RMOL

1.00.1

Generated by Doxygen 1.8.9.1

Sun Jun 28 2015 23:49:49

Contents

1 RMOL Documentation

1.1 Getting Started

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

1.2 RMOL on GitHub

- Project page
- Download RMOL
- Issues/tickets (bugs or features)
  - Open an issue/ticket

1.3 RMOL Development

- Git Repository

```
$ git clone git@github.com:airsim/rmol.git rmolgit # If SSH is allowed
$ git clone https://github.com/airsim/rmol.git rmolgit # If the firewall does not allow SSH
$ cd rmolgit
$ git checkout trunk
```

- Coding Rules
- Documentation Rules
- Test Rules

1.4 External Libraries

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

1.5 Support RMOL

1.6 About RMOL

RMOL is a C++ library of revenue management and optimisation classes and functions. RMOL mainly targets simulation purposes. N

RMOL mainly targets simulation purposes. N Indeed, RMOL is an important component of the Travel Market Simulator. However, it may be used in a stand-alone mode.

RMOL 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 RMOL library originates from the Travel Intelligence (TI) Business Unit (BI) at Amadeus. RMOL is released under the terms of the GNU Lesser General Public License (LGPLv2.1) for you to enjoy.

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

Generated on $datetime

Note

(N) - The RMOL 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 RMOL.

2 People

2.1 Project Admins

- Denis Arnaud <denis.arnaud_stdair at m4x.org> (N)
- Anh Quan Nguyen <aquannguyen+stdair at gmail.com> (N)

2.2 Developers

- Anh Quan Nguyen <aquannguyen+stdair at gmail.com> (N)
- Denis Arnaud <denis.arnaud_stdair at m4x.org> (N)
- Nicolas Bondoux nbondoux@users.sourceforge.net (N)

2.3 Retired Developers

- Patrick Grandjean pgrandjean@users.sourceforge.net (N)
- Benoit Lardeux benlardeux@users.sourceforge.net (N)

- Karim Duval duvalkarim@users.sourceforge.net (N)
- Ngoc-Thach Hoang hoangngocthach@users.sourceforge.net (N)
- Son Nguyen Kim snguyenkim@users.sourceforge.net (N)

2.4 Contributors

- Emmanuel Bastien <os at ebastien.name> (N)
- Christophe Lacombe ddtof@users.sourceforge.net (N)

2.5 Distribution Maintainers

- Fedora/RedHat: Denis Arnaud <denis.arnaud_stdair at m4x.org> (N)
- Debian: Emmanuel Bastien <os at ebastien.name> (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

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.
   ```

2. 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.

3. 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.

4. 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.

5. 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.
2. 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 L↩ IBRARY, 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 L↩ IMITED 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 WIT↩ H YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
2. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL AN↩ Y 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 RMOL 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 RMOL 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.
 *
 * ---------------------------------------------------------------------
 *
 * RMOL - C++ Revenue Management Object Library
 *
 * Copyright (C) 2007-2010  (\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 RMOL is given below sorted in different categories. Many more features and functions exist and for these we refer to the reference documentation.

6.1 Optimisation features

- Dynamic Programming (DP)
- EMSRa and EMSRb
- Network optimisation with Linear Programming (LP)

6.2 Unconstraining

- Inventory censorflag and guillotine
- E-M (Expectation Maximisation)

6.3 Forecasting features

- Exponential Smoothing
- Moving Average

6.4 Overbooking features

- Cancellations and No-Shows
- Cost-based optimisation
- Service-based optimisation

6.5 Other features

- CSV input file parsing

7 Make a Difference

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

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

- Start using RMOL
- Tell your friends about RMOL and help them to get started using it
- If you find a bug, report it to us (on the dedicated GitHub's Web site). 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 RMOL discussion forums on SourceForge. If you know the answer to a question, help others to overcome their RMOL 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 us to port RMOL to new platforms. If you manage to compile RMOL on a new platform, then tell us how you did it.
- Send us your code. If you have a good RMOL compatible code, which you can release under the LGPL, and you think it should be included in RMOL, then send it to us.
- Become an RMOL developer. Send us an e-mail and tell what you can do for RMOL.

8 Make a new release

8.1 Introduction

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

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

- Initialisation
- Release branch maintenance
- Commit and publish the release branch
- Create source packages (tar-balls)
- Upload the documentation to GitHub
- Generate the RPM packages
- Update distributed change log
- Create the binary package, including the documentation
- Files on GitHub
- post_news
- send_announce

8.2 Initialisation

Clone locally the full Git project:

```
$ mkdir -p ~/dev/sim
$ cd ~/dev/sim
$ git clone git@github.com:airsim/rmol.git rmolgit # If SSH is allowed
$ git clone https://github.com/airsim/rmol.git rmolgit # If the firewall does not allow SSH
$ cd rmolgit
$ 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/rmolgit
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 rmol.spec
```

8.4 Commit and publish the release branch

Commit the new release:

```
cd ~/dev/sim/rmolgit
git add -A
git commit -m "[Release 1.00.3] Release of the 1.00.3 version of RMOL."
git push
```

8.5 Create source packages (tar-balls)

Create the distribution packages using the following command:

```
cd ~/dev/sim/rmolgit
git checkout releases
rm -rf build && mkdir -p build
cd build
export INSTALL_BASEDIR=~/dev/deliveries
export RMOL_VER=99.99.99
export LIBSUFFIX_4_CMAKE="-DLIB_SUFFIX=64"
cmake -DCMAKE_INSTALL_PREFIX=${INSTALL_BASEDIR}/rmol-$RMOL_VER \
 -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, `rmol-$R↩MOL_VER.tar.gz` and `rmol-$RMOL_VER.tar.bz2`.

8.6 Generate the RPM packages

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

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

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

```
cp ../rmol.spec ~/dev/packages/SPECS \
  && cp rmol-$RMOL_VER.tar.bz2 ~/dev/packages/SOURCES
cd ~/dev/packages/SPECS
rpmbuild -ba rmol.spec
cd ~/dev/packages
rpmlint -i SPECS/rmol.spec SRPMS/rmol-$RMOL_VER-1.f22.src.rpm \
  RPMS/noarch/rmol-* RPMS/i686/rmol-*
```

8.7 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 `RMOL's Git repository`.

8.8 Create the binary package, including the documentation

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

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

The output binary package will be named, for instance, `rmol-$RMOL_VER-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.

8.9 Files on GitHub

GitHub allows to archive/generate `packages (tar-balls)` corresponding to Git tags.

8.10 Upload the documentation to GitHub

In order to update the Web site files:

```
$ export RMOL_VER=99.99.99
$ cd ~/dev/sim/rmolgit
$ git checkout $RMOL_VER
$ cd build
$ export INSTALL_BASEDIR=~/dev/deliveries
$ if [ -d /usr/lib64 ]; then LIBSUFFIX=64; fi
$ export LIBSUFFIX_4_CMAKE="-DLIB_SUFFIX=$LIBSUFFIX"
$ rm -rf build && mkdir build && cd build
$ cmake -DCMAKE_INSTALL_PREFIX=${INSTALL_BASEDIR}/rmol-$RMOL_VER \
        -DCMAKE_BUILD_TYPE:STRING=Debug -DENABLE_TEST:BOOL=ON \
        -DINSTALL_DOC:BOOL=ON -DRUN_GCOV:BOOL=OFF ${LIBSUFFIX_4_CMAKE} ..
$ make check && make install
$ cd ..
$ git checkout gh-pages
$ rsync -av --del --exclude=.git $INSTALL_BASEDIR/share/doc/rmol/html/ ./
$ git checkout -- .gitignore README.md CNAME
$ git add .
$ git commit -m "[Doc] Updated the documentation for $RMOL_VER"
$ git push
$ git branch -d gh-pages
```

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 rmol-devel rmol-doc
```

9.3 RMOL Requirements

RMOL 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 RMOL.

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 files `CMakeFiles` containing compiler output (useful mainly for debugging `cmake`).

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

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

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

The simplest way to compile this package is:

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

9.5 Compilers and Options

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

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

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

See also

Defining Variables for more details.

9.6 Compiling For Multiple Architectures

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

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

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

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

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

9.7 Installation Names

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

```
cmake -DCMAKE_INSTALL_PREFIX=/home/user/dev/deliveries/rmol-99.99.99 -DLIB_SUFFIX=64 -DCMAKE_BUILD_TYPE:STRING
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/lib64/ccache/gcc
-- Check for working C compiler: /usr/lib64/ccache/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/lib64/ccache/c++
-- Check for working CXX compiler: /usr/lib64/ccache/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Requires Git without specifying any version
-- Current Git revision name: 56c6c98cf2cfb4008a0acd35d08075cf5f79e693 trunk
-- Requires Boost-1.41
```

```
-- Boost version: 1.46.0
-- Found the following Boost libraries:
--   program_options
--   date_time
--   iostreams
--   serialization
--   filesystem
--   unit_test_framework
--   python
-- Found Boost version: 1.46.0
-- Found BoostWrapper: /usr/include (Required is at least version "1.41")
-- Requires MySQL without specifying any version
-- Using mysql-config: /usr/bin/mysql_config
-- Found MySQL: /usr/lib64/mysql/libmysqlclient.so
-- Found MySQL version: 5.5.14
-- Requires SOCI-3.0
-- Using soci-config: /usr/bin/soci-config
-- SOCI headers are buried
-- Found SOCI: /usr/lib64/libsoci_core.so (Required is at least version "3.0")
-- Found SOCIMySQL: /usr/lib64/libsoci_mysql.so (Required is at least version "3.0")
-- Found SOCI with MySQL back-end support version: 3.0.0
-- Requires StdAir-0.35
-- Found StdAir version: 0.37.1
-- Requires Doxygen without specifying any version
-- Found Doxygen: /usr/bin/doxygen
-- Found DoxygenWrapper: /usr/bin/doxygen
-- Found Doxygen version: 1.7.4
-- Had to set the linker language for 'airraclib' to CXX
-- Had to set the linker language for 'rmollib' to CXX
-- Test 'UnconstrainerTest' to be built with 'UnconstrainerTestSuite.cpp'
-- Test 'ForecasterTest' to be built with 'ForecasterTestSuite.cpp'
-- Test 'OptimiseTest' to be built with 'OptimiseTestSuite.cpp'
-- Test 'BOMsForForecasterTest' to be built with 'bomsforforecaster.cpp'
--
-- ============================================================
-- ----------------------------------
-- ---     Project Information    ---
-- ----------------------------------
-- PROJECT_NAME ................... : rmol
-- PACKAGE_PRETTY_NAME ............ : RMOL
-- PACKAGE ........................ : rmol
-- PACKAGE_NAME ................... : RMOL
-- PACKAGE_BRIEF .................. : C++ library of Revenue Management and Optimisation classes and functions
-- PACKAGE_VERSION ................ : 99.99.99
-- GENERIC_LIB_VERSION ............ : 99.99.99
-- GENERIC_LIB_SOVERSION .......... : 99.99
--
-- ----------------------------------
-- ---     Build Configuration    ---
-- ----------------------------------
-- Modules to build ............... : airrac;rmol
-- Libraries to build/install ..... : airraclib;rmollib
-- Binaries to build/install ...... : airrac;rmol
-- Modules to test ................ : rmol
-- Binaries to test ............... : UnconstrainerTesttst;UnconstrainerTesttst;ForecasterTesttst;Unconstraine
--
-- * Module ....................... : airrac
--   + Layers to build ............ : .;basic;bom;factory;command;service
--   + Dependencies on other layers :
--   + Libraries to build/install . : airraclib
--   + Executables to build/install : airrac
--   + Tests to perform ........... :
-- * Module ....................... : rmol
--   + Layers to build ............ : .;basic;bom;factory;command;service
--   + Dependencies on other layers : airraclib
--   + Libraries to build/install . : rmollib
--   + Executables to build/install : rmol
--   + Tests to perform ........... : UnconstrainerTesttst;UnconstrainerTesttst;ForecasterTesttst;Unconstraine
--
-- BUILD_SHARED_LIBS .............. : ON
-- CMAKE_BUILD_TYPE ............... : Debug
--  * CMAKE_C_FLAGS ............... :
--  * CMAKE_CXX_FLAGS ............. : -Wall -Werror
```

```
--  * BUILD_FLAGS ................. :
--  * COMPILE_FLAGS ............... :
-- CMAKE_MODULE_PATH .............. : /home/user/dev/sim/rmol/rmolgithub/config/
-- CMAKE_INSTALL_PREFIX ........... : /home/user/dev/deliveries/rmol-99.99.99
--
-- * Doxygen:
--   - DOXYGEN_VERSION ............. : 1.7.4
--   - DOXYGEN_EXECUTABLE .......... : /usr/bin/doxygen
--   - DOXYGEN_DOT_EXECUTABLE ...... : /usr/bin/dot
--   - DOXYGEN_DOT_PATH ............ : /usr/bin
--
-- ----------------------------------
-- --- Installation Configuration ---
-- ----------------------------------
-- INSTALL_LIB_DIR ................ : /home/user/dev/deliveries/rmol-99.99.99/lib64
-- INSTALL_BIN_DIR ................ : /home/user/dev/deliveries/rmol-99.99.99/bin
-- INSTALL_INCLUDE_DIR ............ : /home/user/dev/deliveries/rmol-99.99.99/include
-- INSTALL_DATA_DIR ............... : /home/user/dev/deliveries/rmol-99.99.99/share
-- INSTALL_SAMPLE_DIR ............. : /home/user/dev/deliveries/rmol-99.99.99/share/rmol/samples
-- INSTALL_DOC .................... : ON
--
-- ----------------------------------
-- ---   Packaging Configuration  ---
-- ----------------------------------
-- CPACK_PACKAGE_CONTACT .......... : Denis Arnaud <denis.arnaud_rmol - at - m4x dot org>
-- CPACK_PACKAGE_VENDOR ........... : Denis Arnaud
-- CPACK_PACKAGE_VERSION .......... : 99.99.99
-- CPACK_PACKAGE_DESCRIPTION_FILE . : /home/user/dev/sim/rmol/rmolgithub/README
-- CPACK_RESOURCE_FILE_LICENSE .... : /home/user/dev/sim/rmol/rmolgithub/COPYING
-- CPACK_GENERATOR ................ : TBZ2
-- CPACK_DEBIAN_PACKAGE_DEPENDS ... :
-- CPACK_SOURCE_GENERATOR ......... : TBZ2;TGZ
-- CPACK_SOURCE_PACKAGE_FILE_NAME . : rmol-99.99.99
--
-- ----------------------------------
-- ---     External libraries    ---
-- ----------------------------------
--
-- * Boost:
--   - Boost_VERSION ............... : 104600
--   - Boost_LIB_VERSION ........... : 1_46
--   - Boost_HUMAN_VERSION ......... : 1.46.0
--   - Boost_INCLUDE_DIRS .......... : /usr/include
--   - Boost required components .. : program_options;date_time;iostreams;serialization;filesystem;unit_test_f
--   - Boost required libraries ... : optimized;/usr/lib64/libboost_iostreams-mt.so;debug;/usr/lib64/libboost_
--
-- * MySQL:
--   - MYSQL_VERSION ............... : 5.5.14
--   - MYSQL_INCLUDE_DIR ........... : /usr/include/mysql
--   - MYSQL_LIBRARIES ............. : /usr/lib64/mysql/libmysqlclient.so
--
-- * SOCI:
--   - SOCI_VERSION ................ : 3.0.0
--   - SOCI_INCLUDE_DIR ............ : /usr/include/soci
--   - SOCIMYSQL_INCLUDE_DIR ....... : /usr/include/soci
--   - SOCI_LIBRARIES .............. : /usr/lib64/libsoci_core.so
--   - SOCIMYSQL_LIBRARIES ......... : /usr/lib64/libsoci_mysql.so
--
-- * StdAir:
--   - STDAIR_VERSION .............. : 0.37.1
--   - STDAIR_BINARY_DIRS .......... : /home/user/dev/deliveries/stdair-0.37.1/bin
--   - STDAIR_EXECUTABLES .......... : stdair
--   - STDAIR_LIBRARY_DIRS ......... : /home/user/dev/deliveries/stdair-0.37.1/lib64
--   - STDAIR_LIBRARIES ............ : stdairlib;stdairuicllib
--   - STDAIR_INCLUDE_DIRS ......... : /home/user/dev/deliveries/stdair-0.37.1/include
--   - STDAIR_SAMPLE_DIR ........... : /home/user/dev/deliveries/stdair-0.37.1/share/stdair/samples
--
-- Change a value with: cmake -D<Variable>=<Value>
-- ================================================================
--
-- Configuring done
-- Generating done
-- Build files have been written to: /home/user/dev/sim/rmol/rmolgithub/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_rmol
[  0%] Built target hdr_cfg_airrac
[ 30%] Built target airraclib
[ 86%] Built target rmollib
[ 90%] Built target BOMsForForecasterTesttst
[ 93%] Built target UnconstrainerTesttst
[ 96%] Built target ForecasterTesttst
[100%] Built target OptimiseTesttst
Scanning dependencies of target check_rmoltst
Test project /home/user/dev/sim/rmol/rmolgithub/build/test/rmol
    Start 1: UnconstrainerTesttst
1/4 Test #1: UnconstrainerTesttst .............   Passed    0.04 sec
    Start 2: ForecasterTesttst
2/4 Test #2: ForecasterTesttst .................   Passed    0.04 sec
    Start 3: OptimiseTesttst
3/4 Test #3: OptimiseTesttst ...................   Passed    0.44 sec
    Start 4: BOMsForForecasterTesttst
4/4 Test #4: BOMsForForecasterTesttst .........   Passed    0.02 sec

100% tests passed, 0 tests failed out of 4

Total Test time (real) =   0.78 sec
[100%] Built target check_rmoltst
Scanning dependencies of target check
[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/rmolgit
rm -rf build && mkdir build
cd build
```

to remove everything.

10 Linking with RMOL

10.1 Table of Contents

10.2 Introduction

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

10.3 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 RMOL based program `my_prog.cpp`, you should use the following command:

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

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

10.4 Using the rmol-config script

RMOL provides a shell script called `rmol-config`, which is installed by default in `$prefix/bin` (`/usr/local/bin`) directory. It can be used to simplify compilation and linking of RMOL 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++ `rmol-config --cflags` -o my_prog_opt my_prog.cpp `rmol-config --libs`
```

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

```
rmol-config --help
```

If the `rmol-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
```

10.5 M4 macro for the GNU Autotools

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

- `RMOL_VERSION` (e.g., defined to 0.23.0)
- `RMOL_CFLAGS` (e.g., defined to `-I${prefix}/include`)
- `RMOL_LIBS` (e.g., defined to `-L${prefix}/lib -lrmol`)

10.6 Using RMOL with dynamic linking

When using static linking some of the library routines in RMOL 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 RMOL library file during your program execution. If you install the RMOL 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=<RMOL installation prefix>/lib:$LD_LIBRARY_PATH
```

11 Test Rules

This section describes rules how the functionality of the IT++ 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 IT++ 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 IT++ 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 IT++ Library

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

```
% make check
```

after successful compilation of the IT++ library.

12 Users Guide

12.1 Table of Contents

12.2 Introduction

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

12.3 Get Started

12.3.1 Get the RMOL library

12.3.2 Build the RMOL project

To build RMOL, go to the top directory (where the RMOL package has been un-packed), and issue the following commands only once:

```
$ export INSTALL_BASEDIR=~/dev/deliveries
$ export RMOL_VER=99.99.99
$ if [ -d /usr/lib64 ]; then LIBSUFFIX=64; fi
$ export LIBSUFFIX_4_CMAKE="-DLIB_SUFFIX=$LIBSUFFIX"
$ rm -rf build && mkdir build && cd build
$ cmake -DCMAKE_INSTALL_PREFIX=${INSTALL_BASEDIR}/rmol-$RMOL_VER \
        -DCMAKE_BUILD_TYPE:STRING=Debug -DENABLE_TEST:BOOL=ON \
        -DINSTALL_DOC:BOOL=ON -DRUN_GCOV:BOOL=OFF ${LIBSUFFIX_4_CMAKE} ..
```

Then, everytime you change the source code:

```
$ make check
```

When everything is fine, install RMOL locally:

```
$ make install
```

12.3.3 Build and Run the Tests

12.3.4 Install the RMOL Project (Binaries, Documentation)

12.4 Exploring the Predefined BOM Tree

RMOL predefines a BOM (Business Object Model) tree specific to the airline IT arena.

12.4.1 Forecaster BOM Tree

- RMOL::EMDetruncator
- RMOL::Detruncator
- RMOL::Forecaster

12.4.2 Optimiser BOM Tree

- RMOL::DPOptimiser
- RMOL::MCOptimiser
- RMOL::Optimiser

12.5 Extending the BOM Tree

13 Supported Systems

13.1 Table of Contents

13.2 Introduction

This page is intended to provide a list of RMOL supported systems, i.e. the systems on which configuration, installation and testing process of the RMOL 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 RMOL library on a system not mentioned below, please let us know, so we could update this database.

13.3 RMOL 0.23.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
- **RMOL release**: 0.23.0
- **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**: RMOL 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
- **RMOL release**: 0.23.1
- **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
```

RMOL 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
- **RMOL release**: 0.23.1
- **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
```

RMOL 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
- **RMOL release**: 0.23.0
- **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**: RMOL 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
- **RMOL release**: 0.23.1
- **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
```

  RMOL configured with:

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

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

13.3.1.6 Red Hat Enterprise Linux with RMOL 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)
- **RMOL release**: 0.23.0
- **External Libraries**: BLAS, CBLAS, LAPACK and FFTW libraries from RMOL 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
- **RMOL release**: 0.23.0
- **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**: RMOL 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
- **RMOL release**: 0.23.0
- **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**: RMOL 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)
- **RMOL release**: 0.23.1
- **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. RMOL 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)
- **RMOL release**: 0.23.1
- **External Libraries**: Installed from Cygwin's repository:
  - `fftw-3.0.1-2`
  - `fftw-dev-3.0.1-1`

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

```
% ./configure --enable-atlas --disable-fftw
```

- **Tests Status**: All tests PASSED
- **Comments**: Only static library can be built. RMOL 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)
- **RMOL release**: 0.23.2
- **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. RMOL 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)
- **RMOL release**: 0.23.2
- **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. RMOL 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 RMOL 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)
- **RMOL release**: 0.23.5
- **External Libraries**: BLAS, CBLAS, LAPACK and FFTW libraries from RMOL External 2.2.0 package
- **Tests Status**: All tests PASSED
- **Comments**: Only static library can be built. RMOL 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
- **RMOL release**: 0.23.5
- **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 RMOL based programs compiled and run with success.
- **Comments**: Only static library can be built. RMOL built by opening the `"win32\rmol.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 RMOL External

- **Platform**: SUNW, Sun-Blade-100 (SPARC)
- **Operating System**: SunOS 5.9 Generic_112233-10
- **Compiler(s)**: g++ (GCC) 3.4.5
- **RMOL release**: 0.23.2
- **External Libraries**: BLAS, CBLAS, LAPACK and FFTW libraries from RMOL 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**: RMOL 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 RMOL Supported Systems (Previous Releases)

14.1 RMOL 3.9.1

14.2 RMOL 3.9.0

14.3 RMOL 3.8.1

15 Tutorials

15.1 Table of Contents

15.2 Introduction

This page contains some tutorial examples that will help you getting started using StdAir. Most examples show how to construct some simple business objects, i.e., instances of the so-named Business Object Model (BOM).

15.2.1 Preparing the StdAir Project for Development

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

15.3 Build a Predefined BOM Tree

A few steps:

- Instanciate the BOM Root Object
- Instanciate the (Airline) Inventory Object
- Link the Inventory Object with the BOM Root

15.3.1 Instanciate the BOM Root Object

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. The corresponding `StdAir` type (class) is `stdair::BomRoot`.

In the following sample, that object is named `ioBomRoot`, and is given as input/output parameter of the `stdair::CmdBomManager::buildSampleBom()` method:

15.3.2 Instanciate the (Airline) Inventory Object

An airline inventory object can then be instantiated. Let us give it the "BA" airline code (corresponding to `British Airways`) as the object key. That is, an object (let us name it `lBAKey`) of type (class) `stdair::InventoryKey` has first to be instantiated.

Thanks to that key, an airline inventory object, i.e. of type (class) `stdair::Inventory`, can be instantiated. Let us name that airline inventory object `lBAInv`.

15.3.3 Link the Inventory Object with the BOM Root

Then, both objects have to be linked: the airline inventory object (`stdair::Inventory`) has to be linked with the root of the BOM tree (`stdair::BomRoot`). That operation is as simple as using the `stdair::FacBomManager::addToListAndMap()` method:

15.3.4 Build Another Airline Inventory

Another airline inventory object, corresponding to the Air France (`Air France`) company, is instantiated the same way:

See the corresponding full program (cmd_bom_manager_cpp) for more details.

15.3.5 Dump The BOM Tree Content

From the `BomRoot` (of type `stdair::BomRoot`) object instance, the list of airline inventories (of type `stdair::Inventory`) can then be retrieved...

... and browsed:

See the corresponding full program (bom_display_cpp) for more details.

15.3.6 Result of the Tutorial Program

When the `stdair.cpp` program is run (with the `-b` option), the output should look like:

```
00001 [D]../../batches/stdair.cpp:243: Welcome to stdair
00002 [D]../../../stdair/command/CmdBomManager.cpp:41: StdAir will build the BOM tree from built-in
       specifications.
00003 [D]../../batches/stdair.cpp:286:
00004 ============================================================
00005 BomRoot:  -- ROOT --
00006 ============================================================
00007 ++++++++++++++++++++++++++++++++++++++++++++++++
00008 Inventory: BA
00009 ++++++++++++++++++++++++++++++++++++++++++++++++
00010 ****************************************
00011 FlightDate: BA9, 2011-Jun-10
00012 ****************************************
00013 ****************************************
00014 Leg-Dates:
00015 ----------
00016 Flight, Leg, BoardDate, BoardTime, OffDate, OffTime, Date Offset, Time Offset, Elapsed, Distance,
       Capacity,
00017 BA9 2011-Jun-10, LHR-BKK, 2011-Jun-10, 21:45:00, 2011-Jun-11, 15:40:00, 11:05:00, 1, 06:50:00, 9900,
       0,
00018 BA9 2011-Jun-10, BKK-SYD, 2011-Jun-11, 17:05:00, 2011-Jun-12, 15:40:00, 09:05:00, 1, 13:30:00, 8100,
       0,
00019 ****************************************
00020 ****************************************
00021 LegCabins:
00022 ----------
00023 Flight, Leg, Cabin, OffedCAP, PhyCAP, RgdADJ, AU, UPR, SS, Staff, WL, Group, CommSpace, AvPool, Avl,
       NAV, GAV, ACP, ETB, BidPrice,
00024 BA9 2011-Jun-10, LHR-BKK 2011-Jun-10, Y, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 9, 0, 0, 3.52965e-319, 0, 0,

00025 BA9 2011-Jun-10, BKK-SYD 2011-Jun-11, Y, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 9, 0, 0, 0, 0, 0,
00026 ****************************************
00027 ****************************************
00028 Buckets:
00029 --------
00030 Flight, Leg, Cabin, Yield, AU/SI, SS, AV,
00031 ****************************************
00032 ****************************************
00033 SegmentCabins:
00034 --------------
00035 Flight, Segment, Cabin, FF, Bkgs, MIN, UPR, CommSpace, AvPool, BP,
00036 BA9 2011-Jun-10, LHR-SYD 2011-Jun-10, Y, EcoSaver, 0, 0, 0, 0, 9, 0,
00037 BA9 2011-Jun-10, LHR-BKK 2011-Jun-10, Y, EcoSaver, 0, 0, 0, 0, 9, 0,
00038 BA9 2011-Jun-10, BKK-SYD 2011-Jun-11, Y, EcoSaver, 0, 0, 0, 0, 9, 0,
00039 ****************************************
```

```
00040 ****************************************
00041 Subclasses:
00042 -----------
00043 Flight, Segment, Cabin, FF, Subclass, MIN/AU (Prot), Nego, NS%, OB%, Bkgs, GrpBks (pdg), StfBkgs,
       WLBkgs, ETB, ClassAvl, RevAvl, SegAvl,
00044 BA9 2011-Jun-10, LHR-SYD 2011-Jun-10, Y, EcoSaver, Q, 0 (0), 0, 0, 0, 0, 0 (0), 0, 0, 0, 0, 0, 0,
00045 ++++++++++++++++++++++++++++++++++++++++++++++++
00046 Inventory: AF
00047 ++++++++++++++++++++++++++++++++++++++++++++++++
00048 ****************************************
00049 FlightDate: AF84, 2011-Mar-20
00050 ****************************************
00051 ****************************************
00052 Leg-Dates:
00053 ----------
00054 Flight, Leg, BoardDate, BoardTime, OffDate, OffTime, Date Offset, Time Offset, Elapsed, Distance,
       Capacity,
00055 AF84 2011-Mar-20, CDG-SFO, 2011-Mar-20, 10:40:00, 2011-Mar-20, 12:50:00, 11:10:00, 0, -09:00:00, 9900,
       0,
00056 ****************************************
00057 ****************************************
00058 LegCabins:
00059 ----------
00060 Flight, Leg, Cabin, OffedCAP, PhyCAP, RgdADJ, AU, UPR, SS, Staff, WL, Group, CommSpace, AvPool, Avl,
       NAV, GAV, ACP, ETB, BidPrice,
00061 AF84 2011-Mar-20, CDG-SFO 2011-Mar-20, Y, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 9, 0, 0, 0, 0, 0,
00062 ****************************************
00063 ****************************************
00064 Buckets:
00065 --------
00066 Flight, Leg, Cabin, Yield, AU/SI, SS, AV,
00067 ****************************************
00068 ****************************************
00069 SegmentCabins:
00070 --------------
00071 Flight, Segment, Cabin, FF, Bkgs, MIN, UPR, CommSpace, AvPool, BP,
00072 AF84 2011-Mar-20, CDG-SFO 2011-Mar-20, Y, EcoSaver, 0, 0, 0, 0, 9, 0,
00073 ****************************************
00074 ****************************************
00075 Subclasses:
00076 -----------
00077 Flight, Segment, Cabin, FF, Subclass, MIN/AU (Prot), Nego, NS%, OB%, Bkgs, GrpBks (pdg), StfBkgs,
       WLBkgs, ETB, ClassAvl, RevAvl, SegAvl,
00078 AF84 2011-Mar-20, CDG-SFO 2011-Mar-20, Y, EcoSaver, Q, 0 (0), 0, 0, 0, 0, 0 (0), 0, 0, 0, 0, 0, 0,
00079
```

See the corresponding full program (batch_stdair_cpp) for more details.

15.4 Extend the Pre-Defined BOM Tree

Now that we master how to instantiate the pre-defined StdAir classes, let us see how to extend that BOM.

15.4.1 Extend an Airline Inventory Object

For instance, let us assume that some (IT) provider (e.g., you) would like to have a specific implementation of the `Inventory` object. The corresponding class has just to extend the `stdair::Inventory` class:

The STL containers have to be defined accordingly too:

See the full class definition (test_archi_inv_hpp) and implementation (test_archi_inv_cpp) for more details.

15.4.2 Build the Specific BOM Objects

The BOM root object (`stdair::BomRoot`) is instantiated the classical way:

Then, the specific implementation of the airline inventory object (`myprovider::Inventory`) can be instantiated the same way as a standard Inventory (`stdair::Inventory`) would be:

Then, the specific implementation of the airline inventory object (`myprovider::Inventory`) is linked to the root of the BOM tree (`stdair::BomRoot`) the same way as the standard Inventory (`stdair::Inventory`) would be:

Another specific airline inventory object is instantiated the same way:

From the `BomRoot` (of type `stdair::BomRoot`) object instance, the list of specific airline inventories (of type `stdair::Inventory`) can then be retrieved...

... and browsed:

15.4.3 Result of the Tutorial Program

When this program is run, the output should look like:

```
00001 Inventory: BA
00002 Inventory: AF
```

See the corresponding full program (StandardAirlineITTestSuite_cpp) for more details.

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

```
  /
// ////////////////////////////////////////////////////////////////////
// Import section
// ////////////////////////////////////////////////////////////////////
// STL
#include <cassert>
#include <limits>
#include <sstream>
#include <fstream>
#include <string>
// Boost Unit Test Framework (UTF)
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MAIN
#define BOOST_TEST_MODULE OptimiseTestSuite
#include <boost/test/unit_test.hpp>
// StdAir
#include <stdair/basic/BasLogParams.hpp>
#include <stdair/basic/BasDBParams.hpp>
#include <stdair/service/Logger.hpp>
// RMOL
#include <rmol/RMOL_Service.hpp>
#include <rmol/config/rmol-paths.hpp>

namespace boost_utf = boost::unit_test;

// (Boost) Unit Test XML Report
std::ofstream utfReportStream ("bomsforforecaster_utfresults.xml");

struct UnitTestConfig {
  UnitTestConfig() {
    boost_utf::unit_test_log.set_stream (utfReportStream);
```

```
    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() {
  }
};

namespace RMOL {

  struct BookingClassData {

    // Attributes
    double _bookingCount;
    double _fare;
    double _sellupFactor;
    bool _censorshipFlag;

    // Constructer
    BookingClassData (const double iBookingCount, const double iFare,
                      const double iSellupFactor, const bool iCensorshipFlag)
      : _bookingCount(iBookingCount), _fare(iFare),
        _sellupFactor(iSellupFactor), _censorshipFlag(iCensorshipFlag) {
    }

    // Getters
    double getFare () const {
      return _fare;
    }

    bool getCensorshipFlag () const {
      return _censorshipFlag;
    }

    // Display
    std::string toString() const {
      std::ostringstream oStr;
      oStr <<  std::endl
           << "[Booking class data information]" << std::endl
           << "Booking counter: " << _bookingCount << std::endl
           << "Fare: " << _fare << std::endl
           << "Sell-up Factor: " << _sellupFactor << std::endl
           << "censorshipFlag: " << _censorshipFlag << std::endl;
      return oStr.str();
    }

  };

  struct BookingClassDataSet {

    typedef std::vector<BookingClassData*> BookingClassDataList_T;

    // Attributes
    int _numberOfClass;
    double _minimumFare;
    bool _censorshipFlag; // true if any of the classes is censored
    BookingClassDataList_T _bookingClassDataList;

    // Constructor
    BookingClassDataSet ()
      : _numberOfClass(0), _minimumFare(0),
        _censorshipFlag(false) {
    }

    // Add BookingClassData
    void addBookingClassData (BookingClassData& ioBookingClassData) {
      _bookingClassDataList.push_back (&ioBookingClassData);
    }

    // Getters
    stdair::NbOfClasses_T getNumberOfClass () const {
      return _bookingClassDataList.size();
    }

    double getMinimumFare () const {
      return _minimumFare;
    }

    bool getCensorshipFlag () const {
      return _censorshipFlag;
    }

    // Setters
    void setMinimumFare (const double iMinFare) {
      _minimumFare = iMinFare;
    }
```

```
    void setCensorshipFlag (const bool iCensorshipFlag) {
      _censorshipFlag = iCensorshipFlag;
    }

    // compute minimum fare
    void updateMinimumFare() {
      double minFare = std::numeric_limits<double>::max();
      BookingClassDataList_T::iterator itBookingClassDataList;
      for (itBookingClassDataList = _bookingClassDataList.begin();
           itBookingClassDataList != _bookingClassDataList.end();
           ++itBookingClassDataList) {
        BookingClassData* lBookingClassData = *itBookingClassDataList;
        assert (lBookingClassData != NULL);

        const double lFare = lBookingClassData->getFare();
        if (lFare < minFare) {
          minFare = lFare;
        }
      }
      //
      setMinimumFare(minFare);
    }

    // compute censorship flag for the data set
    void updateCensorshipFlag () {
      bool censorshipFlag = false;
      BookingClassDataList_T::iterator itBookingClassDataList;
      for (itBookingClassDataList = _bookingClassDataList.begin();
           itBookingClassDataList != _bookingClassDataList.end();
           ++itBookingClassDataList) {
        BookingClassData* lBookingClassData = *itBookingClassDataList;
        assert (lBookingClassData != NULL);

        const bool lCensorshipFlagOfAClass =
          lBookingClassData->getCensorshipFlag();
        if (lCensorshipFlagOfAClass) {
          censorshipFlag = true;
          break;
        }
      }
      //
      setCensorshipFlag(censorshipFlag);
    }

    // Display
    std::string toString() const {
      std::ostringstream oStr;
      oStr << std::endl
           << "[Booking class data set information]" << std::endl
           << "Number of classes: " << _numberOfClass << std::endl
           << "Minimum fare: " << _minimumFare << std::endl
           << "The data of the class set are sensored: " << _censorshipFlag
           << std::endl;
      return oStr.str();
    }

  };

  // /**--------------- BOM : Q-Forecaster ----------------------- */
  // struct QForecaster {

  //   // Function focused BOM

  //   // 1. calculate sell up probability for Q-eq

  //   // 2. calculate Q-Equivalent Booking
  //   double calculateQEqBooking (BookingClassDataSet& iBookingClassDataSet) {
  //     double lQEqBooking = 0.0;
  //     double lMinFare = iBookingClassDataSet.getMinimumFare();


  //     return lQEqBooking;
  //   }

  //   /* Calculate Q-equivalent demand
  //      [<- performed by unconstrainer if necessary (Using ExpMax BOM)]
  //   */


  //   // 3. Partition to each class

  //   //

  // // };

}
```

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

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

BOOST_AUTO_TEST_SUITE (master_test_suite)


BOOST_AUTO_TEST_CASE (rmol_forecaster) {

  // Output log File
  std::string lLogFilename ("bomsforforecaster.log");
  std::ofstream logOutputFile;

  // Open and clean the log outputfile
  logOutputFile.open (lLogFilename.c_str());
  logOutputFile.clear();

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

  // Initialise the RMOL service
  RMOL::RMOL_Service rmolService (lLogParams);

  // Build a sample BOM tree
  rmolService.buildSampleBom();

  // Register BCDataSet
  RMOL::BookingClassDataSet lBookingClassDataSet;

  // Register BookingClassData
  RMOL::BookingClassData QClassData (10, 100, 1, false);
  RMOL::BookingClassData MClassData (5, 150, 0.8, true);
  RMOL::BookingClassData BClassData (0, 200, 0.6, false);
  RMOL::BookingClassData YClassData (0, 300, 0.3, false);

  // Display
  STDAIR_LOG_DEBUG (QClassData.toString());
  STDAIR_LOG_DEBUG (MClassData.toString());
  STDAIR_LOG_DEBUG (BClassData.toString());
  STDAIR_LOG_DEBUG (YClassData.toString());

  // Add BookingClassData into the BCDataSet
  lBookingClassDataSet.addBookingClassData (QClassData);
  lBookingClassDataSet.addBookingClassData (MClassData);
  lBookingClassDataSet.addBookingClassData (BClassData);
  lBookingClassDataSet.addBookingClassData (YClassData);

  // DEBUG
  STDAIR_LOG_DEBUG (lBookingClassDataSet.toString());

  // Number of classes
  const stdair::NbOfClasses_T lNbOfClass = lBookingClassDataSet.getNumberOfClass();

  // DEBUG
  STDAIR_LOG_DEBUG ("Number of Classes: " << lNbOfClass);

  // Minimum fare
  BOOST_CHECK_NO_THROW (lBookingClassDataSet.updateMinimumFare());
  const double lMinFare = lBookingClassDataSet.getMinimumFare();

  // DEBUG
  STDAIR_LOG_DEBUG ("Minimum fare: " << lMinFare);

  // Censorship flag
  BOOST_CHECK_NO_THROW (lBookingClassDataSet.updateCensorshipFlag());
  const bool lCensorshipFlag = lBookingClassDataSet.getCensorshipFlag();

  // DEBUG
  STDAIR_LOG_DEBUG ("Censorship Flag: " << lCensorshipFlag);

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

// End the test suite
BOOST_AUTO_TEST_SUITE_END()
```

17 Command-Line Test to Demonstrate How To Test the RMOL Project

```
  /
// ////////////////////////////////////////////////////////////////////
// Import section
// ////////////////////////////////////////////////////////////////////
// STL
#include <sstream>
#include <fstream>
#include <string>
#include <vector>
#include <cmath>
// Boost Unit Test Framework (UTF)
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MAIN
#define BOOST_TEST_MODULE ForecasterTestSuite
#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>
// RMOL
#include <rmol/RMOL_Service.hpp>

namespace boost_utf = boost::unit_test;

// (Boost) Unit Test XML Report
std::ofstream utfReportStream ("ForecasterTestSuite_utfresults.xml");

struct UnitTestConfig {
  UnitTestConfig() {
    boost_utf::unit_test_log.set_stream (utfReportStream);
    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() {
  }
};


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

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

BOOST_AUTO_TEST_SUITE (master_test_suite)


BOOST_AUTO_TEST_CASE (rmol_forecaster_q_forecasting) {
  const bool lTestFlag = true; //testForecasterHelper(0);
  BOOST_CHECK_EQUAL (lTestFlag, true);
  BOOST_CHECK_MESSAGE (lTestFlag == true,
                       "The test has failed. Please see the log file for "
                       << "more details");
}

// End the test suite
BOOST_AUTO_TEST_SUITE_END()
```

