTok->empty() == true) {
00495       return ioBookingRequestStruct;
00496     }
00497     const std::string lDepartureHourString = *itTok;
00498     ++itTok;
```

```
00499     if (itTok->empty() == true) {
00500       return ioBookingRequestStruct;
00501     }
00502     const std::string lDepartureMinuteString = *itTok;
00503     const bool IsDepartureTimeReadable =
00504       retrieveTime (lDepartureHourString, lDepartureMinuteString,
00505                     lDepartureTime);
00506
00507     if (IsDepartureTimeReadable == false) {
00508       std::cerr << "The default booking request and travel solution list are
      kept."
00509                 << std::endl;
00510       return ioBookingRequestStruct;
00511     }
00512
00513     // Read the request date.
00514     ++itTok;
00515     if (itTok->empty() == true) {
00516         return ioBookingRequestStruct;
00517     }
00518     const std::string lRequestYearString = *itTok;
00519     ++itTok;
00520     if (itTok->empty() == true) {
00521       return ioBookingRequestStruct;
00522     }
00523     const std::string lRequestMonthString = *itTok;
00524     ++itTok;
00525     if (itTok->empty() == true) {
00526       return ioBookingRequestStruct;
00527     }
00528     const std::string lRequestDayString = *itTok;
00529     const bool IsRequestDateReadable =
00530       retrieveDate (lRequestYearString, lRequestMonthString,
00531                     lRequestDayString, lRequestDate);
00532
00533     if (IsRequestDateReadable == false) {
00534       std::cerr << "The default booking request and travel solution list are
      kept."
00535                 << std::endl;
00536       return ioBookingRequestStruct;
00537     }
00538
00539     // Read the request time.
00540     ++itTok;
00541     if (itTok->empty() == true) {
00542       return ioBookingRequestStruct;
00543     }
00544     const std::string lRequestHourString = *itTok;
00545     ++itTok;
00546     if (itTok->empty() == true) {
00547       return ioBookingRequestStruct;
00548     }
00549     const std::string lRequestMinuteString = *itTok;
00550     const bool IsRequestTimeReadable =
00551       retrieveTime (lRequestHourString, lRequestMinuteString,
00552                     lRequestTime);
00553
00554     if (IsRequestTimeReadable == false) {
00555       std::cerr << "The default booking request and travel solution list are
      kept."
00556                 << std::endl;
00557       return ioBookingRequestStruct;
00558     }
00559
00560     // Read the POS.
00561     ++itTok;
00562     if (itTok->empty() == false) {
00563       lPOS = *itTok;
00564       boost::algorithm::to_upper (lPOS);
00565     }
00566
00567     // Read the channel.
00568     ++itTok;
00569     if (itTok->empty() == false) {
00570       lChannel = *itTok;
00571       boost::algorithm::to_upper (lChannel);
00572     }
00573
00574     // Read the trip type.
00575     ++itTok;
00576     if (itTok->empty() == false) {
00577       lTripType = *itTok;
00578       boost::algorithm::to_upper (lTripType);
00579     }
00580
00581     // Read the stay duration.
00582     ++itTok;
```

```
00583     if (itTok->empty() == false) {
00584       try {
00585
00586         lStayDuration = boost::lexical_cast<unsigned short> (*itTok);
00587
00588       } catch (boost::bad_lexical_cast& eCast) {
00589         std::cerr << "The stay duration ('" << *itTok
00590                   << "') cannot be understood." << std::endl;
00591         return ioBookingRequestStruct;
00592       }
00593     }
00594
00595     // At this step we know that all the parameters designed to construct
00596     // the travel solution and the booking request are correct.
00597
00598     // Empty the travel solution list to store a new travel solution.
00599     ioInteractiveTravelSolutionList.pop_front();
00600     // Construct the new travel solution.
00601     stdair::TravelSolutionStruct lTravelSolution;
00602     std::ostringstream oStr;
00603     oStr << lAirlineCode
00604          << stdair::DEFAULT_KEY_FLD_DELIMITER
00605          << lflightNumber
00606          << stdair::DEFAULT_KEY_SUB_FLD_DELIMITER
00607          << lDepartureDate
00608          << stdair::DEFAULT_KEY_FLD_DELIMITER
00609          << lOriginAirport
00610          << stdair::DEFAULT_KEY_SUB_FLD_DELIMITER
00611          << lDestinationAirport
00612          << stdair::DEFAULT_KEY_FLD_DELIMITER
00613          << lDepartureTime;
00614     lTravelSolution.addSegment (oStr.str());
00615     ioInteractiveTravelSolutionList.push_front(lTravelSolution);
00616
00617     // Construct the new booking request.
00618     stdair::DateTime_T lRequestDateTime (lRequestDate, lRequestTime);
00619     const stdair::BookingRequestStruct &lBookingRequestStruct =
00620       stdair::BookingRequestStruct(lOriginAirport,
00621                                    lDestinationAirport,
00622                                    lPOS,
00623                                    lDepartureDate,
00624                                    lRequestDateTime,
00625                                    stdair::CABIN_ECO,
00626                                    stdair::DEFAULT_PARTY_SIZE,
00627                                    lChannel,
00628                                    lTripType,
00629                                    lStayDuration,
00630                                    stdair::FREQUENT_FLYER_MEMBER,
00631                                    lDepartureTime,
00632                                    stdair::DEFAULT_WTP,
00633                                    stdair::DEFAULT_VALUE_OF_TIME,
00634                                    true, 50, true, 50);
00635
00636     return lBookingRequestStruct;
00637   }
00638 }
00639
00640 // ////////////////////////////////////////////////////////////////////
00641 // Analyze the tokens of the 'display' command in order to retrieve
00642 // an airport pair and a departure date.
00643 void parseFlightDateKey (const TokenList_T& iTokenList,
00644                          stdair::AirportCode_T& ioOrigin,
00645                          stdair::AirportCode_T& ioDestination,
00646                          stdair::Date_T& ioDepartureDate) {
00647
00648   TokenList_T::const_iterator itTok = iTokenList.begin();
00649
00650   // Interpret the user input.
00651   if (itTok->empty() == true) {
00652
00653     std::cerr << "Wrong parameters specified. Default paramaters '"
00654               << ioOrigin << "-" << ioDestination
00655               << "/" << ioDepartureDate
00656               << "' are kept."
00657              << std::endl;
00658
00659   } else {
00660
00661     // Read the origin.
00662     ioOrigin = *itTok;
00663     boost::algorithm::to_upper (ioOrigin);
00664
00665     // Read the destination.
00666     ++itTok;
00667     if (itTok->empty() == false) {
00668       ioDestination  = *itTok;
00669       boost::algorithm::to_upper (ioDestination);
```

```
00670     }
00671
00672     // Read the departure date.
00673     ++itTok;
00674     if (itTok->empty() == true) {
00675       return;
00676     }
00677     std::string lYearString = *itTok;
00678     ++itTok;
00679     if (itTok->empty() == true) {
00680       return;
00681     }
00682     std::string lMonthString = *itTok;
00683     ++itTok;
00684     if (itTok->empty() == true) {
00685       return;
00686     }
00687     std::string lDayString = *itTok;
00688     const bool IsDepartureDateReadable =
00689       retrieveDate (lYearString, lMonthString, lDayString,
00690                     ioDepartureDate);
00691     if (IsDepartureDateReadable == false) {
00692       std::cerr << "Default paramaters '"
00693                 << ioOrigin << "-" << ioDestination
00694                 << "/" << ioDepartureDate
00695                 << "' are kept."
00696                 << std::endl;
00697       return;
00698     }
00699   }
00700 }
00701
00702 // //////////////////////////////////////////////////////////
00703 std::string toString (const TokenList_T& iTokenList) {
00704   std::ostringstream oStr;
00705
00706   // Re-create the string with all the tokens, trimmed by read-line
00707   unsigned short idx = 0;
00708   for (TokenList_T::const_iterator itTok = iTokenList.begin();
00709        itTok != iTokenList.end(); ++itTok, ++idx) {
00710     if (idx != 0) {
00711       oStr << " ";
00712     }
00713     oStr << *itTok;
00714   }
00715
00716   return oStr.str();
00717 }
00718
00719 // //////////////////////////////////////////////////////////
00720 TokenList_T extractTokenList (const TokenList_T& iTokenList,
00721                               const std::string& iRegularExpression) {
00722   TokenList_T oTokenList;
00723
00724   // Re-create the string with all the tokens (which had been trimmed
00725   // by read-line)
00726   const std::string lFullLine = toString (iTokenList);
00727
00728   // See the caller for the regular expression
00729   boost::regex expression (iRegularExpression);
00730
00731   std::string::const_iterator start = lFullLine.begin();
00732   std::string::const_iterator end = lFullLine.end();
00733
00734   boost::match_results<std::string::const_iterator> what;
00735   boost::match_flag_type flags = boost::match_default | boost::format_sed;
00736   regex_search (start, end, what, expression, flags);
00737
00738   // Put the matched strings in the list of tokens to be returned back
00739   // to the caller
00740   const unsigned short lMatchSetSize = what.size();
00741   for (unsigned short matchIdx = 1; matchIdx != lMatchSetSize; ++matchIdx) {
00742     const std::string lMatchedString (std::string (what[matchIdx].first,
00743                                                    what[matchIdx].second));
00744     //if (lMatchedString.empty() == false) {
00745     oTokenList.push_back (lMatchedString);
00746     //}
00747   }
00748
00749   // DEBUG
00750   // std::cout << "After (token list): " << oTokenList << std::endl;
00751
00752   return oTokenList;
00753 }
00754
00755 // //////////////////////////////////////////////////////////
00756 // Parse the token list of the 'price' command.