18 Command-Line Test to Demonstrate How To Test the RMOL 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 OptimiseTestSuite
#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>
// RMOL
#include <rmol/basic/BasConst_General.hpp>
#include <rmol/RMOL_Service.hpp>
#include <rmol/config/rmol-paths.hpp>

namespace boost_utf = boost::unit_test;

// (Boost) Unit Test XML Report
std::ofstream utfReportStream ("OptimiseTestSuite_utfresults.xml");

struct UnitTestConfig {
  UnitTestConfig() {
    boost_utf::unit_test_log.set_stream (utfReportStream);
    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() {
  }
};


// ////////////////////////////////////////////////////////////////////
int testOptimiseHelper (const unsigned short optimisationMethodFlag,
                        const bool isBuiltin) {

  // Return value
  int oExpectedBookingLimit = 0;

  // Output log File
  std::ostringstream oStr;
  oStr << "OptimiseTestSuite_" << optimisationMethodFlag << "_" << isBuiltin << ".log";
  const stdair::Filename_T lLogFilename (oStr.str());

  // Number of random draws to be generated (best if greater than 100)
  const int K = RMOL::DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION
      ;

  // Methods of optimisation (0 = Monte-Carlo, 1 = Dynamic Programming,
  // 2 = EMSR, 3 = EMSR-a, 4 = EMSR-b, 5 = EMSR-a with sellup prob.)
  const unsigned short METHOD_FLAG = optimisationMethodFlag;

  // Cabin Capacity (it must be greater then 100 here)
  const double cabinCapacity = 100.0;

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

  // Initialise the RMOL service
  const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);
  RMOL::RMOL_Service rmolService (lLogParams);

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

    // Build the default sample BOM tree and build a dummy BOM tree.
    rmolService.buildSampleBom();

  } else {

    // Parse the optimisation data and build a dummy BOM tree
    const stdair::Filename_T lRMInputFileName (STDAIR_SAMPLE_DIR "/rm02.csv");
    rmolService.parseAndLoad (cabinCapacity, lRMInputFileName);
  }

  switch (METHOD_FLAG) {
  case 0: {
    // DEBUG
    STDAIR_LOG_DEBUG ("Optimisation by Monte-Carlo (MC)");

    // Calculate the optimal protections by the Monte Carlo
    // Integration approach
    rmolService.optimalOptimisationByMCIntegration (K);
    break;
  }

  case 1: {
    // DEBUG
    STDAIR_LOG_DEBUG ("Optimisation by Dynamic Programming (DP)");

    // Calculate the optimal protections by DP.
```

```
    rmolService.optimalOptimisationByDP ();
    break;
  }

  case 2: {
    // DEBUG
    STDAIR_LOG_DEBUG ("Calculate the Bid-Price Vectors (BPV) by EMSR");

    // Calculate the Bid-Price Vector by EMSR
    rmolService.heuristicOptimisationByEmsr ();
    break;
  }

  case 3: {
    // DEBUG
    STDAIR_LOG_DEBUG ("Calculate the Authorisation Levels (AUs) by EMSRa");

    // Calculate the protections by EMSR-a
    // Test the EMSR-a algorithm implementation
    rmolService.heuristicOptimisationByEmsrA ();

    // Return a cumulated booking limit value to test
    // oExpectedBookingLimit = static_cast<int> (lBookingLimitVector.at(2));
    break;
  }

  case 4: {
    // DEBUG
    STDAIR_LOG_DEBUG ("Calculate the Authorisation Levels (AUs) by EMSRb");

    // Calculate the protections by EMSR-b
    rmolService.heuristicOptimisationByEmsrB ();
    break;
  }

  default: rmolService.optimalOptimisationByMCIntegration (K);
  }

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

  return oExpectedBookingLimit;
}


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

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

// ////////////////////////////////////////////////////////////////////
// Tests are based on the following input values
// price; mean; standard deviation;
// 1050; 17.3; 5.8;
// 567; 45.1; 15.0;
// 534; 39.6; 13.2;
// 520; 34.0; 11.3;
// ////////////////////////////////////////////////////////////////////

BOOST_AUTO_TEST_SUITE (master_test_suite)


BOOST_AUTO_TEST_CASE (rmol_optimisation_monte_carlo) {

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

  BOOST_CHECK_NO_THROW (testOptimiseHelper(0, isBuiltin););
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_dynamic_programming) {

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

  BOOST_CHECK_NO_THROW (testOptimiseHelper(1, isBuiltin););
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_bpv) {

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

  BOOST_CHECK_NO_THROW (testOptimiseHelper(2, isBuiltin););
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_a) {
```

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

  BOOST_CHECK_NO_THROW (testOptimiseHelper(3, isBuiltin););
  // const int lBookingLimit = testOptimiseHelper(3);
  // const int lExpectedBookingLimit = 61;
  // BOOST_CHECK_EQUAL (lBookingLimit, lExpectedBookingLimit);
  // BOOST_CHECK_MESSAGE (lBookingLimit == lExpectedBookingLimit,
  //                      "The booking limit is " << lBookingLimit
  //                      << ", but it is expected to be "
  //                      << lExpectedBookingLimit);
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_b) {

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

  BOOST_CHECK_NO_THROW (testOptimiseHelper(4, isBuiltin););
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_monte_carlo_built_in) {

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

  BOOST_CHECK_NO_THROW (testOptimiseHelper(0, isBuiltin););
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_dynamic_programming_built_in) {

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

  BOOST_CHECK_NO_THROW (testOptimiseHelper(1, isBuiltin););
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_bpv_built_in) {

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

  BOOST_CHECK_NO_THROW (testOptimiseHelper(2, isBuiltin););
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_a_built_in) {

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

  BOOST_CHECK_NO_THROW (testOptimiseHelper(3, isBuiltin););
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_b_built_in) {

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

  BOOST_CHECK_NO_THROW (testOptimiseHelper(4, isBuiltin););
}

// End the test suite
BOOST_AUTO_TEST_SUITE_END()
```

19 Command-Line Test to Demonstrate How To Test the RMOL 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 UnconstrainerTestSuite
#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>
// RMOL
#include <rmol/RMOL_Service.hpp>

namespace boost_utf = boost::unit_test;

// (Boost) Unit Test XML Report
std::ofstream utfReportStream ("UnconstrainerTestSuite_utfresults.xml");

struct UnitTestConfig {
  UnitTestConfig() {
    boost_utf::unit_test_log.set_stream (utfReportStream);
    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() {
  }
};


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

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

BOOST_AUTO_TEST_SUITE (master_test_suite)


BOOST_AUTO_TEST_CASE (rmol_unconstraining_em) {
  const bool lTestFlag = true;// testUnconstrainerHelper(0);
  BOOST_CHECK_EQUAL (lTestFlag, true);
  BOOST_CHECK_MESSAGE (lTestFlag == true,
                       "The test has failed. Please see the log file for "
                       << "more details");
}