```

```
00757 TokenList_T extractTokenListForTSAndBR (const TokenList_T& iTokenList) {
00779   const std::string lRegEx("^([[:alpha:]]{2,3})"
00780                            "[[:space:]]+([[:digit:]]{1,4})"
00781                            "[/ ]*"
00782                            "[[:space:]]+([[:digit:]]{2,4})[/-]?"
00783                            "[[:space:]]*([[:alpha:]]{3}|[[:digit:]]{1,2})[/-]?"
00784                            "[[:space:]]*([[:digit:]]{1,2})[[:space:]]*"
00785                            "[[:space:]]+([[:alpha:]]{3})"
00786                            "[[:space:]]+([[:alpha:]]{3})"
00787                            "
      [[:space:]]+([[:digit:]]{1,2})[:]?([[:digit:]]{1,2})"
00788                            "[[:space:]]+([[:digit:]]{2,4})[/-]?"
00789                            "[[:space:]]*([[:alpha:]]{3}|[[:digit:]]{1,2})[/-]?"
00790                            "[[:space:]]*([[:digit:]]{1,2})"
00791                            "
      [[:space:]]+([[:digit:]]{1,2})[:]?([[:digit:]]{1,2})"
00792                            "[[:space:]]+([[:alpha:]]{3})"
00793                            "[[:space:]]+([[:alpha:]]{2})"
00794                            "[[:space:]]+([[:alpha:]]{2})"
00795                            "[[:space:]]+([[:digit:]]{1})$");
00796
00797   //
00798   const TokenList_T& oTokenList = extractTokenList (iTokenList, lRegEx);
00799   return oTokenList;
00800 }
00801
00802 // ////////////////////////////////////////////////////////
00803 // Parse the token list of the 'display' command.
00804 TokenList_T extractTokenListForOriDestDate (const TokenList_T& iTokenList) {
00814   const std::string lRegEx("^([[:alpha:]]{3})"
00815                            "[[:space:]]*[/-]?"
00816                            "[[:space:]]*([[:alpha:]]{3})"
00817                            "[[:space:]]*[/-]?"
00818                            "[[:space:]]*([[:digit:]]{2,4})"
00819                            "[[:space:]]*[/-]?"
00820                            "[[:space:]]*([[:alpha:]]{3}|[[:digit:]]{1,2})"
00821                            "[[:space:]]*[/-]?"
00822                            "[[:space:]]*([[:digit:]]{1,2})$");
00823
00824   //
00825   const TokenList_T& oTokenList = extractTokenList (iTokenList, lRegEx);
00826   return oTokenList;
00827 }
00828
00829 // ///////// M A I N ////////////
00830 int main (int argc, char* argv[]) {
00831
00832   // State whether the BOM tree should be built-in or parsed from an
00833   // input file
00834   bool isBuiltin;
00835
00836   // Fare input file name
00837   stdair::Filename_T lFareInputFilename;
00838
00839   // Readline history
00840   const unsigned int lHistorySize (100);
00841   const std::string lHistoryFilename ("simfqt.hist");
00842   const std::string lHistoryBackupFilename ("simfqt.hist.bak");
00843
00844   // Default parameters for the interactive session
00845   stdair::AirportCode_T lInteractiveOrigin;
00846   stdair::AirportCode_T lInteractiveDestination;
00847   stdair::Date_T lInteractiveDepartureDate;
00848
00849   // Output log File
00850   stdair::Filename_T lLogFilename;
00851
00852   // Call the command-line option parser
00853   const int lOptionParserStatus =
00854     readConfiguration (argc, argv, isBuiltin,
      lFareInputFilename, lLogFilename);
00855
00856   if (lOptionParserStatus == K_SIMFQT_EARLY_RETURN_STATUS
      ) {
00857     return 0;
00858   }
00859
00860   // Set the log parameters
00861   std::ofstream logOutputFile;
00862   // Open and clean the log outputfile
00863   logOutputFile.open (lLogFilename.c_str());
00864   logOutputFile.clear();
00865
00866   // Initialise the fareQuote service
00867   const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);
00868   SIMFQT::SIMFQT_Service simfqtService (lLogParams);
00869
```

```
00870   // DEBUG
00871   STDAIR_LOG_DEBUG ("Welcome to SimFQT display");
00872
00873   // Check wether or not a (CSV) input file should be read
00874   if (isBuiltin == true) {
00875     // Build the sample BOM tree (filled with fares) for Simfqt
00876     simfqtService.buildSampleBom();
00877   } else {
00878     // Build the BOM tree from parsing a fare file
00879     SIMFQT::FareFilePath lFareFilePath (lFareInputFilename)
      ;
00880     simfqtService.parseAndLoad (lFareFilePath);