// End the test suite
BOOST_AUTO_TEST_SUITE_END()
```

20 Namespace Index

20.1 Namespace List

Here is a list of all namespaces with brief descriptions:

21 Hierarchical Index

21.1 Class Hierarchy

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

22 Class Index

22.1 Class List

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

23 File Index

23.1 File List

Here is a list of all files with brief descriptions:

24 Namespace Documentation

24.1 RMOL Namespace Reference

Classes

- class BasedForecasting
- class ConvexHullException
  Convex Hull-related exception.
- class DemandGeneratorList
- class DemandInputPreparation
- class Detruncator
- class DPOptimiser
- class EMDetruncator
- class EmptyBookingClassListException
  Empty Booking Class List of Fare Family exception.
- class EmptyConvexHullException
  Empty convex hull exception.
- class EmptyNestingStructException
  Empty nesting structure in unconstrainer exception.
- class Emsr
- class EmsrUtils
- class FacRmolServiceContext
  Factory for the service context.
- class FareAdjustment
- class FareFamilyDemandVectorSizeException
  Fare Family demand exception.
- class FareFamilyException
  Fare Family-related exception.
- class FirstPolicyNotNullException

Missing policy NULL in convex hull exception.

- class Forecaster
- struct HistoricalBooking

  Structure keeping track, for a given class, of the number of historical bookings and of the censorship flag.
- struct HistoricalBookingHolder
- class HybridForecasting
- class InventoryParser

  Class filling the virtual class list (representing a list of classes/buckets) from a given input inventory.
- class MarginalRevenueTransformation
- class MCOptimiser
- class MissingBookingClassInFareFamilyException

  Missing Booking Class in Fare Family exception.
- class MissingDCPException

  Missing a DCP in unconstrainer exception.
- class NewQFF
- class OldQFF
- class OptimisationException

  Optimisation-related exception.
- class Optimiser
- class OverbookingException

  Overbooking-related exception.
- class PolicyException

  Policy-related exception.
- class PolicyHelper
- class PreOptimiser
- class QForecasting
- class RMOL_Service

  Interface for the RMOL Services.
- class RMOL_ServiceContext

  Inner class holding the context for the RMOL Service object.
- class SegmentSnapshotTableHelper
- class UnconstrainingException

  Unconstraining-related exception.
- class Utilities
- class YieldConvexHullException

  Yield convex hull exception.

Typedefs

- typedef std::list< BucketHolder * > BucketHolderList_T
- typedef std::list< FldDistributionParameters > DistributionParameterList_T
- typedef std::vector< HistoricalBooking > HistoricalBookingVector_T
- typedef boost::shared_ptr< RMOL_Service > RMOL_ServicePtr_T
- typedef std::vector< stdair::Flag_T > FlagVector_T
- typedef std::map< stdair::BookingClass *, stdair::MeanStdDevPair_T > BookingClassMeanStdDevPair↩ Map_T

Variables

- const stdair::AirlineCode_T DEFAULT_RMOL_SERVICE_AIRLINE_CODE = "BA"
- const double DEFAULT_RMOL_SERVICE_CAPACITY = 1.0
- const int DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION = 10000
- const int DEFAULT_PRECISION = 10
- const double DEFAULT_EPSILON = 0.0001
- const double DEFAULT_STOPPING_CRITERION = 0.01
- const double DEFAULT_INITIALIZER_DOUBLE_NEGATIVE = -10.0

24.1.1 Typedef Documentation

24.1.1.1 typedef std::list<BucketHolder*> RMOL::BucketHolderList_T

Define a vector (ordered list) of N bucket/classe holders.

Definition at line 16 of file BucketHolderTypes.hpp.

24.1.1.2 typedef std::list<FldDistributionParameters> RMOL::DistributionParameterList_T

Define the set of parameters, each of one wrapping a pair of distribution parameters (i.e., mean and standard deviation).

Definition at line 16 of file DistributionParameterList.hpp.

24.1.1.3 typedef std::vector<HistoricalBooking> RMOL::HistoricalBookingVector_T

Define a vector (ordered list) of N HistoricalBookings.

Definition at line 16 of file HistoricalBookingHolder.hpp.

24.1.1.4 typedef boost::shared_ptr<RMOL_Service> RMOL::RMOL_ServicePtr_T

Pointer on the RMOL Service handler.

Definition at line 176 of file RMOL_Types.hpp.

24.1.1.5 typedef std::vector<stdair::Flag_T> RMOL::FlagVector_T

Define the vector of censorship flags.

Definition at line 179 of file RMOL_Types.hpp.

24.1.1.6 typedef std::map<stdair::BookingClass*, stdair::MeanStdDevPair_T> RMOL::BookingClassMeanStdDevPairMap_T

Define the map between booking class and demand.

Definition at line 182 of file RMOL_Types.hpp.

24.1.2 Variable Documentation

24.1.2.1 const stdair::AirlineCode_T RMOL::DEFAULT_RMOL_SERVICE_AIRLINE_CODE = "BA"

Default airline code for the RMOL_Service.

Definition at line 10 of file BasConst.cpp.

24.1.2.2 const double RMOL::DEFAULT_RMOL_SERVICE_CAPACITY = 1.0

Default capacity for the RMOL_Service.

Definition at line 13 of file BasConst.cpp.

24.1.2.3 const int RMOL::DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION = 10000

Default value for the number of draws within the Monte-Carlo Integration algorithm.

Definition at line 17 of file BasConst.cpp.

Referenced by RMOL::MCOptimiser::optimisationByMCIntegration().

24.1.2.4 const int RMOL::DEFAULT_PRECISION = 10

Default value for the precision of the integral computation in the Dynamic Programming algorithm (100 means that the precision will be 0.01).

Default value for the precision of the integral computation in the Dynamic Programming algorithm.

Definition at line 22 of file BasConst.cpp.

24.1.2.5 const double RMOL::DEFAULT_EPSILON = 0.0001

Default epsilon value to qualify a denominator

Definition at line 25 of file BasConst.cpp.

24.1.2.6 const double RMOL::DEFAULT_STOPPING_CRITERION = 0.01

Default stopping value for an iterative algorithm.

Definition at line 28 of file BasConst.cpp.

24.1.2.7 const double RMOL::DEFAULT_INITIALIZER_DOUBLE_NEGATIVE = -10.0

Default negative value used to initialze a double variable.

Definition at line 31 of file BasConst.cpp.

24.2 stdair Namespace Reference

Forward declarations.

24.2.1 Detailed Description

Forward declarations.

Forward declarations.

25 Class Documentation

25.1 RMOL::BasedForecasting Class Reference

`#include <rmol/command/BasedForecasting.hpp>`

Static Public Member Functions

- static bool forecast (stdair::SegmentCabin &, const stdair::Date_T &, const stdair::DTD_T &, const stdair::↩ UnconstrainingMethod &, const stdair::NbOfSegments_T &)
- static void prepareHistoricalBooking (const stdair::SegmentCabin &, const stdair::BookingClass &, const stdair::SegmentSnapshotTable &, HistoricalBookingHolder &, const stdair::DCP_T &, const stdair::DCP_T &, const stdair::NbOfSegments_T &, const stdair::NbOfSegments_T &)

25.1.1 Detailed Description

Class wrapping the forecasting algorithms.

Definition at line 21 of file BasedForecasting.hpp.

25.1.2 Member Function Documentation

25.1.2.1 bool RMOL::BasedForecasting::forecast (stdair::SegmentCabin & *ioSegmentCabin,* const stdair::Date_T & *iCurrentDate,* const stdair::DTD_T & *iCurrentDTD,* const stdair::UnconstrainingMethod & *iUnconstrainingMethod,* const stdair::NbOfSegments_T & *iNbOfDepartedSegments*) [static]

Forecast demand for a segment cabin. Compute for the current DTD the mean and the standard deviation of unconstrained bookings of similar flights.

Parameters

stdair::SegmentCabin&	Current Segment Cabin
const	stdair::Date_T& Current Date
const	stdair::DTD_T& Current DTD
const	stdair::UnconstrainingMethod& Method used for the unconstraining
const	stdair::NbOfSegments_T& Number of usable historical segments

Definition at line 31 of file BasedForecasting.cpp.

References RMOL::Utilities::computeDistributionParameters(), RMOL::HistoricalBookingHolder::getNbOfFlights(), RMOL::SegmentSnapshotTableHelper::getNbOfSegmentAlreadyPassedThisDTD(), RMOL::HistoricalBookingHolder::getUnconstrainedDemand(), prepareHistoricalBooking(), and RMOL::Detruncator::unconstrain().

25.1.2.2 void RMOL::BasedForecasting::prepareHistoricalBooking (const stdair::SegmentCabin & *iSegmentCabin,* const stdair::BookingClass & *iBookingClass,* const stdair::SegmentSnapshotTable & *iSegmentSnapshotTable,* HistoricalBookingHolder & *ioHBHolder,* const stdair::DCP_T & *iDCPBegin,* const stdair::DCP_T & *iDCPEnd,* const stdair::NbOfSegments_T & *iSegmentBegin,* const stdair::NbOfSegments_T & *iSegmentEnd*) [static]

Prepare the historical booking figures for a given cabin

Parameters

const	stdair::DCP_T& DCP range start
const	stdair::DCP_T& DCP range end
const	stdair::NbOfSegments_T& Segment range start index
const	stdair::NbOfSegments_T& Segment range end index

Definition at line 121 of file BasedForecasting.cpp.

References RMOL::HistoricalBookingHolder::addHistoricalBooking().

Referenced by forecast().

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

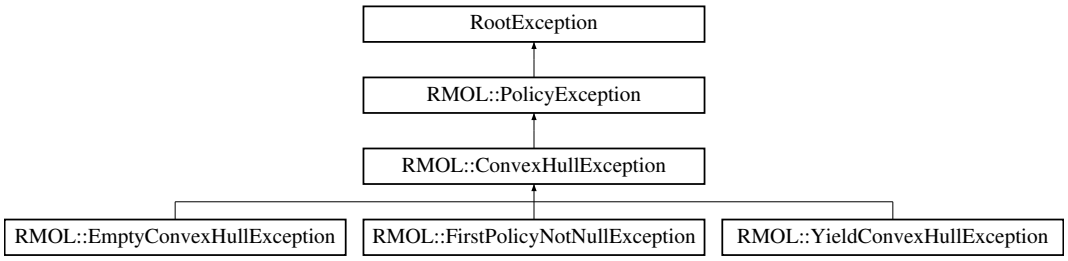
- rmol/command/BasedForecasting.hpp
- rmol/command/BasedForecasting.cpp

25.2 RMOL::ConvexHullException Class Reference

Convex Hull-related exception.

`#include <rmol/RMOL_Types.hpp>`

Inheritance diagram for RMOL::ConvexHullException:



Public Member Functions

- ConvexHullException (const std::string &iWhat)

25.2.1 Detailed Description

Convex Hull-related exception.

Definition at line 93 of file RMOL_Types.hpp.

25.2.2 Constructor & Destructor Documentation

25.2.2.1 RMOL::ConvexHullException::ConvexHullException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 96 of file RMOL_Types.hpp.

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

- rmol/RMOL_Types.hpp

25.3 RMOL::DemandGeneratorList Class Reference

`#include <rmol/bom/old/DemandGeneratorList.hpp>`

Public Member Functions

- DemandGeneratorList ()
- DemandGeneratorList (const DemandGeneratorList &)
- DemandGeneratorList (const DistributionParameterList_T &)
- virtual ~DemandGeneratorList ()
- void generateVariateList (VariateList_T &) const

Protected Types

- typedef std::list< Gaussian > DemandGeneratorList_T

25.3.1 Detailed Description

Wrapper around a set of Gaussian Random Generators.

Definition at line 17 of file DemandGeneratorList.hpp.

25.3.2 Member Typedef Documentation

25.3.2.1 typedef std::list<Gaussian> RMOL::DemandGeneratorList::DemandGeneratorList_T [protected]

Define a (ordered) set of Gaussian Random Generators.

Definition at line 20 of file DemandGeneratorList.hpp.

25.3.3 Constructor & Destructor Documentation

25.3.3.1 RMOL::DemandGeneratorList::DemandGeneratorList ( )

Constructors.

Definition at line 10 of file DemandGeneratorList.cpp.

25.3.3.2 RMOL::DemandGeneratorList::DemandGeneratorList (const DemandGeneratorList & *iDemandGeneratorList*)

Definition at line 17 of file DemandGeneratorList.cpp.

25.3.3.3 RMOL::DemandGeneratorList::DemandGeneratorList (const DistributionParameterList_T & *iDistributionParameterList*)

List of distribution parameters (mean, standard deviation).

Definition at line 25 of file DemandGeneratorList.cpp.

25.3.3.4 RMOL::DemandGeneratorList::~DemandGeneratorList ( ) [virtual]

Destructors.

Definition at line 30 of file DemandGeneratorList.cpp.

25.3.4 Member Function Documentation

25.3.4.1 void RMOL::DemandGeneratorList::generateVariateList (VariateList_T & *ioVariateList*) const

Definition at line 50 of file DemandGeneratorList.cpp.

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

- rmol/bom/old/DemandGeneratorList.hpp
- rmol/bom/old/DemandGeneratorList.cpp

25.4 RMOL::DemandInputPreparation Class Reference

`#include <rmol/command/DemandInputPreparation.hpp>`

Static Public Member Functions

- static bool prepareDemandInput (const stdair::SegmentCabin &)

25.4.1 Detailed Description

Class wrapping the pre-optimisation algorithms.

Definition at line 21 of file DemandInputPreparation.hpp.

25.4.2 Member Function Documentation

25.4.2.1 bool RMOL::DemandInputPreparation::prepareDemandInput (const stdair::SegmentCabin & *iSegmentCabin*) [static]

Prepare the demand input for the optimser.

Definition at line 23 of file DemandInputPreparation.cpp.

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

- rmol/command/DemandInputPreparation.hpp
- rmol/command/DemandInputPreparation.cpp

25.5 RMOL::Detruncator Class Reference

`#include <rmol/command/Detruncator.hpp>`

Static Public Member Functions

- static void unconstrain (HistoricalBookingHolder &, const stdair::UnconstrainingMethod &)

25.5.1 Detailed Description

Class wrapping the principal unconstraining algorithms and some accessory algorithms.

Definition at line 20 of file Detruncator.hpp.

25.5.2 Member Function Documentation

25.5.2.1 void RMOL::Detruncator::unconstrain (HistoricalBookingHolder & *ioHBHolder,* const stdair::UnconstrainingMethod & *iMethod*) [static]

Unconstrain booking figures between two DCP's.

Definition at line 17 of file Detruncator.cpp.

References RMOL::EMDetruncator::unconstrain().

Referenced by RMOL::QForecasting::forecast(), RMOL::HybridForecasting::forecast(), RMOL::OldQFF::forecast(), and RMOL::BasedForecasting::forecast().

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

- rmol/command/Detruncator.hpp
- rmol/command/Detruncator.cpp

25.6 RMOL::DPOptimiser Class Reference

`#include <rmol/bom/DPOptimiser.hpp>`

Static Public Member Functions

- static void optimalOptimisationByDP (stdair::LegCabin &)
- static double cdfGaussianQ (const double, const double)

25.6.1 Detailed Description

Utility methods for the Dynamic Programming algorithms.

Definition at line 17 of file DPOptimiser.hpp.

25.6.2 Member Function Documentation

25.6.2.1 void RMOL::DPOptimiser::optimalOptimisationByDP (stdair::LegCabin & *ioLegCabin*) [static]

Dynamic Programming to compute the cumulative protection levels and booking limits (described in the book Revenue Management - Talluri & Van Ryzin, p.41-42).

Definition at line 22 of file DPOptimiser.cpp.

Referenced by RMOL::Optimiser::optimalOptimisationByDP().

25.6.2.2 static double RMOL::DPOptimiser::cdfGaussianQ (const double , const double) [static]

Compute the cdf_Q of a gaussian.

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

- rmol/bom/DPOptimiser.hpp
- rmol/bom/DPOptimiser.cpp

25.7 RMOL::EMDetruncator Class Reference

`#include <rmol/bom/EMDetruncator.hpp>`

Static Public Member Functions

- static void unconstrain (HistoricalBookingHolder &)

25.7.1 Detailed Description

Utility for the Expectation-Maximisation algorithm.

Definition at line 12 of file EMDetruncator.hpp.

25.7.2 Member Function Documentation

25.7.2.1 void RMOL::EMDetruncator::unconstrain (HistoricalBookingHolder & *ioHistoricalBookingHolder*) [static]

Unconstrain the censored booking data using the Expection-Maximisation algorithm.

Definition at line 20 of file EMDetruncator.cpp.

References RMOL::HistoricalBookingHolder::getDemandMean(), RMOL::HistoricalBookingHolder::getListOfToBeUnconstrainedFlags(), RMOL::HistoricalBookingHolder::getNbOfFlights(), RMOL::HistoricalBookingHolder::getNbOfUncensoredBookings(), RMOL::HistoricalBookingHolder::getNbOfUncensoredData(), RMOL::HistoricalBookingHolder::getStandardDeviation(), RMOL::HistoricalBookingHolder::getUncensoredStandardDeviation(), RMOL::HistoricalBookingHolder::getUnconstrainedDemand(), and RMOL::HistoricalBookingHolder::setUnconstrainedDemand().

Referenced by RMOL::Detruncator::unconstrain().

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

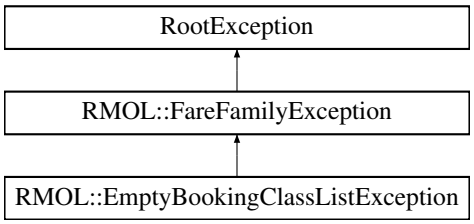
- rmol/bom/EMDetruncator.hpp
- rmol/bom/EMDetruncator.cpp

25.8 RMOL::EmptyBookingClassListException Class Reference

Empty Booking Class List of Fare Family exception.

`#include <rmol/RMOL_Types.hpp>`

Inheritance diagram for RMOL::EmptyBookingClassListException:



Public Member Functions

- EmptyBookingClassListException (const std::string &iWhat)

25.8.1 Detailed Description

Empty Booking Class List of Fare Family exception.

Definition at line 144 of file RMOL_Types.hpp.

25.8.2 Constructor & Destructor Documentation

25.8.2.1 RMOL::EmptyBookingClassListException::EmptyBookingClassListException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 147 of file RMOL_Types.hpp.

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

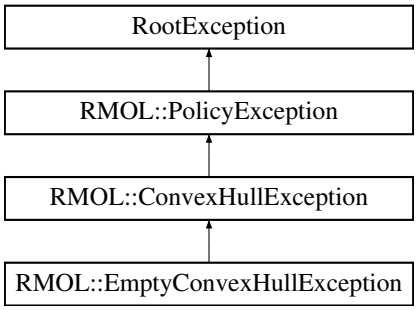
- rmol/RMOL_Types.hpp

25.9 RMOL::EmptyConvexHullException Class Reference

Empty convex hull exception.

`#include <rmol/RMOL_Types.hpp>`

Inheritance diagram for RMOL::EmptyConvexHullException:



Public Member Functions

- EmptyConvexHullException (const std::string &iWhat)

25.9.1 Detailed Description

Empty convex hull exception.

Definition at line 103 of file RMOL_Types.hpp.

25.9.2 Constructor & Destructor Documentation

25.9.2.1 RMOL::EmptyConvexHullException::EmptyConvexHullException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 106 of file RMOL_Types.hpp.

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

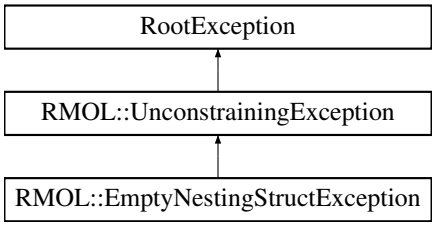
- rmol/RMOL_Types.hpp

25.10 RMOL::EmptyNestingStructException Class Reference

Empty nesting structure in unconstrainer exception.

`#include <rmol/RMOL_Types.hpp>`

Inheritance diagram for RMOL::EmptyNestingStructException:



Public Member Functions

- EmptyNestingStructException (const std::string &iWhat)

25.10.1 Detailed Description

Empty nesting structure in unconstrainer exception.

Definition at line 52 of file RMOL_Types.hpp.

25.10.2 Constructor & Destructor Documentation

25.10.2.1 RMOL::EmptyNestingStructException::EmptyNestingStructException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 55 of file RMOL_Types.hpp.

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

- rmol/RMOL_Types.hpp

25.11 RMOL::Emsr Class Reference

`#include <rmol/bom/Emsr.hpp>`

Static Public Member Functions

- static void heuristicOptimisationByEmsr (stdair::LegCabin &)
- static void heuristicOptimisationByEmsrA (stdair::LegCabin &)
- static void heuristicOptimisationByEmsrB (stdair::LegCabin &)

25.11.1 Detailed Description

Class Implementing the EMSR algorithm for Bid-Price Vector computing.

Definition at line 18 of file Emsr.hpp.

25.11.2 Member Function Documentation

25.11.2.1 void RMOL::Emsr::heuristicOptimisationByEmsr (stdair::LegCabin & *ioLegCabin*) [static]

Compute the Bid-Price Vector using the EMSR algorithm. Then compute the protection levels and booking limits by using the BPV.

For each class/bucket j with yield pj and demand Dj, compute pj∗Pr(Dj>=x) with x the capacity index. This value is called the EMSR (Expected Marginal Seat Revenue) of the class/bucket j with the remaining capacity of x. Thus, we have for each class/bucket a list of EMSR values. We merge all these lists and sort the values from high to low in order to obtain the BPV.

Definition at line 108 of file Emsr.cpp.

References RMOL::EmsrUtils::computeEmsrValue().

Referenced by RMOL::Optimiser::heuristicOptimisationByEmsr().

25.11.2.2 void RMOL::Emsr::heuristicOptimisationByEmsrA (stdair::LegCabin & *ioLegCabin*) [static]

Calculate the optimal protections for the set of buckets/classes given in input, and update those buckets accordingly.

Definition at line 21 of file Emsr.cpp.

References RMOL::EmsrUtils::computeProtectionLevel().

Referenced by RMOL::Optimiser::heuristicOptimisationByEmsrA().

25.11.2.3 void RMOL::Emsr::heuristicOptimisationByEmsrB (stdair::LegCabin & *ioLegCabin*) [static]

Complute the protection levels and booking limites by using the EMSR-b algorithm.

Definition at line 64 of file Emsr.cpp.

References RMOL::EmsrUtils::computeAggregatedVirtualClass(), and RMOL::EmsrUtils::computeProtectionLevel().

Referenced by RMOL::Optimiser::heuristicOptimisationByEmsrB().

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

- rmol/bom/Emsr.hpp
- rmol/bom/Emsr.cpp

25.12 RMOL::EmsrUtils Class Reference

`#include <rmol/bom/EmsrUtils.hpp>`

Static Public Member Functions

- static void computeAggregatedVirtualClass (stdair::VirtualClassStruct &, stdair::VirtualClassStruct &)
- static const stdair::ProtectionLevel_T computeProtectionLevel (stdair::VirtualClassStruct &, stdair::VirtualClassStruct &)
- static const double computeEmsrValue (double, stdair::VirtualClassStruct &)

25.12.1 Detailed Description

Forward declarations.

Definition at line 19 of file EmsrUtils.hpp.

25.12.2 Member Function Documentation

25.12.2.1 void RMOL::EmsrUtils::computeAggregatedVirtualClass (stdair::VirtualClassStruct & *ioAggregatedVirtualClass*, stdair::VirtualClassStruct & *ioCurrentVirtualClass*) [static]

Compute the aggregated class/bucket of classes/buckets 1,..,j for EMSR-b algorithm.

Definition at line 19 of file EmsrUtils.cpp.

Referenced by RMOL::Emsr::heuristicOptimisationByEmsrB().

25.12.2.2 const stdair::ProtectionLevel_T RMOL::EmsrUtils::computeProtectionLevel (stdair::VirtualClassStruct & *ioAggregatedVirtualClass*, stdair::VirtualClassStruct & *ioNextVirtualClass*) [static]

Compute the protection level using the Little-Wood formular.

Definition at line 53 of file EmsrUtils.cpp.

Referenced by RMOL::Emsr::heuristicOptimisationByEmsrA(), and RMOL::Emsr::heuristicOptimisationByEmsrB().

25.12.2.3 const double RMOL::EmsrUtils::computeEmsrValue (double *iCapacity*, stdair::VirtualClassStruct & *ioVirtualClass*) [static]

Compute the EMSR value of a class/bucket.

Definition at line 80 of file EmsrUtils.cpp.

Referenced by RMOL::Emsr::heuristicOptimisationByEmsr().

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

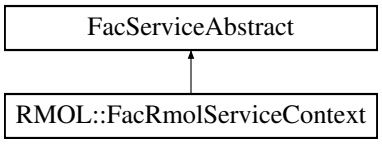
- rmol/bom/EmsrUtils.hpp
- rmol/bom/EmsrUtils.cpp

25.13 RMOL::FacRmolServiceContext Class Reference

Factory for the service context.

`#include <rmol/factory/FacRmolServiceContext.hpp>`

Inheritance diagram for RMOL::FacRmolServiceContext:



Public Member Functions

- ~FacRmolServiceContext ()
- RMOL_ServiceContext & create ()

Static Public Member Functions

- static FacRmolServiceContext & instance ()

Protected Member Functions

- FacRmolServiceContext ()

25.13.1 Detailed Description

Factory for the service context.

Definition at line 22 of file FacRmolServiceContext.hpp.

25.13.2 Constructor & Destructor Documentation

25.13.2.1 RMOL::FacRmolServiceContext::~FacRmolServiceContext ( )

Destructor.

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

Definition at line 17 of file FacRmolServiceContext.cpp.

25.13.2.2 RMOL::FacRmolServiceContext::FacRmolServiceContext ( ) [inline], [protected]

Default Constructor.

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

Definition at line 57 of file FacRmolServiceContext.hpp.

Referenced by instance().

25.13.3 Member Function Documentation

25.13.3.1 FacRmolServiceContext & RMOL::FacRmolServiceContext::instance ( ) [static]

Provide the unique instance.

The singleton is instantiated when first used.

Returns

FacServiceContext&

Definition at line 22 of file FacRmolServiceContext.cpp.

References FacRmolServiceContext().

25.13.3.2 RMOL_ServiceContext & RMOL::FacRmolServiceContext::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 FacRmolServiceContext.cpp.

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

- rmol/factory/FacRmolServiceContext.hpp
- rmol/factory/FacRmolServiceContext.cpp

25.14 RMOL::FareAdjustment Class Reference

`#include <rmol/command/FareAdjustment.hpp>`

Static Public Member Functions

- static bool adjustYield (const stdair::SegmentCabin &)

25.14.1 Detailed Description

Class wrapping the pre-optimisation algorithms.

Definition at line 21 of file FareAdjustment.hpp.

25.14.2 Member Function Documentation

25.14.2.1 bool RMOL::FareAdjustment::adjustYield (const stdair::SegmentCabin & *iSegmentCabin*) [static]

Prepare the demand input for the optimser.

Definition at line 23 of file FareAdjustment.cpp.

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

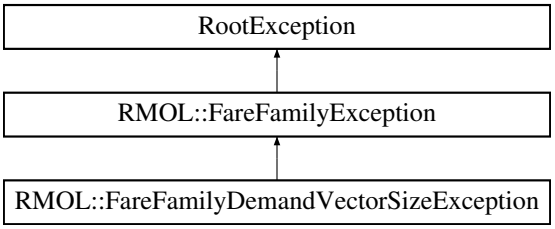
- rmol/command/FareAdjustment.hpp
- rmol/command/FareAdjustment.cpp

25.15 RMOL::FareFamilyDemandVectorSizeException Class Reference

Fare Family demand exception.

`#include <rmol/RMOL_Types.hpp>`

Inheritance diagram for RMOL::FareFamilyDemandVectorSizeException:



Public Member Functions

- FareFamilyDemandVectorSizeException (const std::string &iWhat)

25.15.1 Detailed Description

Fare Family demand exception.

Definition at line 164 of file RMOL_Types.hpp.

25.15.2 Constructor & Destructor Documentation

25.15.2.1 RMOL::FareFamilyDemandVectorSizeException::FareFamilyDemandVectorSizeException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 167 of file RMOL_Types.hpp.

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

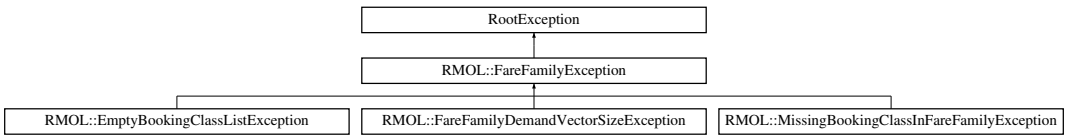
- rmol/RMOL_Types.hpp

25.16 RMOL::FareFamilyException Class Reference

Fare Family-related exception.

`#include <rmol/RMOL_Types.hpp>`

Inheritance diagram for RMOL::FareFamilyException:



Public Member Functions

- FareFamilyException (const std::string &iWhat)

25.16.1 Detailed Description

Fare Family-related exception.

Definition at line 134 of file RMOL_Types.hpp.

25.16.2 Constructor & Destructor Documentation

25.16.2.1 RMOL::FareFamilyException::FareFamilyException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 137 of file RMOL_Types.hpp.

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

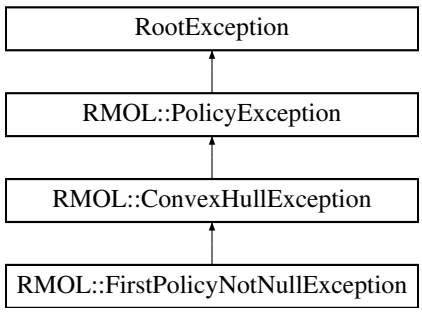
- rmol/RMOL_Types.hpp

25.17 RMOL::FirstPolicyNotNullException Class Reference

Missing policy NULL in convex hull exception.

`#include <rmol/RMOL_Types.hpp>`

Inheritance diagram for RMOL::FirstPolicyNotNullException:



Public Member Functions

- FirstPolicyNotNullException (const std::string &iWhat)

25.17.1 Detailed Description

Missing policy NULL in convex hull exception.

Definition at line 113 of file RMOL_Types.hpp.

25.17.2 Constructor & Destructor Documentation

25.17.2.1 RMOL::FirstPolicyNotNullException::FirstPolicyNotNullException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 116 of file RMOL_Types.hpp.

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

- rmol/RMOL_Types.hpp

25.18 RMOL::Forecaster Class Reference

`#include <rmol/command/Forecaster.hpp>`

Static Public Member Functions

- static bool forecast (stdair::FlightDate &, const stdair::DateTime_T &, const stdair::UnconstrainingMethod &, const stdair::ForecastingMethod &)

25.18.1 Detailed Description

Class wrapping the forecasting algorithms.

Definition at line 22 of file Forecaster.hpp.

25.18.2 Member Function Documentation

25.18.2.1 bool RMOL::Forecaster::forecast (stdair::FlightDate & *ioFlightDate,* const stdair::DateTime_T & *iEventTime,* const stdair::UnconstrainingMethod & *iUnconstrainingMethod,* const stdair::ForecastingMethod & *iForecastingMethod*) [static]

Forecast demand for a flight-date.

Definition at line 35 of file Forecaster.cpp.

Referenced by RMOL::RMOL_Service::optimise().

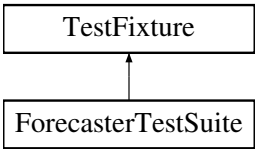
The documentation for this class was generated from the following files:

- rmol/command/Forecaster.hpp
- rmol/command/Forecaster.cpp

25.19 ForecasterTestSuite Class Reference

`#include <test/rmol/ForecasterTestSuite.hpp>`

Inheritance diagram for ForecasterTestSuite:



Public Member Functions

- void testQForecaster ()
- ForecasterTestSuite ()

Protected Attributes

- std::stringstream _describeKey

25.19.1 Detailed Description

Definition at line 6 of file ForecasterTestSuite.hpp.

25.19.2 Constructor & Destructor Documentation

25.19.2.1 ForecasterTestSuite::ForecasterTestSuite ( )

Constructor.

25.19.3 Member Function Documentation

25.19.3.1 void ForecasterTestSuite::testQForecaster ( )

Test Q-forecaster.

25.19.4 Member Data Documentation

25.19.4.1 std::stringstream ForecasterTestSuite::_describeKey [protected]

Definition at line 19 of file ForecasterTestSuite.hpp.

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

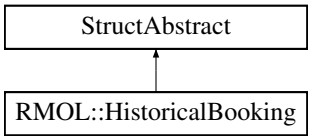
- test/rmol/ForecasterTestSuite.hpp

25.20 RMOL::HistoricalBooking Struct Reference

Structure keeping track, for a given class, of the number of historical bookings and of the censorship flag.

`#include <rmol/bom/HistoricalBooking.hpp>`

Inheritance diagram for RMOL::HistoricalBooking:



Public Member Functions

- const stdair::NbOfBookings_T & getNbOfBookings () const
- const stdair::NbOfBookings_T & getUnconstrainedDemand () const
- const stdair::Flag_T & getFlag () const
- void setUnconstrainedDemand (const stdair::NbOfBookings_T &iDemand)
- void setParameters (const stdair::NbOfBookings_T, const stdair::Flag_T)
- void toStream (std::ostream &ioOut) const
- const std::string describe () const
- void display () const
- HistoricalBooking (const stdair::NbOfBookings_T, const stdair::Flag_T)
- HistoricalBooking ()
- HistoricalBooking (const HistoricalBooking &)
- virtual ~HistoricalBooking ()

25.20.1 Detailed Description

Structure keeping track, for a given class, of the number of historical bookings and of the censorship flag.

Definition at line 17 of file HistoricalBooking.hpp.

25.20.2 Constructor & Destructor Documentation

25.20.2.1 RMOL::HistoricalBooking::HistoricalBooking (const stdair::NbOfBookings_T *iNbOfBookings*, const stdair::Flag_T *iFlag*)

Main constructor.

Definition at line 21 of file HistoricalBooking.cpp.

25.20.2.2 RMOL::HistoricalBooking::HistoricalBooking ( )

Default constructor.

Definition at line 15 of file HistoricalBooking.cpp.

25.20.2.3 RMOL::HistoricalBooking::HistoricalBooking (const HistoricalBooking & *iHistoricalBooking*)

Copy constructor.

Definition at line 29 of file HistoricalBooking.cpp.

25.20.2.4 RMOL::HistoricalBooking::~HistoricalBooking ( ) [virtual]

Destructor.

Definition at line 36 of file HistoricalBooking.cpp.

25.20.3 Member Function Documentation

25.20.3.1 const stdair::NbOfBookings_T& RMOL::HistoricalBooking::getNbOfBookings ( ) const [inline]

Getter for the booking.

Definition at line 22 of file HistoricalBooking.hpp.

Referenced by RMOL::HistoricalBookingHolder::calculateExpectedDemand(), RMOL::HistoricalBookingHolder::getHistoricalBooking(), RMOL::HistoricalBookingHolder::getNbOfUncensoredBookings(), RMOL::HistoricalBookingHolder::getUncensoredStandardDeviation(), toStream(), and RMOL::HistoricalBookingHolder::toStream().

25.20.3.2 const stdair::NbOfBookings_T& RMOL::HistoricalBooking::getUnconstrainedDemand ( ) const [inline]

Getter for the unconstrained bookings.

Definition at line 26 of file HistoricalBooking.hpp.

Referenced by RMOL::HistoricalBookingHolder::getDemandMean(), RMOL::HistoricalBookingHolder::getStandardDeviation(), RMOL::HistoricalBookingHolder::getUnconstrainedDemand(), toStream(), and RMOL::HistoricalBookingHolder::toStream().

25.20.3.3 const stdair::Flag_T& RMOL::HistoricalBooking::getFlag ( ) const [inline]

Getter for the flag of censorship: "false" means that the bookings are not censored.

Definition at line 31 of file HistoricalBooking.hpp.

Referenced by RMOL::HistoricalBookingHolder::getCensorshipFlag(), RMOL::HistoricalBookingHolder::getListOfToBeUnconstrainedFlags(), RMOL::HistoricalBookingHolder::getNbOfUncensoredBookings(), toStream(), and RMOL::HistoricalBookingHolder::toStream().

25.20.3.4 void RMOL::HistoricalBooking::setUnconstrainedDemand (const stdair::NbOfBookings_T & *iDemand*) [inline]

Setter for the unconstraining demand.

Definition at line 38 of file HistoricalBooking.hpp.

25.20.3.5 void RMOL::HistoricalBooking::setParameters (const stdair::NbOfBookings_T *iNbOfBookings*, const stdair::Flag_T *iFlag*)

Setter for all parameters.

Definition at line 41 of file HistoricalBooking.cpp.

25.20.3.6 void RMOL::HistoricalBooking::toStream (std::ostream & *ioOut*) const

Dump a Business Object into an output stream.

Parameters

ostream&	the output stream

Returns

ostream& the output stream.

Definition at line 57 of file HistoricalBooking.cpp.

References getFlag(), getNbOfBookings(), and getUnconstrainedDemand().

Referenced by display().

25.20.3.7 const std::string RMOL::HistoricalBooking::describe ( ) const

Give a description of the structure (for display purposes).

Definition at line 48 of file HistoricalBooking.cpp.

25.20.3.8 void RMOL::HistoricalBooking::display ( ) const

Display on standard output.

Definition at line 66 of file HistoricalBooking.cpp.

References toStream().

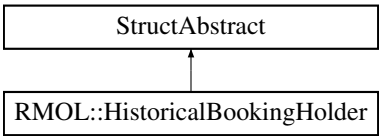
The documentation for this struct was generated from the following files:

- rmol/bom/HistoricalBooking.hpp
- rmol/bom/HistoricalBooking.cpp

25.21 RMOL::HistoricalBookingHolder Struct Reference

`#include <rmol/bom/HistoricalBookingHolder.hpp>`

Inheritance diagram for RMOL::HistoricalBookingHolder:



Public Member Functions

- const short getNbOfFlights () const
- const short getNbOfUncensoredData () const
- const stdair::NbOfBookings_T getNbOfUncensoredBookings () const

- const double getUncensoredStandardDeviation (const double &iMeanOfUncensoredBookings, const short iNbOfUncensoredData) const
- const double getDemandMean () const
- const double getStandardDeviation (const double) const
- const std::vector< bool > getListOfToBeUnconstrainedFlags () const
- const stdair::NbOfBookings_T & getHistoricalBooking (const short i) const
- const stdair::NbOfBookings_T & getUnconstrainedDemand (const short i) const
- const stdair::Flag_T & getCensorshipFlag (const short i) const
- const stdair::NbOfBookings_T & getUnconstrainedDemandOnFirstElement () const
- const stdair::NbOfBookings_T calculateExpectedDemand (const double, const double, const short, const stdair::NbOfBookings_T) const
- void setUnconstrainedDemand (const stdair::NbOfBookings_T &iExpectedDemand, const short i)
- void addHistoricalBooking (const HistoricalBooking &iHistoricalBooking)
- void toStream (std::ostream &ioOut) const
- const std::string describe () const
- void display () const
- virtual ~HistoricalBookingHolder ()
- HistoricalBookingHolder ()

25.21.1 Detailed Description

Holder of a HistoricalBookingList object (for memory allocation and recollection purposes).

Definition at line 23 of file HistoricalBookingHolder.hpp.

25.21.2 Constructor & Destructor Documentation

25.21.2.1 RMOL::HistoricalBookingHolder::~HistoricalBookingHolder ( ) [virtual]

Destructor.

Definition at line 23 of file HistoricalBookingHolder.cpp.

25.21.2.2 RMOL::HistoricalBookingHolder::HistoricalBookingHolder ( )

Constructor.
Protected to force the use of the Factory.

Definition at line 19 of file HistoricalBookingHolder.cpp.

25.21.3 Member Function Documentation

25.21.3.1 const short RMOL::HistoricalBookingHolder::getNbOfFlights ( ) const

Get number of flights.

Definition at line 28 of file HistoricalBookingHolder.cpp.

Referenced by RMOL::QForecasting::forecast(), RMOL::HybridForecasting::forecast(), RMOL::OldQFF::forecast(), RMOL::BasedForecasting::forecast(), and RMOL::EMDetruncator::unconstrain().

25.21.3.2 const short RMOL::HistoricalBookingHolder::getNbOfUncensoredData ( ) const

Get number of uncensored booking data.

Definition at line 33 of file HistoricalBookingHolder.cpp.

Referenced by RMOL::EMDetruncator::unconstrain().

25.21.3.3 const stdair::NbOfBookings_T RMOL::HistoricalBookingHolder::getNbOfUncensoredBookings ( ) const

Get number of uncensored bookings.

Definition at line 49 of file HistoricalBookingHolder.cpp.

References RMOL::HistoricalBooking::getFlag(), and RMOL::HistoricalBooking::getNbOfBookings().

Referenced by RMOL::EMDetruncator::unconstrain().

25.21.3.4 const double RMOL::HistoricalBookingHolder::getUncensoredStandardDeviation (const double & *iMeanOfUncensoredBookings,* const short *iNbOfUncensoredData*) const

Get standard deviation of uncensored bookings.

Definition at line 69 of file HistoricalBookingHolder.cpp.

References RMOL::HistoricalBooking::getNbOfBookings().

Referenced by RMOL::EMDetruncator::unconstrain().

25.21.3.5 const double RMOL::HistoricalBookingHolder::getDemandMean ( ) const

Get mean of historical demand.

Definition at line 95 of file HistoricalBookingHolder.cpp.

References RMOL::HistoricalBooking::getUnconstrainedDemand().

Referenced by RMOL::EMDetruncator::unconstrain().

25.21.3.6 const double RMOL::HistoricalBookingHolder::getStandardDeviation (const double *iDemandMean*) const

Get standard deviation of demand.

Definition at line 116 of file HistoricalBookingHolder.cpp.

References RMOL::HistoricalBooking::getUnconstrainedDemand().

Referenced by RMOL::EMDetruncator::unconstrain().

25.21.3.7 const std::vector< bool > RMOL::HistoricalBookingHolder::getListOfToBeUnconstrainedFlags ( ) const

Get the list of flags of need to be unconstrained.

Definition at line 140 of file HistoricalBookingHolder.cpp.

References RMOL::HistoricalBooking::getFlag().

Referenced by RMOL::EMDetruncator::unconstrain().

25.21.3.8 const stdair::NbOfBookings_T & RMOL::HistoricalBookingHolder::getHistoricalBooking (const short *i*) const

Get the historical booking of the (i+1)-th flight.

Definition at line 161 of file HistoricalBookingHolder.cpp.

References RMOL::HistoricalBooking::getNbOfBookings().

25.21.3.9 const stdair::NbOfBookings_T & RMOL::HistoricalBookingHolder::getUnconstrainedDemand (const short *i*) const

Get the unconstraining demand of the (i+1)-th flight.

Definition at line 169 of file HistoricalBookingHolder.cpp.

References RMOL::HistoricalBooking::getUnconstrainedDemand().

Referenced by RMOL::QForecasting::forecast(), RMOL::HybridForecasting::forecast(), RMOL::OldQFF::forecast(), RMOL::BasedForecasting::forecast(), getUnconstrainedDemandOnFirstElement(), and RMOL::EMDetruncator::unconstrain().

25.21.3.10 const stdair::Flag_T & RMOL::HistoricalBookingHolder::getCensorshipFlag (const short *i*) const

Get the flag of the (i+1)-th flight.

Definition at line 177 of file HistoricalBookingHolder.cpp.

References RMOL::HistoricalBooking::getFlag().

25.21.3.11 const stdair::NbOfBookings_T& RMOL::HistoricalBookingHolder::getUnconstrainedDemandOnFirstElement ( ) const [inline]

Get the unconstraining demand of the first flight.

Definition at line 60 of file HistoricalBookingHolder.hpp.

References getUnconstrainedDemand().

25.21.3.12 const stdair::NbOfBookings_T RMOL::HistoricalBookingHolder::calculateExpectedDemand (const double *iMean,* const double *iSD,* const short *i,* const stdair::NbOfBookings_T *iDemand*) const

Calculate the expected demand.

Definition at line 191 of file HistoricalBookingHolder.cpp.

References RMOL::HistoricalBooking::getNbOfBookings().

25.21.3.13 void RMOL::HistoricalBookingHolder::setUnconstrainedDemand (const stdair::NbOfBookings_T & *iExpectedDemand,* const short *i*)

Set the expected historical demand of the (i+1)-th flight.

Definition at line 185 of file HistoricalBookingHolder.cpp.

Referenced by RMOL::EMDetruncator::unconstrain().

25.21.3.14 void RMOL::HistoricalBookingHolder::addHistoricalBooking (const HistoricalBooking & *iHistoricalBooking*)

Add a HistoricalBooking object to the holder.

Definition at line 236 of file HistoricalBookingHolder.cpp.

Referenced by RMOL::BasedForecasting::prepareHistoricalBooking(), RMOL::QForecasting::preparePrice↩OrientedHistoricalBooking(), and RMOL::HybridForecasting::prepareProductOrientedHistoricalBooking().

25.21.3.15 void RMOL::HistoricalBookingHolder::toStream (std::ostream & *ioOut*) const

Dump a Business Object into an output stream.

Parameters

ostream&	the output stream

Returns

ostream& the output stream.

Definition at line 241 of file HistoricalBookingHolder.cpp.

References RMOL::HistoricalBooking::getFlag(), RMOL::HistoricalBooking::getNbOfBookings(), and RMOL::↩HistoricalBooking::getUnconstrainedDemand().

Referenced by display().

25.21.3.16 const std::string RMOL::HistoricalBookingHolder::describe ( ) const

Give a description of the structure (for display purposes).

Definition at line 265 of file HistoricalBookingHolder.cpp.

25.21.3.17 void RMOL::HistoricalBookingHolder::display ( ) const

Display on standard output.

Definition at line 273 of file HistoricalBookingHolder.cpp.

References toStream().

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

- rmol/bom/HistoricalBookingHolder.hpp
- rmol/bom/HistoricalBookingHolder.cpp

25.22 RMOL::HybridForecasting Class Reference

`#include <rmol/command/HybridForecasting.hpp>`

Static Public Member Functions

- static bool forecast (stdair::SegmentCabin &, const stdair::Date_T &, const stdair::DTD_T &, const stdair::↩ UnconstrainingMethod &, const stdair::NbOfSegments_T &)
- static void prepareProductOrientedHistoricalBooking (const stdair::SegmentCabin &, const stdair::Booking↩ Class &, const stdair::SegmentSnapshotTable &, HistoricalBookingHolder &, const stdair::DCP_T &, const stdair::DCP_T &, const stdair::NbOfSegments_T &, const stdair::NbOfSegments_T &)

25.22.1 Detailed Description

Class wrapping the forecasting algorithms.

Definition at line 21 of file HybridForecasting.hpp.

25.22.2 Member Function Documentation

25.22.2.1 bool RMOL::HybridForecasting::forecast (stdair::SegmentCabin & *ioSegmentCabin,* const stdair::Date_T & *iCurrentDate,* const stdair::DTD_T & *iCurrentDTD,* const stdair::UnconstrainingMethod & *iUnconstrainingMethod,* const stdair::NbOfSegments_T & *iNbOfDepartedSegments*) [static]

Forecast demand for a segment cabin.

Parameters

stdair::↩ SegmentCabin&	Current Segment Cabin
const	stdair::Date_T& Current Date
const	stdair::DTD_T& Current DTD
const	stdair::UnconstrainingMethod& Method used for the unconstraining
const	stdair::NbOfSegments_T& Number of usable historical segments

Definition at line 32 of file HybridForecasting.cpp.

References RMOL::Utilities::computeDistributionParameters(), RMOL::QForecasting::forecast(), RMOL::↩ HistoricalBookingHolder::getNbOfFlights(), RMOL::SegmentSnapshotTableHelper::getNbOfSegmentAlready↩ PassedThisDTD(), RMOL::HistoricalBookingHolder::getUnconstrainedDemand(), prepareProductOriented↩ HistoricalBooking(), and RMOL::Detruncator::unconstrain().

25.22.2.2 void RMOL::HybridForecasting::prepareProductOrientedHistoricalBooking (const stdair::SegmentCabin & *iSegmentCabin,* const stdair::BookingClass & *iBookingClass,* const stdair::SegmentSnapshotTable & *iSegmentSnapshotTable,* HistoricalBookingHolder & *ioHBHolder,* const stdair::DCP_T & *iDCPBegin,* const stdair::DCP_T & *iDCPEnd,* const stdair::NbOfSegments_T & *iSegmentBegin,* const stdair::NbOfSegments_T & *iSegmentEnd*) [static]

Prepare the historical product-oriented booking figures for a given cabin

Parameters

const	stdair::DCP_T& DCP range start
const	stdair::DCP_T& DCP range end
const	stdair::NbOfSegments_T& Segment range start index
const	stdair::NbOfSegments_T& Segment range end index

Definition at line 125 of file HybridForecasting.cpp.

References RMOL::HistoricalBookingHolder::addHistoricalBooking().

Referenced by forecast().

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

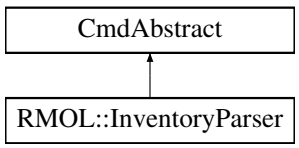
- rmol/command/HybridForecasting.hpp
- rmol/command/HybridForecasting.cpp

25.23 RMOL::InventoryParser Class Reference

Class filling the virtual class list (representing a list of classes/buckets) from a given input inventory.

`#include <rmol/command/InventoryParser.hpp>`

Inheritance diagram for RMOL::InventoryParser:



Static Public Member Functions

- static bool parseInputFileAndBuildBom (const std::string &iInputFileName, stdair::BomRoot &)

25.23.1 Detailed Description

Class filling the virtual class list (representing a list of classes/buckets) from a given input inventory.

Definition at line 25 of file InventoryParser.hpp.

25.23.2 Member Function Documentation

25.23.2.1 bool RMOL::InventoryParser::parseInputFileAndBuildBom (const std::string & *iInputFileName*, stdair::BomRoot & *ioBomRoot*) [static]

Parse the input values from a CSV-formatted inventory file.

Parameters

const	std::string& iInputFileName Inventory file to be parsed.
stdair::Bom↩ Root&	The BOM tree.

Returns

bool Whether or not the parsing was successful.

Definition at line 36 of file InventoryParser.cpp.

Referenced by RMOL::RMOL_Service::parseAndLoad().

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

- rmol/command/InventoryParser.hpp
- rmol/command/InventoryParser.cpp

25.24 RMOL::MarginalRevenueTransformation Class Reference

`#include <rmol/command/MarginalRevenueTransformation.hpp>`

Static Public Member Functions

- static bool prepareDemandInput (stdair::SegmentCabin &)

25.24.1 Detailed Description

Class wrapping the pre-optimisation algorithms.

Definition at line 21 of file MarginalRevenueTransformation.hpp.

25.24.2 Member Function Documentation

25.24.2.1 bool RMOL::MarginalRevenueTransformation::prepareDemandInput (stdair::SegmentCabin & *ioSegmentCabin*) [static]

Prepare the demand input for the optimser.

Definition at line 28 of file MarginalRevenueTransformation.cpp.

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

- rmol/command/MarginalRevenueTransformation.hpp
- rmol/command/MarginalRevenueTransformation.cpp

25.25 RMOL::MCOptimiser Class Reference

`#include <rmol/bom/MCOptimiser.hpp>`

Static Public Member Functions

- static void optimalOptimisationByMCIntegration (stdair::LegCabin &)
- static stdair::GeneratedDemandVector_T generateDemandVector (const stdair::MeanValue_T &, const stdair::StdDevValue_T &, const stdair::NbOfSamples_T &)
- static void optimisationByMCIntegration (stdair::LegCabin &)

25.25.1 Detailed Description

Utility methods for the Monte-Carlo algorithms.

Definition at line 19 of file MCOptimiser.hpp.

25.25.2 Member Function Documentation

25.25.2.1 void RMOL::MCOptimiser::optimalOptimisationByMCIntegration (stdair::LegCabin & *ioLegCabin*) [static]

Calculate the optimal protections for the set of buckets/classes given in input, and update those buckets accordingly. The Monte Carlo Integration algorithm (see The Theory and Practice of Revenue Management, by Kalyan T. Talluri and Garret J. van Ryzin, Kluwer Academic Publishers, for the details) is used.

Definition at line 28 of file MCOptimiser.cpp.

Referenced by RMOL::Optimiser::optimalOptimisationByMCIntegration().

25.25.2.2 stdair::GeneratedDemandVector_T RMOL::MCOptimiser::generateDemandVector (const stdair::MeanValue_T & *iMean,* const stdair::StdDevValue_T & *iStdDev,* const stdair::NbOfSamples_T & *K*) [static]

Monte-Carlo

Definition at line 154 of file MCOptimiser.cpp.

Referenced by optimisationByMCIntegration().

25.25.2.3 void RMOL::MCOptimiser::optimisationByMCIntegration (stdair::LegCabin & *ioLegCabin*) [static]

Definition at line 175 of file MCOptimiser.cpp.

References RMOL::DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION, and generateDemandVector().

Referenced by RMOL::Optimiser::optimiseUsingOnDForecast().

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

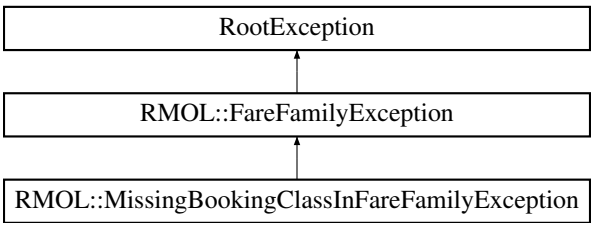
- rmol/bom/MCOptimiser.hpp
- rmol/bom/MCOptimiser.cpp

25.26 RMOL::MissingBookingClassInFareFamilyException Class Reference

Missing Booking Class in Fare Family exception.

`#include <rmol/RMOL_Types.hpp>`

Inheritance diagram for RMOL::MissingBookingClassInFareFamilyException:



Public Member Functions

- MissingBookingClassInFareFamilyException (const std::string &iWhat)

25.26.1 Detailed Description

Missing Booking Class in Fare Family exception.

Definition at line 154 of file RMOL_Types.hpp.

25.26.2 Constructor & Destructor Documentation

25.26.2.1 RMOL::MissingBookingClassInFareFamilyException::MissingBookingClassInFareFamilyException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 157 of file RMOL_Types.hpp.

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

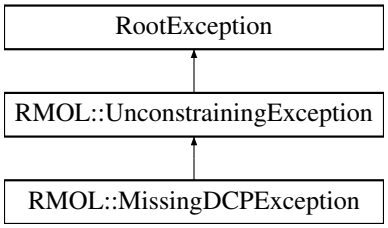
- rmol/RMOL_Types.hpp

25.27 RMOL::MissingDCPException Class Reference

Missing a DCP in unconstrainer exception.

`#include <rmol/RMOL_Types.hpp>`

Inheritance diagram for RMOL::MissingDCPException:



Public Member Functions

- MissingDCPException (const std::string &iWhat)

25.27.1 Detailed Description

Missing a DCP in unconstrainer exception.

Definition at line 62 of file RMOL_Types.hpp.

25.27.2 Constructor & Destructor Documentation

25.27.2.1 RMOL::MissingDCPException::MissingDCPException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 65 of file RMOL_Types.hpp.

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

- rmol/RMOL_Types.hpp

25.28 RMOL::NewQFF Class Reference

`#include <rmol/command/NewQFF.hpp>`

Static Public Member Functions

- static bool forecast (stdair::SegmentCabin &, const stdair::Date_T &, const stdair::DTD_T &, const stdair::↩ UnconstrainingMethod &, const stdair::NbOfSegments_T &)

25.28.1 Detailed Description

Class wrapping the forecasting algorithms.

Definition at line 23 of file NewQFF.hpp.

25.28.2 Member Function Documentation

25.28.2.1 bool RMOL::NewQFF::forecast (stdair::SegmentCabin & *ioSegmentCabin,* const stdair::Date_T & *iCurrentDate,* const stdair::DTD_T & *iCurrentDTD,* const stdair::UnconstrainingMethod & *iUnconstrainingMethod,* const stdair::NbOfSegments_T & *iNbOfDepartedSegments*) [static]

Forecast demand for a segment cabin.

Parameters

stdair::↩ SegmentCabin&	Current Segment Cabin
const	stdair::Date_T& Current Date
const	stdair::DTD_T& Current DTD
const	stdair::UnconstrainingMethod& Method used for the unconstraining
const	stdair::NbOfSegments_T& Number of usable historical segments

Definition at line 31 of file NewQFF.cpp.

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

- rmol/command/NewQFF.hpp
- rmol/command/NewQFF.cpp

25.29 RMOL::OldQFF Class Reference

`#include <rmol/command/OldQFF.hpp>`

Static Public Member Functions

- static bool forecast (stdair::SegmentCabin &, const stdair::Date_T &, const stdair::DTD_T &, const stdair::↩ UnconstrainingMethod &, const stdair::NbOfSegments_T &)

25.29.1 Detailed Description

Class wrapping the forecasting algorithms.

Definition at line 23 of file OldQFF.hpp.

25.29.2 Member Function Documentation

25.29.2.1 bool RMOL::OldQFF::forecast (stdair::SegmentCabin & *ioSegmentCabin,* const stdair::Date_T & *iCurrentDate,* const stdair::DTD_T & *iCurrentDTD,* const stdair::UnconstrainingMethod & *iUnconstrainingMethod,* const stdair::NbOfSegments_T & *iNbOfDepartedSegments*) [static]

Forecast demand for a segment cabin.

Parameters

stdair::SegmentCabin&	Current Segment Cabin
const	stdair::Date_T& Current Date
const	stdair::DTD_T& Current DTD
const	stdair::UnconstrainingMethod& Method used for the unconstraining
const	stdair::NbOfSegments_T& Number of usable historical segments

Definition at line 31 of file OldQFF.cpp.

References RMOL::Utilities::computeDistributionParameters(), RMOL::Utilities::computeSellUpFactorCurves(), RMOL::HistoricalBookingHolder::getNbOfFlights(), RMOL::SegmentSnapshotTableHelper::getNbOfSegmentAlreadyPassedThisDTD(), RMOL::HistoricalBookingHolder::getUnconstrainedDemand(), and RMOL::Detruncator::unconstrain().

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

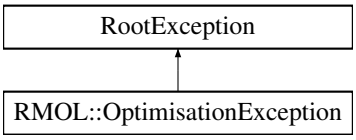
- rmol/command/OldQFF.hpp
- rmol/command/OldQFF.cpp

25.30 RMOL::OptimisationException Class Reference

Optimisation-related exception.

`#include <rmol/RMOL_Types.hpp>`

Inheritance diagram for RMOL::OptimisationException:



Public Member Functions

- OptimisationException (const std::string &iWhat)

25.30.1 Detailed Description

Optimisation-related exception.

Definition at line 72 of file RMOL_Types.hpp.

25.30.2 Constructor & Destructor Documentation

25.30.2.1 RMOL::OptimisationException::OptimisationException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 75 of file RMOL_Types.hpp.

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

- rmol/RMOL_Types.hpp

25.31 RMOL::Optimiser Class Reference

`#include <rmol/command/Optimiser.hpp>`

Static Public Member Functions

- static void optimalOptimisationByMCIntegration (const stdair::NbOfSamples_T &, stdair::LegCabin &)
- static void optimalOptimisationByDP (stdair::LegCabin &)
- static void heuristicOptimisationByEmsr (stdair::LegCabin &)
- static void heuristicOptimisationByEmsrA (stdair::LegCabin &)
- static void heuristicOptimisationByEmsrB (stdair::LegCabin &)
- static bool optimise (stdair::FlightDate &, const stdair::OptimisationMethod &)
- static bool buildVirtualClassListForLegBasedOptimisation (stdair::LegCabin &)
- static double optimiseUsingOnDForecast (stdair::FlightDate &, const bool &iReduceFluctuations=false)

25.31.1 Detailed Description

Class wrapping the optimisation algorithms.

Definition at line 20 of file Optimiser.hpp.

25.31.2 Member Function Documentation

25.31.2.1 void RMOL::Optimiser::optimalOptimisationByMCIntegration (const stdair::NbOfSamples_T & *K*, stdair::LegCabin & *ioLegCabin*) [static]

Monte Carlo Integration algorithm.
Calculate the optimal protections for the set of buckets/classes given in input, and update those buckets accordingly. The Monte Carlo Integration algorithm (see The Theory and Practice of Revenue Management, by Kalyan T. Talluri and Garret J. van Ryzin, Kluwer Academic Publishers, for the details) is used. Hence, K is the number of random draws to perform. 100 is a minimum for K, as statistics must be drawn from those random generations.

Definition at line 30 of file Optimiser.cpp.

References RMOL::MCOptimiser::optimalOptimisationByMCIntegration().

Referenced by RMOL::RMOL_Service::optimalOptimisationByMCIntegration().

25.31.2.2 void RMOL::Optimiser::optimalOptimisationByDP (stdair::LegCabin & *ioLegCabin*) [static]

Dynamic Programming.

Definition at line 64 of file Optimiser.cpp.

References RMOL::DPOptimiser::optimalOptimisationByDP().

25.31.2.3 void RMOL::Optimiser::heuristicOptimisationByEmsr (stdair::LegCabin & *ioLegCabin*) [static]

EMRS algorithm.

Definition at line 69 of file Optimiser.cpp.

References RMOL::Emsr::heuristicOptimisationByEmsr().

Referenced by RMOL::RMOL_Service::heuristicOptimisationByEmsr().

25.31.2.4 void RMOL::Optimiser::heuristicOptimisationByEmsrA (stdair::LegCabin & *ioLegCabin*) [static]

EMRS-a algorithm.

Definition at line 74 of file Optimiser.cpp.

References RMOL::Emsr::heuristicOptimisationByEmsrA().

Referenced by RMOL::RMOL_Service::heuristicOptimisationByEmsrA().

25.31.2.5 void RMOL::Optimiser::heuristicOptimisationByEmsrB (stdair::LegCabin & *ioLegCabin*) [static]

EMRS-b algorithm.

Definition at line 79 of file Optimiser.cpp.

References RMOL::Emsr::heuristicOptimisationByEmsrB().

Referenced by RMOL::RMOL_Service::heuristicOptimisationByEmsrB().

25.31.2.6 bool RMOL::Optimiser::optimise (stdair::FlightDate & *ioFlightDate,* const stdair::OptimisationMethod & *iOptimisationMethod*) [static]

Optimise a flight-date using leg-based Monte Carlo Integration.

Definition at line 84 of file Optimiser.cpp.

Referenced by RMOL::RMOL_Service::optimise().

25.31.2.7 bool RMOL::Optimiser::buildVirtualClassListForLegBasedOptimisation (stdair::LegCabin & *ioLegCabin*) [static]

Build the virtual class list for the given leg-cabin.

Definition at line 164 of file Optimiser.cpp.

25.31.2.8 double RMOL::Optimiser::optimiseUsingOnDForecast (stdair::FlightDate & *ioFlightDate,* const bool & *iReduceFluctuations* = `false`) [static]

Optimiser

Definition at line 247 of file Optimiser.cpp.

References RMOL::MCOptimiser::optimisationByMCIntegration().

Referenced by RMOL::RMOL_Service::optimiseOnD(), RMOL::RMOL_Service::optimiseOnDUsingAdvancedRM↩Cooperation(), and RMOL::RMOL_Service::optimiseOnDUsingRMCooperation().

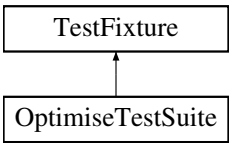
The documentation for this class was generated from the following files:

- rmol/command/Optimiser.hpp
- rmol/command/Optimiser.cpp

25.32 OptimiseTestSuite Class Reference

`#include <test/rmol/OptimiseTestSuite.hpp>`

Inheritance diagram for OptimiseTestSuite:



Public Member Functions

- void testOptimiseMC ()
- void testOptimiseDP ()

- void testOptimiseEMSR ()
- void testOptimiseEMSRa ()
- void testOptimiseEMSRb ()
- OptimiseTestSuite ()

Protected Attributes

- std::stringstream _describeKey

25.32.1 Detailed Description

Definition at line 6 of file OptimiseTestSuite.hpp.

25.32.2 Constructor & Destructor Documentation

25.32.2.1 OptimiseTestSuite::OptimiseTestSuite ( )

Test some error detection functionalities. Constructor.

25.32.3 Member Function Documentation

25.32.3.1 void OptimiseTestSuite::testOptimiseMC ( )

Test the Monte-Carlo (MC) Optimisation functionality.

25.32.3.2 void OptimiseTestSuite::testOptimiseDP ( )

Test the Dynamic Programming (DP) Optimisation functionality.

25.32.3.3 void OptimiseTestSuite::testOptimiseEMSR ( )

Test the Expected Marginal Seat Revenue (EMSR) Optimisation functionality.

25.32.3.4 void OptimiseTestSuite::testOptimiseEMSRa ( )

Test the Expected Marginal Seat Revenue, variant a (EMSR-a), Optimisation functionality.

25.32.3.5 void OptimiseTestSuite::testOptimiseEMSRb ( )

Test the Expected Marginal Seat Revenue, variant b (EMSR-b), Optimisation functionality.

25.32.4 Member Data Documentation

25.32.4.1 std::stringstream OptimiseTestSuite::_describeKey [protected]

Definition at line 43 of file OptimiseTestSuite.hpp.

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

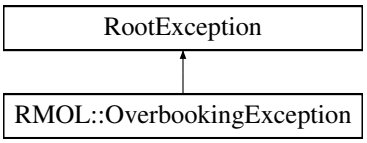
- test/rmol/OptimiseTestSuite.hpp

25.33 RMOL::OverbookingException Class Reference

Overbooking-related exception.

`#include <rmol/RMOL_Types.hpp>`

Inheritance diagram for RMOL::OverbookingException:



Public Member Functions

- OverbookingException (const std::string &iWhat)

25.33.1 Detailed Description

Overbooking-related exception.

Definition at line 32 of file RMOL_Types.hpp.

25.33.2 Constructor & Destructor Documentation

25.33.2.1 RMOL::OverbookingException::OverbookingException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 35 of file RMOL_Types.hpp.

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

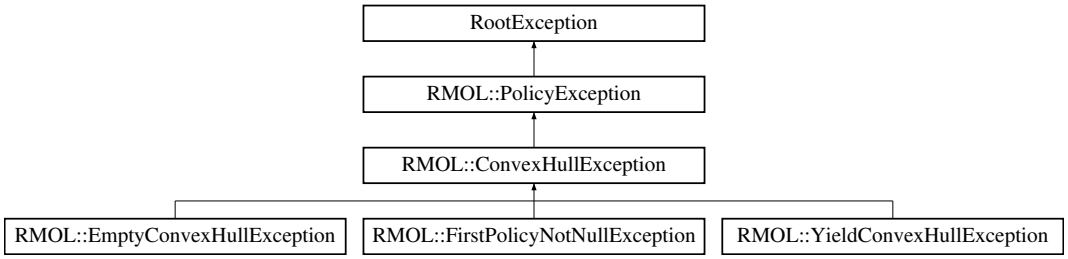
- rmol/RMOL_Types.hpp

25.34 RMOL::PolicyException Class Reference

Policy-related exception.

`#include <rmol/RMOL_Types.hpp>`

Inheritance diagram for RMOL::PolicyException:



Public Member Functions

- PolicyException (const std::string &iWhat)

25.34.1 Detailed Description

Policy-related exception.

Definition at line 82 of file RMOL_Types.hpp.

25.34.2 Constructor & Destructor Documentation

25.34.2.1 RMOL::PolicyException::PolicyException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 85 of file RMOL_Types.hpp.

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

- rmol/RMOL_Types.hpp

25.35 RMOL::PolicyHelper Class Reference

`#include <rmol/bom/PolicyHelper.hpp>`

Static Public Member Functions

- static void diffBetweenTwoPolicies (stdair::NestingNode &, const stdair::Policy &, const stdair::Policy &)
- static void computeLastNode (stdair::NestingNode &, const stdair::Policy &, const stdair::SegmentCabin &)
- static bool isNested (const stdair::Policy &, const stdair::Policy &)

25.35.1 Detailed Description

Class holding helper methods.

Definition at line 28 of file PolicyHelper.hpp.

25.35.2 Member Function Documentation

25.35.2.1 void RMOL::PolicyHelper::diffBetweenTwoPolicies (stdair::NestingNode & *ioNode,* const stdair::Policy & *iFirstPolicy,* const stdair::Policy & *iSecondPolicy*) [static]

Find the booking class list representing the difference between two Policies (first minus second)

Definition at line 24 of file PolicyHelper.cpp.

25.35.2.2 void RMOL::PolicyHelper::computeLastNode (stdair::NestingNode & *ioNode,* const stdair::Policy & *iPolicy,* const stdair::SegmentCabin & *iSegmentCabin*) [static]

Compute the list of the booking class which is not in the node.

Definition at line 164 of file PolicyHelper.cpp.

25.35.2.3 bool RMOL::PolicyHelper::isNested (const stdair::Policy & *iFirstPolicy,* const stdair::Policy & *iSecondPolicy*) [static]

Check if the first policy is nested under the second policy.

Definition at line 220 of file PolicyHelper.cpp.

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

- rmol/bom/PolicyHelper.hpp
- rmol/bom/PolicyHelper.cpp

25.36 RMOL::PreOptimiser Class Reference

`#include <rmol/command/PreOptimiser.hpp>`

Static Public Member Functions

- static bool preOptimise (stdair::FlightDate &, const stdair::PreOptimisationMethod &)

25.36.1 Detailed Description

Class wrapping the pre-optimisation algorithms.

Definition at line 22 of file PreOptimiser.hpp.

25.36.2 Member Function Documentation

25.36.2.1 bool RMOL::PreOptimiser::preOptimise (stdair::FlightDate & *ioFlightDate,* const stdair::PreOptimisationMethod & *iPreOptimisationMethod*) [static]

Prepare the demand input for the optimser.

Definition at line 30 of file PreOptimiser.cpp.

Referenced by RMOL::RMOL_Service::optimise().

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

- rmol/command/PreOptimiser.hpp
- rmol/command/PreOptimiser.cpp

25.37 RMOL::QForecasting Class Reference

`#include <rmol/command/QForecasting.hpp>`

Static Public Member Functions

- static bool forecast (stdair::SegmentCabin &, const stdair::Date_T &, const stdair::DTD_T &, const stdair::↩ UnconstrainingMethod &, const stdair::NbOfSegments_T &)
- static void preparePriceOrientedHistoricalBooking (const stdair::SegmentCabin &, const stdair::Segment↩ SnapshotTable &, HistoricalBookingHolder &, const stdair::DCP_T &, const stdair::DCP_T &, const stdair::↩ NbOfSegments_T &, const stdair::NbOfSegments_T &, const stdair::BookingClassSellUpCurveMap_T &)

25.37.1 Detailed Description

Class wrapping the optimisation algorithms.

Definition at line 23 of file QForecasting.hpp.

25.37.2 Member Function Documentation

25.37.2.1 bool RMOL::QForecasting::forecast (stdair::SegmentCabin & *ioSegmentCabin,* const stdair::Date_T & *iCurrentDate,* const stdair::DTD_T & *iCurrentDTD,* const stdair::UnconstrainingMethod & *iUnconstrainingMethod,* const stdair::NbOfSegments_T & *iNbOfDepartedSegments*) [static]

Forecast demand for a flight-date.

Parameters

const	stdair::Date_T& Current Date
const	stdair::NbOfSegments_T& Number of usable historical segments

Definition at line 31 of file QForecasting.cpp.

References RMOL::Utilities::computeDispatchingFactorCurves(), RMOL::Utilities::computeDistributionParameters(), RMOL::Utilities::computeSellUpFactorCurves(), RMOL::Utilities::dispatchDemandForecast(), RMOL::Utilities::dispatchDemandForecastForFA(), RMOL::HistoricalBookingHolder::getNbOfFlights(), RMOL::SegmentSnapshotTableHelper::getNbOfSegmentAlreadyPassedThisDTD(), RMOL::HistoricalBookingHolder::getUnconstrainedDemand(), preparePriceOrientedHistoricalBooking(), and RMOL::Detruncator::unconstrain().

Referenced by RMOL::HybridForecasting::forecast().

25.37.2.2 void RMOL::QForecasting::preparePriceOrientedHistoricalBooking (const stdair::SegmentCabin & *iSegmentCabin,* const stdair::SegmentSnapshotTable & *iSegmentSnapshotTable,* HistoricalBookingHolder & *ioHBHolder,* const stdair::DCP_T & *iDCPBegin,* const stdair::DCP_T & *iDCPEnd,* const stdair::NbOfSegments_T & *iSegmentBegin,* const stdair::NbOfSegments_T & *iSegmentEnd,* const stdair::BookingClassSellUpCurveMap_T & *iBCSellUpCurveMap*) [static]

Prepare the historical price-oriented booking figures for a given cabin

Parameters

const	stdair::DCP_T& DCP range start
const	stdair::DCP_T& DCP range end
const	stdair::NbOfSegments_T& Segment range start index
const	stdair::NbOfSegments_T& Segment range end index

Definition at line 136 of file QForecasting.cpp.

References RMOL::HistoricalBookingHolder::addHistoricalBooking().

Referenced by forecast().

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

- rmol/command/QForecasting.hpp
- rmol/command/QForecasting.cpp

25.38 RMOL::RMOL_Service Class Reference

Interface for the RMOL Services.

`#include <rmol/RMOL_Service.hpp>`

Public Member Functions

- RMOL_Service (const stdair::BasLogParams &, const stdair::BasDBParams &)
- RMOL_Service (const stdair::BasLogParams &)
- RMOL_Service (stdair::STDAIR_ServicePtr_T)
- void parseAndLoad (const stdair::CabinCapacity_T &iCabinCapacity, const stdair::Filename_T &iDemandAndClassDataFile)
- void setUpStudyStatManager ()
- ~RMOL_Service ()
- void buildSampleBom ()
- void clonePersistentBom ()
- void buildComplementaryLinks (stdair::BomRoot &)
- void optimalOptimisationByMCIntegration (const int K)
- void optimalOptimisationByDP ()

- void heuristicOptimisationByEmsr ()
- void heuristicOptimisationByEmsrA ()
- void heuristicOptimisationByEmsrB ()
- void heuristicOptimisationByMCIntegrationForQFF ()
- void heuristicOptimisationByEmsrBForQFF ()
- void MRTForNewQFF ()
- const stdair::SegmentCabin & retrieveDummySegmentCabin (const bool isForFareFamilies=false)
- bool optimise (stdair::FlightDate &, const stdair::DateTime_T &, const stdair::UnconstrainingMethod &, const stdair::ForecastingMethod &, const stdair::PreOptimisationMethod &, const stdair::OptimisationMethod &, const stdair::PartnershipTechnique &)
- void forecastOnD (const stdair::DateTime_T &)
- stdair::YieldFeatures * getYieldFeatures (const stdair::OnDDate &, const stdair::CabinCode_T &, stdair::BomRoot &)
- void forecastOnD (const stdair::YieldFeatures &, stdair::OnDDate &, const stdair::CabinCode_T &, const stdair::DTD_T &, stdair::BomRoot &)
- void setOnDForecast (const stdair::AirlineClassList &, const stdair::MeanValue_T &, const stdair::StdDevValue_T &, stdair::OnDDate &, const stdair::CabinCode_T &, stdair::BomRoot &)
- void setOnDForecast (const stdair::AirlineCode_T &, const stdair::Date_T &, const stdair::AirportCode_T &, const stdair::AirportCode_T &, const stdair::CabinCode_T &, const stdair::ClassCode_T &, const stdair::MeanValue_T &, const stdair::StdDevValue_T &, const stdair::Yield_T &, stdair::BomRoot &)
- void setOnDForecast (const stdair::AirlineCodeList_T &, const stdair::AirlineCode_T &, const stdair::Date_T &, const stdair::AirportCode_T &, const stdair::AirportCode_T &, const stdair::CabinCode_T &, const stdair::ClassCodeList_T &, const stdair::MeanValue_T &, const stdair::StdDevValue_T &, const stdair::Yield_T &, stdair::BomRoot &)
- void resetDemandInformation (const stdair::DateTime_T &)
- void resetDemandInformation (const stdair::DateTime_T &, const stdair::Inventory &)
- void projectAggregatedDemandOnLegCabins (const stdair::DateTime_T &)
- void projectOnDDemandOnLegCabinsUsingYP (const stdair::DateTime_T &)
- void projectOnDDemandOnLegCabinsUsingDA (const stdair::DateTime_T &)
- void projectOnDDemandOnLegCabinsUsingDYP (const stdair::DateTime_T &)
- void projectOnDDemandOnLegCabinsUsingDYP (const stdair::DateTime_T &, const stdair::Inventory &)
- void optimiseOnD (const stdair::DateTime_T &)
- void optimiseOnDUsingRMCooperation (const stdair::DateTime_T &)
- void optimiseOnDUsingAdvancedRMCooperation (const stdair::DateTime_T &)
- void updateBidPrice (const stdair::DateTime_T &)
- void updateBidPrice (const stdair::FlightDate &, stdair::BomRoot &)
- std::string jsonExport (const stdair::AirlineCode_T &, const stdair::FlightNumber_T &, const stdair::Date_T &iDepartureDate) const
- std::string csvDisplay () const

25.38.1 Detailed Description

Interface for the RMOL Services.

Definition at line 43 of file RMOL_Service.hpp.

25.38.2 Constructor & Destructor Documentation

25.38.2.1 RMOL::RMOL_Service::RMOL_Service (const stdair::BasLogParams & *iLogParams,* const stdair::BasDBParams & *iDBParams*)

Constructor.

The initRmolService() 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.

Moreover, database connection parameters are given, so that a session can be created on the corresponding database.

Parameters

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

Definition at line 85 of file RMOL_Service.cpp.

25.38.2.2 RMOL::RMOL_Service::RMOL_Service (const stdair::BasLogParams & *iLogParams*)

Constructor.

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

Moreover, 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 64 of file RMOL_Service.cpp.

25.38.2.3 RMOL::RMOL_Service::RMOL_Service (stdair::STDAIR_ServicePtr_T *ioSTDAIRServicePtr*)

Constructor.

The initRmolService() 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 RMOL_Service is itself being initialised by another library service such as AIRINV_Service).

Parameters

STDAIR_↩ ServicePtr_T	the shared pointer of stdair service.

Definition at line 107 of file RMOL_Service.cpp.

25.38.2.4 RMOL::RMOL_Service::∼RMOL_Service ( )

Destructor.

Definition at line 124 of file RMOL_Service.cpp.

25.38.3 Member Function Documentation

25.38.3.1 void RMOL::RMOL_Service::parseAndLoad (const stdair::CabinCapacity_T & *iCabinCapacity*, const stdair::Filename_T & *iDemandAndClassDataFile*)

Parse the optimisation-related data and load them into memory.

First, the STDAIR_Service::buildDummyInventory() method is called, for RMOL and with the given cabin capacity, in order to build the miminum required flight-date structure in order to perform an optimisation on a leg-cabin.

The CSV input file describes the problem to be optimised, i.e.:

- the demand specifications for all the booking classes (mean and standard deviations for the demand distribution); the yields corresponding to those booking classes.

That CSV file is parsed and instantiated in memory accordingly. The leg-cabin capacity has been set at the initialisation of the (RMOL) service.

Parameters

const	stdair::CabinCapacity& Capacity of the leg-cabin to be optimised.
const	stdair::Filename_T& (CSV) input file.

Definition at line 201 of file RMOL_Service.cpp.

References buildComplementaryLinks(), clonePersistentBom(), and RMOL::InventoryParser::parseInputFileAnd↩ BuildBom().

Referenced by main().

25.38.3.2 void RMOL::RMOL_Service::setUpStudyStatManager ( )

Set up the StudyStatManager.

25.38.3.3 void RMOL::RMOL_Service::buildSampleBom ( )

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

See also

stdair::CmdBomManager::buildSampleBom() for more details.

Definition at line 260 of file RMOL_Service.cpp.

References buildComplementaryLinks(), and clonePersistentBom().

Referenced by main().

25.38.3.4 void RMOL::RMOL_Service::clonePersistentBom ( )

Clone the persistent BOM object.

Definition at line 319 of file RMOL_Service.cpp.

References buildComplementaryLinks().

Referenced by buildSampleBom(), and parseAndLoad().

25.38.3.5 void RMOL::RMOL_Service::buildComplementaryLinks (stdair::BomRoot & *ioBomRoot*)

Build all the complementary links in the given bom root object. Build the links between dummy leg cabin and dummy segment cabin.

Definition at line 357 of file RMOL_Service.cpp.

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

25.38.3.6 void RMOL::RMOL_Service::optimalOptimisationByMCIntegration (const int *K*)

Single resource optimization using the Monte Carlo algorithm.

Definition at line 382 of file RMOL_Service.cpp.

References RMOL::Optimiser::optimalOptimisationByMCIntegration().

Referenced by optimise().

25.38.3.7 void RMOL::RMOL_Service::optimalOptimisationByDP ( )

Single resource optimization using dynamic programming.

Definition at line 426 of file RMOL_Service.cpp.

Referenced by optimise().

25.38.3.8 void RMOL::RMOL_Service::heuristicOptimisationByEmsr ( )

Single resource optimization using EMSR heuristic.

Definition at line 430 of file RMOL_Service.cpp.

References RMOL::Optimiser::heuristicOptimisationByEmsr().

Referenced by optimise().

25.38.3.9 void RMOL::RMOL_Service::heuristicOptimisationByEmsrA ( )

Single resource optimization using EMSR-a heuristic.

Definition at line 475 of file RMOL_Service.cpp.

References RMOL::Optimiser::heuristicOptimisationByEmsrA().

Referenced by optimise().

25.38.3.10 void RMOL::RMOL_Service::heuristicOptimisationByEmsrB ( )

Single resource optimization using EMSR-b heuristic.

Definition at line 500 of file RMOL_Service.cpp.

References RMOL::Optimiser::heuristicOptimisationByEmsrB().

Referenced by optimise().

25.38.3.11 void RMOL::RMOL_Service::heuristicOptimisationByMCIntegrationForQFF ( )

Single resource optimization using the Monte Carlo algorithm for QFF method.

25.38.3.12 void RMOL::RMOL_Service::heuristicOptimisationByEmsrBForQFF ( )

Single resource optimization using EMSR-b heuristic for QFF method.

25.38.3.13 void RMOL::RMOL_Service::MRTForNewQFF ( )

Single resource pre-optimization using Marginal Revenue Transformation for QFF method.

25.38.3.14 const stdair::SegmentCabin & RMOL::RMOL_Service::retrieveDummySegmentCabin (const bool *isForFareFamilies* = `false`)

Retrieve one sample segment-cabin of the dummy inventory of "XX".

Parameters

const	bool Boolean to choose the sample segment-cabin. True: the dummy segment-cabin with fare families. False: the dummy segment-cabin without fare families. By default the value is false.

Definition at line 525 of file RMOL_Service.cpp.

25.38.3.15 bool RMOL::RMOL_Service::optimise (stdair::FlightDate & *ioFlightDate,* const stdair::DateTime_T & *iRMEventTime,* const stdair::UnconstrainingMethod & *iUnconstrainingMethod,* const stdair::ForecastingMethod & *iForecastingMethod,* const stdair::PreOptimisationMethod & *iPreOptimisationMethod,* const stdair::OptimisationMethod & *iOptimisationMethod,* const stdair::PartnershipTechnique & *iPartnershipTechnique*)

Optimise (revenue management) an flight-date/network-date

Definition at line 547 of file RMOL_Service.cpp.

References RMOL::Forecaster::forecast(), forecastOnD(), RMOL::Optimiser::optimise(), optimiseOnD(), optimise↩OnDUsingAdvancedRMCooperation(), optimiseOnDUsingRMCooperation(), RMOL::PreOptimiser::preOptimise(), projectAggregatedDemandOnLegCabins(), projectOnDDemandOnLegCabinsUsingDYP(), projectOnDDemand↩

OnLegCabinsUsingYP(), resetDemandInformation(), and updateBidPrice().

25.38.3.16 void RMOL::RMOL_Service::forecastOnD (const stdair::DateTime_T & *iRMEventTime*)

Forecaster

Definition at line 648 of file RMOL_Service.cpp.

References getYieldFeatures().

Referenced by optimise().

25.38.3.17 stdair::YieldFeatures * RMOL::RMOL_Service::getYieldFeatures (const stdair::OnDDate & *iOnDDate,* const stdair::CabinCode_T & *iCabinCode,* stdair::BomRoot & *iBomRoot*)

Definition at line 723 of file RMOL_Service.cpp.

Referenced by forecastOnD().

25.38.3.18 void RMOL::RMOL_Service::forecastOnD (const stdair::YieldFeatures & *iYieldFeatures,* stdair::OnDDate & *iOnDDate,* const stdair::CabinCode_T & *iCabinCode,* const stdair::DTD_T & *iDTD,* stdair::BomRoot & *iBomRoot*)

Definition at line 796 of file RMOL_Service.cpp.

References setOnDForecast().

25.38.3.19 void RMOL::RMOL_Service::setOnDForecast (const stdair::AirlineClassList & *iAirlineClassList,* const stdair::MeanValue_T & *iMeanValue,* const stdair::StdDevValue_T & *iStdDevValue,* stdair::OnDDate & *iOnDDate,* const stdair::CabinCode_T & *iCabinCode,* stdair::BomRoot & *iBomRoot*)

Definition at line 911 of file RMOL_Service.cpp.

Referenced by forecastOnD().

25.38.3.20 void RMOL::RMOL_Service::setOnDForecast (const stdair::AirlineCode_T & *iAirlineCode,* const stdair::Date_T & *iDepartureDate,* const stdair::AirportCode_T & *iOrigin,* const stdair::AirportCode_T & *iDestination,* const stdair::CabinCode_T & *iCabinCode,* const stdair::ClassCode_T & *iClassCode,* const stdair::MeanValue_T & *iMeanValue,* const stdair::StdDevValue_T & *iStdDevValue,* const stdair::Yield_T & *iYield,* stdair::BomRoot & *iBomRoot*)

Definition at line 970 of file RMOL_Service.cpp.

25.38.3.21 void RMOL::RMOL_Service::setOnDForecast (const stdair::AirlineCodeList_T & *iAirlineCodeList,* const stdair::AirlineCode_T & *iAirlineCode,* const stdair::Date_T & *iDepartureDate,* const stdair::AirportCode_T & *iOrigin,* const stdair::AirportCode_T & *iDestination,* const stdair::CabinCode_T & *iCabinCode,* const stdair::ClassCodeList_T & *iClassCodeList,* const stdair::MeanValue_T & *iMeanValue,* const stdair::StdDevValue_T & *iStdDevValue,* const stdair::Yield_T & *iYield,* stdair::BomRoot & *iBomRoot*)

Definition at line 1034 of file RMOL_Service.cpp.

25.38.3.22 void RMOL::RMOL_Service::resetDemandInformation (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1151 of file RMOL_Service.cpp.

Referenced by optimise(), optimiseOnDUsingAdvancedRMCooperation(), and optimiseOnDUsingRMCooperation().

25.38.3.23 void RMOL::RMOL_Service::resetDemandInformation (const stdair::DateTime_T & *iRMEventTime,* const stdair::Inventory & *iInventory*)

Definition at line 1177 of file RMOL_Service.cpp.

25.38.3.24 void RMOL::RMOL_Service::projectAggregatedDemandOnLegCabins (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1227 of file RMOL_Service.cpp.

Referenced by optimise().

25.38.3.25 void RMOL::RMOL_Service::projectOnDDemandOnLegCabinsUsingYP (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1332 of file RMOL_Service.cpp.

Referenced by optimise().

25.38.3.26 void RMOL::RMOL_Service::projectOnDDemandOnLegCabinsUsingDA (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1609 of file RMOL_Service.cpp.

25.38.3.27 void RMOL::RMOL_Service::projectOnDDemandOnLegCabinsUsingDYP (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1765 of file RMOL_Service.cpp.

Referenced by optimise(), optimiseOnDUsingAdvancedRMCooperation(), and optimiseOnDUsingRMCooperation().

25.38.3.28 void RMOL::RMOL_Service::projectOnDDemandOnLegCabinsUsingDYP (const stdair::DateTime_T & *iRMEventTime,* const stdair::Inventory & *iInventory*)

Definition at line 1791 of file RMOL_Service.cpp.

25.38.3.29 void RMOL::RMOL_Service::optimiseOnD (const stdair::DateTime_T & *iRMEventTime*)

Optimiser

Definition at line 1431 of file RMOL_Service.cpp.

References RMOL::Optimiser::optimiseUsingOnDForecast().

Referenced by optimise().

25.38.3.30 void RMOL::RMOL_Service::optimiseOnDUsingRMCooperation (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1907 of file RMOL_Service.cpp.

References RMOL::Optimiser::optimiseUsingOnDForecast(), projectOnDDemandOnLegCabinsUsingDYP(), and resetDemandInformation().

Referenced by optimise().

25.38.3.31 void RMOL::RMOL_Service::optimiseOnDUsingAdvancedRMCooperation (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1967 of file RMOL_Service.cpp.

References RMOL::Optimiser::optimiseUsingOnDForecast(), projectOnDDemandOnLegCabinsUsingDYP(), resetDemandInformation(), and updateBidPrice().

Referenced by optimise().

25.38.3.32 void RMOL::RMOL_Service::updateBidPrice (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1480 of file RMOL_Service.cpp.

Referenced by optimise(), and optimiseOnDUsingAdvancedRMCooperation().

25.38.3.33 void RMOL::RMOL_Service::updateBidPrice (const stdair::FlightDate & *iFlightDate,* stdair::BomRoot & *iBomRoot*)

Definition at line 1528 of file RMOL_Service.cpp.

25.38.3.34 std::string RMOL::RMOL_Service::jsonExport (const stdair::AirlineCode_T & , const stdair::FlightNumber_T & , const stdair::Date_T & *iDepartureDate*) const

Recursively dump, in the returned string and in JSON format, the flight-date corresponding to the parameters given as input.

Parameters

const	stdair::AirlineCode_T& Airline code of the flight to dump.
const	stdair::FlightNumber_T& Flight number of the flight to dump.
const	stdair::Date_T& Departure date of a flight to dump.

Returns

std::string Output string in which the BOM tree is JSON-ified.

25.38.3.35 std::string RMOL::RMOL_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.

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

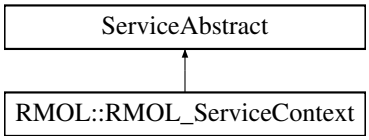
- rmol/RMOL_Service.hpp
- rmol/service/RMOL_Service.cpp

25.39 RMOL::RMOL_ServiceContext Class Reference

Inner class holding the context for the RMOL Service object.

`#include <rmol/service/RMOL_ServiceContext.hpp>`

Inheritance diagram for RMOL::RMOL_ServiceContext:



Friends

- class RMOL_Service
- class FacRmolServiceContext

25.39.1 Detailed Description

Inner class holding the context for the RMOL Service object.

Definition at line 29 of file RMOL_ServiceContext.hpp.

25.39.2 Friends And Related Function Documentation

25.39.2.1 friend class RMOL_Service [friend]

The RMOL_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 35 of file RMOL_ServiceContext.hpp.

25.39.2.2 friend class FacRmolServiceContext [friend]

Definition at line 36 of file RMOL_ServiceContext.hpp.

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

- rmol/service/RMOL_ServiceContext.hpp
- rmol/service/RMOL_ServiceContext.cpp

25.40 RMOL::SegmentSnapshotTableHelper Class Reference

`#include <rmol/bom/SegmentSnapshotTableHelper.hpp>`

Static Public Member Functions

- static stdair::NbOfSegments_T getNbOfSegmentAlreadyPassedThisDTD (const stdair::SegmentSnapshot↩ Table &, const stdair::DTD_T &, const stdair::Date_T &)
- static bool hasPassedThisDTD (const stdair::SegmentCabin &, const stdair::DTD_T &, const stdair::Date_T &)

25.40.1 Detailed Description

Class representing the actual business functions for an airline guillotine block.

Definition at line 23 of file SegmentSnapshotTableHelper.hpp.

25.40.2 Member Function Documentation

25.40.2.1 stdair::NbOfSegments_T RMOL::SegmentSnapshotTableHelper::getNbOfSegmentAlreadyPassedThisDTD (const stdair::SegmentSnapshotTable & *iGB*, const stdair::DTD_T & *iDTD*, const stdair::Date_T & *iCurrentDate*) [static]

Retrieve the number of similar segments which already passed the given DTD.

Definition at line 20 of file SegmentSnapshotTableHelper.cpp.

References hasPassedThisDTD().

Referenced by RMOL::QForecasting::forecast(), RMOL::HybridForecasting::forecast(), RMOL::OldQFF::forecast(), RMOL::BasedForecasting::forecast(), and RMOL::Utilities::getNbOfDepartedSimilarSegments().

25.40.2.2 bool RMOL::SegmentSnapshotTableHelper::hasPassedThisDTD (const stdair::SegmentCabin & *iSegmentCabin*, const stdair::DTD_T & *iDTD*, const stdair::Date_T & *iCurrentDate*) [static]

Check if the given segment has passed the given DTD.

Definition at line 42 of file SegmentSnapshotTableHelper.cpp.

Referenced by getNbOfSegmentAlreadyPassedThisDTD().

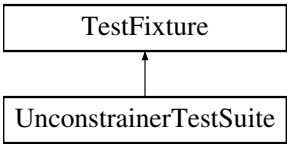
The documentation for this class was generated from the following files:

- rmol/bom/SegmentSnapshotTableHelper.hpp
- rmol/bom/SegmentSnapshotTableHelper.cpp

25.41 UnconstrainerTestSuite Class Reference

`#include <test/rmol/UnconstrainerTestSuite.hpp>`

Inheritance diagram for UnconstrainerTestSuite:



Public Member Functions

- void testUnconstrainingByEM ()
- UnconstrainerTestSuite ()

Protected Attributes

- std::stringstream _describeKey

25.41.1 Detailed Description

Definition at line 6 of file UnconstrainerTestSuite.hpp.

25.41.2 Constructor & Destructor Documentation

25.41.2.1 UnconstrainerTestSuite::UnconstrainerTestSuite ( )

Constructor.

25.41.3 Member Function Documentation

25.41.3.1 void UnconstrainerTestSuite::testUnconstrainingByEM ( )

Test data unconstraining by Expectation Maximization.

25.41.4 Member Data Documentation

25.41.4.1 std::stringstream UnconstrainerTestSuite::_describeKey [protected]

Definition at line 19 of file UnconstrainerTestSuite.hpp.

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

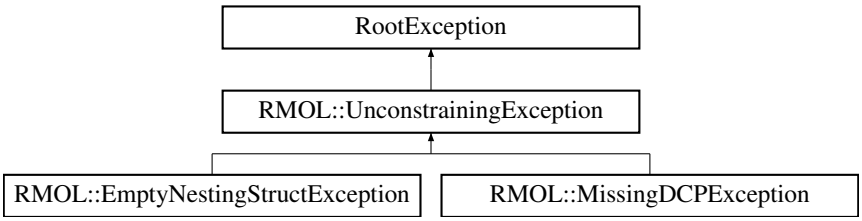
- test/rmol/UnconstrainerTestSuite.hpp

25.42 RMOL::UnconstrainingException Class Reference

Unconstraining-related exception.

`#include <rmol/RMOL_Types.hpp>`

Inheritance diagram for RMOL::UnconstrainingException:



Public Member Functions

- UnconstrainingException (const std::string &iWhat)

25.42.1 Detailed Description

Unconstraining-related exception.

Definition at line 42 of file RMOL_Types.hpp.

25.42.2 Constructor & Destructor Documentation

25.42.2.1 RMOL::UnconstrainingException::UnconstrainingException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 45 of file RMOL_Types.hpp.

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

- rmol/RMOL_Types.hpp

25.43 RMOL::Utilities Class Reference

`#include <rmol/bom/Utilities.hpp>`

Static Public Member Functions

- static void computeDistributionParameters (const stdair::UncDemVector_T &, stdair::MeanValue_T &, stdair::StdDevValue_T &)
- static stdair::DCPList_T buildRemainingDCPList (const stdair::DTD_T &)
- static stdair::DCPList_T buildPastDCPList (const stdair::DTD_T &)
- static stdair::NbOfSegments_T getNbOfDepartedSimilarSegments (const stdair::SegmentCabin &, const stdair::Date_T &)
- static stdair::BookingClassSellUpCurveMap_T computeSellUpFactorCurves (const stdair::FRAT5Curve_T &, const stdair::BookingClassList_T &)
- static stdair::BookingClassDispatchingCurveMap_T computeDispatchingFactorCurves (const stdair::FRA↩T5Curve_T &, const stdair::BookingClassList_T &)
- static void dispatchDemandForecast (const stdair::BookingClassDispatchingCurveMap_T &, const stdair::↩MeanValue_T &, const stdair::StdDevValue_T &, const stdair::DTD_T &)
- static void dispatchDemandForecastForFA (const stdair::BookingClassSellUpCurveMap_T &, const stdair↩::MeanValue_T &, const stdair::StdDevValue_T &, const stdair::DTD_T &)

25.43.1 Detailed Description

Class holding helper methods.

Definition at line 20 of file Utilities.hpp.

25.43.2 Member Function Documentation

25.43.2.1 void RMOL::Utilities::computeDistributionParameters (const stdair::UncDemVector_T & *iVector,* stdair::MeanValue_T & *ioMean,* stdair::StdDevValue_T & *ioStdDev*) [static]

Compute the mean and the standard deviation from a set of samples.

Definition at line 27 of file Utilities.cpp.

Referenced by RMOL::QForecasting::forecast(), RMOL::HybridForecasting::forecast(), RMOL::OldQFF::forecast(), and RMOL::BasedForecasting::forecast().

25.43.2.2 stdair::DCPList_T RMOL::Utilities::buildRemainingDCPList (const stdair::DTD_T & *iDTD*) [static]

Build the list of remaining DCP's for the segment-date.

Definition at line 59 of file Utilities.cpp.

25.43.2.3 stdair::DCPList_T RMOL::Utilities::buildPastDCPList (const stdair::DTD_T & *iDTD*) [static]

Build the list of past DCP's for the segment-date.

Definition at line 84 of file Utilities.cpp.

25.43.2.4 stdair::NbOfSegments_T RMOL::Utilities::getNbOfDepartedSimilarSegments (const stdair::SegmentCabin & *iSegmentCabin,* const stdair::Date_T & *iEventDate*) [static]

Retrieve the number of departed similar segments.

Definition at line 104 of file Utilities.cpp.

References RMOL::SegmentSnapshotTableHelper::getNbOfSegmentAlreadyPassedThisDTD().

25.43.2.5 stdair::BookingClassSellUpCurveMap_T RMOL::Utilities::computeSellUpFactorCurves (const stdair::FRAT5Curve_T & *iFRAT5Curve,* const stdair::BookingClassList_T & *iBCList*) [static]

Precompute the sell-up factors for each class and each DCP.

Definition at line 116 of file Utilities.cpp.

Referenced by RMOL::QForecasting::forecast(), and RMOL::OldQFF::forecast().

25.43.2.6 stdair::BookingClassDispatchingCurveMap_T RMOL::Utilities::computeDispatchingFactorCurves (const stdair::FRAT5Curve_T & *iFRAT5Curve,* const stdair::BookingClassList_T & *iBCList*) [static]

Precompute the dispatching factors for each class and each DCP.

Definition at line 177 of file Utilities.cpp.

Referenced by RMOL::QForecasting::forecast().

25.43.2.7 void RMOL::Utilities::dispatchDemandForecast (const stdair::BookingClassDispatchingCurveMap_T & *iBCDispatchingCurveMap,* const stdair::MeanValue_T & *iMean,* const stdair::StdDevValue_T & *iStdDev,* const stdair::DTD_T & *iCurrentDCP*) [static]

Dispatching the demand forecast to all classes.

Definition at line 253 of file Utilities.cpp.

Referenced by RMOL::QForecasting::forecast().

25.43.2.8 void RMOL::Utilities::dispatchDemandForecastForFA (const stdair::BookingClassSellUpCurveMap_T & *iBCSellUpCurveMap,* const stdair::MeanValue_T & *iMean,* const stdair::StdDevValue_T & *iStdDev,* const stdair::DTD_T & *iCurrentDCP*) [static]

Dispatching the demand forecast to all classes for FA.

Definition at line 286 of file Utilities.cpp.

Referenced by RMOL::QForecasting::forecast().

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

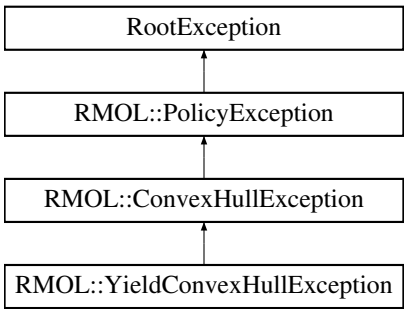
- rmol/bom/Utilities.hpp
- rmol/bom/Utilities.cpp

25.44 RMOL::YieldConvexHullException Class Reference

Yield convex hull exception.

`#include <rmol/RMOL_Types.hpp>`

Inheritance diagram for RMOL::YieldConvexHullException:



Public Member Functions

- YieldConvexHullException (const std::string &iWhat)

25.44.1 Detailed Description

Yield convex hull exception.

Definition at line 123 of file RMOL_Types.hpp.

25.44.2 Constructor & Destructor Documentation

25.44.2.1 RMOL::YieldConvexHullException::YieldConvexHullException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 126 of file RMOL_Types.hpp.

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

- rmol/RMOL_Types.hpp

26 File Documentation