00881   }
00882
00883   // DEBUG: Display the whole BOM tree
00884   const std::string& lCSVDump = simfqtService.csvDisplay();
00885   STDAIR_LOG_DEBUG (lCSVDump);
00886
00887   // DEBUG
00888   STDAIR_LOG_DEBUG ("=====================================================");
00889   STDAIR_LOG_DEBUG ("=       Beginning of the interactive session       =");
00890   STDAIR_LOG_DEBUG ("=====================================================");
00891
00892   // Initialise the GNU readline wrapper
00893   swift::SReadline lReader (lHistoryFilename, lHistorySize);
00894   initReadline (lReader);
00895
00896   // Now we can ask user for a line
00897   std::string lUserInput;
00898   bool EndOfInput (false);
00899   Command_T::Type_T lCommandType (Command_T::NOP);
00900
00901   while (lCommandType != Command_T::QUIT && EndOfInput == false) {
00902
00903     stdair::TravelSolutionList_T lInteractiveTravelSolutionList;
00904     stdair::TravelSolutionStruct lInteractiveTravelSolution;
00905
00906     // Update the default booking request.
00907     // If there is an input file, we want the CRS booking request (defined in
       stdair).
00908     // If not, we want the default booking request.
00909     const bool isCRSBookingRequest = !isBuiltin;
00910     const stdair::BookingRequestStruct& lInteractiveBookingRequest =
00911       simfqtService.buildBookingRequest (isCRSBookingRequest);
00912
00913     // Update the default parameters for the following interactive session.
00914     if (isBuiltin == true) {
00915       lInteractiveOrigin = "LHR";
00916       lInteractiveDestination = "SYD";
00917       lInteractiveDepartureDate = stdair::Date_T(2011,06,10);
00918       simfqtService.buildSampleTravelSolutions (lInteractiveTravelSolutionList)
      ;
00919     } else {
00920       lInteractiveOrigin = "SIN";
00921       lInteractiveDestination = "BKK";
00922       lInteractiveDepartureDate = stdair::Date_T(2010,01,30);
00923       //
00924       const std::string lBA9_SegmentDateKey ("SQ, 970, 2010-01-30, SIN, BKK,
       07:10");
00925
00926       // Add the segment date key to the travel solution.
00927       lInteractiveTravelSolution.addSegment (lBA9_SegmentDateKey);
00928
00929       // Add the travel solution to the list
00930       lInteractiveTravelSolutionList.push_back (lInteractiveTravelSolution);
00931     }
00932
00933     // Prompt.
00934     std::ostringstream oPromptStr;
00935     oPromptStr << "simfqt "
00936                << "> ";
00937     // The last parameter could be ommited.
00938     TokenList_T lTokenListByReadline;
00939     lUserInput = lReader.GetLine (oPromptStr.str(), lTokenListByReadline,
00940                                   EndOfInput);
00941
00942     // The history could be saved to an arbitrary file at any time.
00943     lReader.SaveHistory (lHistoryBackupFilename);
00944
00945     if (EndOfInput) {
00946       std::cout << std::endl;
00947       break;
00948     }
00949
00950     // Interpret the user input.
00951     lCommandType = extractCommand (lTokenListByReadline);
00952
```

```
00953     switch (lCommandType) {
00954
00955       // ////////////////////////////// Help ////////////////////////
00956     case Command_T::HELP: {
00957       // Search for information to display default parameters lists.
00958       // Get the first travel solution.
00959       stdair::TravelSolutionStruct& lTravelSolutionStruct =
00960         lInteractiveTravelSolutionList.front();
00961       // Get the segment-path of the first travel solution.
00962       const stdair::SegmentPath_T& lSegmentPath =
00963         lTravelSolutionStruct.getSegmentPath();
00964       // Get the first segment of the first travel solution.
00965       const std::string& lSegmentDateKey = lSegmentPath.front();
00966       // Get the parsed key of the first segment of the first travel solution.
00967       const stdair::ParsedKey& lParsedKey =
00968         stdair::BomKeyManager::extractKeys (lSegmentDateKey);
00969       // Get the request date time
00970       const stdair::DateTime_T& lRequestDateTime =
00971         lInteractiveBookingRequest.getRequestDateTime();
00972       const stdair::Time_T lRequestTime =
00973         lRequestDateTime.time_of_day();
00974       std::cout << std::endl;
00975       // Display help.