```
#include <rmol/basic/BasConst_General.hpp>
#include <rmol/basic/BasConst_RMOL_Service.hpp>
```

Namespaces

- RMOL

Variables

- const stdair::AirlineCode_T RMOL::DEFAULT_RMOL_SERVICE_AIRLINE_CODE = "BA"
- const double RMOL::DEFAULT_RMOL_SERVICE_CAPACITY = 1.0
- const int RMOL::DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION = 10000
- const int RMOL::DEFAULT_PRECISION = 10
- const double RMOL::DEFAULT_EPSILON = 0.0001
- const double RMOL::DEFAULT_STOPPING_CRITERION = 0.01
- const double RMOL::DEFAULT_INITIALIZER_DOUBLE_NEGATIVE = -10.0

26.16 BasConst.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 #include <rmol/basic/BasConst_General.hpp>
```

```
00005 #include <rmol/basic/BasConst_RMOL_Service.hpp>
00006
00007 namespace RMOL {
00008
00010   const stdair::AirlineCode_T DEFAULT_RMOL_SERVICE_AIRLINE_CODE = "BA";
00011
00013   const double DEFAULT_RMOL_SERVICE_CAPACITY = 1.0;
00014
00017   const int DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION = 10000;
00018
00022   const int DEFAULT_PRECISION = 10;
00023
00025   const double DEFAULT_EPSILON = 0.0001;
00026
00028   const double DEFAULT_STOPPING_CRITERION = 0.01;
00029
00031   const double DEFAULT_INITIALIZER_DOUBLE_NEGATIVE = -10.0;
00032 }
```

26.17 rmol/basic/BasConst_General.hpp File Reference

```
#include <stdair/stdair_types.hpp>
```

Namespaces

- RMOL

26.18 BasConst_General.hpp

```
00001 #ifndef __RMOL_BAS_BASCONST_GENERAL_HPP
00002 #define __RMOL_BAS_BASCONST_GENERAL_HPP
00003
00004 // ////////////////////////////////////////////////////////////////////
00005 // Import section
00006 // ////////////////////////////////////////////////////////////////////
00007 // StdAir
00008 #include <stdair/stdair_types.hpp>
00009
00010 namespace RMOL {
00011
00014   extern const int DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION;
00015
00018   extern const int DEFAULT_PRECISION;
00019
00021   extern const double DEFAULT_EPSILON;
00022
00024   extern const double DEFAULT_STOPPING_CRITERION;
00025
00027   extern const double DEFAULT_INITIALIZER_DOUBLE_NEGATIVE;
00028 }
00029 #endif // __RMOL_BAS_BASCONST_GENERAL_HPP
```

26.19 rmol/basic/BasConst_RMOL_Service.hpp File Reference

```
#include <vector>
#include <stdair/stdair_basic_types.hpp>
#include <rmol/RMOL_Types.hpp>
```

Namespaces

- RMOL

26.20 BasConst_RMOL_Service.hpp

```
00001 #ifndef __RMOL_BAS_BASCONST_RMOL_SERVICE_HPP
```

```
00002 #define __RMOL_BAS_BASCONST_RMOL_SERVICE_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <vector>
00009 // StdAir
00010 #include <stdair/stdair_basic_types.hpp>
00011 // RMOL
00012 #include <rmol/RMOL_Types.hpp>
00013
00014 namespace RMOL {
00015
00017   extern const stdair::AirlineCode_T DEFAULT_RMOL_SERVICE_AIRLINE_CODE;
00018
00020   extern const double DEFAULT_RMOL_SERVICE_CAPACITY;
00021
00022 }
00023 #endif // __RMOL_BAS_BASCONST_RMOL_SERVICE_HPP
```

26.21 rmol/batches/rmol.cpp File Reference

```
#include <cassert>
#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <boost/date_time/posix_time/posix_time.hpp>
#include <boost/date_time/gregorian/gregorian.hpp>
#include <boost/program_options.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/basic/BasConst_General.hpp>
#include <rmol/RMOL_Service.hpp>
#include <rmol/config/rmol-paths.hpp>
```

Functions

- const std::string K_RMOL_DEFAULT_LOG_FILENAME ("rmol.log")
- const std::string K_RMOL_DEFAULT_INPUT_FILENAME (STDAIR_SAMPLE_DIR"/rm01.csv")
- template<class T >
  std::ostream & operator<< (std::ostream &os, const std::vector< T > &v)
- int readConfiguration (int argc, char *argv[], int &ioRandomDraws, double &ioCapacity, short &ioMethod, bool &ioIsBuiltin, std::string &ioInputFilename, std::string &ioLogFilename)
- void optimise (RMOL::RMOL_Service &rmolService, const short &iMethod, const int &iRandomDraws)
- int main (int argc, char *argv[])

Variables

- const bool K_RMOL_DEFAULT_BUILT_IN_INPUT = false
- const int K_RMOL_DEFAULT_RANDOM_DRAWS = RMOL::DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION
- const double K_RMOL_DEFAULT_CAPACITY = 500.0
- const short K_RMOL_DEFAULT_METHOD = 0
- const int K_RMOL_EARLY_RETURN_STATUS = 99

26.21.1 Function Documentation

26.21.1.1 const std::string K_RMOL_DEFAULT_LOG_FILENAME ("rmol.log")

Default name and location for the log file.

Referenced by readConfiguration().

26.21.1.2 const std::string K_RMOL_DEFAULT_INPUT_FILENAME (STDAIR_SAMPLE_DIR"/rm01.csv")

Default name and location for the (CSV) input file.

Referenced by readConfiguration().

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

Definition at line 48 of file rmol.cpp.

26.21.1.4 int readConfiguration (int *argc,* char * *argv[ ],* int & *ioRandomDraws,* double & *ioCapacity,* short & *ioMethod,* bool & *ioIsBuiltin,* std::string & *ioInputFilename,* std::string & *ioLogFilename*)

Read and parse the command line options.

Definition at line 58 of file rmol.cpp.

References K_RMOL_DEFAULT_BUILT_IN_INPUT, K_RMOL_DEFAULT_CAPACITY, K_RMOL_DEFAULT_IN↩PUT_FILENAME(), K_RMOL_DEFAULT_LOG_FILENAME(), K_RMOL_DEFAULT_METHOD, K_RMOL_DEFA↩ULT_RANDOM_DRAWS, and K_RMOL_EARLY_RETURN_STATUS.

Referenced by main().

26.21.1.5 void optimise (RMOL::RMOL_Service & *rmolService,* const short & *iMethod,* const int & *iRandomDraws*)

Definition at line 168 of file rmol.cpp.

References RMOL::RMOL_Service::heuristicOptimisationByEmsr(), RMOL::RMOL_Service::heuristicOptimisation↩ByEmsrA(), RMOL::RMOL_Service::heuristicOptimisationByEmsrB(), RMOL::RMOL_Service::optimalOptimisation↩ByDP(), and RMOL::RMOL_Service::optimalOptimisationByMCIntegration().

Referenced by main().

26.21.1.6 int main (int *argc,* char * *argv[ ]*)

Definition at line 205 of file rmol.cpp.

References RMOL::RMOL_Service::buildSampleBom(), K_RMOL_EARLY_RETURN_STATUS, optimise(), RM↩OL::RMOL_Service::parseAndLoad(), and readConfiguration().

26.21.2 Variable Documentation

26.21.2.1 const bool K_RMOL_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/–input option.

Definition at line 24 of file rmol.cpp.

Referenced by readConfiguration().

26.21.2.2 const int K_RMOL_DEFAULT_RANDOM_DRAWS = RMOL::DEFAULT_NUMBER_OF_DRAWS_FOR_MC_↩SIMULATION

Default number of random draws to be generated (best if over 100).

Definition at line 30 of file rmol.cpp.

Referenced by readConfiguration().

26.21.2.3 const double K_RMOL_DEFAULT_CAPACITY = 500.0

Default value for the capacity of the resource (e.g., a flight cabin).

Definition at line 33 of file rmol.cpp.

Referenced by readConfiguration().

26.21.2.4 const short K_RMOL_DEFAULT_METHOD = 0

Default name and location for the Revenue Management method to be used.

- 0 = Monte-Carlo
- 1 = Dynamic Programming
- 2 = EMSR
- 3 = EMSR-a
- 4 = EMSR-b

Definition at line 44 of file rmol.cpp.

Referenced by readConfiguration().

26.21.2.5 const int K_RMOL_EARLY_RETURN_STATUS = 99

Early return status (so that it can be differentiated from an error).

Definition at line 55 of file rmol.cpp.

Referenced by main(), and readConfiguration().

26.22 rmol.cpp

```
00001 // STL
00002 #include <cassert>
00003 #include <iostream>
00004 #include <sstream>
00005 #include <fstream>
00006 #include <string>
00007 // Boost (Extended STL)
00008 #include <boost/date_time/posix_time/posix_time.hpp>
00009 #include <boost/date_time/gregorian/gregorian.hpp>
00010 #include <boost/program_options.hpp>
00011 // StdAir
00012 #include <stdair/service/Logger.hpp>
00013 // RMOL
00014 #include <rmol/basic/BasConst_General.hpp>
00015 #include <rmol/RMOL_Service.hpp>
00016 #include <rmol/config/rmol-paths.hpp>
00017
00018 // //////// Constants //////
00020 const std::string K_RMOL_DEFAULT_LOG_FILENAME ("rmol.log");
00021
00024 const bool K_RMOL_DEFAULT_BUILT_IN_INPUT = false;
00025
00027 const std::string K_RMOL_DEFAULT_INPUT_FILENAME (STDAIR_SAMPLE_DIR "/rm01.csv"
      );
00028
00030 const int K_RMOL_DEFAULT_RANDOM_DRAWS =
      RMOL::DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION;
00031
00033 const double K_RMOL_DEFAULT_CAPACITY = 500.0;
00034
00044 const short K_RMOL_DEFAULT_METHOD = 0;
00045
00046 // ///////// Parsing of Options & Configuration /////////
00047 // A helper function to simplify the main part.
00048 template<class T> std::ostream& operator<< (std::ostream& os,
00049                                             const std::vector<T>& v) {
00050   std::copy (v.begin(), v.end(), std::ostream_iterator<T> (std::cout, " "));
00051   return os;
00052 }
00053
00055 const int K_RMOL_EARLY_RETURN_STATUS = 99;
00056
```

```
00058 int readConfiguration(int argc, char* argv[],
00059                       int& ioRandomDraws, double& ioCapacity,
00060                       short& ioMethod, bool& ioIsBuiltin,
00061                       std::string& ioInputFilename, std::string& ioLogFilename){
00062
00063   // Default for the built-in input
00064   ioIsBuiltin = K_RMOL_DEFAULT_BUILT_IN_INPUT;
00065
00066   // Declare a group of options that will be allowed only on command line
00067   boost::program_options::options_description generic ("Generic options");
00068   generic.add_options()
00069     ("prefix", "print installation prefix")
00070     ("version,v", "print version string")
00071     ("help,h", "produce help message");
00072
00073   // Declare a group of options that will be allowed both on command
00074   // line and in config file
00075   boost::program_options::options_description config ("Configuration");
00076   config.add_options()
00077     ("draws,d",
00078      boost::program_options::value<int>(&ioRandomDraws)->default_value(
      K_RMOL_DEFAULT_RANDOM_DRAWS),
00079      "Number of to-be-generated random draws")
00080     ("capacity,c",
00081      boost::program_options::value<double>(&ioCapacity)->default_value(
      K_RMOL_DEFAULT_CAPACITY),
00082      "Resource capacity (e.g., for a flight leg)")
00083     ("method,m",
00084      boost::program_options::value<short>(&ioMethod)->default_value(
      K_RMOL_DEFAULT_METHOD),
00085      "Revenue Management method to be used (0 = Monte-Carlo, 1 = Dynamic Programming, 2 = EMSR, 3 = EMSR-a,
       4 = EMSR-b)")
00086     ("builtin,b",
00087      "The cabin set up can be either built-in or parsed from an input file. That latter must then be given
       with the -i/--input option")
00088     ("input,i",
00089      boost::program_options::value< std::string >(&ioInputFilename)->default_value(
      K_RMOL_DEFAULT_INPUT_FILENAME),
00090      "(CSV) input file for the demand distribution parameters and resource (leg-cabin) capacities")
00091     ("log,l",
00092      boost::program_options::value< std::string >(&ioLogFilename)->default_value(
      K_RMOL_DEFAULT_LOG_FILENAME),
00093      "Filename for the logs")
00094     ;
00095
00096   // Hidden options, will be allowed both on command line and
00097   // in config file, but will not be shown to the user.
00098   boost::program_options::options_description hidden ("Hidden options");
00099   hidden.add_options()
00100     ("copyright",
00101      boost::program_options::value< std::vector<std::string> >(),
00102      "Show the copyright (license)");
00103
00104   boost::program_options::options_description cmdline_options;
00105   cmdline_options.add(generic).add(config).add(hidden);
00106
00107   boost::program_options::options_description config_file_options;
00108   config_file_options.add(config).add(hidden);
00109
00110   boost::program_options::options_description visible ("Allowed options");
00111   visible.add(generic).add(config);
00112
00113   boost::program_options::positional_options_description p;
00114   p.add ("copyright", -1);
00115
00116   boost::program_options::variables_map vm;
00117   boost::program_options::
00118     store (boost::program_options::command_line_parser (argc, argv).
00119            options (cmdline_options).positional(p).run(), vm);
00120
00121   std::ifstream ifs ("rmol.cfg");
00122   boost::program_options::store (parse_config_file (ifs, config_file_options),
00123                                  vm);
00124   boost::program_options::notify (vm);
00125
00126   if (vm.count ("help")) {
00127     std::cout << visible << std::endl;
00128     return K_RMOL_EARLY_RETURN_STATUS;
00129   }
00130
00131   if (vm.count ("version")) {
00132     std::cout << PACKAGE_NAME << ", version " << PACKAGE_VERSION << std::endl;
00133     return K_RMOL_EARLY_RETURN_STATUS;
00134   }
00135
00136   if (vm.count ("prefix")) {
00137     std::cout << "Installation prefix: " << PREFIXDIR << std::endl;
```

```
00138     return K_RMOL_EARLY_RETURN_STATUS;
00139   }
00140
00141   if (vm.count ("builtin")) {
00142     ioIsBuiltin = true;
00143   }
00144   const std::string isBuiltinStr = (ioIsBuiltin == true)?"yes":"no";
00145   std::cout << "The BOM should be built-in? " << isBuiltinStr << std::endl;
00146
00147   if (ioIsBuiltin == false) {
00148     if (vm.count ("input")) {
00149       ioInputFilename = vm["input"].as< std::string >();
00150       std::cout << "Input filename is: " << ioInputFilename << std::endl;
00151     }
00152   }
00153
00154   if (vm.count ("log")) {
00155     ioLogFilename = vm["log"].as< std::string >();
00156     std::cout << "Log filename is: " << ioLogFilename << std::endl;
00157   }
00158
00159   std::cout << "The number of random draws is: " << ioRandomDraws << std::endl;
00160   std::cout << "The resource capacity is: " << ioCapacity << std::endl;
00161   std::cout << "The optimisation method is: " << ioMethod << std::endl;
00162   std::cout << std::endl;
00163
00164   return 0;
00165 }
00166
00167 // ////////////////////////////////////////////////////
00168 void optimise (RMOL::RMOL_Service& rmolService,
00169                const short& iMethod, const int& iRandomDraws) {
00170
00171   switch (iMethod) {
00172   case 0: {
00173     // Calculate the optimal protections by the Monte Carlo
00174     // Integration approach
00175     rmolService.optimalOptimisationByMCIntegration (iRandomDraws);
00176     break;
00177   }
00178   case 1: {
00179     // Calculate the optimal protections by DP.
00180     rmolService.optimalOptimisationByDP ();
00181     break;
00182   }
00183   case 2: {
00184     // Calculate the Bid-Price Vector by EMSR
00185     rmolService.heuristicOptimisationByEmsr ();
00186     break;
00187   }
00188   case 3: {
00189     // Calculate the protections by EMSR-a
00190     rmolService.heuristicOptimisationByEmsrA ();
00191     break;
00192   }
00193   case 4: {
00194     // Calculate the protections by EMSR-b
00195     rmolService.heuristicOptimisationByEmsrB ();
00196     break;
00197   }
00198   default: {
00199     rmolService.optimalOptimisationByMCIntegration (iRandomDraws);
00200   }
00201   }
00202 }
00203
00204 // ///////// M A I N ////////////
00205 int main (int argc, char* argv[]) {
00206
00207   // Number of random draws to be generated (best if greater than 100)
00208   int lRandomDraws = 0;
00209
00210   // Cabin Capacity (it must be greater then 100 here)
00211   double lCapacity = 0.0;
00212
00213   // Methods of optimisation (0 = Monte-Carlo, 1 = Dynamic Programming,
00214   // 2 = EMSR, 3 = EMSR-a, 4 = EMSR-b)
00215   short lMethod = 0;
00216
00217   // Built-in
00218   bool isBuiltin;
00219
00220   // Input file name
00221   std::string lInputFilename;
00222
00223   // Output log File
00224   std::string lLogFilename;
```

```
00225
00226   // Call the command-line option parser
00227   const int lOptionParserStatus =
00228     readConfiguration (argc, argv, lRandomDraws, lCapacity, lMethod,
00229                        isBuiltin, lInputFilename, lLogFilename);
00230
00231   if (lOptionParserStatus == K_RMOL_EARLY_RETURN_STATUS) {
00232     return 0;
00233   }
00234
00235   // Set the log parameters
00236   std::ofstream logOutputFile;
00237   // Open and clean the log outputfile
00238   logOutputFile.open (lLogFilename.c_str());
00239   logOutputFile.clear();
00240
00241   // Initialise the log stream
00242   const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);
00243
00244   // Initialise the RMOL service
00245   RMOL::RMOL_Service rmolService (lLogParams);
00246
00247   if (isBuiltin == true) {
00248     // DEBUG
00249     STDAIR_LOG_DEBUG ("No input file has been given."
00250                       "A sample BOM tree will therefore be built.");
00251
00252     // Build a sample BOM tree
00253     rmolService.buildSampleBom();
00254
00255   } else {
00256     // DEBUG
00257     STDAIR_LOG_DEBUG ("RMOL will parse " << lInputFilename
00258                       << " and build the corresponding BOM tree.");
00259
00260     //
00261     rmolService.parseAndLoad (lCapacity, lInputFilename);
00262   }
00263
00264   // Launch the optimisation
00265   optimise (rmolService, lMethod, lRandomDraws);
00266
00267   //
00268   logOutputFile.close();
00269
00270   return 0;
00271 }
```

26.23 rmol/bom/BucketHolderTypes.hpp File Reference

```
#include <list>
#include <map>
#include <stdair/stdair_basic_types.hpp>
```

Namespaces

- RMOL

Typedefs

- typedef std::list< BucketHolder * > RMOL::BucketHolderList_T

26.24 BucketHolderTypes.hpp

```
00001 #ifndef __RMOL_BUCKETHOLDERTYPES_HPP
00002 #define __RMOL_BUCKETHOLDERTYPES_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <list>
```

```
00009 #include <map>
00010 // STDAIR
00011 #include <stdair/stdair_basic_types.hpp>
00012
00013 namespace RMOL {
00014
00016   class BucketHolder;
00017
00019   typedef std::list<BucketHolder*> BucketHolderList_T;
00020
00023   typedef std::map<const stdair::MapKey_T, BucketHolder*>;
00024 }
00025 #endif // __RMOL_BUCKETHOLDERTYPES_HPP
```

26.25 rmol/bom/DistributionParameterList.hpp File Reference

```
#include <list>
#include <rmol/field/FldDistributionParameters.hpp>
```

Namespaces

- RMOL

Typedefs

- typedef std::list< FldDistributionParameters > RMOL::DistributionParameterList_T

26.26 DistributionParameterList.hpp

```
00001 #ifndef __RMOL_DISTRIBUTIONPARAMETERLIST_HPP
00002 #define __RMOL_DISTRIBUTIONPARAMETERLIST_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <list>
00009 // RMOL
00010 #include <rmol/field/FldDistributionParameters.hpp>
00011
00012 namespace RMOL {
00013
00016   typedef std::list<FldDistributionParameters> DistributionParameterList_T;
00017
00018 }
00019 #endif // __RMOL_DISTRIBUTIONPARAMETERLIST_HPP
```

26.27 rmol/bom/DPOptimiser.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <vector>
#include <cmath>
#include <boost/math/distributions/normal.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/VirtualClassStruct.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/basic/BasConst_General.hpp>
#include <rmol/bom/DPOptimiser.hpp>
```

Namespaces

- RMOL

26.28 DPOptimiser.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 #include <vector>
00008 #include <cmath>
00009 // Boost Math
00010 #include <boost/math/distributions/normal.hpp>
00011 // StdAir
00012 #include <stdair/bom/LegCabin.hpp>
00013 #include <stdair/bom/VirtualClassStruct.hpp>
00014 #include <stdair/service/Logger.hpp>
00015 // RMOL
00016 #include <rmol/basic/BasConst_General.hpp>
00017 #include <rmol/bom/DPOptimiser.hpp>
00018
00019 namespace RMOL {
00020
00021   // ////////////////////////////////////////////////////////////////////
00022   void DPOptimiser::optimalOptimisationByDP (stdair::LegCabin&
      ioLegCabin) {
00023     // // Number of classes/buckets: n
00024     // const short nbOfClasses = ioBucketHolder.getSize();
00025
00026     // // Number of values of x to compute for each Vj(x).
00027     // const int maxValue = static_cast<int> (iCabinCapacity * DEFAULT_PRECISION);
00028
00029     // // Vector of the Expected Maximal Revenue (Vj).
00030     // std::vector< std::vector<double> > MERVectorHolder;
00031
00032     // // Vector of V_0(x).
00033     // std::vector<double> initialMERVector (maxValue+1, 0.0);
00034     // MERVectorHolder.push_back (initialMERVector);
00035
00036     // // Current cumulative protection level (y_j * DEFAULT_PRECISION).
00037     // // Initialise with y_0 = 0.
00038     // int currentProtection = 0;
00039
00040     // int currentBucketIndex = 1;
00041     // ioBucketHolder.begin();
00042
00043     // while (currentProtection < maxValue && currentBucketIndex < nbOfClasses) {
00044     // //while (currentBucketIndex == 1) {
00045     //   bool protectionChanged = false;
00046     //   double nextProtection = 0.0;
00047     //   std::vector<double> currentMERVector;
00048     //   // double testGradient = 10000;
00049
00050     //   Bucket& currentBucket = ioBucketHolder.getCurrentBucket();
00051     //   const double meanDemand = currentBucket.getMean();
00052     //   const double SDDemand = currentBucket.getStandardDeviation();
00053     //   const double currentYield = currentBucket.getAverageYield();
00054     //   const double errorFactor = 1.0;
00055
00056     //   Bucket& nextBucket = ioBucketHolder.getNextBucket();
00057     //   const double nextYield = nextBucket.getAverageYield();
00058
00059     //   // For x <= currentProtection (y_(j-1)), V_j(x) = V_(j-1)(x).
00060     //   for (int x = 0; x <= currentProtection; ++x) {
00061     //     const double MERValue = MERVectorHolder.at(currentBucketIndex-1).at(x);
00062     //     currentMERVector.push_back (MERValue);
00063     //   }
00064
00065     //   //
00066     //   boost::math::normal lNormalDistribution (meanDemand, SDDemand);
00067
00068     //   // Vector of gaussian pdf values.
00069     //   std::vector<double> pdfVector;
00070     //   for (int s = 0; s <= maxValue - currentProtection; ++s) {
00071     //     const double pdfValue =
00072     //       boost::math::pdf (lNormalDistribution, s/DEFAULT_PRECISION);
00073     //     pdfVector.push_back (pdfValue);
00074     //   }
00075
00076     //   // Vector of gaussian cdf values.
```

```
00077     //   std::vector<double> cdfVector;
00078     //   for (int s = 0; s <= maxValue - currentProtection; ++s) {
00079     //     const double cdfValue =
00080     //       boost::math::cdf (boost::math::complement (lNormalDistribution,
00081     //                                                  s/DEFAULT_PRECISION));
00082     //     cdfVector.push_back (cdfValue);
00083     //   }
00084
00085     //   // Compute V_j(x) for x > currentProtection (y_(j-1)).
00086     //   for (int x = currentProtection + 1; x <= maxValue; ++x) {
00087     //     const double lowerBound = static_cast<double> (x - currentProtection);
00088
00089     //     // Compute the first integral in the V_j(x) formulation (see
00090     //     // the memo of Jerome Contant).
00091     //     const double power1 =
00092     //       - 0.5 * meanDemand * meanDemand / (SDDemand * SDDemand);
00093     //     const double e1 = std::exp (power1);
00094     //     const double power2 =
00095     //       - 0.5 * (lowerBound / DEFAULT_PRECISION - meanDemand)
00096     //       * (lowerBound / DEFAULT_PRECISION - meanDemand)
00097     //       / (SDDemand * SDDemand);
00098     //     const double e2 = std::exp (power2);
00099
00100    //     const double cdfValue0 =
00101    //       boost::math::cdf (boost::math::complement (lNormalDistribution,
00102    //                                                  0.0));
00103    //     const double cdfValue1 =
00104    //       boost::math::cdf(boost::math::complement(lNormalDistribution,
00105    //                                                lowerBound/DEFAULT_PRECISION));
00106    //     const double integralResult1 = currentYield
00107    //       * ((e1 - e2) * SDDemand / sqrt (2 * 3.14159265)
00108    //          + meanDemand * (cdfValue0 - cdfValue1));
00109
00110    //     double integralResult2 = 0.0;
00111
00112    //     for (int s = 0; s < lowerBound; ++s) {
00113    //       const double partialResult =
00114    //         2 * MERVectorHolder.at(currentBucketIndex-1).at(x-s)
00115    //         * pdfVector.at(s);
00116
00117    //       integralResult2 += partialResult;
00118    //     }
00119    //     integralResult2 -= MERVectorHolder.at(currentBucketIndex-1).at(x) *
00120    //       pdfVector.at(0);
00121
00122    //     const int intLowerBound = static_cast<int>(lowerBound);
00123    //     integralResult2 +=
00124    //       MERVectorHolder.at(currentBucketIndex-1).at(x - intLowerBound) *
00125    //       pdfVector.at(intLowerBound);
00126
00127    //     integralResult2 /= 2 * DEFAULT_PRECISION;
00128    //     /*
00129    //     for (int s = 0; s < lowerBound; ++s) {
00130    //       const double partialResult =
00131    //         (MERVectorHolder.at(currentBucketIndex-1).at(x-s) +
00132    //          MERVectorHolder.at(currentBucketIndex-1).at(x-s-1)) *
00133    //         (cdfVector.at(s+1) - cdfVector.at(s)) / 2;
00134    //       integralResult2 += partialResult;
00135    //     }
00136    //     */
00137    //     const double firstElement = integralResult1 + integralResult2;
00138
00139    //     // Compute the second integral in the V_j(x) formulation (see
00140    //     // the memo of Jerome Contant).
00141    //     const double constCoefOfSecondElement =
00142    //       currentYield * lowerBound / DEFAULT_PRECISION
00143    //       + MERVectorHolder.at(currentBucketIndex-1).at(currentProtection);
00144
00145    //     const double secondElement = constCoefOfSecondElement
00146    //       * boost::math::cdf(boost::math::complement(lNormalDistribution,
00147    //                                                  lowerBound/DEFAULT_PRECISION));
00148
00149    //     const double MERValue = (firstElement + secondElement) / errorFactor;
00150
00151
00152    //     assert (currentMERVector.size() > 0);
00153    //     const double lastMERValue = currentMERVector.back();
00154
00155    //     const double currentGradient =
00156    //       (MERValue - lastMERValue) * DEFAULT_PRECISION;
00157
00158    //     //assert (currentGradient >= 0);
00159    //     if (currentGradient < -0) {
00160    //       std::ostringstream ostr;
00161    //       ostr << currentGradient  << std::endl
00162    //            << "x = " << x << std::endl
00163    //            << "class: " << currentBucketIndex << std::endl;
```

```
00164     //       STDAIR_LOG_DEBUG (ostr.str());
00165     //     }
00166     //
00167     //     /*
00168     //     assert (currentGradient <= testGradient);
00169     //     testGradient = currentGradient;
00170     //     */
00171     //     if (protectionChanged == false && currentGradient <= nextYield) {
00172     //       nextProtection = x - 1;
00173     //       protectionChanged = true;
00174     //     }
00175     //
00176     //      if (protectionChanged == true && currentGradient > nextYield) {
00177     //       protectionChanged = false;
00178     //     }
00179     //
00180     //     if (protectionChanged == false && x == maxValue) {
00181     //       nextProtection = maxValue;
00182     //     }
00183     //
00184     //     currentMERVector.push_back (MERValue);
00185     //   }
00186     //
00187     //   // DEBUG
00188     //   STDAIR_LOG_DEBUG ("Vmaxindex = " << currentMERVector.back());
00189     //
00190     //   MERVectorHolder.push_back (currentMERVector);
00191     //
00192     //   const double realProtection = nextProtection / DEFAULT_PRECISION;
00193     //   const double bookingLimit = iCabinCapacity - realProtection;
00194     //
00195     //   currentBucket.setCumulatedProtection (realProtection);
00196     //   nextBucket.setCumulatedBookingLimit (bookingLimit);
00197     //
00198     //   currentProtection = static_cast<int> (std::floor (nextProtection));
00199     //
00200     //   ioBucketHolder.iterate();
00201     //   ++currentBucketIndex;
00202     // }
00203   }
00204
00205 }
```

26.29 rmol/bom/DPOptimiser.hpp File Reference

```
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::DPOptimiser

Namespaces

- stdair

  Forward declarations.
- RMOL

26.30 DPOptimiser.hpp

```
00001 #ifndef __RMOL_BOM_DPOPTIMISER_HPP
00002 #define __RMOL_BOM_DPOPTIMISER_HPP
00003
00004 // //////////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////////
00007 // RMOL
00008 #include <rmol/RMOL_Types.hpp>
00009
00011 namespace stdair {
00012   class LegCabin;
00013 }
00014
```

```
00015 namespace RMOL {
00017   class DPOptimiser {
00018   public:
00019
00025     static void optimalOptimisationByDP (stdair::LegCabin&);
00026
00030     static double cdfGaussianQ (const double, const double);
00031   };
00032 }
00033 #endif // __RMOL_BOM_DPOPTIMISER_HPP
```

26.31 rmol/bom/EMDetruncator.cpp File Reference

```
#include <iostream>
#include <cmath>
#include <vector>
#include <cassert>
#include <stdair/stdair_basic_types.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/HistoricalBookingHolder.hpp>
#include <rmol/bom/EMDetruncator.hpp>
```

Namespaces

- RMOL

26.32 EMDetruncator.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <iostream>
00006 #include <cmath>
00007 #include <vector>
00008 #include <cassert>
00009 // StdAir
00010 #include <stdair/stdair_basic_types.hpp>
00011 #include <stdair/service/Logger.hpp>
00012 // RMOL
00013 #include <rmol/bom/HistoricalBookingHolder.hpp>
00014 #include <rmol/bom/EMDetruncator.hpp>
00015
00016 namespace RMOL {
00017
00018   // ////////////////////////////////////////////////////////////////////
00019   void EMDetruncator::unconstrain
00020   (HistoricalBookingHolder& ioHistoricalBookingHolder) {
00021
00022     // Number of flights.
00023     const short lNbOfFlights =
00024       ioHistoricalBookingHolder.getNbOfFlights();
00025
00026     // Number of uncensored booking data.
00027     const short lNbOfUncensoredData =
00028       ioHistoricalBookingHolder.getNbOfUncensoredData();
00029
00030     if (lNbOfUncensoredData > 1) {
00031       // Number of uncensored bookings.
00032       const stdair::NbOfBookings_T lNbOfUncensoredBookings =
00033         ioHistoricalBookingHolder.getNbOfUncensoredBookings();
00034
00035       const double lMeanOfUncensoredBookings =
00036         static_cast<double>(lNbOfUncensoredBookings/lNbOfUncensoredData);
00037
00038       const double lStdDevOfUncensoredBookings =
00039         ioHistoricalBookingHolder.getUncensoredStandardDeviation
00040         (lMeanOfUncensoredBookings, lNbOfUncensoredData);
00041
00042       std::vector<bool> toBeUnconstrained =
00043         ioHistoricalBookingHolder.getListOfToBeUnconstrainedFlags();
00044
00045       double lDemandMean = lMeanOfUncensoredBookings;
```

```
00046       double lStdDev = lStdDevOfUncensoredBookings;
00047
00048       // DEBUG
00049       // STDAIR_LOG_DEBUG ("mean: " << lDemandMean << ", std: " << lStdDev);
00050
00051       if (lStdDev != 0) {
00052         bool stopUnconstraining = false;
00053         while (stopUnconstraining == false) {
00054           stopUnconstraining = true;
00055
00056           for (short i = 0; i < lNbOfFlights; ++i) {
00057             if (toBeUnconstrained.at(i) == true) {
00058               // Get the unconstrained demand of the (i+1)-th flight.
00059               const stdair::NbOfBookings_T demand =
00060                 ioHistoricalBookingHolder.getUnconstrainedDemand (i);
00061               //STDAIR_LOG_DEBUG ("demand: " << demand);
00062
00063               // Execute the Expectation step.
00064               const stdair::NbOfBookings_T expectedDemand =
00065                 ioHistoricalBookingHolder.
00066                 calculateExpectedDemand (lDemandMean, lStdDev, i, demand);
00067               //STDAIR_LOG_DEBUG ("expected: " << expectedDemand);
00068
00069               double absDiff =
00070                 static_cast<double>(expectedDemand - demand);
00071
00072               if (absDiff < 0) {
00073                 absDiff = - absDiff;
00074               }
00075               if (absDiff < 0.001) {
00076                 toBeUnconstrained.at (i) = false;
00077               }
00078               else {
00079                 stopUnconstraining = false;
00080               }
00081
00082               ioHistoricalBookingHolder.setUnconstrainedDemand (expectedDemand,
00083                                                                 i);
00084             }
00085           }
00086
00087           if (stopUnconstraining == false) {
00088             lDemandMean = ioHistoricalBookingHolder.getDemandMean();
00089             lStdDev =
00090               ioHistoricalBookingHolder.getStandardDeviation (lDemandMean);
00091           }
00092         }
00093       }
00094     }
00095
00096   }
00097 }
```

26.33 rmol/bom/EMDetruncator.hpp File Reference

Classes

- class RMOL::EMDetruncator

Namespaces

- RMOL

26.34 EMDetruncator.hpp

```
00001 #ifndef __RMOL_BOM_EMDETRUNCATOR_HPP
00002 #define __RMOL_BOM_EMDETRUNCATOR_HPP
00003
00004 // //////////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////////
00007 namespace RMOL {
00008   // Forward declarations.
00009   struct HistoricalBookingHolder;
00010
00012   class EMDetruncator {
00013   public:
```

```
00016     static void unconstrain (HistoricalBookingHolder&);
00017   };
00018 }
00019 #endif // __RMOL_BOM_EMDETRUNCATOR_HPP
```

26.35 rmol/bom/Emsr.cpp File Reference

```
#include <assert.h>
#include <iostream>
#include <cmath>
#include <list>
#include <algorithm>
#include <stdair/stdair_rm_types.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/VirtualClassStruct.hpp>
#include <rmol/bom/Emsr.hpp>
#include <rmol/bom/EmsrUtils.hpp>
```

Namespaces

- RMOL

26.36 Emsr.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // C
00005 #include <assert.h>
00006 // STL
00007 #include <iostream>
00008 #include <cmath>
00009 #include <list>
00010 #include <algorithm>
00011 // StdAir
00012 #include <stdair/stdair_rm_types.hpp>
00013 #include <stdair/bom/LegCabin.hpp>
00014 #include <stdair/bom/VirtualClassStruct.hpp>
00015 // RMOL
00016 #include <rmol/bom/Emsr.hpp>
00017 #include <rmol/bom/EmsrUtils.hpp>
00018
00019 namespace RMOL {
00020   // ////////////////////////////////////////////////////////////////////
00021   void Emsr::heuristicOptimisationByEmsrA (stdair::LegCabin& ioLegCabin)
   {
00022     stdair::VirtualClassList_T& lVirtualClassList =
00023       ioLegCabin.getVirtualClassList ();
00024     const stdair::CabinCapacity_T& lCabinCapacity =
00025       ioLegCabin.getOfferedCapacity();
00026
00032     stdair::VirtualClassList_T::iterator itVC =lVirtualClassList.begin();
00033     assert (itVC != lVirtualClassList.end());
00034
00035     stdair::VirtualClassStruct& lFirstVC = *itVC;
00036     lFirstVC.setCumulatedBookingLimit (lCabinCapacity);
00037     ++itVC;
00038     for (; itVC != lVirtualClassList.end(); ++itVC) {
00039       stdair::VirtualClassStruct& lNextVC = *itVC;
00040
00041       // Initialise the protection for class/bucket j.
00042       stdair::ProtectionLevel_T lProtectionLevel = 0.0;
00043
00044       for(stdair::VirtualClassList_T::iterator itHigherVC =
00045             lVirtualClassList.begin(); itHigherVC != itVC; ++itHigherVC) {
00046         stdair::VirtualClassStruct& lHigherVC = *itHigherVC;
00047         const double lPartialProtectionLevel =
00048           EmsrUtils::computeProtectionLevel (lHigherVC, lNextVC);
00049         lProtectionLevel += lPartialProtectionLevel;
00050       }
00051       stdair::VirtualClassList_T::iterator itCurrentVC = itVC; --itCurrentVC;
00052       stdair::VirtualClassStruct& lCurrentVC = *itCurrentVC;
```

```
00053       lCurrentVC.setCumulatedProtection (lProtectionLevel);
00054
00055       // Compute the booking limit for the class/bucket j+1 (can be negative).
00056       const double lBookingLimit = lCabinCapacity - lProtectionLevel;
00057
00058       // Set the booking limit for class/bucket j+1.
00059       lNextVC.setCumulatedBookingLimit (lBookingLimit);
00060     }
00061   }
00062
00063   // ////////////////////////////////////////////////////////////////////
00064   void Emsr::heuristicOptimisationByEmsrB (stdair::LegCabin& ioLegCabin)
      {
00065     stdair::VirtualClassList_T& lVirtualClassList =
00066       ioLegCabin.getVirtualClassList ();
00067     const stdair::CabinCapacity_T& lCabinCapacity =
00068       ioLegCabin.getOfferedCapacity();
00069
00075     stdair::VirtualClassList_T::iterator itVC =lVirtualClassList.begin();
00076     assert (itVC != lVirtualClassList.end());
00077
00078     stdair::VirtualClassStruct& lFirstVC = *itVC;
00079     lFirstVC.setCumulatedBookingLimit (lCabinCapacity);
00080     ++itVC;
00081     stdair::VirtualClassStruct lAggregatedVC = lFirstVC;
00082     for (; itVC != lVirtualClassList.end(); ++itVC) {
00083       stdair::VirtualClassStruct& lNextVC = *itVC;
00084
00085       // Compute the protection level for the aggregated class/bucket
00086       // using the Little-Wood formular.
00087       const stdair::ProtectionLevel_T lProtectionLevel =
00088         EmsrUtils::computeProtectionLevel (lAggregatedVC, lNextVC);
00089
00090       // Set the protection level for class/bucket j.
00091       stdair::VirtualClassList_T::iterator itCurrentVC = itVC; --itCurrentVC;
00092       stdair::VirtualClassStruct& lCurrentVC = *itCurrentVC;
00093       lCurrentVC.setCumulatedProtection (lProtectionLevel);
00094
00095       // Compute the booking limit for class/bucket j+1 (can be negative).
00096       const double lBookingLimit = lCabinCapacity - lProtectionLevel;
00097
00098       // Set the booking limit for class/bucket j+1.
00099       lNextVC.setCumulatedBookingLimit (lBookingLimit);
00100
00101       // Compute the aggregated class/bucket of classes/buckets 1,..,j.
00102       EmsrUtils::computeAggregatedVirtualClass (lAggregatedVC,
      lNextVC);
00103
00104     }
00105   }
00106
00107   // ////////////////////////////////////////////////////////////////////
00108   void Emsr::heuristicOptimisationByEmsr (stdair::LegCabin& ioLegCabin) {
00109     stdair::VirtualClassList_T& lVirtualClassList =
00110       ioLegCabin.getVirtualClassList ();
00111     const stdair::CabinCapacity_T& lCapacity = ioLegCabin.getOfferedCapacity();
00112     ioLegCabin.emptyBidPriceVector();
00113     stdair::BidPriceVector_T& lBidPriceVector =
00114       ioLegCabin.getBidPriceVector();
00115
00116     // Cabin capacity in integer.
00117     const int lCabinCapacity = static_cast<const int> (lCapacity);
00118
00119     // List of all EMSR values.
00120     stdair::EmsrValueList_T lEmsrValueList;
00121
00127     for (stdair::VirtualClassList_T::iterator itVC = lVirtualClassList.begin();
00128          itVC != lVirtualClassList.end(); ++itVC) {
00129       stdair::VirtualClassStruct& lCurrentVC = *itVC;
00130       for (int k = 1; k <= lCabinCapacity; ++k) {
00131         const double emsrValue = EmsrUtils::computeEmsrValue (k, lCurrentVC);
00132         lEmsrValueList.push_back(emsrValue);
00133       }
00134     }
00135
00136     // Sort the EMSR values from high to low.
00137     std::sort(lEmsrValueList.rbegin(), lEmsrValueList.rend());
00138
00139     // Sanity check
00140     const int lEmsrValueListSize = lEmsrValueList.size();
00141     assert (lEmsrValueListSize >= lCabinCapacity);
00142
00143     // Copy the EMSR sorted values to the BPV.
00144     stdair::EmsrValueList_T::const_iterator itCurrentValue =
00145       lEmsrValueList.begin();
00146     for (int j = 0; j < lCabinCapacity; ++j, ++itCurrentValue) {
00147       const double lBidPrice = *itCurrentValue;
```

```
00148       lBidPriceVector.push_back (lBidPrice);
00149     }
00150     lEmsrValueList.clear();
00151
00152     // Build the protection levels and booking limits.
00153     if (lVirtualClassList.size() > 1) {
00154       int lCapacityIndex = 0;
00155       for (stdair::VirtualClassList_T::iterator itVC =lVirtualClassList.begin();
00156            itVC != lVirtualClassList.end();) {
00157         stdair::VirtualClassStruct& lCurrentVC = *itVC;
00158         if (itVC != lVirtualClassList.end()) {
00159           ++itVC;
00160         }
00161         stdair::VirtualClassStruct& lNextVC = *itVC;
00162         const stdair::Yield_T lNextYield = lNextVC.getYield();
00163         while ((lCapacityIndex < lCabinCapacity)
00164                && (lBidPriceVector.at(lCapacityIndex) > lNextYield)) {
00165           ++lCapacityIndex;
00166         }
00167         lCurrentVC.setCumulatedProtection (lCapacityIndex);
00168         lNextVC.setCumulatedBookingLimit (lCapacity - lCapacityIndex);
00169       }
00170     }
00171   }
00172
00173 }
```

26.37 rmol/bom/Emsr.hpp File Reference

```
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::Emsr

Namespaces

- stdair

  Forward declarations.
- RMOL

26.38 Emsr.hpp

```
00001 #ifndef __RMOL_EMSR_HPP
00002 #define __RMOL_EMSR_HPP
00003
00004 // ////////////////////////////////////////////////////////////////////
00005 // Import section
00006 // ////////////////////////////////////////////////////////////////////
00007 // RMOL
00008 #include <rmol/RMOL_Types.hpp>
00009
00011 namespace stdair {
00012   class LegCabin;
00013 }
00014
00015 namespace RMOL {
00016
00018   class Emsr {
00019   public:
00031     static void heuristicOptimisationByEmsr (stdair::LegCabin&);
00032
00037     static void heuristicOptimisationByEmsrA (stdair::LegCabin&);
00038
00043     static void heuristicOptimisationByEmsrB (stdair::LegCabin&);
00044
00045   };
00046 }
00047 #endif // __RMOL_EMSR_HPP
```

26.39 rmol/bom/EmsrUtils.cpp File Reference

```
#include <cassert>
#include <cmath>
#include <boost/math/distributions/normal.hpp>
#include <stdair/stdair_maths_types.hpp>
#include <stdair/bom/VirtualClassStruct.hpp>
#include <rmol/bom/EmsrUtils.hpp>
#include <rmol/basic/BasConst_General.hpp>
```

Namespaces

- RMOL

26.40 EmsrUtils.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <cmath>
00007 // Boost Math
00008 #include <boost/math/distributions/normal.hpp>
00009 // StdAir
00010 #include <stdair/stdair_maths_types.hpp>
00011 #include <stdair/bom/VirtualClassStruct.hpp>
00012 // RMOL
00013 #include <rmol/bom/EmsrUtils.hpp>
00014 #include <rmol/basic/BasConst_General.hpp>
00015
00016 namespace RMOL {
00017   // ////////////////////////////////////////////////////////////////////
00018   void EmsrUtils::computeAggregatedVirtualClass
00019   (stdair::VirtualClassStruct& ioAggregatedVirtualClass,
00020    stdair::VirtualClassStruct& ioCurrentVirtualClass) {
00021     // Retrieve the demand mean, demand standard deviation and average
00022     // yield of the classes/buckets.
00023     const stdair::MeanValue_T lAggregatedMean =
00024       ioAggregatedVirtualClass.getMean();
00025     const stdair::MeanValue_T lCurrentMean = ioCurrentVirtualClass.getMean();
00026     const stdair::StdDevValue_T lAggregatedSD =
00027       ioAggregatedVirtualClass.getStdDev();
00028     const stdair::StdDevValue_T lCurrentSD = ioCurrentVirtualClass.getStdDev();
00029     const stdair::Yield_T lAggregatedYield =
00030       ioAggregatedVirtualClass.getYield();
00031     const stdair::Yield_T lCurrentYield = ioCurrentVirtualClass.getYield();
00032
00033     // Compute the new demand mean, new demand standard deviation and
00034     // new average yield for the new aggregated class/bucket.
00035     const stdair::MeanValue_T lNewMean = lAggregatedMean + lCurrentMean;
00036     const stdair::StdDevValue_T lNewSD =
00037       std::sqrt (lAggregatedSD*lAggregatedSD + lCurrentSD*lCurrentSD);
00038     stdair::Yield_T lNewYield = lCurrentYield;
00039     if (lNewMean > 0) {
00040       lNewYield = (lAggregatedYield*lAggregatedMean +
00041                    lCurrentYield*lCurrentMean) / lNewMean;
00042     }
00043     // Set the new yield range for the new aggregated class/bucket.
00044     ioAggregatedVirtualClass.setYield(lNewYield);
00045
00046     // Set the new demand for the new aggregated class/bucket.
00047     ioAggregatedVirtualClass.setMean (lNewMean);
00048     ioAggregatedVirtualClass.setStdDev (lNewSD);
00049   }
00050
00051   // ////////////////////////////////////////////////////////////////////
00052   const stdair::ProtectionLevel_T EmsrUtils::
00053   computeProtectionLevel (stdair::VirtualClassStruct& ioAggregatedVirtualClass,
00054                           stdair::VirtualClassStruct& ioNextVirtualClass) {
00055     // Retrive the mean & standard deviation of the aggregated
00056     // class/bucket and the average yield of all the two
00057     // classes/buckets.
00058     const stdair::MeanValue_T lMean = ioAggregatedVirtualClass.getMean();
00059     const stdair::StdDevValue_T lSD = ioAggregatedVirtualClass.getStdDev();
00060     const stdair::Yield_T lAggreatedYield = ioAggregatedVirtualClass.getYield();
```

```
00061     const stdair::Yield_T lNextYield = ioNextVirtualClass.getYield();
00062     assert (lAggreatedYield != 0);
00063
00064     // Compute the yield ratio between the higher bucket and the current one
00065     const double lYieldRatio = lNextYield / lAggreatedYield;
00066
00070     boost::math::normal lNormalDistribution (lMean, lSD);
00071     const stdair::ProtectionLevel_T lProtection =
00072       boost::math::quantile (boost::math::complement (lNormalDistribution,
00073                                                       lYieldRatio));
00074
00075     return lProtection;
00076   }
00077
00078   // ////////////////////////////////////////////////////////////////////
00079   const double EmsrUtils::
00080   computeEmsrValue (double iCapacity,
00081                     stdair::VirtualClassStruct& ioVirtualClass){
00082     // Retrieve the average yield, mean and standard deviation of the
00083     // demand of the class/bucket.
00084     const stdair::MeanValue_T lMean = ioVirtualClass.getMean();
00085     const stdair::StdDevValue_T lSD = ioVirtualClass.getStdDev();
00086     const stdair::Yield_T lYield = ioVirtualClass.getYield();
00087
00088     // Compute the EMSR value = lYield * Pr (demand >= iCapacity).
00089     boost::math::normal lNormalDistribution (lMean, lSD);
00090     const double emsrValue =
00091       lYield * boost::math::cdf (boost::math::complement (lNormalDistribution,
00092                                                           iCapacity));
00093
00094     return emsrValue;
00095   }
00096 }
```

26.41 rmol/bom/EmsrUtils.hpp File Reference

```
#include <stdair/stdair_inventory_types.hpp>
```

Classes

- class RMOL::EmsrUtils

Namespaces

- stdair

  Forward declarations.
- RMOL

26.42 EmsrUtils.hpp

```
00001 #ifndef __RMOL_EMSRUTILS_HPP
00002 #define __RMOL_EMSRUTILS_HPP
00003
00004 // ////////////////////////////////////////////////////////////////////
00005 // Import section
00006 // ////////////////////////////////////////////////////////////////////
00007 // StdAir
00008 #include <stdair/stdair_inventory_types.hpp>
00009
00010 // Forward declarations.
00011 namespace stdair {
00012   struct VirtualClassStruct;
00013 }
00014
00015 namespace RMOL {
00016
00019   class EmsrUtils {
00020   public:
00023     static void computeAggregatedVirtualClass (stdair::VirtualClassStruct&,
00024                                                stdair::VirtualClassStruct&);
00025
00027     static const stdair::ProtectionLevel_T computeProtectionLevel (
```

```
      stdair::VirtualClassStruct&, stdair::VirtualClassStruct&);
00028
00030     static const double computeEmsrValue (double, stdair::VirtualClassStruct&);
00031   };
00032 }
00033 #endif // __RMOL_EMSRUTILS_HPP
```

26.43 rmol/bom/HistoricalBooking.cpp File Reference

```
#include <sstream>
#include <cassert>
#include <iomanip>
#include <iostream>
#include <rmol/bom/HistoricalBooking.hpp>
```

Namespaces

- RMOL

26.44 HistoricalBooking.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <sstream>
00006 #include <cassert>
00007 #include <iomanip>
00008 #include <iostream>
00009 // RMOL
00010 #include <rmol/bom/HistoricalBooking.hpp>
00011
00012 namespace RMOL {
00013
00014   // ////////////////////////////////////////////////////////////////////
00015   HistoricalBooking::HistoricalBooking () :
00016     _numberOfBookings (0.0),_unconstrainedDemand (0.0), _flag (false) {
00017   }
00018
00019   // ////////////////////////////////////////////////////////////////////
00020   HistoricalBooking::
00021   HistoricalBooking (const stdair::NbOfBookings_T iNbOfBookings,
00022                      const stdair::Flag_T iFlag)
00023     : _numberOfBookings (iNbOfBookings),
00024       _unconstrainedDemand (iNbOfBookings), _flag (iFlag) {
00025   }
00026
00027   // ////////////////////////////////////////////////////////////////////
00028   HistoricalBooking::HistoricalBooking
00029   (const HistoricalBooking& iHistoricalBooking) :
00030     _numberOfBookings (iHistoricalBooking.getNbOfBookings()),
00031     _unconstrainedDemand (iHistoricalBooking.getUnconstrainedDemand()),
00032     _flag (iHistoricalBooking.getFlag()) {
00033   }
00034
00035   // ////////////////////////////////////////////////////////////////////
00036   HistoricalBooking::~HistoricalBooking() {
00037   }
00038
00039   // ////////////////////////////////////////////////////////////////////
00040   void HistoricalBooking::setParameters
00041   (const stdair::NbOfBookings_T iNbOfBookings, const stdair::Flag_T iFlag) {
00042     _numberOfBookings = iNbOfBookings;
00043     _unconstrainedDemand = iNbOfBookings;
00044     _flag = iFlag;
00045   }
00046
00047   // ////////////////////////////////////////////////////////////////////
00048   const std::string HistoricalBooking::describe() const {
00049     std::ostringstream ostr;
00050     ostr << "Struct of hitorical booking, unconstrained demand and flag of "
00051          << "censorship for a FlightDate/Class.";
00052
00053     return ostr.str();
```

```
00054   }
00055
00056   // ////////////////////////////////////////////////////////////////////
00057   void HistoricalBooking::toStream (std::ostream& ioOut) const {
00058     const stdair::NbOfBookings_T bj = getNbOfBookings();
00059     const stdair::NbOfBookings_T uj = getUnconstrainedDemand();
00060     const stdair::Flag_T fj = getFlag();
00061     ioOut << std::fixed << std::setprecision (2)
00062           << bj << "; " << uj << "; " << fj << std::endl;
00063   }
00064
00065   // ////////////////////////////////////////////////////////////////////
00066   void HistoricalBooking::display () const {
00067     toStream (std::cout);
00068   }
00069 }
```

26.45 rmol/bom/HistoricalBooking.hpp File Reference

```
#include <stdair/stdair_basic_types.hpp>
#include <stdair/basic/StructAbstract.hpp>
```

Classes

- struct RMOL::HistoricalBooking

  Structure keeping track, for a given class, of the number of historical bookings and of the censorship flag.