00976       std::cout << "Commands: " << std::endl;
00977       std::cout << " help" << "\t\t" << "Display this help" << std::endl;
00978       std::cout << " quit" << "\t\t" << "Quit the application" << std::endl;
00979       std::cout << " list" << "\t\t"
00980                 << "List all the fare rule O&Ds and the corresponding date
      ranges" << std::endl;
00981       std::cout << " display" << "\t"
00982                 << "Display all fare rules for an O&D and a departure date. \n"
      << "\t\t"
00983                 << "If no parameters specified or wrong list of parameters,
      default values are used: \n"<< "\t\t"
00984                 << "      display " <<  lInteractiveOrigin << " "
00985                 << lInteractiveDestination << " "
00986                 << lInteractiveDepartureDate << std::endl;
00987       std::cout << " price" << "\t\t"
00988                 << "Price the travel solution corresponding to a booking
      request. \n" << "\t\t"
00989                 << "If no parameters specified or wrong list of parameters,
      default value are used: \n" << "\t\t"
00990                 << "      price "
00991                 << lParsedKey._airlineCode << " "
00992                 << lParsedKey._flightNumber << " "
00993                 << lParsedKey._departureDate << " "
00994                 << lParsedKey._boardingPoint << " "
00995                 << lParsedKey._offPoint << " "
00996                 << lParsedKey._boardingTime << " "
00997                 << lRequestDateTime.date() << " "
00998                 << lRequestTime.hours() << ":" << lRequestTime.minutes() << " "

00999                 << lInteractiveBookingRequest.getPOS() << " "
01000                 << lInteractiveBookingRequest.getBookingChannel() << " "
01001                 << lInteractiveBookingRequest.getTripType() << " "
01002                 << lInteractiveBookingRequest.getStayDuration() << std::endl;
01003       std::cout << std::endl;
01004       break;
01005     }
01006
01007       // ////////////////////////////// Quit ////////////////////////
01008     case Command_T::QUIT: {
01009       break;
01010     }
01011
01012       // ////////////////////////////// List ////////////////////////
01013     case Command_T::LIST: {
01014
01015       // Get the list of all airport pairs and date ranges for which
01016       // there are fares available.
01017       const std::string& lAirportPairDateListStr =
01018         simfqtService.list ();
01019
01020       if (lAirportPairDateListStr.empty() == false) {
01021         std::cout << lAirportPairDateListStr << std::endl;
01022         STDAIR_LOG_DEBUG (lAirportPairDateListStr);
01023
01024       } else {
01025         std::cerr << "There is no result for airport pairs and date ranges."
01026                   << "Make sure your input file is not empty."
01027                   << std::endl;
01028       }
01029
01030       break;
01031     }
01032
01033       // ////////////////////////////// Display ////////////////////////
```

```
01034     case Command_T::DISPLAY: {
01035
01036       // If no parameters are entered by the user, keep default ones.
01037       if (lTokenListByReadline.empty() == true) {
01038
01039         std::cout << "No parameters specified. Default paramaters '"
01040                   << lInteractiveOrigin << "-" << lInteractiveDestination
01041                   << "/" << lInteractiveDepartureDate
01042                   << "' are kept."
01043                   << std::endl;
01044
01045       } else {
01046
01047         // Find the best match corresponding to the given parameters.
01048         TokenList_T lTokenList =
01049           extractTokenListForOriDestDate (lTokenListByReadline);
01050
01051         // Parse the best match, and give default values in case the
01052         // user does not specify all the parameters or does not
01053         // specify some of them correctly.
01054         parseFlightDateKey (lTokenList, lInteractiveOrigin,
01055                             lInteractiveDestination, lInteractiveDepartureDate)
      ;
01056
01057       }
01058
01059       // Check whether the selected airportpair-date is valid:
01060       // i.e. if there are corresponding fare rules.
01061       const bool isAirportPairDateValid =
01062         simfqtService.check (lInteractiveOrigin, lInteractiveDestination,
01063                              lInteractiveDepartureDate);
01064
01065       if (isAirportPairDateValid == false) {
01066         std::ostringstream oFDKStr;
01067         oFDKStr << "The airport pair/departure date: "
01068                 << lInteractiveOrigin << "-" << lInteractiveDestination
01069                 << "/" << lInteractiveDepartureDate
01070                 << " does not correpond to any fare rule.\n"
01071                 << "Make sure it exists with the 'list' command.";
01072         std::cout << oFDKStr.str() << std::endl;
01073         STDAIR_LOG_ERROR (oFDKStr.str());
01074
01075         break;
01076       }
01077
01078       // Display the list of corresponding fare rules.
01079       std::cout << "List of fare rules for "
01080                 << lInteractiveOrigin << "-"
01081                 << lInteractiveDestination << "/"
01082                 << lInteractiveDepartureDate
01083                 << std::endl;
01084
01085       const std::string& lFareRuleListStr =
01086         simfqtService.csvDisplay (lInteractiveOrigin,
01087                                   lInteractiveDestination,
01088                                   lInteractiveDepartureDate);
01089
01090       assert (lFareRuleListStr.empty() == false);
01091       std::cout << lFareRuleListStr << std::endl;
01092       STDAIR_LOG_DEBUG (lFareRuleListStr);
01093
01094       break;
01095     }
01096
01097       // ////////////////////////////// Price ////////////////////////
01098     case Command_T::PRICE: {
01099
01100       // If no parameters are entered by the user, keep default ones.
01101       if (lTokenListByReadline.empty() == true) {
01102
01103         lInteractiveTravelSolution = lInteractiveTravelSolutionList.front();
01104
01105         std::cout << "No parameters specified. Default booking request and
       default travel solution list are kept.\n"
01106                   << "Booking request: << "
01107                   << lInteractiveBookingRequest.display()  << " >>"
01108                   << "\nTravel Solution: << "
01109                   << lInteractiveTravelSolution.display() << " >>"
01110                   << "\n********** \n"
01111                   << "Fare quote"
01112                   << "\n**********"
01113                  << std::endl;
01114
01115         // Try to fareQuote the sample list of travel solutions.
01116         try {
01117         simfqtService.quotePrices (lInteractiveBookingRequest,
01118                                    lInteractiveTravelSolutionList);
```

```
01119         } catch (stdair::ObjectNotFoundException& E) {
01120           std::cerr << "The given travel solution corresponding to the given
      booking request can not be priced.\n"
01121                     << E.what()
01122                     << std::endl;
01123           break;
01124         }
01125       } else {
01126
01127         // Find the best match corresponding to the given parameters.
01128         TokenList_T lTokenList =
01129           extractTokenListForTSAndBR (lTokenListByReadline);
01130
01131         // Parse the best match, and give default values in case the
01132         // user does not specify all the parameters or does not
01133         // specify some of them correctly.
01134         stdair::BookingRequestStruct lFinalBookingRequest
01135           = parseTravelSolutionAndBookingRequestKey (lTokenList,
01136
      lInteractiveTravelSolutionList,
01137                                                    lInteractiveBookingRequest
      );
01138
01139
01140         assert (lInteractiveTravelSolutionList.size() >= 1);
01141         lInteractiveTravelSolution = lInteractiveTravelSolutionList.front();
01142
01143         // Display the booking request and the first travel solution
01144         // before pricing.
01145         std::cout << "Booking request: << "
01146                   << lFinalBookingRequest.display()  << " >>"
01147                   << "\nTravel Solution: << "
01148                   << lInteractiveTravelSolution.display() << " >>"
01149                   << "\n********** \n"
01150                   << "Fare quote"
01151                   << "\n**********"
01152                 << std::endl;
01153
01154         // Try to fareQuote the sample list of travel solutions.
01155         try {
01156           simfqtService.quotePrices (lFinalBookingRequest,
01157                                      lInteractiveTravelSolutionList);
01158         } catch (stdair::ObjectNotFoundException& E) {
01159           std::cerr << "The given travel solution corresponding to the given
      booking request can not be priced.\n"
01160                     << E.what()
01161                     << std::endl;
01162           break;
01163         }
01164       }
01165
01166       // Display the first travel solution after pricing:
01167       // one or more fare option have been added.
01168       lInteractiveTravelSolution = lInteractiveTravelSolutionList.front();
01169       std::cout << "Travel Solution: << "
01170                 << lInteractiveTravelSolution.display() << " >>\n"
01171                 << std::endl;
01172
01173       break;
01174     }
01175
01176       // ///////////////////////////// Default / No value ///////////////////////
01177     case Command_T::NOP: {
01178       break;
01179     }
01180     case Command_T::LAST_VALUE:
01181     default: {
01182       // DEBUG
01183       std::ostringstream oStr;
01184       oStr << "The '" << lUserInput << "' command is not yet understood.\n"
01185            << "Type help to have more information." << std::endl;
01186
01187       STDAIR_LOG_DEBUG (oStr.str());
01188       std::cout << oStr.str() << std::endl;
01189     }
01190     }
01191   }
01192
01193   // DEBUG
01194   STDAIR_LOG_DEBUG ("End of the session. Exiting.");
01195   std::cout << "End of the session. Exiting." << std::endl;
01196
01197   // Close the Log outputFile
01198   logOutputFile.close();
01199
01200   /*
01201     Note: as that program is not intended to be run on a server in
```

```
01202     production, it is better not to catch the exceptions. When it
01203     happens (that an exception is throwned), that way we get the
01204     call stack.