Namespaces

- RMOL

26.46 HistoricalBooking.hpp

```
00001 #ifndef __RMOL_BOM_HISTORICALBOOKING_HPP
00002 #define __RMOL_BOM_HISTORICALBOOKING_HPP
00003
00004 // ////////////////////////////////////////////////////////////////////
00005 // Import section
00006 // ////////////////////////////////////////////////////////////////////
00007 // StdAir
00008 #include <stdair/stdair_basic_types.hpp>
00009 #include <stdair/basic/StructAbstract.hpp>
00010
00011 namespace RMOL {
00012
00017   struct HistoricalBooking : public stdair::StructAbstract {
00018
00019   public:
00020     // ////////////////// Getters ////////////////////////
00022     const stdair::NbOfBookings_T& getNbOfBookings() const {
00023       return _numberOfBookings;
00024     }
00026     const stdair::NbOfBookings_T& getUnconstrainedDemand() const {
00027       return _unconstrainedDemand;
00028     }
00031     const stdair::Flag_T& getFlag() const {
00032       return _flag;
00033     }
00034
00035   public:
00036     // ////////////////////// Setters ///////////////////////
00038     void setUnconstrainedDemand (const stdair::NbOfBookings_T& iDemand) {
00039       _unconstrainedDemand = iDemand;
00040     }
00041
00043     void setParameters (const stdair::NbOfBookings_T, const stdair::Flag_T);
00044
00045   public:
00046     // ///////// Display Methods //////////
00052     void toStream (std::ostream& ioOut) const;
00053
00057     const std::string describe() const;
```

```
00058
00062     void display () const;
00063
00064   public:
00065     // ////////// Constructors and destructor. ////////////
00069     HistoricalBooking (const stdair::NbOfBookings_T, const stdair::Flag_T);
00073     HistoricalBooking();
00077     HistoricalBooking (const HistoricalBooking&);
00078
00082     virtual ~HistoricalBooking();
00083
00084   private:
00085     // ////////////// Attributes ////////////
00089     stdair::NbOfBookings_T _numberOfBookings;
00090
00094     stdair::NbOfBookings_T _unconstrainedDemand;
00095
00099     stdair::Flag_T _flag;
00100   };
00101 }
00102 #endif // __RMOL_BOM_HISTORICALBOOKING_HPP
```

26.47 rmol/bom/HistoricalBookingHolder.cpp File Reference

```
#include <sstream>
#include <iostream>
#include <iomanip>
#include <cmath>
#include <cassert>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/HistoricalBooking.hpp>
#include <rmol/bom/HistoricalBookingHolder.hpp>
```

Namespaces

- RMOL

26.48 HistoricalBookingHolder.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <sstream>
00006 #include <iostream>
00007 #include <iomanip>
00008 #include <cmath>
00009 #include <cassert>
00010 // StdAir
00011 #include <stdair/service/Logger.hpp>
00012 // RMOL
00013 #include <rmol/bom/HistoricalBooking.hpp>
00014 #include <rmol/bom/HistoricalBookingHolder.hpp>
00015
00016 namespace RMOL {
00017
00018   // ////////////////////////////////////////////////////////////////////
00019   HistoricalBookingHolder::HistoricalBookingHolder () {
00020   }
00021
00022   // ////////////////////////////////////////////////////////////////////
00023   HistoricalBookingHolder::~HistoricalBookingHolder () {
00024     _historicalBookingVector.clear();
00025   }
00026
00027   // ////////////////////////////////////////////////////////////////////
00028   const short HistoricalBookingHolder::getNbOfFlights () const {
00029     return _historicalBookingVector.size();
00030   }
00031
00032   // ////////////////////////////////////////////////////////////////////
00033   const short HistoricalBookingHolder::getNbOfUncensoredData
      () const {
```

```
00034     short lResult = 0;
00035     const short lSize = _historicalBookingVector.size();
00036
00037     for (short ite = 0; ite < lSize; ++ite) {
00038       const stdair::Flag_T lFlag = _historicalBookingVector.at(ite).getFlag ();
00039       if (lFlag == false) {
00040         ++ lResult;
00041       }
00042     }
00043
00044     return lResult;
00045   }
00046
00047   // ////////////////////////////////////////////////////////////////////
00048   const stdair::NbOfBookings_T HistoricalBookingHolder::
00049   getNbOfUncensoredBookings () const {
00050     stdair::NbOfBookings_T lResult = 0;
00051     const short lSize = _historicalBookingVector.size();
00052
00053     for (short ite = 0; ite < lSize; ++ite) {
00054       const HistoricalBooking& lHistorialBooking =
00055         _historicalBookingVector.at (ite);
00056       const stdair::Flag_T lFlag = lHistorialBooking.getFlag ();
00057       if (lFlag == false) {
00058         const stdair::NbOfBookings_T& lBooking =
00059           lHistorialBooking.getNbOfBookings ();
00060         lResult += lBooking;
00061       }
00062     }
00063
00064     return lResult;
00065   }
00066
00067   // ////////////////////////////////////////////////////////////////////
00068   const double HistoricalBookingHolder::
00069   getUncensoredStandardDeviation (const double& iMeanOfUncensoredBookings,
00070                                   const short iNbOfUncensoredData) const {
00071
00072     double lResult = 0;
00073     const short lSize = _historicalBookingVector.size();
00074
00075     for (short ite = 0; ite < lSize; ++ite) {
00076       const stdair::Flag_T lFlag = _historicalBookingVector.at(ite).getFlag ();
00077       if (lFlag == false) {
00078         const HistoricalBooking& lHistorialBooking =
00079           _historicalBookingVector.at (ite);
00080
00081         const stdair::NbOfBookings_T& lBooking =
00082           lHistorialBooking.getNbOfBookings ();
00083
00084         lResult += (lBooking - iMeanOfUncensoredBookings)
00085           * (lBooking - iMeanOfUncensoredBookings);
00086       }
00087     }
00088     lResult /= (iNbOfUncensoredData - 1);
00089     lResult = std::sqrt (lResult);
00090
00091     return lResult;
00092   }
00093
00094   // ////////////////////////////////////////////////////////////////////
00095   const double HistoricalBookingHolder::getDemandMean () const {
00096     double lResult = 0;
00097     const short lSize = _historicalBookingVector.size();
00098
00099     for (short ite = 0; ite < lSize; ++ite) {
00100       const HistoricalBooking& lHistorialBooking =
00101         _historicalBookingVector.at(ite);
00102
00103       const stdair::NbOfBookings_T& lDemand =
00104         lHistorialBooking.getUnconstrainedDemand ();
00105
00106       lResult += static_cast<double>(lDemand);
00107     }
00108
00109     lResult /= lSize;
00110
00111     return lResult;
00112   }
00113
00114   // ////////////////////////////////////////////////////////////////////
00115   const double HistoricalBookingHolder::getStandardDeviation
00116   (const double iDemandMean) const {
00117     double lResult = 0;
00118     const short lSize = _historicalBookingVector.size();
00119
00120     for (short ite = 0; ite < lSize; ++ite) {
```

```
00121       const HistoricalBooking& lHistorialBooking =
00122         _historicalBookingVector.at(ite);
00123
00124       const stdair::NbOfBookings_T& lDemand =
00125         lHistorialBooking.getUnconstrainedDemand ();
00126
00127       const double lDoubleDemand = static_cast<double> (lDemand);
00128       lResult += (lDoubleDemand - iDemandMean) * (lDoubleDemand - iDemandMean);
00129     }
00130
00131     lResult /= (lSize - 1);
00132
00133     lResult = std::sqrt (lResult);
00134
00135     return lResult;
00136   }
00137
00138   // ////////////////////////////////////////////////////////////////////
00139   const std::vector<bool> HistoricalBookingHolder::
00140   getListOfToBeUnconstrainedFlags () const {
00141     std::vector<bool> lResult;
00142     const short lSize = _historicalBookingVector.size();
00143
00144     for (short ite = 0; ite < lSize; ++ite) {
00145       const HistoricalBooking& lHistorialBooking =
00146         _historicalBookingVector.at(ite);
00147       const stdair::Flag_T lFlag = lHistorialBooking.getFlag ();
00148       if (lFlag == true) {
00149         lResult.push_back(true);
00150       }
00151       else {
00152         lResult.push_back(false);
00153       }
00154     }
00155
00156     return lResult;
00157   }
00158
00159   // ////////////////////////////////////////////////////////////////////
00160   const stdair::NbOfBookings_T& HistoricalBookingHolder::
00161   getHistoricalBooking (const short i) const {
00162     const HistoricalBooking& lHistorialBooking =
00163       _historicalBookingVector.at(i);
00164     return lHistorialBooking.getNbOfBookings();
00165   }
00166
00167   // ////////////////////////////////////////////////////////////////////
00168   const stdair::NbOfBookings_T& HistoricalBookingHolder::
00169   getUnconstrainedDemand (const short i) const {
00170     const HistoricalBooking& lHistorialBooking =
00171       _historicalBookingVector.at(i);
00172     return lHistorialBooking.getUnconstrainedDemand();
00173   }
00174
00175   // ////////////////////////////////////////////////////////////////////
00176   const stdair::Flag_T& HistoricalBookingHolder::
00177   getCensorshipFlag (const short i) const {
00178     const HistoricalBooking& lHistorialBooking =
00179       _historicalBookingVector.at(i);
00180     return lHistorialBooking.getFlag();
00181   }
00182
00183   // ////////////////////////////////////////////////////////////////////
00184   void HistoricalBookingHolder::setUnconstrainedDemand
00185   (const stdair::NbOfBookings_T& iExpectedDemand, const short i) {
00186     _historicalBookingVector.at(i).setUnconstrainedDemand(iExpectedDemand);
00187   }
00188
00189   // ////////////////////////////////////////////////////////////////////
00190   const stdair::NbOfBookings_T HistoricalBookingHolder::calculateExpectedDemand
00191   (const double iMean, const double iSD,
00192    const short i, const stdair::NbOfBookings_T iDemand) const {
00193
00194     const HistoricalBooking lHistorialBooking =
00195       _historicalBookingVector.at(i);
00196     const double lBooking =
00197       static_cast <double> (lHistorialBooking.getNbOfBookings());
00198     double e, d1, d2;
00199
00200     e = - (lBooking - iMean) * (lBooking - iMean) * 0.625 / (iSD * iSD);
00201     //STDAIR_LOG_DEBUG ("e: " << e);
00202     e = exp (e);
00203     //STDAIR_LOG_DEBUG ("e: " << e);
00204
00205     double s = std::sqrt (1 - e);
00206     //STDAIR_LOG_DEBUG ("s: " << s);
00207
```

```
00208     if (lBooking >= iMean) {
00209       if (e < 0.01) {
00210         return iDemand;
00211       }
00212       d1 = 0.5 * (1 - s);
00213     }
00214     else {
00215       d1 = 0.5 * (1 + s);
00216     }
00217     //STDAIR_LOG_DEBUG ("d1: " << d1);
00218
00219     e = - (lBooking - iMean) * (lBooking - iMean) * 0.5 / (iSD * iSD);
00220     e = exp (e);
00221     d2 = e * iSD / std::sqrt(2 * 3.14159265);
00222     //STDAIR_LOG_DEBUG ("d2: " << d2);
00223
00224     if (d1 == 0) {
00225       return iDemand;
00226     }
00227
00228     const stdair::NbOfBookings_T lDemand =
00229       static_cast<stdair::NbOfBookings_T> (iMean + d2/d1);
00230
00231     return lDemand;
00232   }
00233
00234   // ////////////////////////////////////////////////////////////////////
00235   void HistoricalBookingHolder::addHistoricalBooking
00236   (const HistoricalBooking& iHistoricalBooking) {
00237     _historicalBookingVector.push_back(iHistoricalBooking);
00238   }
00239
00240   // ////////////////////////////////////////////////////////////////////
00241   void HistoricalBookingHolder::toStream (std::ostream& ioOut) const {
00242     const short lSize = _historicalBookingVector.size();
00243
00244     ioOut << "Historical Booking; Unconstrained Demand; Flag" << std::endl;
00245
00246     for (short ite = 0; ite < lSize; ++ite) {
00247       const HistoricalBooking& lHistorialBooking =
00248         _historicalBookingVector.at(ite);
00249
00250       const stdair::NbOfBookings_T& lBooking =
00251         lHistorialBooking.getNbOfBookings();
00252
00253       const stdair::NbOfBookings_T& lDemand =
00254         lHistorialBooking.getUnconstrainedDemand();
00255
00256       const stdair::Flag_T lFlag = lHistorialBooking.getFlag();
00257
00258       ioOut << lBooking << "    "
00259             << lDemand << "    "
00260             << lFlag << std::endl;
00261     }
00262   }
00263
00264   // ////////////////////////////////////////////////////////////////////
00265   const std::string HistoricalBookingHolder::describe() const {
00266     std::ostringstream ostr;
00267     ostr << "Holder of HistoricalBooking structs.";
00268
00269     return ostr.str();
00270   }
00271
00272   // ////////////////////////////////////////////////////////////////////
00273   void HistoricalBookingHolder::display() const {
00274     toStream (std::cout);
00275   }
00276 }
```

26.49 rmol/bom/HistoricalBookingHolder.hpp File Reference

```
#include <iostream>
#include <vector>
#include <stdair/stdair_basic_types.hpp>
#include <stdair/basic/StructAbstract.hpp>
```

Classes

- struct RMOL::HistoricalBookingHolder

Namespaces

- RMOL

Typedefs

- typedef std::vector< HistoricalBooking > RMOL::HistoricalBookingVector_T

26.50 HistoricalBookingHolder.hpp

```
00001 #ifndef __RMOL_BOM_HISTORICALBOOKINGHOLDER_HPP
00002 #define __RMOL_BOM_HISTORICALBOOKINGHOLDER_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <iostream>
00009 #include <vector>
00010 // StdAir
00011 #include <stdair/stdair_basic_types.hpp>
00012 #include <stdair/basic/StructAbstract.hpp>
00013
00014 namespace RMOL {
00016   struct HistoricalBooking;
00017
00019   typedef std::vector<HistoricalBooking> HistoricalBookingVector_T;
00020
00023   struct HistoricalBookingHolder : public stdair::StructAbstract {
00024
00025   public:
00026     // ////// Getters //////
00028     const short getNbOfFlights () const;
00029
00031     const short getNbOfUncensoredData () const;
00032
00034     const stdair::NbOfBookings_T getNbOfUncensoredBookings () const;
00035
00037     const double getUncensoredStandardDeviation
00038     (const double& iMeanOfUncensoredBookings,
00039      const short iNbOfUncensoredData) const;
00040
00042     const double getDemandMean () const;
00043
00045     const double getStandardDeviation (const double) const;
00046
00048     const std::vector<bool> getListOfToBeUnconstrainedFlags() const;
00049
00051     const stdair::NbOfBookings_T& getHistoricalBooking (const short i) const;
00052
00054     const stdair::NbOfBookings_T& getUnconstrainedDemand (const short i) const;
00055
00057     const stdair::Flag_T& getCensorshipFlag (const short i) const;
00058
00060     const stdair::NbOfBookings_T& getUnconstrainedDemandOnFirstElement(
      ) const {
00061       return getUnconstrainedDemand (0);
00062     }
00063
00065     const stdair::NbOfBookings_T calculateExpectedDemand (const double,
00066                                                           const double,
00067                                                           const short,
00068                                                 const stdair::NbOfBookings_T) const;
00069
00071     void setUnconstrainedDemand (const stdair::NbOfBookings_T& iExpectedDemand,
00072                                  const short i);
00073
00075     void addHistoricalBooking (const HistoricalBooking&
      iHistoricalBooking);
00076
00080     void toStream (std::ostream& ioOut) const;
00081
00082     // ///////// Display Methods //////////
```

```
00084     const std::string describe() const;
00085
00087     void display () const;
00088
00090     virtual ~HistoricalBookingHolder();
00091
00092   public:
00095     HistoricalBookingHolder ();
00096
00097   private:
00099     HistoricalBookingVector_T _historicalBookingVector;
00100
00101   protected:
00102   };
00103 }
00104 #endif // __RMOL_BOM_HISTORICALBOOKINGHOLDER_HPP
00105
```

26.51 rmol/bom/MCOptimiser.cpp File Reference

```
#include <cassert>
#include <string>
#include <sstream>
#include <algorithm>
#include <cmath>
#include <stdair/stdair_basic_types.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/bom/VirtualClassStruct.hpp>
#include <stdair/service/Logger.hpp>
#include <stdair/basic/RandomGeneration.hpp>
#include <stdair/basic/BasConst_General.hpp>
#include <rmol/basic/BasConst_General.hpp>
#include <rmol/bom/MCOptimiser.hpp>
```

Namespaces

- RMOL

26.52 MCOptimiser.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <string>
00007 #include <sstream>
00008 #include <algorithm>
00009 #include <cmath>
00010 // StdAir
00011 #include <stdair/stdair_basic_types.hpp>
00012 #include <stdair/bom/BomManager.hpp>
00013 #include <stdair/bom/LegCabin.hpp>
00014 #include <stdair/bom/SegmentCabin.hpp>
00015 #include <stdair/bom/BookingClass.hpp>
00016 #include <stdair/bom/VirtualClassStruct.hpp>
00017 #include <stdair/service/Logger.hpp>
00018 #include <stdair/basic/RandomGeneration.hpp>
00019 #include <stdair/basic/BasConst_General.hpp>
00020 // RMOL
00021 #include <rmol/basic/BasConst_General.hpp>
00022 #include <rmol/bom/MCOptimiser.hpp>
00023
00024 namespace RMOL {
00025
00026   // // ////////////////////////////////////////////////////////////////////
```

```
00027   void MCOptimiser::
00028   optimalOptimisationByMCIntegration (stdair::LegCabin& ioLegCabin) {
00029     // Retrieve the segment-cabin
00030     const stdair::SegmentCabinList_T lSegmentCabinList =
00031       stdair::BomManager::getList<stdair::SegmentCabin> (ioLegCabin);
00032     stdair::SegmentCabinList_T::const_iterator itSC = lSegmentCabinList.begin();
00033     assert (itSC != lSegmentCabinList.end());
00034     const stdair::SegmentCabin* lSegmentCabin_ptr = *itSC;
00035     assert (lSegmentCabin_ptr != NULL);
00036
00037     // Retrieve the class list.
00038     const stdair::BookingClassList_T lBookingClassList =
00039       stdair::BomManager::getList<stdair::BookingClass> (*lSegmentCabin_ptr);
00040
00041     // Retrieve the remaining cabin capacity.
00042     const stdair::Availability_T& lCap = ioLegCabin.getAvailabilityPool();
00043     const int lCapacity = static_cast<const int> (lCap);
00044     const stdair::UnsignedIndex_T lCapacityIndex =
00045       static_cast<const stdair::UnsignedIndex_T> ((lCapacity+abs(lCapacity))/2);
00046
00047     // Retrieve the virtual class list.
00048     stdair::VirtualClassList_T& lVCList = ioLegCabin.getVirtualClassList();
00049
00050     // Parse the virtual class list and compute the protection levels.
00051     stdair::VirtualClassList_T::iterator itCurrentVC = lVCList.begin();
00052     assert (itCurrentVC != lVCList.end());
00053     stdair::VirtualClassList_T::iterator itNextVC = itCurrentVC; ++itNextVC;
00054
00055     // Initialise  the partial sum holder with the demand sample of the first
00056     // virtual class.
00057     stdair::VirtualClassStruct& lFirstVC = *itCurrentVC;
00058     stdair::GeneratedDemandVector_T lPartialSumHolder =
00059       lFirstVC.getGeneratedDemandVector();
00060
00061     // Initialise the booking limit for the first class, which is equal to
00062     // the remaining capacity.
00063     lFirstVC.setCumulatedBookingLimit (lCap);
00064
00065     // Initialise bid price vector with the first element (index 0) equal to
00066     // the highest yield.
00067     ioLegCabin.emptyBidPriceVector();
00068     stdair::BidPriceVector_T& lBPV = ioLegCabin.getBidPriceVector();
00069     //const stdair::Yield_T& y1 = lFirstVC.getYield ();
00070     //lBPV.push_back (y1);
00071     stdair::UnsignedIndex_T idx = 1;
00072
00073     for (; itNextVC != lVCList.end(); ++itCurrentVC, ++itNextVC) {
00074       // Get the yields of the two classes.
00075       stdair::VirtualClassStruct& lCurrentVC = *itCurrentVC;
00076       stdair::VirtualClassStruct& lNextVC = *itNextVC;
00077       const stdair::Yield_T& yj = lCurrentVC.getYield ();
00078       const stdair::Yield_T& yj1 = lNextVC.getYield ();
00079
00080       // Consistency check: the yield/price of a higher class/bucket
00081       // (with the j index lower) must be higher.
00082       assert (yj > yj1);
00083
00084       // Sort the partial sum holder.
00085       std::sort (lPartialSumHolder.begin(), lPartialSumHolder.end());
00086       const stdair::UnsignedIndex_T K = lPartialSumHolder.size ();
00087
00088       // Compute the optimal index lj = floor {[y(j)-y(j+1)]/y(j) . K}
00089       const double ljdouble = std::floor (K * (yj - yj1) / yj);
00090       stdair::UnsignedIndex_T lj =
00091         static_cast<stdair::UnsignedIndex_T> (ljdouble);
00092
00093       // Consistency check.
00094       assert (lj >= 1 && lj < K);
00095
00096       //  The optimal protection: p(j) = 1/2 [S(j,lj) + S(j, lj+1)]
00097       const double sjl = lPartialSumHolder.at (lj - 1);
00098       const double sjlp1 = lPartialSumHolder.at (lj + 1 - 1);
00099       const double pj = (sjl + sjlp1) / 2;
00100
00101       // Set the cumulated protection level for the current class.
00102       lCurrentVC.setCumulatedProtection (pj);
00103       // Set the cumulated booking limit for the next class.
00104       lNextVC.setCumulatedBookingLimit (lCap - pj);
00105
00110       const stdair::UnsignedIndex_T pjint = static_cast<const int> (pj);
00111       stdair::GeneratedDemandVector_T::iterator itLowerBound =
00112         lPartialSumHolder.begin();
00113       for (; idx <= pjint && idx <= lCapacityIndex; ++idx) {
00114         itLowerBound =
00115           std::lower_bound (itLowerBound, lPartialSumHolder.end(), idx);
00116         const stdair::UnsignedIndex_T pos =
00117           itLowerBound - lPartialSumHolder.begin();
```

```
00118
00119         const stdair::BidPrice_T lBP = yj * (K - pos) / K;
00120         lBPV.push_back (lBP);
00121       }
00122
00123       // Update the partial sum holder.
00124       const stdair::GeneratedDemandVector_T& lNextPSH =
00125         lNextVC.getGeneratedDemandVector();
00126       assert (K <= lNextPSH.size());
00127       for (stdair::UnsignedIndex_T i = 0; i < K - lj; ++i) {
00128         lPartialSumHolder.at(i) = lPartialSumHolder.at(i + lj) + lNextPSH.at(i);
00129       }
00130       lPartialSumHolder.resize (K - lj);
00131     }
00132
00137     stdair::VirtualClassStruct& lLastVC = *itCurrentVC;
00138     const stdair::Yield_T& yn = lLastVC.getYield();
00139     stdair::GeneratedDemandVector_T::iterator itLowerBound =
00140       lPartialSumHolder.begin();
00141     for (; idx <= lCapacityIndex; ++idx) {
00142       itLowerBound =
00143         std::lower_bound (itLowerBound, lPartialSumHolder.end(), idx);
00144       const stdair::UnsignedIndex_T pos =
00145         itLowerBound - lPartialSumHolder.begin();
00146       const stdair::UnsignedIndex_T K = lPartialSumHolder.size();
00147       const stdair::BidPrice_T lBP = yn * (K - pos) / K;
00148       lBPV.push_back (lBP);
00149     }
00150   }
00151
00152   // ////////////////////////////////////////////////////////////////////
00153   stdair::GeneratedDemandVector_T MCOptimiser::
00154   generateDemandVector (const stdair::MeanValue_T& iMean,
00155                         const stdair::StdDevValue_T& iStdDev,
00156                         const stdair::NbOfSamples_T& K) {
00157     stdair::GeneratedDemandVector_T oDemandVector;
00158     if (iStdDev > 0) {
00159       stdair::RandomGeneration lGenerator (stdair::DEFAULT_RANDOM_SEED);
00160       for (unsigned int i = 0; i < K; ++i) {
00161         stdair::RealNumber_T lDemandSample =
00162           lGenerator.generateNormal (iMean, iStdDev);
00163         oDemandVector.push_back (lDemandSample);
00164       }
00165     } else {
00166       for (unsigned int i = 0; i < K; ++i) {
00167         oDemandVector.push_back (iMean);
00168       }
00169     }
00170     return oDemandVector;
00171   }
00172
00173   // ////////////////////////////////////////////////////////////////////////
00174   void MCOptimiser::
00175   optimisationByMCIntegration (stdair::LegCabin& ioLegCabin) {
00176     // Number of MC samples
00177     stdair::NbOfSamples_T K = DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION
   ;
00178
00179     const stdair::YieldLevelDemandMap_T& lYieldDemandMap =
00180       ioLegCabin.getYieldLevelDemandMap();
00181     assert (!lYieldDemandMap.empty());
00182
00183     std::ostringstream oStr;
00184     oStr << "Yield list ";
00185     for (stdair::YieldLevelDemandMap_T::const_iterator itYD =
00186           lYieldDemandMap.begin();
00187          itYD != lYieldDemandMap.end(); ++itYD) {
00188       const stdair::Yield_T& y = itYD->first;
00189       oStr << y << " ";
00190     }
00191
00192     STDAIR_LOG_DEBUG (oStr.str());
00193     ioLegCabin.emptyBidPriceVector();
00194     stdair::BidPriceVector_T& lBidPriceVector =
00195       ioLegCabin.getBidPriceVector();
00196     const stdair::Availability_T& lAvailabilityPool =
00197       ioLegCabin.getAvailabilityPool();
00198     // Initialise the minimal bid price to 1.0 (just to avoid problems
00199     // of division by zero).
00200     const stdair::BidPrice_T& lMinBP = 1.0;
00201
00202     stdair::YieldLevelDemandMap_T::const_reverse_iterator itCurrentYD =
00203       lYieldDemandMap.rbegin();
00204     stdair::YieldLevelDemandMap_T::const_reverse_iterator itNextYD = itCurrentYD;
00205     ++itNextYD;
00206
00207     // Initialise the partial sum holder
```

```
00208     stdair::MeanStdDevPair_T lMeanStdDevPair = itCurrentYD->second;
00209     stdair::GeneratedDemandVector_T lPartialSumHolder =
00210       generateDemandVector(lMeanStdDevPair.first, lMeanStdDevPair.second, K);
00211
00212     stdair::UnsignedIndex_T idx = 1;
00213     for (; itNextYD!=lYieldDemandMap.rend(); ++itCurrentYD, ++itNextYD) {
00214       const stdair::Yield_T& yj = itCurrentYD->first;
00215       const stdair::Yield_T& yj1 = itNextYD->first;
00216       // Consistency check: the yield/price of a higher class/bucket
00217       // (with the j index lower) must be higher.
00218       assert (yj > yj1);
00219       // Sort the partial sum holder.
00220       std::sort (lPartialSumHolder.begin(), lPartialSumHolder.end());
00221       // STDAIR_LOG_DEBUG ("Partial sums : max = " << lPartialSumHolder.back()
00222       //                   << " min = " << lPartialSumHolder.front());
00223       K = lPartialSumHolder.size ();
00224       // Compute the optimal index lj = floor {[y(j)-y(j+1)]/y(j) . K}
00225       const double ljdouble = std::floor (K * (yj - yj1) / yj);
00226       stdair::UnsignedIndex_T lj =
00227         static_cast<stdair::UnsignedIndex_T> (ljdouble);
00228       // Consistency check.
00229       assert (lj >= 1 && lj < K);
00230       //  The optimal protection: p(j) = 1/2 [S(j,lj) + S(j, lj+1)]
00231       const double sjl = lPartialSumHolder.at (lj - 1);
00232       const double sjlp1 = lPartialSumHolder.at (lj + 1 - 1);
00233       const double pj = (sjl + sjlp1) / 2;
00238       const stdair::UnsignedIndex_T pjint = static_cast<const int> (pj);
00239       stdair::GeneratedDemandVector_T::iterator itLowerBound =
00240         lPartialSumHolder.begin();
00241       for (; idx <= pjint && idx <= lAvailabilityPool; ++idx) {
00242         itLowerBound =
00243           std::lower_bound (itLowerBound, lPartialSumHolder.end(), idx);
00244         const stdair::UnsignedIndex_T pos =
00245           itLowerBound - lPartialSumHolder.begin();
00246
00247         const stdair::BidPrice_T lBP = yj * (K - pos) / K;
00248         lBidPriceVector.push_back (lBP);
00249       }
00250       // Update the partial sum holder.
00251       lMeanStdDevPair = itNextYD->second;
00252       const stdair::GeneratedDemandVector_T& lNextDV =
00253         generateDemandVector (lMeanStdDevPair.first,
00254                               lMeanStdDevPair.second, K - lj);
00255       for (stdair::UnsignedIndex_T i = 0; i < K - lj; ++i) {
00256         lPartialSumHolder.at(i) = lPartialSumHolder.at(i + lj) + lNextDV.at(i);
00257       }
00258       lPartialSumHolder.resize (K - lj);
00259     }
00268     // STDAIR_LOG_DEBUG ("Partial sums : max = " << lPartialSumHolder.back()
00269     //                   << " min = " << lPartialSumHolder.front());
00270
00271     std::sort (lPartialSumHolder.begin(), lPartialSumHolder.end());
00272     const stdair::Yield_T& yn = itCurrentYD->first;
00273     stdair::GeneratedDemandVector_T::iterator itLowerBound =
00274       lPartialSumHolder.begin();
00275     K = lPartialSumHolder.size();
00276
00277     bool lMinBPReached = false;
00278     for (; idx <= lAvailabilityPool; ++idx) {
00279       itLowerBound =
00280         std::lower_bound (itLowerBound, lPartialSumHolder.end(), idx);
00281
00282       if (!lMinBPReached) {
00283         const stdair::UnsignedIndex_T pos =
00284           itLowerBound - lPartialSumHolder.begin();
00285         stdair::BidPrice_T lBP = yn * (K - pos) / K;
00286
00287         if (lBP < lMinBP) {
00288           lBP = lMinBP; lMinBPReached = true;
00289         }
00290
00291         lBidPriceVector.push_back (lBP);
00292
00293       } else {
00294         lBidPriceVector.push_back (lMinBP);
00295       }
00296
00297     }
00298
00299     // Updating the bid price values
00300     ioLegCabin.updatePreviousBidPrice();
00301     ioLegCabin.setCurrentBidPrice (lBidPriceVector.back());
00302
00303     // Compute and display the bid price variation after optimisation
00304     const stdair::BidPrice_T lPreviousBP = ioLegCabin.getPreviousBidPrice();
00305     stdair::BidPrice_T lNewBP = ioLegCabin.getCurrentBidPrice();
00306     // Check
```

```
00307     assert (lPreviousBP != 0);
00308     stdair::BidPrice_T lBidPriceDelta = lNewBP - lPreviousBP;
00309
00310     double lBidPriceVariation = 100*lBidPriceDelta/lPreviousBP;
00311
00312     STDAIR_LOG_DEBUG ("Bid price: previous value " << lPreviousBP
00313                       << ", new value " << lNewBP
00314                       << ", variation " << lBidPriceVariation << " %"
00315                       <<", BPV size " << lBidPriceVector.size());
00316   }
00317
00318 }
```

26.53 rmol/bom/MCOptimiser.hpp File Reference

```
#include <rmol/RMOL_Types.hpp>
#include <stdair/stdair_maths_types.hpp>
#include <stdair/stdair_rm_types.hpp>
```

Classes

- class RMOL::MCOptimiser

Namespaces

- stdair

  Forward declarations.
- RMOL

26.54 MCOptimiser.hpp

```
00001 #ifndef __RMOL_BOM_MCUTILS_HPP
00002 #define __RMOL_BOM_MCUTILS_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // RMOL
00008 #include <rmol/RMOL_Types.hpp>
00009 #include <stdair/stdair_maths_types.hpp>
00010 #include <stdair/stdair_rm_types.hpp>
00011
00012 // Forward declarations.
00013 namespace stdair {
00014   class LegCabin;
00015 }
00016
00017 namespace RMOL {
00019   class MCOptimiser {
00020   public:
00021
00030     static void optimalOptimisationByMCIntegration (stdair::LegCabin&);
00031
00035     static stdair::GeneratedDemandVector_T
00036     generateDemandVector (const stdair::MeanValue_T&,
00037                           const stdair::StdDevValue_T&,
00038                           const stdair::NbOfSamples_T&);
00039
00040     static void optimisationByMCIntegration (stdair::LegCabin&);
00041
00042   };
00043 }
00044 #endif // __RMOL_BOM_MCUTILS_HPP
```

26.55 rmol/bom/old/DemandGeneratorList.cpp File Reference

```
#include <rmol/bom/DemandGeneratorList.hpp>
```

Namespaces

- RMOL

26.56 DemandGeneratorList.cpp

```
00001 // //////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////////////////////////////////////
00004 // RMOL
00005 #include <rmol/bom/DemandGeneratorList.hpp>
00006
00007 namespace RMOL {
00008
00009   // //////////////////////////////////////////////////////////////////////
00010   DemandGeneratorList::DemandGeneratorList () {
00011     const DistributionParameterList_T aDistributionParameterList;
00012     init (aDistributionParameterList);
00013   }
00014
00015   // //////////////////////////////////////////////////////////////////////
00016   DemandGeneratorList::
00017   DemandGeneratorList (const DemandGeneratorList&
      iDemandGeneratorList) {
00018     // TODO: copy the distribution parameters of the input generator list
00019     const DistributionParameterList_T aDistributionParameterList;
00020     init (aDistributionParameterList);
00021   }
00022
00023   // //////////////////////////////////////////////////////////////////////
00024   DemandGeneratorList::
00025   DemandGeneratorList (const DistributionParameterList_T&
      iDistributionParameterList) {
00026     init (iDistributionParameterList);
00027   }
00028
00029   // //////////////////////////////////////////////////////////////////////
00030   DemandGeneratorList::~DemandGeneratorList () {
00031   }
00032
00033   // //////////////////////////////////////////////////////////////////////
00034   void DemandGeneratorList::
00035   init (const DistributionParameterList_T& iDistributionParameterList) {
00036
00037     DistributionParameterList_T::const_iterator itParams =
00038       iDistributionParameterList.begin();
00039     for ( ; itParams != iDistributionParameterList.end(); itParams++) {
00040       const FldDistributionParameters& aParams = *itParams;
00041
00042       const Gaussian gaussianGenerator (aParams);
00043
00044       _demandGeneratorList.push_back (gaussianGenerator);
00045     }
00046   }
00047
00048   // //////////////////////////////////////////////////////////////////////
00049   void DemandGeneratorList::
00050   generateVariateList (VariateList_T& ioVariateList) const {
00051
00052     // Iterate on the (number of) classes/buckets, n
00053     DemandGeneratorList_T::const_iterator itGenerator =
00054       _demandGeneratorList.begin();
00055     for ( ; itGenerator != _demandGeneratorList.end(); itGenerator++) {
00056       const Gaussian& gaussianGenerator = *itGenerator;
00057
00058       // Generate a random variate following the Gaussian distribution
00059       const double generatedVariate = gaussianGenerator.generateVariate ();
00060       ioVariateList.push_back (generatedVariate);
00061     }
00062   }
00063
00064 }
```

26.57 rmol/bom/old/DemandGeneratorList.hpp File Reference

```
#include <list>
#include <rmol/bom/VariateList.hpp>
#include <rmol/bom/DistributionParameterList.hpp>
#include <rmol/bom/Gaussian.hpp>
```

Classes

- class RMOL::DemandGeneratorList

Namespaces

- RMOL

26.58 DemandGeneratorList.hpp

```
00001 #ifndef __RMOL_DEMANDGENERATORLIST_HPP
00002 #define __RMOL_DEMANDGENERATORLIST_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <list>
00009 // RMOL
00010 #include <rmol/bom/VariateList.hpp>
00011 #include <rmol/bom/DistributionParameterList.hpp>
00012 #include <rmol/bom/Gaussian.hpp>
00013
00014 namespace RMOL {
00015
00017   class DemandGeneratorList {
00018   protected:
00020     typedef std::list<Gaussian> DemandGeneratorList_T;
00021
00022   public:
00024     DemandGeneratorList ();
00025     DemandGeneratorList (const DemandGeneratorList&);
00027     DemandGeneratorList (const DistributionParameterList_T&);
00028
00030     virtual ~DemandGeneratorList();
00031
00033     void generateVariateList (VariateList_T&) const;
00034
00035   private:
00036     DemandGeneratorList_T _demandGeneratorList;
00037
00039     void init (const DistributionParameterList_T&);
00040
00041   };
00042 }
00043 #endif // __RMOL_DEMANDGENERATORLIST_HPP
```

26.59 rmol/bom/PolicyHelper.cpp File Reference

```
#include <cassert>
#include <cmath>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/basic/BasConst_Yield.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/Policy.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/bom/FareFamily.hpp>
#include <stdair/bom/NestingNode.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/factory/FacBomManager.hpp>
#include <rmol/bom/PolicyHelper.hpp>
```

Namespaces

- RMOL

26.60 PolicyHelper.cpp

```
00001
00002 // //////////////////////////////////////////////////////////////////////
00003 // Import section
00004 // //////////////////////////////////////////////////////////////////////
00005 // STL
00006 #include <cassert>
00007 #include <cmath>
00008 // StdAir
00009 #include <stdair/basic/BasConst_Inventory.hpp>
00010 #include <stdair/basic/BasConst_Yield.hpp>
00011 #include <stdair/bom/SegmentCabin.hpp>
00012 #include <stdair/bom/Policy.hpp>
00013 #include <stdair/bom/BookingClass.hpp>
00014 #include <stdair/bom/FareFamily.hpp>
00015 #include <stdair/bom/NestingNode.hpp>
00016 #include <stdair/bom/BomManager.hpp>
00017 #include <stdair/factory/FacBomManager.hpp>
00018 // RMOL
00019 #include <rmol/bom/PolicyHelper.hpp>
00020
00021 namespace RMOL {
00022   // //////////////////////////////////////////////////////////////////////
00023   void PolicyHelper::
00024   diffBetweenTwoPolicies (stdair::NestingNode& ioNode,
00025                           const stdair::Policy& iFirstPolicy,
00026                           const stdair::Policy& iSecondPolicy) {
00027
00028     // Retrieve the booking class list of the first policy
00029     const stdair::BookingClassList_T& lFirstBCList =
00030           stdair::BomManager::getList<stdair::BookingClass> (iFirstPolicy);
00031
00032     // Browse the booking class list
00033     for (stdair::BookingClassList_T::const_iterator itBC = lFirstBCList.begin();
00034          itBC != lFirstBCList.end(); ++itBC){
00035       const stdair::BookingClass* iFirstPolicyBC_ptr = *itBC;
00036       const stdair::BookingClassKey& lFirstPolicyBCKey =
00037         iFirstPolicyBC_ptr->getKey();
00038       const stdair::ClassCode_T& lFirstPolicyClassCode =
00039         lFirstPolicyBCKey.getClassCode();
00040       // Retrieve the fare family of the booking class and the list of booking
00041       // class of this fare family
00042       const stdair::FareFamily& lFareFamily =
00043         stdair::BomManager::getParent<stdair::FareFamily> (*iFirstPolicyBC_ptr);
00044
00045       // Retrieve the list of booking class between the both booking classes
00046       diffBetweenBookingClassAndPolicy (ioNode, lFareFamily,
00047                                         lFirstPolicyClassCode,
00048                                         iSecondPolicy);
00049     }
00050   }
00051
00052   // //////////////////////////////////////////////////////////////////////
00053   const bool PolicyHelper::
00054   intersectionBetweenPolicyAndBookingClassList (const stdair::BookingClassList_T& iBCList,
00055                                                 const stdair::Policy& iPolicy,
00056                                                 stdair::ClassCode_T& oClassCode) {
00057     bool isInBookingClassList = false;
00058
00059     // Retrieve the booking classes of the policy
00060     const bool hasAListOfBC =
00061       stdair::BomManager::hasList<stdair::BookingClass> (iPolicy);
00062     if (hasAListOfBC == false) {
00063       return isInBookingClassList;
00064     }
00065     const stdair::BookingClassList_T& lPolicyBookingClassList =
00066       stdair::BomManager::getList<stdair::BookingClass> (iPolicy);
00067
00068     // Browse the booking class list of the fare family
00069     stdair::BookingClassList_T::const_iterator itFFBC = iBCList.begin();
00070     for (; itFFBC != iBCList.end(); ++itFFBC) {
00071       stdair::BookingClass* lFFBC_ptr = *itFFBC;
00072       const stdair::BookingClassKey& lFFBCKey = lFFBC_ptr->getKey();
00073       const stdair::ClassCode_T& lFFClassCode = lFFBCKey.getClassCode();
00074       // Compare the booking class with booking classes of policy
00075       stdair::BookingClassList_T::const_iterator itPolicyBC =
00076         lPolicyBookingClassList.begin();
00077       for (; itPolicyBC != lPolicyBookingClassList.end(); ++itPolicyBC){
```

```
00078         const stdair::BookingClass* lPolicyBC_ptr = *itPolicyBC;
00079         const stdair::BookingClassKey& lPolicyBCKey = lPolicyBC_ptr->getKey();
00080         const stdair::ClassCode_T& lPolicyClassCode =
00081           lPolicyBCKey.getClassCode();
00082         if (lPolicyClassCode == lFFClassCode) {
00083           oClassCode = lPolicyClassCode;
00084           isInBookingClassList = true;
00085           return isInBookingClassList;
00086         }
00087       }
00088     }
00089     // If the policy has not any booking class in the fare family,
00090     // return false
00091     return isInBookingClassList;
00092   }
00093
00094   // ////////////////////////////////////////////////////////////////////
00095   void PolicyHelper::
00096   diffBetweenBookingClassAndPolicy (stdair::NestingNode& ioNode,
00097                                     const stdair::FareFamily& iFareFamily,
00098                                     const stdair::ClassCode_T& iFirstPolicyClassCode,
00099                                     const stdair::Policy& iSecondPolicy) {
00100     const stdair::BookingClassList_T& lFFBCList =
00101         stdair::BomManager::getList<stdair::BookingClass> (iFareFamily);
00102     const bool isEmptyBookingClassList = lFFBCList.empty();
00103     if (isEmptyBookingClassList == true) {
00104       std::ostringstream ostr;
00105       ostr << "The booking class list of the fare family "
00106            << iFareFamily.describeKey() << " is empty.";
00107       STDAIR_LOG_DEBUG(ostr.str());
00108       throw EmptyBookingClassListException (ostr.str());
00109     }
00110
00111     // Retrieve the reverse iterator for the first booking class
00112     stdair::BookingClassList_T::const_reverse_iterator ritBC;
00113     for (ritBC = lFFBCList.rbegin(); ritBC != lFFBCList.rend(); ++ritBC) {
00114       const stdair::BookingClass* lBC_ptr = *ritBC;
00115       assert (lBC_ptr != NULL);
00116       const stdair::BookingClassKey& lBookingClassKey = lBC_ptr->getKey();
00117       const stdair::ClassCode_T& lClassCode = lBookingClassKey.getClassCode();
00118       if (iFirstPolicyClassCode == lClassCode) {
00119         break;
00120       }
00121     }
00122     if (ritBC == lFFBCList.rend()) {
00123       std::ostringstream ostr;
00124       ostr << "The booking class " << iFirstPolicyClassCode
00125            << "is not in the Fare Family " << iFareFamily.describeKey();
00126       STDAIR_LOG_DEBUG(ostr.str());
00127       throw MissingBookingClassInFareFamilyException (ostr.str());
00128     }
00129     assert(ritBC != lFFBCList.rend());
00130
00131     // Retrieve the booking class of the second policy in the same
00132     // fare family than the current booking class
00133     stdair::ClassCode_T lSecondPolicyClassCode;
00134     const bool hasABookingClassIn =
00135       intersectionBetweenPolicyAndBookingClassList(lFFBCList,
00136                                                    iSecondPolicy,
00137                                                    lSecondPolicyClassCode);
00138     // Add booking class between the first booking class and
00139     // the second booking class
00140
00141     if (hasABookingClassIn == false) {
00142       for (; ritBC != lFFBCList.rend(); ++ritBC) {
00143         stdair::BookingClass* lBC_ptr = *ritBC;
00144         stdair::FacBomManager::addToList (ioNode, *lBC_ptr);
00145       }
00146     } else {
00147
00148       for (; ritBC != lFFBCList.rend(); ++ritBC) {
00149         stdair::BookingClass* lBC_ptr = *ritBC;
00150         assert (lBC_ptr != NULL);
00151         const stdair::BookingClassKey& lBookingClassKey = lBC_ptr->getKey();
00152         const stdair::ClassCode_T& lClassCode = lBookingClassKey.getClassCode();
00153         if (lSecondPolicyClassCode == lClassCode) {
00154           break;
00155         }
00156         stdair::FacBomManager::addToList (ioNode, *lBC_ptr);
00157       }
00158       assert(ritBC != lFFBCList.rend());
00159     }
00160   }
00161
00162   // ////////////////////////////////////////////////////////////////////
00163   void PolicyHelper::
00164   computeLastNode (stdair::NestingNode& ioNode,
```

```
00165                   const stdair::Policy& iPolicy,
00166                   const stdair::SegmentCabin& iSegmentCabin) {
00167     // Compare the number of booking classes in the policy and the number
00168     // of fare families of the segment-cabin.
00169     ioNode.setYield(stdair::DEFAULT_YIELD_VALUE);
00170     const stdair::BookingClassList_T& lBCList =
00171       stdair::BomManager::getList<stdair::BookingClass> (iPolicy);
00172     const stdair::NbOfClasses_T lNbOfClasses = lBCList.size();
00173     const stdair::FareFamilyList_T& lFFList =
00174       stdair::BomManager::getList<stdair::FareFamily> (iSegmentCabin);
00175     const stdair::NbOfFareFamilies_T lNbOfFFs = lFFList.size();
00176     assert (lNbOfFFs >= lNbOfClasses);
00177
00178     // Number of closed fare families in the policy.
00179     const stdair::NbOfFareFamilies_T lNbOfClosedFFs = lNbOfFFs - lNbOfClasses;
00180     stdair::FareFamilyList_T::const_reverse_iterator itFF = lFFList.rbegin();
00181     for (unsigned i=0; i<lNbOfClosedFFs; ++i, ++itFF) {
00182       const stdair::FareFamily* lFF_ptr = *itFF;
00183       assert (lFF_ptr != NULL);
00184       const stdair::BookingClassList_T& lCurrentBCList =
00185         stdair::BomManager::getList<stdair::BookingClass> (*lFF_ptr);
00186       for (stdair::BookingClassList_T::const_reverse_iterator itCurrentBC =
00187              lCurrentBCList.rbegin(); itCurrentBC != lCurrentBCList.rend();
00188            ++itCurrentBC) {
00189         stdair::BookingClass* lCurrentBC_ptr = *itCurrentBC;
00190         assert (lCurrentBC_ptr != NULL);
00191         stdair::FacBomManager::addToList (ioNode, *lCurrentBC_ptr);
00192       }
00193     }
00194
00195     //
00196     for (stdair::BookingClassList_T::const_reverse_iterator itBC =
00197            lBCList.rbegin(); itBC != lBCList.rend(); ++itBC) {
00198       const stdair::BookingClass* lBC_ptr = *itBC;
00199       assert (lBC_ptr != NULL);
00200       const stdair::FareFamily& lFF =
00201         stdair::BomManager::getParent<stdair::FareFamily> (*lBC_ptr);
00202
00203       const stdair::BookingClassList_T& lCurrentBCList =
00204         stdair::BomManager::getList<stdair::BookingClass> (lFF);
00205       for (stdair::BookingClassList_T::const_reverse_iterator itCurrentBC =
00206              lCurrentBCList.rbegin(); itCurrentBC != lCurrentBCList.rend();
00207            ++itCurrentBC) {
00208         stdair::BookingClass* lCurrentBC_ptr = *itCurrentBC;
00209         assert (lCurrentBC_ptr != NULL);
00210         if (lCurrentBC_ptr->describeKey() != lBC_ptr->describeKey()) {
00211           stdair::FacBomManager::addToList (ioNode, *lCurrentBC_ptr);
00212         } else {
00213           break;
00214         }
00215       }
00216     }
00217   }
00218
00219   // ////////////////////////////////////////////////////////////////////
00220   bool PolicyHelper::isNested (const stdair::Policy& iFirstPolicy,
00221                                const stdair::Policy& iSecondPolicy) {
00222     bool isNested = false;
00223     // The number of classes in the first policy should be smaller or equal
00224     // to the number of classes in the second one.
00225     const bool hasAListOfBCFirstPolicy =
00226       stdair::BomManager::hasList<stdair::BookingClass> (iFirstPolicy);
00227     // All policies are nested with the empty policy
00228     if (hasAListOfBCFirstPolicy == false) {
00229       isNested = true;
00230       return isNested;
00231     }
00232     const stdair::BookingClassList_T& lFirstBCList =
00233       stdair::BomManager::getList<stdair::BookingClass> (iFirstPolicy);
00234     const bool hasAListOfBCSecondPolicy =
00235       stdair::BomManager::hasList<stdair::BookingClass> (iSecondPolicy);
00236     // The empty policy is not nested
00237     if (hasAListOfBCSecondPolicy == false) {
00238       return isNested;
00239     }
00240     const stdair::BookingClassList_T& lSecondBCList =
00241       stdair::BomManager::getList<stdair::BookingClass> (iSecondPolicy);
00242     if (lFirstBCList.size() > lSecondBCList.size()) {
00243       return isNested;
00244     }
00245
00246     // Browse the two lists of booking classes and verify if the pairs
00247     // of classes are in order.
00248     stdair::BookingClassList_T::const_iterator itFirstBC = lFirstBCList.begin();
00249     for (stdair::BookingClassList_T::const_iterator itSecondBC =
00250            lSecondBCList.begin(); itFirstBC != lFirstBCList.end();
00251          ++itFirstBC, ++itSecondBC) {
```

```
00252       const stdair::BookingClass* lFirstBC_ptr = *itFirstBC;
00253       assert (lFirstBC_ptr != NULL);
00254       const std::string lFirstKey = lFirstBC_ptr->describeKey();
00255       const stdair::BookingClass* lSecondBC_ptr = *itSecondBC;
00256       assert (lSecondBC_ptr != NULL);
00257       const std::string lSecondKey = lSecondBC_ptr->describeKey();
00258       if (lFirstKey == lSecondKey) {
00259         break;
00260       }
00261
00262       // Retrieve the parent FF and its booking class list.
00263       const stdair::FareFamily& lFF =
00264         stdair::BomManager::getParent<stdair::FareFamily> (*lFirstBC_ptr);
00265       const stdair::BookingClassList_T& lBCList =
00266         stdair::BomManager::getList<stdair::BookingClass> (lFF);
00267       for (stdair::BookingClassList_T::const_iterator itBC = lBCList.begin();
00268            itBC != lBCList.end(); ++itBC) {
00269         const stdair::BookingClass* lBC_ptr = *itBC;
00270         assert (lBC_ptr != NULL);
00271         const std::string lKey = lBC_ptr->describeKey();
00272         if (lFirstKey == lKey) {
00273           break;
00274         } else if (lSecondKey == lKey) {
00275           return isNested;
00276         }
00277       }
00278     }
00279
00280     isNested = true;
00281     return isNested;
00282   }
00283 }
```

26.61 rmol/bom/PolicyHelper.hpp File Reference

```
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/bom/PolicyTypes.hpp>
#include <stdair/bom/BookingClassTypes.hpp>
#include <stdair/bom/FareFamilyTypes.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::PolicyHelper

Namespaces

- stdair

  Forward declarations.
- RMOL

26.62 PolicyHelper.hpp

```
00001 #ifndef __RMOL_BOM_POLICYHELPER_HPP
00002 #define __RMOL_BOM_POLICYHELPER_HPP
00003 // //////////////////////////////////////////////////////////////////
00004 // Import section
00005 // //////////////////////////////////////////////////////////////////
00006 // StdAir
00007 #include <stdair/stdair_inventory_types.hpp>
00008 #include <stdair/bom/PolicyTypes.hpp>
00009 #include <stdair/bom/BookingClassTypes.hpp>
00010 #include <stdair/bom/FareFamilyTypes.hpp>
00011 // RMOL
00012 #include <rmol/RMOL_Types.hpp>
00013
00014 // Forward declarations
00015 namespace stdair {
00016   class SegmentCabin;
00017   class Policy;
```

```
00018   class FareFamily;
00019   class BookingClass;
00020   class NestingNode;
00021 }
00022
00023 namespace RMOL {
00024
00028   class PolicyHelper {
00029   public:
00030
00035     static void
00036     diffBetweenTwoPolicies (stdair::NestingNode&, const stdair::Policy&,
00037                             const stdair::Policy&);
00038
00042     static void
00043     computeLastNode (stdair::NestingNode&, const stdair::Policy&,
00044                      const stdair::SegmentCabin&);
00045
00049     static bool isNested (const stdair::Policy&, const stdair::Policy&);
00050
00051
00052   private:
00053
00058     static const bool
00059     intersectionBetweenPolicyAndBookingClassList (const stdair::BookingClassList_T&,
00060                                                   const stdair::Policy&,
00061                                                   stdair::ClassCode_T&);
00062
00066     static void
00067     diffBetweenBookingClassAndPolicy (stdair::NestingNode&,
00068                                       const stdair::FareFamily&,
00069                                       const stdair::ClassCode_T&,
00070                                       const stdair::Policy&);
00071
00072   };
00073
00074 }
00075
00076 #endif // __RMOL_BOM_POLICYHELPER_HPP
```

26.63 rmol/bom/SegmentSnapshotTableHelper.cpp File Reference

```
#include <cassert>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/bom/SegmentSnapshotTable.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/SegmentSnapshotTableHelper.hpp>
```

Namespaces

- RMOL

26.64 SegmentSnapshotTableHelper.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 // StdAir
00007 #include <stdair/basic/BasConst_Inventory.hpp>
00008 #include <stdair/bom/BomManager.hpp>
00009 #include <stdair/bom/SegmentDate.hpp>
00010 #include <stdair/bom/SegmentCabin.hpp>
00011 #include <stdair/bom/BookingClass.hpp>
00012 #include <stdair/bom/SegmentSnapshotTable.hpp>
00013 #include <stdair/service/Logger.hpp>
00014 // RMOL
```

```
00015 #include <rmol/bom/SegmentSnapshotTableHelper.hpp>
00016
00017 namespace RMOL {
00018   // ////////////////////////////////////////////////////////////////////
00019   stdair::NbOfSegments_T SegmentSnapshotTableHelper::
00020   getNbOfSegmentAlreadyPassedThisDTD (const
      stdair::SegmentSnapshotTable& iGB,
00021                                       const stdair::DTD_T& iDTD,
00022                                       const stdair::Date_T& iCurrentDate) {
00023     stdair::NbOfSegments_T oNbOfSegments = 0;
00024
00025     // Browse the list of segments and check if it has passed the given DTD.
00026     const stdair::SegmentCabinIndexMap_T& lSCMap=iGB.getSegmentCabinIndexMap();
00027     for (stdair::SegmentCabinIndexMap_T::const_iterator itSC = lSCMap.begin();
00028          itSC != lSCMap.end(); ++itSC) {
00029       const stdair::SegmentCabin* lSC_ptr = itSC->first;
00030       assert (lSC_ptr != NULL);
00031
00032       if (hasPassedThisDTD (*lSC_ptr, iDTD, iCurrentDate) == true) {
00033         ++oNbOfSegments;
00034       }
00035     }
00036
00037     return oNbOfSegments;
00038   }
00039
00040   // ////////////////////////////////////////////////////////////////////
00041   bool SegmentSnapshotTableHelper::
00042   hasPassedThisDTD (const stdair::SegmentCabin& iSegmentCabin,
00043                     const stdair::DTD_T& iDTD,
00044                     const stdair::Date_T& iCurrentDate) {
00045     // Retrieve the boarding date.
00046     const stdair::SegmentDate& lSegmentDate =
00047       stdair::BomManager::getParent<stdair::SegmentDate> (iSegmentCabin);
00048     const stdair::Date_T& lBoardingDate = lSegmentDate.getBoardingDate();
00049
00050     // Compare the date offset between the boarding date and the current date
00051     // to the DTD.
00052     stdair::DateOffset_T lDateOffset = lBoardingDate - iCurrentDate;
00053     stdair::DTD_T lDateOffsetInDays = lDateOffset.days();
00054     if (iDTD > lDateOffsetInDays) {
00055       return true;
00056     } else {
00057       return false;
00058     }
00059   }
00060 }
```

26.65 rmol/bom/SegmentSnapshotTableHelper.hpp File Reference

```
#include <string>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/stdair_date_time_types.hpp>
```

Classes

- class RMOL::SegmentSnapshotTableHelper

Namespaces

- stdair

  Forward declarations.
- RMOL

26.66 SegmentSnapshotTableHelper.hpp

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

```
00006 // //////////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <string>
00009 // StdAir
00010 #include <stdair/stdair_inventory_types.hpp>
00011 #include <stdair/stdair_date_time_types.hpp>
00012
00013 // Forward declarations
00014 namespace stdair {
00015   class SegmentSnapshotTable;
00016   class SegmentCabin;
00017 }
00018
00019 namespace RMOL {
00020
00023   class SegmentSnapshotTableHelper {
00024   public:
00025     // ********** Business Methods *********
00030     static stdair::NbOfSegments_T getNbOfSegmentAlreadyPassedThisDTD (
      const stdair::SegmentSnapshotTable&, const stdair::DTD_T&, const stdair::Date_T&);
00031
00035     static bool hasPassedThisDTD (const stdair::SegmentCabin&,
00036                                   const stdair::DTD_T&, const stdair::Date_T&);
00037   };
00038
00039 }
00040 #endif // __RMOL_BOM_SEGMENTSNAPSHOTTABLEHELPER_HPP
```

26.67 rmol/bom/Utilities.cpp File Reference

```
#include <cassert>
#include <string>
#include <numeric>
#include <algorithm>
#include <cmath>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/FareFamily.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/bom/BookingClassTypes.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/basic/BasConst_General.hpp>
#include <rmol/bom/Utilities.hpp>
#include <rmol/bom/SegmentSnapshotTableHelper.hpp>
```

Namespaces

- RMOL

26.68 Utilities.cpp

```
00001
00002 // //////////////////////////////////////////////////////////////////////
00003 // Import section
00004 // //////////////////////////////////////////////////////////////////////
00005 // STL
00006 #include <cassert>
00007 #include <string>
00008 #include <numeric>
00009 #include <algorithm>
00010 #include <cmath>
00011 // StdAir
00012 #include <stdair/basic/BasConst_Inventory.hpp>
00013 #include <stdair/bom/BomManager.hpp>
00014 #include <stdair/bom/SegmentCabin.hpp>
00015 #include <stdair/bom/FareFamily.hpp>
00016 #include <stdair/bom/BookingClass.hpp>
00017 #include <stdair/bom/BookingClassTypes.hpp>
```

```
00018 #include <stdair/service/Logger.hpp>
00019 // RMOL
00020 #include <rmol/basic/BasConst_General.hpp>
00021 #include <rmol/bom/Utilities.hpp>
00022 #include <rmol/bom/SegmentSnapshotTableHelper.hpp>
00023
00024 namespace RMOL {
00025   // ////////////////////////////////////////////////////////////////////
00026   void Utilities::
00027   computeDistributionParameters (const stdair::UncDemVector_T& iVector,
00028                                  stdair::MeanValue_T& ioMean,
00029                                  stdair::StdDevValue_T& ioStdDev) {
00030     ioMean = 0.0; ioStdDev = 0.0;
00031     const stdair::NbOfSamples_T lNbOfSamples = iVector.size();
00032     assert (lNbOfSamples > 1);
00033
00034     // Compute the mean
00035     for (stdair::UncDemVector_T::const_iterator itSample = iVector.begin();
00036          itSample != iVector.end(); ++itSample) {
00037       //STDAIR_LOG_NOTIFICATION (*itSample);
00038       ioMean += *itSample;
00039     }
00040     ioMean /= lNbOfSamples;
00041
00042     // Compute the standard deviation
00043     for (stdair::UncDemVector_T::const_iterator itSample = iVector.begin();
00044          itSample != iVector.end(); ++itSample) {
00045       const stdair::MeanValue_T& lSample = *itSample;
00046       ioStdDev += ((lSample - ioMean) * (lSample - ioMean));
00047     }
00048     ioStdDev /= (lNbOfSamples - 1);
00049     ioStdDev = std::sqrt (ioStdDev);
00050
00051     // Sanity check
00052     if (ioStdDev == 0) {
00053       ioStdDev = 0.1;
00054     }
00055   }
00056
00057   // ////////////////////////////////////////////////////////////////////
00058   stdair::DCPList_T Utilities::
00059   buildRemainingDCPList (const stdair::DTD_T& iDTD) {
00060     stdair::DCPList_T oDCPList;
00061
00062     const stdair::DCPList_T lWholeDCPList = stdair::DEFAULT_DCP_LIST;
00063     stdair::DCPList_T::const_iterator itDCP = lWholeDCPList.begin();
00064     while (itDCP != lWholeDCPList.end()) {
00065       const stdair::DCP_T& lDCP = *itDCP;
00066       if (iDTD >= lDCP) {
00067         break;
00068       }
00069       ++itDCP;
00070     }
00071     assert (itDCP != lWholeDCPList.end());
00072
00073     oDCPList.push_back (iDTD);
00074     ++itDCP;
00075     for (; itDCP != lWholeDCPList.end(); ++itDCP) {
00076       oDCPList.push_back (*itDCP);
00077     }
00078
00079     return oDCPList;
00080   }
00081
00082   // ////////////////////////////////////////////////////////////////////
00083   stdair::DCPList_T Utilities::
00084   buildPastDCPList (const stdair::DTD_T& iDTD) {
00085     stdair::DCPList_T oDCPList;
00086
00087     const stdair::DCPList_T lWholeDCPList = stdair::DEFAULT_DCP_LIST;
00088     stdair::DCPList_T::const_iterator itDCP = lWholeDCPList.begin();
00089     while (itDCP != lWholeDCPList.end()) {
00090       const stdair::DCP_T& lDCP = *itDCP;
00091       if (iDTD <= lDCP) {
00092         oDCPList.push_back (lDCP);
00093         ++itDCP;
00094       } else {
00095         break;
00096       }
00097     }
00098
00099     return oDCPList;
00100   }
00101
00102   // ////////////////////////////////////////////////////////////////////
00103   stdair::NbOfSegments_T Utilities::
00104   getNbOfDepartedSimilarSegments (const stdair::SegmentCabin& iSegmentCabin
```

```
,
00105                                 const stdair::Date_T& iEventDate) {
00106     stdair::DTD_T lDTD = 0;
00107     // Retrieve the guillotine block.
00108     const stdair::SegmentSnapshotTable& lSegmentSnapshotTable =
00109       iSegmentCabin.getSegmentSnapshotTable();
00110     return SegmentSnapshotTableHelper::
00111       getNbOfSegmentAlreadyPassedThisDTD (lSegmentSnapshotTable,
      lDTD, iEventDate);
00112   }
00113
00114   // ////////////////////////////////////////////////////////////////////
00115   stdair::BookingClassSellUpCurveMap_T Utilities::
00116   computeSellUpFactorCurves (const stdair::FRAT5Curve_T& iFRAT5Curve,
00117                              const stdair::BookingClassList_T& iBCList) {
00118     stdair::BookingClassSellUpCurveMap_T oBCSellUpFactorMap;
00119
00120     // Initialise a sell-up factor curve of 1.0 values
00121     stdair::SellUpCurve_T lBasedSellUpCurve;
00122     for (stdair::FRAT5Curve_T::const_iterator itFRAT5 = iFRAT5Curve.begin();
00123          itFRAT5 != iFRAT5Curve.end(); ++itFRAT5) {
00124       const stdair::DTD_T& lDTD = itFRAT5->first;
00125       lBasedSellUpCurve.insert(stdair::SellUpCurve_T::value_type(lDTD, 1.0));
00126     }
00127
00128     // Retrieve the classes from low to high and compute the distributions of
00129     // product-oriented and price-oriented demand.
00130     // Retrieve the lowest class.
00131     stdair::BookingClassList_T::const_reverse_iterator itCurrentClass =
00132       iBCList.rbegin();
00133     assert (itCurrentClass != iBCList.rend());
00134
00135     // If there is only one class in the cabin, all the sell-up factors
00136     // will be 1.
00137     stdair::BookingClass* lLowestBC_ptr = *itCurrentClass;
00138     assert (lLowestBC_ptr != NULL);
00139     const stdair::Yield_T& lLowestYield = lLowestBC_ptr->getYield();
00140     bool insert = oBCSellUpFactorMap.
00141       insert (stdair::BookingClassSellUpCurveMap_T::
00142               value_type(lLowestBC_ptr, lBasedSellUpCurve)).second;
00143     assert (insert == true);
00144     ++itCurrentClass;
00145
00146     // Compute the demand for higher class using the formula
00147     // Pro_sell_up_from_Q_to_F = e ^ ((y_F/y_Q - 1) * ln (0.5) / (FRAT5 - 1))
00148     for (; itCurrentClass != iBCList.rend(); ++itCurrentClass) {
00149       stdair::BookingClass* lCurrentBC_ptr = *itCurrentClass;
00150       assert (lCurrentBC_ptr != NULL);
00151       const stdair::Yield_T& lCurrentYield = lCurrentBC_ptr->getYield();
00152
00153       // Compute the sell-up factor curve for the current class.
00154       stdair::SellUpCurve_T lCurrentSellUpCurve;
00155       for (stdair::FRAT5Curve_T::const_iterator itFRAT5 = iFRAT5Curve.begin();
00156            itFRAT5 != iFRAT5Curve.end(); ++itFRAT5) {
00157         const stdair::DTD_T& lDTD = itFRAT5->first;
00158         const stdair::FRAT5_T& lFRAT5 = itFRAT5->second;
00159         const double lSellUpCoef = log(0.5)/(lFRAT5-1);
00160         const stdair::SellupProbability_T lSellUpFactor =
00161           exp ((lCurrentYield/lLowestYield - 1.0) * lSellUpCoef);
00162         const bool isInsertionSuccessful =
00163           lCurrentSellUpCurve.insert (stdair::SellUpCurve_T::value_type(lDTD, lSellUpFactor)).second;
00164         assert (isInsertionSuccessful == true);
00165       }
00166       const bool isInsertionSuccessful = oBCSellUpFactorMap.
00167         insert (stdair::BookingClassSellUpCurveMap_T::
00168                 value_type(lCurrentBC_ptr, lCurrentSellUpCurve)).second;
00169       assert (isInsertionSuccessful == true);
00170     }
00171     return oBCSellUpFactorMap;
00172   }
00173
00174
00175   // ////////////////////////////////////////////////////////////////////
00176   stdair::BookingClassDispatchingCurveMap_T Utilities::
00177   computeDispatchingFactorCurves (const stdair::FRAT5Curve_T& iFRAT5Curve,
00178                                   const stdair::BookingClassList_T& iBCList) {
00179     stdair::BookingClassDispatchingCurveMap_T oBCDispatchingFactorMap;
00180
00181     // Initialise a sell-up factor curve of 1.0 values
00182     stdair::DispatchingCurve_T lBasedDispatchingCurve;
00183     for (stdair::FRAT5Curve_T::const_iterator itFRAT5 = iFRAT5Curve.begin();
00184          itFRAT5 != iFRAT5Curve.end(); ++itFRAT5) {
00185       const stdair::DTD_T& lDTD = itFRAT5->first;
00186       lBasedDispatchingCurve.insert(stdair::DispatchingCurve_T::value_type(lDTD, 1.0));
00187     }
00188
00189     // Retrieve the classes from low to high and compute the distributions of
```

```
00190     // product-oriented and price-oriented demand.
00191     // Retrieve the lowest class.
00192     stdair::BookingClassList_T::const_reverse_iterator itCurrentClass =
00193       iBCList.rbegin();
00194     assert (itCurrentClass != iBCList.rend());
00195     stdair::BookingClassList_T::const_reverse_iterator itNextClass =
00196       itCurrentClass; ++itNextClass;
00197
00198     // If there is only one class in the cabin, all the sell-up factors
00199     // will be 1.
00200     stdair::BookingClass* lLowestBC_ptr = *itCurrentClass;
00201     assert (lLowestBC_ptr != NULL);
00202     const stdair::Yield_T& lLowestYield = lLowestBC_ptr->getYield();
00203     if (itNextClass == iBCList.rend()) {
00204       bool insert = oBCDispatchingFactorMap.
00205         insert (stdair::BookingClassDispatchingCurveMap_T::
00206                 value_type(lLowestBC_ptr, lBasedDispatchingCurve)).second;
00207       assert (insert == true);
00208     } else {
00209       // Compute the demand for higher class using the formula
00210       // Pro_sell_up_from_Q_to_F = e ^ ((y_F/y_Q - 1) * ln (0.5) / (FRAT5 - 1))
00211       for (; itNextClass != iBCList.rend(); ++itCurrentClass, ++itNextClass) {
00212         stdair::BookingClass* lCurrentBC_ptr = *itCurrentClass;
00213         stdair::BookingClass* lNextBC_ptr = *itNextClass;
00214         assert (lNextBC_ptr != NULL);
00215         const stdair::Yield_T& lNextYield = lNextBC_ptr->getYield();
00216
00217         // Compute the sell-up factor curve for the current class.
00218         stdair::DispatchingCurve_T lCurrentDispatchingCurve;
00219         for (stdair::FRAT5Curve_T::const_iterator itFRAT5 = iFRAT5Curve.begin();
00220          itFRAT5 != iFRAT5Curve.end(); ++itFRAT5) {
00221           const stdair::DTD_T& lDTD = itFRAT5->first;
00222           const stdair::FRAT5_T& lFRAT5 = itFRAT5->second;
00223           const double lDispatchingCoef = log(0.5)/(lFRAT5-1);
00224           double lDispatchingFactor =
00225             exp ((lNextYield/lLowestYield - 1.0) * lDispatchingCoef);
00226           stdair::DispatchingCurve_T::iterator itBasedDispatching =
00227             lBasedDispatchingCurve.find (lDTD);
00228           assert (itBasedDispatching != lBasedDispatchingCurve.end());
00229           double& lBasedFactor = itBasedDispatching->second;
00230           bool insert = lCurrentDispatchingCurve.insert (stdair::DispatchingCurve_T::value_type(lDTD,
      lBasedFactor - lDispatchingFactor)).second;
00231           assert (insert == true);
00232           lBasedFactor = lDispatchingFactor;
00233         }
00234         bool insert = oBCDispatchingFactorMap.
00235           insert (stdair::BookingClassDispatchingCurveMap_T::
00236                   value_type(lCurrentBC_ptr, lCurrentDispatchingCurve)).second;
00237         assert (insert == true);
00238       }
00239
00240       // Compute the sell-up factor curve for the highest class (which is the
00241       // "current class")
00242       stdair::BookingClass* lCurrentBC_ptr = *itCurrentClass;
00243       bool insert = oBCDispatchingFactorMap.
00244         insert (stdair::BookingClassDispatchingCurveMap_T::
00245                 value_type(lCurrentBC_ptr, lBasedDispatchingCurve)).second;
00246       assert (insert == true);
00247     }
00248     return oBCDispatchingFactorMap;
00249   }
00250
00251   // ////////////////////////////////////////////////////////////////////
00252   void Utilities::dispatchDemandForecast
00253   (const stdair::BookingClassDispatchingCurveMap_T& iBCDispatchingCurveMap,
00254    const stdair::MeanValue_T& iMean,
00255    const stdair::StdDevValue_T& iStdDev,
00256    const stdair::DTD_T& iCurrentDCP) {
00257     for (stdair::BookingClassDispatchingCurveMap_T::const_iterator itBCDC =
00258            iBCDispatchingCurveMap.begin();
00259          itBCDC != iBCDispatchingCurveMap.end(); ++itBCDC) {
00260       stdair::BookingClass* lBC_ptr = itBCDC->first;
00261       assert (lBC_ptr != NULL);
00262       const stdair::DispatchingCurve_T& lDispatchingCurve = itBCDC->second;
00263       stdair::DispatchingCurve_T::const_iterator itDispatchingFactor =
00264         lDispatchingCurve.find (iCurrentDCP);
00265       assert (itDispatchingFactor != lDispatchingCurve.end());
00266       const double& lDF = itDispatchingFactor->second;
00267
00268       const stdair::MeanValue_T& lCurrentMean = lBC_ptr->getPriceDemMean();
00269       const stdair::StdDevValue_T& lCurrentStdDev = lBC_ptr->getPriceDemStdDev();
00270
00271       const stdair::MeanValue_T lAdditionalMean = iMean * lDF;
00272       const stdair::StdDevValue_T lAdditionalStdDev = iStdDev * std::sqrt (lDF);
00273
00274       const stdair::MeanValue_T lNewMean = lCurrentMean + lAdditionalMean;
00275       const stdair::StdDevValue_T lNewStdDev =
```

```
00276         std::sqrt (lCurrentStdDev * lCurrentStdDev
00277                    + lAdditionalStdDev * lAdditionalStdDev);
00278
00279       lBC_ptr->setPriceDemMean (lNewMean);
00280       lBC_ptr->setPriceDemStdDev (lNewStdDev);
00281     }
00282   }
00283
00284   // ////////////////////////////////////////////////////////////////////
00285   void Utilities::dispatchDemandForecastForFA
00286   (const stdair::BookingClassSellUpCurveMap_T& iBCSellUpCurveMap,
00287    const stdair::MeanValue_T& iMean,
00288    const stdair::StdDevValue_T& iStdDev,
00289    const stdair::DTD_T& iCurrentDCP) {
00290     for (stdair::BookingClassSellUpCurveMap_T::const_iterator itBCSU =
00291            iBCSellUpCurveMap.begin();
00292          itBCSU != iBCSellUpCurveMap.end(); ++itBCSU) {
00293       stdair::BookingClass* lBC_ptr = itBCSU->first;
00294       assert (lBC_ptr != NULL);
00295       const stdair::SellUpCurve_T& lSellUpCurve = itBCSU->second;
00296       stdair::SellUpCurve_T::const_iterator itSellUpFactor =
00297         lSellUpCurve.find (iCurrentDCP);
00298       assert (itSellUpFactor != lSellUpCurve.end());
00299       const stdair::SellupProbability_T& lSU = itSellUpFactor->second;
00300
00301       const stdair::MeanValue_T& lCurrentMean = lBC_ptr->getCumuPriceDemMean();
00302       const stdair::StdDevValue_T& lCurrentStdDev =
00303         lBC_ptr->getCumuPriceDemStdDev();
00304
00305       const stdair::MeanValue_T lAdditionalMean = iMean * lSU;
00306       const stdair::StdDevValue_T lAdditionalStdDev = iStdDev * std::sqrt (lSU);
00307
00308       const stdair::MeanValue_T lNewMean = lCurrentMean + lAdditionalMean;
00309       const stdair::StdDevValue_T lNewStdDev =
00310         std::sqrt (lCurrentStdDev * lCurrentStdDev
00311                    + lAdditionalStdDev * lAdditionalStdDev);
00312
00313       lBC_ptr->setCumuPriceDemMean (lNewMean);
00314       lBC_ptr->setCumuPriceDemStdDev (lNewStdDev);
00315     }
00316   }
00317 }
```

26.69 rmol/bom/Utilities.hpp File Reference

```
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/bom/FareFamilyTypes.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::Utilities

Namespaces

- stdair

  Forward declarations.
- RMOL

26.70 Utilities.hpp

```
00001 #ifndef __RMOL_BOM_UTILITIES_HPP
00002 #define __RMOL_BOM_UTILITIES_HPP
00003 // ////////////////////////////////////////////////////////////////////
00004 // Import section
00005 // ////////////////////////////////////////////////////////////////////
00006 // StdAir
00007 #include <stdair/stdair_inventory_types.hpp>
00008 #include <stdair/bom/FareFamilyTypes.hpp>
00009 // RMOL
00010 #include <rmol/RMOL_Types.hpp>
```

```
00011
00012 // Forward declarations
00013 namespace stdair {
00014   class SegmentCabin;
00015 }
00016
00017 namespace RMOL {
00018
00020   class Utilities {
00021   public:
00023     static void computeDistributionParameters (const stdair::UncDemVector_T&,
00024                                                stdair::MeanValue_T&,
00025                                                stdair::StdDevValue_T&);
00026
00030     static stdair::DCPList_T buildRemainingDCPList (const stdair::DTD_T&);
00031
00035     static stdair::DCPList_T buildPastDCPList (const stdair::DTD_T&);
00036
00040     static stdair::NbOfSegments_T
00041     getNbOfDepartedSimilarSegments (const stdair::SegmentCabin&,
00042                                     const stdair::Date_T&);
00043
00047     static stdair::BookingClassSellUpCurveMap_T
00048     computeSellUpFactorCurves (const stdair::FRAT5Curve_T&,
00049                                const stdair::BookingClassList_T&);
00050
00054     static stdair::BookingClassDispatchingCurveMap_T
00055     computeDispatchingFactorCurves (const stdair::FRAT5Curve_T&,
00056                                     const stdair::BookingClassList_T&);
00057
00061     static void
00062     dispatchDemandForecast (const stdair::BookingClassDispatchingCurveMap_T&,
00063                             const stdair::MeanValue_T&,
00064                             const stdair::StdDevValue_T&,
00065                             const stdair::DTD_T&);
00066
00070     static void
00071     dispatchDemandForecastForFA (const stdair::BookingClassSellUpCurveMap_T&,
00072                                  const stdair::MeanValue_T&,
00073                                  const stdair::StdDevValue_T&,
00074                                  const stdair::DTD_T&);
00075   };
00076
00077 }
00078
00079 #endif // __RMOL_BOM_UTILITIES_HPP
```

26.71 rmol/command/BasedForecasting.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <cmath>
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/basic/RandomGeneration.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/LegDate.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/SegmentSnapshotTable.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/Utilities.hpp>
#include <rmol/bom/SegmentSnapshotTableHelper.hpp>
#include <rmol/bom/HistoricalBookingHolder.hpp>
#include <rmol/bom/HistoricalBooking.hpp>
#include <rmol/command/BasedForecasting.hpp>
#include <rmol/command/Detruncator.hpp>
```