01205   */
01206
01207   return 0;
01208 }
```

23.61 test/simfqt/FQTTestSuite.cpp File Reference

23.62 FQTTestSuite.cpp

```
00001
00005 // ////////////////////////////////////////////////////////////////////
00006 // Import section
00007 // ////////////////////////////////////////////////////////////////////
00008 // STL
00009 #include <sstream>
00010 #include <fstream>
00011 #include <string>
00012 // Boost Unit Test Framework (UTF)
00013 #define BOOST_TEST_DYN_LINK
00014 #define BOOST_TEST_MAIN
00015 #define BOOST_TEST_MODULE FQTTestSuite
00016 #include <boost/test/unit_test.hpp>
00017 // StdAir
00018 #include <stdair/basic/BasLogParams.hpp>
00019 #include <stdair/basic/BasDBParams.hpp>
00020 #include <stdair/basic/BasFileMgr.hpp>
00021 #include <stdair/service/Logger.hpp>
00022 #include <stdair/bom/TravelSolutionStruct.hpp>
00023 #include <stdair/bom/BookingRequestStruct.hpp>
00024 // SimFQT
00025 #include <simfqt/SIMFQT_Service.hpp>
00026 #include <simfqt/config/simfqt-paths.hpp>
00027
00028 namespace boost_utf = boost::unit_test;
00029
00033 struct UnitTestConfig {
00035   UnitTestConfig() {
00036     static std::ofstream _test_log ("FQTTestSuite_utfresults.xml");
00037     boost_utf::unit_test_log.set_stream (_test_log);
00038     boost_utf::unit_test_log.set_format (boost_utf::XML);
00039     boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
00040     //boost_utf::unit_test_log.set_threshold_level
      (boost_utf::log_successful_tests);
00041   }
00042
00044   ~UnitTestConfig() {
00045   }
00046 };
00047
00048 // ////////////////////////////////////////////////////////////////////
00052 void testFareQuoterHelper (const unsigned short iTestFlag,
00053                            const stdair::Filename_T iFareInputFilename,
00054                            const bool isBuiltin) {
00055
00056   // Output log File
00057   std::ostringstream oStr;
00058   oStr << "FQTTestSuite_" << iTestFlag << ".log";
00059   const stdair::Filename_T lLogFilename (oStr.str());
00060
00061   // Set the log parameters
00062   std::ofstream logOutputFile;
00063   // Open and clean the log outputfile
00064   logOutputFile.open (lLogFilename.c_str());
00065   logOutputFile.clear();
00066
00067   // Initialise the SimFQT service object
00068   const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG,
00069                                          logOutputFile);
00070
00071   // Initialise the Simfqt service object
00072   SIMFQT::SIMFQT_Service simfqtService (lLogParams);
00073
00074   // Check wether or not a (CSV) input file should be read
00075   if (isBuiltin == true) {
00076
00077     // Build the default sample BOM tree (filled with fares) for Simfqt
00078     simfqtService.buildSampleBom();
00079
00080   } else {
00081
```

```
00082     // Build the BOM tree from parsing the fare input file
00083     SIMFQT::FareFilePath lFareFilePath (iFareInputFilename)
    ;
00084     simfqtService.parseAndLoad (lFareFilePath);
00085   }
00086
00087   // Build a sample list of travel solutions and a booking request.
00088   stdair::TravelSolutionList_T lTravelSolutionList;
00089   simfqtService.buildSampleTravelSolutions (lTravelSolutionList);
00090   stdair::BookingRequestStruct lBookingRequest =
00091     simfqtService.buildBookingRequest();
00092
00093   // Try to fareQuote the sample list of travel solutions
00094   simfqtService.quotePrices (lBookingRequest, lTravelSolutionList);
00095
00096   // Close the log file
00097   logOutputFile.close();
00098
00099 }
00100
00101 // /////////////// Main: Unit Test Suite //////////////
00102
00103 // Set the UTF configuration (re-direct the output to a specific file)
00104 BOOST_GLOBAL_FIXTURE (UnitTestConfig);
00105
00106 // Start the test suite
00107 BOOST_AUTO_TEST_SUITE (master_test_suite)
00108
00109
00112 BOOST_AUTO_TEST_CASE (simfqt_simple_pricing_test) {
00113
00114   // Input file name
00115   const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR
      "/fare01.csv");
00116
00117   // State whether the BOM tree should be built-in or parsed from an input file
00118   const bool isBuiltin = false;
00119
00120   // Try to fareQuote the sample default list of travel solutions
00121   BOOST_CHECK_NO_THROW (testFareQuoterHelper (0, lFareInputFilename, isBuiltin)
    );
00122
00123 }
00124
00129 BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_01) {
00130
00131   // Input file name
00132   const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR
      "/fareError01.csv");
00133
00134   // State whether the BOM tree should be built-in or parsed from an input file
00135   const bool isBuiltin = false;
00136
00137   // Try to fareQuote the sample default list of travel solutions
00138   BOOST_CHECK_THROW (testFareQuoterHelper (1, lFareInputFilename, isBuiltin),
00139                      SIMFQT::AirportPairNotFoundException
    );
00140 }
00141
00146 BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_02) {
00147
00148   // Input file name
00149   const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR
      "/fareError02.csv");
00150
00151   // State whether the BOM tree should be built-in or parsed from an input file
00152   const bool isBuiltin = false;
00153
00154   // Try to fareQuote the sample default list of travel solutions
00155   BOOST_CHECK_THROW (testFareQuoterHelper (2, lFareInputFilename, isBuiltin),
00156                      SIMFQT::PosOrChannelNotFoundException
    );
00157 }
00158
00163 BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_03) {
00164
00165   // Input file name
00166   const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR
      "/fareError03.csv");
00167
00168   // State whether the BOM tree should be built-in or parsed from an input file
00169   const bool isBuiltin = false;
00170
00171   // Try to fareQuote the sample default list of travel solutions
00172   BOOST_CHECK_THROW (testFareQuoterHelper (3, lFareInputFilename, isBuiltin),
00173                      SIMFQT::FlightDateNotFoundException
    );
```

```
00174 }
00175
00180 BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_04) {
00181
00182   // Input file name
00183   const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR
      "/fareError04.csv");
00184
00185   // State whether the BOM tree should be built-in or parsed from an input file
00186   const bool isBuiltin = false;
00187
00188   // Try to fareQuote the sample default list of travel solutions
00189   BOOST_CHECK_THROW (testFareQuoterHelper (4, lFareInputFilename, isBuiltin),
00190                      SIMFQT::FlightTimeNotFoundException
      );
00191 }
00192
00197 BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_05) {
00198
00199   // Input file name
00200   const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR
      "/fareError05.csv");
00201
00202   // State whether the BOM tree should be built-in or parsed from an input file
00203   const bool isBuiltin = false;
00204
00205   // Try to fareQuote the sample default list of travel solutions
00206   BOOST_CHECK_THROW (testFareQuoterHelper (5, lFareInputFilename, isBuiltin),
00207                      SIMFQT::FeaturesNotFoundException
      );
00208 }
00209
00214 BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_06) {
00215
00216   // Input file name
00217   const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR
      "/fareError06.csv");
00218
00219   // State whether the BOM tree should be built-in or parsed from an input file
00220   const bool isBuiltin = false;
00221
00222   // Try to fareQuote the sample default list of travel solutions
00223   BOOST_CHECK_THROW (testFareQuoterHelper (6, lFareInputFilename, isBuiltin),
00224                      SIMFQT::AirlineNotFoundException
      );
00225 }
00226
00231 BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_07) {
00232
00233   // Input file name
00234   const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR
      "/fareError07.csv");
00235
00236   // State whether the BOM tree should be built-in or parsed from an input file
00237   const bool isBuiltin = false;
00238
00239   // Try to fareQuote the sample default list of travel solutions
00240   BOOST_CHECK_THROW (testFareQuoterHelper (7, lFareInputFilename, isBuiltin),
00241                      SIMFQT::FareFileParsingFailedException
      );
00242 }
00243
00248 BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_08) {
00249
00250   // Input file name
00251   const stdair::Filename_T lFareInputFilename (STDAIR_SAMPLE_DIR
      "/missingFile.csv");
00252
00253   // State whether the BOM tree should be built-in or parsed from an input file
00254   const bool isBuiltin = false;
00255
00256   // Try to fareQuote the sample default list of travel solutions
00257   BOOST_CHECK_THROW (testFareQuoterHelper (8, lFareInputFilename, isBuiltin),
00258                      SIMFQT::FareInputFileNotFoundException
      );
00259 }
00260
00265 BOOST_AUTO_TEST_CASE (simfqt_error_pricing_test_09) {
00266
00267   // Input file name
00268   const stdair::Filename_T lEmptyInputFilename (STDAIR_SAMPLE_DIR
      "/ ");
00269
00270   // State whether the BOM tree should be built-in or parsed from an input file
00271   const bool isBuiltin = true;
00272
00273   // Try to fareQuote the sample default list of travel solutions
```

```
00274   BOOST_CHECK_NO_THROW(testFareQuoterHelper (9, lEmptyInputFilename, isBuiltin)
      );
00275 }
00276
00277
00278 // End the test suite
00279 BOOST_AUTO_TEST_SUITE_END()
00280
00281
```