Namespaces

- RMOL

26.72 BasedForecasting.cpp

```
00001 // //////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 #include <cmath>
00008 // StdAir
00009 #include <stdair/basic/BasConst_General.hpp>
00010 #include <stdair/basic/BasConst_Inventory.hpp>
00011 #include <stdair/basic/RandomGeneration.hpp>
00012 #include <stdair/bom/BomManager.hpp>
00013 #include <stdair/bom/LegDate.hpp>
00014 #include <stdair/bom/SegmentDate.hpp>
00015 #include <stdair/bom/LegCabin.hpp>
00016 #include <stdair/bom/SegmentCabin.hpp>
00017 #include <stdair/bom/SegmentSnapshotTable.hpp>
00018 #include <stdair/bom/BookingClass.hpp>
00019 #include <stdair/service/Logger.hpp>
00020 // RMOL
00021 #include <rmol/bom/Utilities.hpp>
00022 #include <rmol/bom/SegmentSnapshotTableHelper.hpp>
00023 #include <rmol/bom/HistoricalBookingHolder.hpp>
00024 #include <rmol/bom/HistoricalBooking.hpp>
00025 #include <rmol/command/BasedForecasting.hpp>
00026 #include <rmol/command/Detruncator.hpp>
00027
00028 namespace RMOL {
00029   // //////////////////////////////////////////////////////////////////////
00030   bool BasedForecasting::
00031   forecast (stdair::SegmentCabin& ioSegmentCabin,
00032             const stdair::Date_T& iCurrentDate,
00033             const stdair::DTD_T& iCurrentDTD,
00034             const stdair::UnconstrainingMethod& iUnconstrainingMethod,
00035             const stdair::NbOfSegments_T& iNbOfDepartedSegments) {
00036
00037     // Retrieve the snapshot table.
00038     const stdair::SegmentSnapshotTable& lSegmentSnapshotTable =
00039       ioSegmentCabin.getSegmentSnapshotTable();
00040
00041     // Retrieve the booking class list.
00042     const stdair::BookingClassList_T& lBCList =
00043       stdair::BomManager::getList<stdair::BookingClass>(ioSegmentCabin);
00044
00045     // Browse all remaining DCP's and do unconstraining and forecasting for
00046     // all demand.
00047     const stdair::DCPList_T lWholeDCPList = stdair::DEFAULT_DCP_LIST;
00048     stdair::DCPList_T::const_iterator itDCP = lWholeDCPList.begin();
00049     stdair::DCPList_T::const_iterator itNextDCP = itDCP; ++itNextDCP;
00050     for (; itNextDCP != lWholeDCPList.end(); ++itDCP, ++itNextDCP) {
00051       const stdair::DCP_T& lCurrentDCP = *itDCP;
00052       const stdair::DCP_T& lNextDCP = *itNextDCP;
00053
00054       // The end of the interval is after the current DTD.
00055       if (lNextDCP < iCurrentDTD) {
00056         // Get the number of similar segments which has already passed the
00057         // (lNextDCP+1)
00058         const stdair::NbOfSegments_T& lNbOfUsableSegments =
00059           SegmentSnapshotTableHelper::
00060           getNbOfSegmentAlreadyPassedThisDTD (
      lSegmentSnapshotTable,
00061                                               lNextDCP+1,
00062                                               iCurrentDate);
00063         stdair::NbOfSegments_T lSegmentBegin = 0;
00064         const stdair::NbOfSegments_T lSegmentEnd = lNbOfUsableSegments-1;
00065         if (iNbOfDepartedSegments > 52) {
00066           lSegmentBegin = iNbOfDepartedSegments - 52;
00067         }
00068
00069         // Browse the list of booking classes and forecast the product-oriented
00070         // demand for each class.
00071         for (stdair::BookingClassList_T::const_iterator itBC = lBCList.begin();
00072              itBC != lBCList.end(); ++itBC) {
00073           stdair::BookingClass* lBC_ptr = *itBC;
00074           assert (lBC_ptr != NULL);
00075
00076           // Retrieve the historical product-oriented bookings for the
```

```
00077           // given class.
00078           HistoricalBookingHolder lHBHolder;
00079           prepareHistoricalBooking (ioSegmentCabin, *lBC_ptr,
00080                                     lSegmentSnapshotTable,
00081                                     lHBHolder,
00082                                     lCurrentDCP, lNextDCP,
00083                                     lSegmentBegin, lSegmentEnd);
00084
00085           // Unconstrain the historical bookings.
00086           Detruncator::unconstrain (lHBHolder, iUnconstrainingMethod);
00087
00088           // Retrieve the historical unconstrained demand and perform the
00089           // forecasting.
00090           stdair::UncDemVector_T lUncDemVector;
00091           const short lNbOfHistoricalFlights = lHBHolder.getNbOfFlights();
00092           for (short i = 0; i < lNbOfHistoricalFlights; ++i) {
00093             const stdair::NbOfBookings_T& lUncDemand =
00094               lHBHolder.getUnconstrainedDemand (i);
00095             lUncDemVector.push_back (lUncDemand);
00096           }
00097           stdair::MeanValue_T lMean = 0.0;
00098           stdair::StdDevValue_T lStdDev = 0.0;
00099           Utilities::computeDistributionParameters (lUncDemVector,
00100                                                     lMean, lStdDev);
00101
00102           // Add the demand forecast to the booking class.
00103           const stdair::MeanValue_T& lCurrentMean = lBC_ptr->getProductDemMean();
00104           const stdair::StdDevValue_T& lCurrentStdDev =
00105             lBC_ptr->getProductDemStdDev();
00106
00107           const stdair::MeanValue_T lNewMean = lCurrentMean + lMean;
00108           const stdair::StdDevValue_T lNewStdDev =
00109             std::sqrt (lCurrentStdDev * lCurrentStdDev + lStdDev * lStdDev);
00110
00111           lBC_ptr->setProductDemMean (lNewMean);
00112           lBC_ptr->setProductDemStdDev (lNewStdDev);
00113         }
00114       }
00115     }
00116     return true;
00117   }
00118
00119   // ////////////////////////////////////////////////////////////////////
00120   void BasedForecasting::prepareHistoricalBooking
00121     (const stdair::SegmentCabin& iSegmentCabin,
00122      const stdair::BookingClass& iBookingClass,
00123      const stdair::SegmentSnapshotTable& iSegmentSnapshotTable,
00124      HistoricalBookingHolder& ioHBHolder,
00125      const stdair::DCP_T& iDCPBegin, const stdair::DCP_T& iDCPEnd,
00126      const stdair::NbOfSegments_T& iSegmentBegin,
00127      const stdair::NbOfSegments_T& iSegmentEnd) {
00128
00129     // Retrieve the booking class index within the snapshot table
00130     const stdair::ClassIndex_T& lClassIdx =
00131       iSegmentSnapshotTable.getClassIndex (iBookingClass.describeKey());
00132
00133     // Retrieve the gross daily booking and availability snapshots.
00134     const stdair::ConstSegmentCabinDTDRangeSnapshotView_T lPriceBookingView =
00135       iSegmentSnapshotTable.getConstSegmentCabinDTDRangePriceOrientedGrossBookingSnapshotView (
      iSegmentBegin, iSegmentEnd, iDCPEnd, iDCPBegin);
00136     const stdair::ConstSegmentCabinDTDRangeSnapshotView_T lProductBookingView =
00137       iSegmentSnapshotTable.getConstSegmentCabinDTDRangeProductOrientedGrossBookingSnapshotView (
      iSegmentBegin, iSegmentEnd, iDCPEnd, iDCPBegin);
00138     const stdair::ConstSegmentCabinDTDRangeSnapshotView_T lAvlView =
00139       iSegmentSnapshotTable.getConstSegmentCabinDTDRangeAvailabilitySnapshotView (iSegmentBegin,
      iSegmentEnd, iDCPEnd, iDCPBegin);
00140
00141     // Browse the list of segments and build the historical booking holder.
00142     const stdair::ClassIndexMap_T& lVTIdxMap =
00143       iSegmentSnapshotTable.getClassIndexMap();
00144     const stdair::NbOfClasses_T lNbOfClasses = lVTIdxMap.size();
00145
00146     for (short i = 0; i <= iSegmentEnd-iSegmentBegin; ++i) {
00147       stdair::Flag_T lCensorshipFlag = false;
00148       const short lNbOfDTDs = iDCPBegin - iDCPEnd + 1;
00149       const stdair::UnsignedIndex_T lIdx = i*lNbOfClasses + lClassIdx;
00150
00151       // Parse the DTDs during the period and compute the censorship flag
00152       for (short j = 0; j < lNbOfDTDs; ++j) {
00153         // Check if the data has been censored during this day.
00154         // STDAIR_LOG_DEBUG ("i: " << i << ", NbOfClasses: " << lNbOfClasses
00155         //                   << ", ClassIdx: " << iClassIdx << ", j: " << j);
00156         if (lAvlView[lIdx][j] < 1.0) {
00157           lCensorshipFlag = true;
00158           break;
00159         }
00160       }
```

```
00161
00162       // Retrieve the historical bookings
00163       stdair::NbOfBookings_T lNbOfHistoricalBkgs = 0.0;
00164       for (short j = 0; j < lNbOfDTDs; ++j) {
00165         lNbOfHistoricalBkgs +=
00166           lPriceBookingView[lIdx][j] + lProductBookingView[lIdx][j];
00167       }
00168       HistoricalBooking lHistoricalBkg (lNbOfHistoricalBkgs, lCensorshipFlag);
00169       ioHBHolder.addHistoricalBooking (lHistoricalBkg);
00170     }
00171   }
00172
00173 }
```

26.73 rmol/command/BasedForecasting.hpp File Reference

```
#include <stdair/stdair_inventory_types.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::BasedForecasting

Namespaces

- stdair

  Forward declarations.
- RMOL

26.74 BasedForecasting.hpp

```
00001 #ifndef __RMOL_COMMAND_BASEDFORECASTING_HPP
00002 #define __RMOL_COMMAND_BASEDFORECASTING_HPP
00003
00004 // ////////////////////////////////////////////////////////////////////
00005 // Import section
00006 // ////////////////////////////////////////////////////////////////////
00007 // StdAir
00008 #include <stdair/stdair_inventory_types.hpp>
00009 // RMOL
00010 #include <rmol/RMOL_Types.hpp>
00011
00012 // Forward declarations
00013 namespace stdair {
00014   class SegmentCabin;
00015   class BookingClass;
00016   class SegmentSnapshotTable;
00017 }
00018
00019 namespace RMOL {
00021   class BasedForecasting {
00022   public:
00034     static bool forecast (stdair::SegmentCabin&, const stdair::Date_T&,
00035                           const stdair::DTD_T&,
00036                           const stdair::UnconstrainingMethod&,
00037                           const stdair::NbOfSegments_T&);
00038
00047     static void prepareHistoricalBooking
00048     (const stdair::SegmentCabin&, const stdair::BookingClass&,
00049      const stdair::SegmentSnapshotTable&, HistoricalBookingHolder&,
00050      const stdair::DCP_T&, const stdair::DCP_T&,
00051      const stdair::NbOfSegments_T&, const stdair::NbOfSegments_T&);
00052   };
00053 }
00054 #endif // __RMOL_COMMAND_BASEDFORECASTING_HPP
```

26.75 rmol/command/DemandInputPreparation.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <cmath>
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/Utilities.hpp>
#include <rmol/command/DemandInputPreparation.hpp>
```

Namespaces

- RMOL

26.76 DemandInputPreparation.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 #include <cmath>
00008 // StdAir
00009 #include <stdair/basic/BasConst_General.hpp>
00010 #include <stdair/basic/BasConst_Inventory.hpp>
00011 #include <stdair/bom/BomManager.hpp>
00012 #include <stdair/bom/SegmentCabin.hpp>
00013 #include <stdair/bom/BookingClass.hpp>
00014 #include <stdair/service/Logger.hpp>
00015 // RMOL
00016 #include <rmol/bom/Utilities.hpp>
00017 #include <rmol/command/DemandInputPreparation.hpp>
00018
00019 namespace RMOL {
00020
00021   // ////////////////////////////////////////////////////////////////////
00022   bool DemandInputPreparation::
00023   prepareDemandInput (const stdair::SegmentCabin& iSegmentCabin) {
00024     bool isSucceeded = true;
00025
00026     // Browse the list of booking classes and sum the price-oriented
00027     // demand foreast and the product-oriented demand forecast.
00028     const stdair::BookingClassList_T& lBCList =
00029       stdair::BomManager::getList<stdair::BookingClass> (iSegmentCabin);
00030     for (stdair::BookingClassList_T::const_iterator itBC = lBCList.begin();
00031          itBC != lBCList.end(); ++itBC) {
00032       stdair::BookingClass* lBC_ptr = *itBC;
00033       assert (lBC_ptr != NULL);
00034
00035       const stdair::MeanValue_T& lPriceDemMean = lBC_ptr->getPriceDemMean();
00036       const stdair::StdDevValue_T& lPriceStdDev = lBC_ptr->getPriceDemStdDev();
00037       const stdair::MeanValue_T& lProductDemMean = lBC_ptr->getProductDemMean();
00038       const stdair::StdDevValue_T& lProductStdDev =
00039         lBC_ptr->getProductDemStdDev();
00040
00041       const stdair::MeanValue_T lNewMeanValue = lPriceDemMean + lProductDemMean;
00042       const stdair::StdDevValue_T lNewStdDev =
00043         std::sqrt(lPriceStdDev*lPriceStdDev + lProductStdDev*lProductStdDev);
00044
00045       lBC_ptr->setMean (lNewMeanValue);
00046       lBC_ptr->setStdDev (lNewStdDev);
00047     }
00048
00049     return isSucceeded;
00050   }
00051
00052 }
```

26.77 rmol/command/DemandInputPreparation.hpp File Reference

```
#include <map>
#include <stdair/stdair_inventory_types.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::DemandInputPreparation

Namespaces

- stdair

  Forward declarations.

- RMOL

26.78 DemandInputPreparation.hpp

```
00001 #ifndef __RMOL_COMMAND_DEMANDINPUTPREPARATION_HPP
00002 #define __RMOL_COMMAND_DEMANDINPUTPREPARATION_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <map>
00009 // StdAir
00010 #include <stdair/stdair_inventory_types.hpp>
00011 // RMOL
00012 #include <rmol/RMOL_Types.hpp>
00013
00014 // Forward declarations
00015 namespace stdair {
00016   class SegmentCabin;
00017 }
00018
00019 namespace RMOL {
00021   class DemandInputPreparation {
00022   public:
00026     static bool prepareDemandInput (const stdair::SegmentCabin&);
00027   };
00028 }
00029 #endif // __RMOL_COMMAND_DEMANDINPUTPREPARATION_HPP
```

26.79 rmol/command/Detruncator.cpp File Reference

```
#include <cassert>
#include <stdair/basic/UnconstrainingMethod.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/HistoricalBookingHolder.hpp>
#include <rmol/bom/EMDetruncator.hpp>
#include <rmol/command/Detruncator.hpp>
```

Namespaces

- RMOL

26.80 Detruncator.cpp

```
00001 // //////////////////////////////////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 // StdAir
00007 #include <stdair/basic/UnconstrainingMethod.hpp>
00008 #include <stdair/service/Logger.hpp>
00009 // RMOL
00010 #include <rmol/bom/HistoricalBookingHolder.hpp>
00011 #include <rmol/bom/EMDetruncator.hpp>
00012 #include <rmol/command/Detruncator.hpp>
00013
00014 namespace RMOL {
00015   // //////////////////////////////////////////////////////////////////
00016   void Detruncator::
00017   unconstrain (HistoricalBookingHolder& ioHBHolder,
00018                const stdair::UnconstrainingMethod& iMethod) {
00019     const stdair::UnconstrainingMethod::EN_UnconstrainingMethod& lUnconstrainingMethod =
00020       iMethod.getMethod();
00021     switch (lUnconstrainingMethod) {
00022     case stdair::UnconstrainingMethod::EM: {
00023       EMDetruncator::unconstrain (ioHBHolder);
00024       break;
00025     }
00026     default: {
00027       assert (false);
00028       break;
00029     }
00030     }
00031   }
00032 }
00033
```

26.81 rmol/command/Detruncator.hpp File Reference

```
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/basic/UnconstrainingMethod.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::Detruncator

Namespaces

- RMOL

26.82 Detruncator.hpp

```
00001 #ifndef __RMOL_COMMAND_DETRUNCATOR_HPP
00002 #define __RMOL_COMMAND_DETRUNCATOR_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // StdAir
00008 #include <stdair/stdair_inventory_types.hpp>
00009 #include <stdair/basic/UnconstrainingMethod.hpp>
00010 // RMOL
00011 #include <rmol/RMOL_Types.hpp>
00012
00013
00014 namespace RMOL {
00015   // Forward declarations.
00016   struct HistoricalBookingHolder;
00017
00020   class Detruncator {
00021   public:
00025     static void unconstrain (HistoricalBookingHolder&,
```

```
00026                 const stdair::UnconstrainingMethod&);
00027
00028   };
00029 }
00030 #endif // __RMOL_COMMAND_DETRUNCATOR_HPP
00031
00032
```

26.83 rmol/command/FareAdjustment.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <cmath>
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/Utilities.hpp>
#include <rmol/command/FareAdjustment.hpp>
```

Namespaces

- RMOL

26.84 FareAdjustment.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 #include <cmath>
00008 // StdAir
00009 #include <stdair/basic/BasConst_General.hpp>
00010 #include <stdair/basic/BasConst_Inventory.hpp>
00011 #include <stdair/bom/BomManager.hpp>
00012 #include <stdair/bom/SegmentCabin.hpp>
00013 #include <stdair/bom/BookingClass.hpp>
00014 #include <stdair/service/Logger.hpp>
00015 // RMOL
00016 #include <rmol/bom/Utilities.hpp>
00017 #include <rmol/command/FareAdjustment.hpp>
00018
00019 namespace RMOL {
00020
00021   // ////////////////////////////////////////////////////////////////////
00022   bool FareAdjustment::
00023   adjustYield (const stdair::SegmentCabin& iSegmentCabin) {
00024     return false;
00025   }
00026
00027 }
```

26.85 rmol/command/FareAdjustment.hpp File Reference

```
#include <map>
#include <stdair/stdair_inventory_types.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::FareAdjustment

Namespaces

- stdair

  Forward declarations.
- RMOL

26.86 FareAdjustment.hpp

```
00001 #ifndef __RMOL_COMMAND_FAREADJUSTMENT_HPP
00002 #define __RMOL_COMMAND_FAREADJUSTMENT_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <map>
00009 // StdAir
00010 #include <stdair/stdair_inventory_types.hpp>
00011 // RMOL
00012 #include <rmol/RMOL_Types.hpp>
00013
00014 // Forward declarations
00015 namespace stdair {
00016   class SegmentCabin;
00017 }
00018
00019 namespace RMOL {
00021   class FareAdjustment {
00022   public:
00026     static bool adjustYield (const stdair::SegmentCabin&);
00027   };
00028 }
00029 #endif // __RMOL_COMMAND_FAREADJUSTMENT_HPP
```

26.87 rmol/command/Forecaster.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <cmath>
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/FlightDate.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/SegmentSnapshotTable.hpp>
#include <stdair/bom/FareFamily.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/Utilities.hpp>
#include <rmol/bom/SegmentSnapshotTableHelper.hpp>
#include <rmol/bom/HistoricalBookingHolder.hpp>
#include <rmol/bom/HistoricalBooking.hpp>
#include <rmol/command/BasedForecasting.hpp>
#include <rmol/command/Forecaster.hpp>
#include <rmol/command/QForecasting.hpp>
#include <rmol/command/HybridForecasting.hpp>
#include <rmol/command/OldQFF.hpp>
#include <rmol/command/NewQFF.hpp>
```

Namespaces

- RMOL

26.88 Forecaster.cpp

```
00001 // //////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 #include <cmath>
00008 // StdAir
00009 #include <stdair/basic/BasConst_General.hpp>
00010 #include <stdair/basic/BasConst_Inventory.hpp>
00011 #include <stdair/bom/BomManager.hpp>
00012 #include <stdair/bom/FlightDate.hpp>
00013 #include <stdair/bom/SegmentDate.hpp>
00014 #include <stdair/bom/SegmentCabin.hpp>
00015 #include <stdair/bom/SegmentSnapshotTable.hpp>
00016 #include <stdair/bom/FareFamily.hpp>
00017 #include <stdair/bom/BookingClass.hpp>
00018 #include <stdair/service/Logger.hpp>
00019 // RMOL
00020 #include <rmol/bom/Utilities.hpp>
00021 #include <rmol/bom/SegmentSnapshotTableHelper.hpp>
00022 #include <rmol/bom/HistoricalBookingHolder.hpp>
00023 #include <rmol/bom/HistoricalBooking.hpp>
00024 #include <rmol/command/BasedForecasting.hpp>
00025 #include <rmol/command/Forecaster.hpp>
00026 #include <rmol/command/QForecasting.hpp>
00027 #include <rmol/command/HybridForecasting.hpp>
00028 #include <rmol/command/OldQFF.hpp>
00029 #include <rmol/command/NewQFF.hpp>
00030
00031 namespace RMOL {
00032
00033   // //////////////////////////////////////////////////////////////////////
00034   bool Forecaster::
00035   forecast (stdair::FlightDate& ioFlightDate,
00036             const stdair::DateTime_T& iEventTime,
00037             const stdair::UnconstrainingMethod& iUnconstrainingMethod,
00038             const stdair::ForecastingMethod& iForecastingMethod) {
00039     // Build the offset dates.
00040     const stdair::Date_T& lEventDate = iEventTime.date();
00041
00042     //
00043     bool isSucceeded = true;
00044     const stdair::SegmentDateList_T& lSDList =
00045       stdair::BomManager::getList<stdair::SegmentDate> (ioFlightDate);
00046     for (stdair::SegmentDateList_T::const_iterator itSD = lSDList.begin();
00047          itSD != lSDList.end(); ++itSD) {
00048       stdair::SegmentDate* lSD_ptr = *itSD;
00049       assert (lSD_ptr != NULL);
00050
00051       // TODO: Take into account the case where the segment departure date
00052       // is not the same as the flight departure date.
00053       // const stdair::Date_T& lBoardingDate = lSD_ptr->getBoardingDate();
00054       // const stdair::DateOffset_T lSegmentDateOffset =
00055       //   lBoardingDate - lEventDate;
00056       // const stdair::DTD_T lSegmentDTD = lSegmentDateOffset.days();
00057
00058       //
00059       const stdair::SegmentCabinList_T& lSCList =
00060         stdair::BomManager::getList<stdair::SegmentCabin> (*lSD_ptr);
00061       for (stdair::SegmentCabinList_T::const_iterator itSC = lSCList.begin();
00062            itSC != lSCList.end(); ++itSC) {
00063         stdair::SegmentCabin* lSC_ptr = *itSC;
00064         assert (lSC_ptr != NULL);
00065
00066         //
00067         // STDAIR_LOG_NOTIFICATION (ioFlightDate.getDepartureDate()
00068         //                          << ";" << lSegmentDTD);
00069         bool isForecasted = forecast (*lSC_ptr, lEventDate,
00070                                       iUnconstrainingMethod,
00071                                       iForecastingMethod);
00072         if (isForecasted == false) {
00073           isSucceeded = false;
00074         }
00075       }
00076     }
00077
```

```
00078     return isSucceeded;
00079   }
00080
00081   // ////////////////////////////////////////////////////////////////////
00082   bool Forecaster::
00083   forecast (stdair::SegmentCabin& ioSegmentCabin,
00084             const stdair::Date_T& iEventDate,
00085             const stdair::UnconstrainingMethod& iUnconstrainingMethod,
00086             const stdair::ForecastingMethod& iForecastingMethod) {
00087     // Retrieve the number of departed similar segments.
00088     stdair::NbOfSegments_T lNbOfDepartedSegments =
00089       Utilities::getNbOfDepartedSimilarSegments (ioSegmentCabin,
      iEventDate);
00090
00091     // DEBUG
00092     // STDAIR_LOG_DEBUG ("Nb of similar departed segments: "
00093     //                   << lNbOfDepartedSegments);
00094
00095     // If the number of departed segments are less than two, there
00096     // will be no forecast, and thus no optimisation.
00097     if (lNbOfDepartedSegments < 2) {
00098       return false;
00099     } else {
00100       setDemandForecastsToZero (ioSegmentCabin);
00101       const stdair::SegmentDate& lSegmentDate =
00102         stdair::BomManager::getParent<stdair::SegmentDate> (ioSegmentCabin);
00103       const stdair::Date_T& lBoardingDate = lSegmentDate.getBoardingDate();
00104       const stdair::DateOffset_T lDateOffset = lBoardingDate - iEventDate;
00105       const stdair::DTD_T& lDaysBeforeDeparture = lDateOffset.days();
00106
00107       // If the forecasting method is QFF (old or new), but there are
00108       // not more than two fare families in the cabin, hybrid
00109       // forecasting will be used.
00110       const stdair::ForecastingMethod::EN_ForecastingMethod lForecastingMethod =
00111         iForecastingMethod.getMethod();
00112       switch (lForecastingMethod) {
00113       case stdair::ForecastingMethod::Q_FORECASTING: {
00114         return QForecasting::forecast (ioSegmentCabin, iEventDate,
00115                                        lDaysBeforeDeparture,
00116                                        iUnconstrainingMethod,
00117                                        lNbOfDepartedSegments);
00118       }
00119       case stdair::ForecastingMethod::HYBRID_FORECASTING: {
00120         return HybridForecasting::forecast (ioSegmentCabin, iEventDate,
00121                                             lDaysBeforeDeparture,
00122                                             iUnconstrainingMethod,
00123                                             lNbOfDepartedSegments);
00124       }
00125       case stdair::ForecastingMethod::NEW_QFF: {
00126         if (ioSegmentCabin.getFareFamilyStatus()==false) {
00127
00128           return HybridForecasting::forecast (ioSegmentCabin, iEventDate,
00129                                               lDaysBeforeDeparture,
00130                                               iUnconstrainingMethod,
00131                                               lNbOfDepartedSegments);
00132         } else {
00133           return NewQFF::forecast (ioSegmentCabin, iEventDate,
00134                                    lDaysBeforeDeparture, iUnconstrainingMethod,
00135                                    lNbOfDepartedSegments);
00136         }
00137       }
00138       case stdair::ForecastingMethod::OLD_QFF: {
00139         if (ioSegmentCabin.getFareFamilyStatus()==false) {
00140
00141           return HybridForecasting::forecast (ioSegmentCabin, iEventDate,
00142                                               lDaysBeforeDeparture,
00143                                               iUnconstrainingMethod,
00144                                               lNbOfDepartedSegments);
00145         } else {
00146           return OldQFF::forecast (ioSegmentCabin, iEventDate,
00147                                    lDaysBeforeDeparture, iUnconstrainingMethod,
00148                                    lNbOfDepartedSegments);
00149         }
00150       }
00151       case stdair::ForecastingMethod::BASED_FORECASTING: {
00152         return BasedForecasting::forecast (ioSegmentCabin, iEventDate,
00153                                            lDaysBeforeDeparture,
00154                                            iUnconstrainingMethod,
00155                                            lNbOfDepartedSegments);
00156       }
00157       default:{
00158         assert (false);
00159         break;
00160       }
00161       }
00162       return false;
00163     }
```

```
00164   }
00165
00166   // ////////////////////////////////////////////////////////////////////
00167   void Forecaster::
00168   setDemandForecastsToZero(const stdair::SegmentCabin& iSegmentCabin) {
00169     // Set the demand forecast for all classes and fare families to zero.
00170     const stdair::FareFamilyList_T& lFFList =
00171       stdair::BomManager::getList<stdair::FareFamily> (iSegmentCabin);
00172     for (stdair::FareFamilyList_T::const_iterator itFF = lFFList.begin();
00173          itFF != lFFList.end(); ++itFF) {
00174       stdair::FareFamily* lFF_ptr = *itFF;
00175       assert (lFF_ptr != NULL);
00176       lFF_ptr->setMean (0.0);
00177       lFF_ptr->setStdDev (0.0);
00178
00179       const stdair::BookingClassList_T& lBCList =
00180         stdair::BomManager::getList<stdair::BookingClass> (*lFF_ptr);
00181       for (stdair::BookingClassList_T::const_iterator itBC = lBCList.begin();
00182            itBC != lBCList.end(); ++itBC) {
00183         stdair::BookingClass* lBC_ptr = *itBC;
00184         assert (lBC_ptr != NULL);
00185         lBC_ptr->setMean (0.0);
00186         lBC_ptr->setStdDev (0.0);
00187         lBC_ptr->setPriceDemMean (0.0);
00188         lBC_ptr->setPriceDemStdDev (0.0);
00189         lBC_ptr->setProductDemMean (0.0);
00190         lBC_ptr->setProductDemStdDev (0.0);
00191         lBC_ptr->setCumuPriceDemMean (0.0);
00192         lBC_ptr->setCumuPriceDemStdDev (0.0);
00193       }
00194     }
00195   }
00196 }
```

26.89 rmol/command/Forecaster.hpp File Reference

```
#include <map>
#include <stdair/stdair_inventory_types.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::Forecaster

Namespaces

- stdair

  Forward declarations.
- RMOL

26.90 Forecaster.hpp

```
00001 #ifndef __RMOL_COMMAND_FORECASTER_HPP
00002 #define __RMOL_COMMAND_FORECASTER_HPP
00003
00004 // ////////////////////////////////////////////////////////////////////
00005 // Import section
00006 // ////////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <map>
00009 // StdAir
00010 #include <stdair/stdair_inventory_types.hpp>
00011 // RMOL
00012 #include <rmol/RMOL_Types.hpp>
00013
00014 // Forward declarations
00015 namespace stdair {
00016   class FlightDate;
00017   class SegmentCabin;
00018 }
00019
```

```
00020 namespace RMOL {
00022   class Forecaster {
00023   public:
00027     static bool forecast (stdair::FlightDate&, const stdair::DateTime_T&,
00028                           const stdair::UnconstrainingMethod&,
00029                           const stdair::ForecastingMethod&);
00030
00031   private:
00035     static bool forecast (stdair::SegmentCabin&, const stdair::Date_T&,
00036                           const stdair::UnconstrainingMethod&,
00037                           const stdair::ForecastingMethod&);
00038
00042     static void setDemandForecastsToZero (const stdair::SegmentCabin&);
00043
00044   };
00045 }
00046 #endif // __RMOL_COMMAND_FORECASTER_HPP
```

26.91 rmol/command/HybridForecasting.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <cmath>
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/basic/RandomGeneration.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/LegDate.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/SegmentSnapshotTable.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/Utilities.hpp>
#include <rmol/bom/SegmentSnapshotTableHelper.hpp>
#include <rmol/bom/HistoricalBookingHolder.hpp>
#include <rmol/bom/HistoricalBooking.hpp>
#include <rmol/command/QForecasting.hpp>
#include <rmol/command/HybridForecasting.hpp>
#include <rmol/command/Detruncator.hpp>
```

Namespaces

- RMOL

26.92 HybridForecasting.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 #include <cmath>
00008 // StdAir
00009 #include <stdair/basic/BasConst_General.hpp>
00010 #include <stdair/basic/BasConst_Inventory.hpp>
00011 #include <stdair/basic/RandomGeneration.hpp>
00012 #include <stdair/bom/BomManager.hpp>
00013 #include <stdair/bom/LegDate.hpp>
00014 #include <stdair/bom/SegmentDate.hpp>
00015 #include <stdair/bom/LegCabin.hpp>
00016 #include <stdair/bom/SegmentCabin.hpp>
00017 #include <stdair/bom/SegmentSnapshotTable.hpp>
00018 #include <stdair/bom/BookingClass.hpp>
```

```
00019 #include <stdair/service/Logger.hpp>
00020 // RMOL
00021 #include <rmol/bom/Utilities.hpp>
00022 #include <rmol/bom/SegmentSnapshotTableHelper.hpp>
00023 #include <rmol/bom/HistoricalBookingHolder.hpp>
00024 #include <rmol/bom/HistoricalBooking.hpp>
00025 #include <rmol/command/QForecasting.hpp>
00026 #include <rmol/command/HybridForecasting.hpp>
00027 #include <rmol/command/Detruncator.hpp>
00028
00029 namespace RMOL {
00030   // ////////////////////////////////////////////////////////////////////
00031   bool HybridForecasting::
00032   forecast (stdair::SegmentCabin& ioSegmentCabin,
00033             const stdair::Date_T& iCurrentDate,
00034             const stdair::DTD_T& iCurrentDTD,
00035             const stdair::UnconstrainingMethod& iUnconstrainingMethod,
00036             const stdair::NbOfSegments_T& iNbOfDepartedSegments) {
00037     // Call QForecasting to treat the price-oriented demand.
00038     QForecasting::forecast (ioSegmentCabin, iCurrentDate, iCurrentDTD,
00039                             iUnconstrainingMethod, iNbOfDepartedSegments);
00040
00041     // Retrieve the snapshot table.
00042     const stdair::SegmentSnapshotTable& lSegmentSnapshotTable =
00043       ioSegmentCabin.getSegmentSnapshotTable();
00044
00045     // Retrieve the booking class list.
00046     const stdair::BookingClassList_T& lBCList =
00047       stdair::BomManager::getList<stdair::BookingClass>(ioSegmentCabin);
00048
00049     // Browse all remaining DCP's and do unconstraining, forecasting for
00050     // all product-oriented demand.
00051     const stdair::DCPList_T lWholeDCPList = stdair::DEFAULT_DCP_LIST;
00052     stdair::DCPList_T::const_iterator itDCP = lWholeDCPList.begin();
00053     stdair::DCPList_T::const_iterator itNextDCP = itDCP; ++itNextDCP;
00054     for (; itNextDCP != lWholeDCPList.end(); ++itDCP, ++itNextDCP) {
00055       const stdair::DCP_T& lCurrentDCP = *itDCP;
00056       const stdair::DCP_T& lNextDCP = *itNextDCP;
00057
00058       // The end of the interval is after the current DTD.
00059       if (lNextDCP < iCurrentDTD) {
00060         // Get the number of similar segments which has already passed the
00061         // (lNextDCP+1)
00062         const stdair::NbOfSegments_T& lNbOfUsableSegments =
00063           SegmentSnapshotTableHelper::
00064           getNbOfSegmentAlreadyPassedThisDTD (
    lSegmentSnapshotTable,
00065                                               lNextDCP+1,
00066                                               iCurrentDate);
00067         stdair::NbOfSegments_T lSegmentBegin = 0;
00068         const stdair::NbOfSegments_T lSegmentEnd = lNbOfUsableSegments-1;
00069         if (iNbOfDepartedSegments > 52) {
00070           lSegmentBegin = iNbOfDepartedSegments - 52;
00071         }
00072
00073         // Browse the list of booking classes and forecast the product-oriented
00074         // demand for each class.
00075         for (stdair::BookingClassList_T::const_iterator itBC = lBCList.begin();
00076              itBC != lBCList.end(); ++itBC) {
00077           stdair::BookingClass* lBC_ptr = *itBC;
00078           assert (lBC_ptr != NULL);
00079
00080           // Retrieve the historical product-oriented bookings for the
00081           // given class.
00082           HistoricalBookingHolder lHBHolder;
00083           prepareProductOrientedHistoricalBooking (ioSegmentCabin, *
    lBC_ptr,
00084                                                    lSegmentSnapshotTable,
00085                                                    lHBHolder,
00086                                                    lCurrentDCP, lNextDCP,
00087                                                    lSegmentBegin, lSegmentEnd);
00088
00089           // Unconstrain the historical bookings.
00090           Detruncator::unconstrain (lHBHolder, iUnconstrainingMethod);
00091
00092           // Retrieve the historical unconstrained demand and perform the
00093           // forecasting.
00094           stdair::UncDemVector_T lUncDemVector;
00095           const short lNbOfHistoricalFlights = lHBHolder.getNbOfFlights();
00096           for (short i = 0; i < lNbOfHistoricalFlights; ++i) {
00097             const stdair::NbOfBookings_T& lUncDemand =
00098               lHBHolder.getUnconstrainedDemand (i);
00099             lUncDemVector.push_back (lUncDemand);
00100           }
00101           stdair::MeanValue_T lMean = 0.0;
00102           stdair::StdDevValue_T lStdDev = 0.0;
00103           Utilities::computeDistributionParameters (lUncDemVector,
```

```
00104                                                 lMean, lStdDev);
00105
00106           // Add the demand forecast to the booking class.
00107           const stdair::MeanValue_T& lCurrentMean = lBC_ptr->getProductDemMean();
00108           const stdair::StdDevValue_T& lCurrentStdDev =
00109             lBC_ptr->getProductDemStdDev();
00110
00111           const stdair::MeanValue_T lNewMean = lCurrentMean + lMean;
00112           const stdair::StdDevValue_T lNewStdDev =
00113             std::sqrt (lCurrentStdDev * lCurrentStdDev + lStdDev * lStdDev);
00114
00115           lBC_ptr->setProductDemMean (lNewMean);
00116           lBC_ptr->setProductDemStdDev (lNewStdDev);
00117         }
00118       }
00119     }
00120     return true;
00121   }
00122
00123   // ////////////////////////////////////////////////////////////////////
00124   void HybridForecasting::prepareProductOrientedHistoricalBooking
00125     (const stdair::SegmentCabin& iSegmentCabin,
00126      const stdair::BookingClass& iBookingClass,
00127      const stdair::SegmentSnapshotTable& iSegmentSnapshotTable,
00128      HistoricalBookingHolder& ioHBHolder,
00129      const stdair::DCP_T& iDCPBegin, const stdair::DCP_T& iDCPEnd,
00130      const stdair::NbOfSegments_T& iSegmentBegin,
00131      const stdair::NbOfSegments_T& iSegmentEnd) {
00132
00133     // Retrieve the booking class index within the snapshot table
00134     const stdair::ClassIndex_T& lClassIdx =
00135       iSegmentSnapshotTable.getClassIndex (iBookingClass.describeKey());
00136
00137     // Retrieve the gross daily booking and availability snapshots.
00138     const stdair::ConstSegmentCabinDTDRangeSnapshotView_T lBookingView =
00139       iSegmentSnapshotTable.getConstSegmentCabinDTDRangeProductOrientedGrossBookingSnapshotView (
      iSegmentBegin, iSegmentEnd, iDCPEnd, iDCPBegin);
00140     const stdair::ConstSegmentCabinDTDRangeSnapshotView_T lAvlView =
00141       iSegmentSnapshotTable.getConstSegmentCabinDTDRangeAvailabilitySnapshotView (iSegmentBegin,
      iSegmentEnd, iDCPEnd, iDCPBegin);
00142
00143     // Browse the list of segments and build the historical booking holder.
00144     const stdair::ClassIndexMap_T& lVTIdxMap =
00145       iSegmentSnapshotTable.getClassIndexMap();
00146     const stdair::NbOfClasses_T lNbOfClasses = lVTIdxMap.size();
00147
00148     for (short i = 0; i <= iSegmentEnd-iSegmentBegin; ++i) {
00149       stdair::Flag_T lCensorshipFlag = false;
00150       const short lNbOfDTDs = iDCPBegin - iDCPEnd + 1;
00151       const stdair::UnsignedIndex_T lIdx = i*lNbOfClasses + lClassIdx;
00152
00153       // Parse the DTDs during the period and compute the censorship flag
00154       for (short j = 0; j < lNbOfDTDs; ++j) {
00155         // Check if the data has been censored during this day.
00156         // STDAIR_LOG_DEBUG ("i: " << i << ", NbOfClasses: " << lNbOfClasses
00157         //                   << ", ClassIdx: " << iClassIdx << ", j: " << j);
00158         if (lAvlView[lIdx][j] < 1.0) {
00159           lCensorshipFlag = true;
00160           break;
00161         }
00162       }
00163
00164       // Retrieve the historical product-oriented bookings
00165       stdair::NbOfBookings_T lNbOfHistoricalBkgs = 0.0;
00166       for (short j = 0; j < lNbOfDTDs; ++j) {
00167         lNbOfHistoricalBkgs += lBookingView[lIdx][j];
00168       }
00169       HistoricalBooking lHistoricalBkg (lNbOfHistoricalBkgs, lCensorshipFlag);
00170       ioHBHolder.addHistoricalBooking (lHistoricalBkg);
00171     }
00172   }
00173
00174 }
```

26.93 rmol/command/HybridForecasting.hpp File Reference

```
#include <stdair/stdair_inventory_types.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::HybridForecasting

Namespaces

- stdair

  Forward declarations.

- RMOL

26.94 HybridForecasting.hpp

```
00001 #ifndef __RMOL_COMMAND_HYBRIDFORECASTING_HPP
00002 #define __RMOL_COMMAND_HYBRIDFORECASTING_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // StdAir
00008 #include <stdair/stdair_inventory_types.hpp>
00009 // RMOL
00010 #include <rmol/RMOL_Types.hpp>
00011
00012 // Forward declarations
00013 namespace stdair {
00014   class SegmentCabin;
00015   class BookingClass;
00016   class SegmentSnapshotTable;
00017 }
00018
00019 namespace RMOL {
00021   class HybridForecasting {
00022   public:
00032     static bool forecast (stdair::SegmentCabin&, const stdair::Date_T&,
00033                           const stdair::DTD_T&,
00034                           const stdair::UnconstrainingMethod&,
00035                           const stdair::NbOfSegments_T&);
00036
00045     static void prepareProductOrientedHistoricalBooking
00046     (const stdair::SegmentCabin&, const stdair::BookingClass&,
00047      const stdair::SegmentSnapshotTable&, HistoricalBookingHolder&,
00048      const stdair::DCP_T&, const stdair::DCP_T&,
00049      const stdair::NbOfSegments_T&, const stdair::NbOfSegments_T&);
00050   };
00051 }
00052 #endif // __RMOL_COMMAND_HYBRIDFORECASTING_HPP
```

26.95 rmol/command/InventoryParser.cpp File Reference

```
#include <sstream>
```

```
#include <fstream>
#include <cassert>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/stdair_maths_types.hpp>
#include <stdair/stdair_exceptions.hpp>
#include <stdair/basic/BasConst_DefaultObject.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/basic/BasFileMgr.hpp>
#include <stdair/bom/BomRetriever.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/BomRoot.hpp>
#include <stdair/bom/Inventory.hpp>
#include <stdair/bom/FlightDate.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/LegDate.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/bom/VirtualClassStruct.hpp>
#include <stdair/factory/FacBom.hpp>
#include <stdair/factory/FacBomManager.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/command/InventoryParser.hpp>
```

Namespaces

- RMOL

26.96 InventoryParser.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <sstream>
00006 #include <fstream>
00007 #include <cassert>
00008 // StdAir
00009 #include <stdair/stdair_inventory_types.hpp>
00010 #include <stdair/stdair_maths_types.hpp>
00011 #include <stdair/stdair_exceptions.hpp>
00012 #include <stdair/basic/BasConst_DefaultObject.hpp>
00013 #include <stdair/basic/BasConst_Inventory.hpp>
00014 #include <stdair/basic/BasFileMgr.hpp>
00015 #include <stdair/bom/BomRetriever.hpp>
00016 #include <stdair/bom/BomManager.hpp>
00017 #include <stdair/bom/BomRoot.hpp>
00018 #include <stdair/bom/Inventory.hpp>
00019 #include <stdair/bom/FlightDate.hpp>
00020 #include <stdair/bom/SegmentDate.hpp>
00021 #include <stdair/bom/SegmentCabin.hpp>
00022 #include <stdair/bom/LegDate.hpp>
00023 #include <stdair/bom/LegCabin.hpp>
00024 #include <stdair/bom/BookingClass.hpp>
00025 #include <stdair/bom/VirtualClassStruct.hpp>
00026 #include <stdair/factory/FacBom.hpp>
00027 #include <stdair/factory/FacBomManager.hpp>
00028 #include <stdair/service/Logger.hpp>
00029 // RMOL
00030 #include <rmol/command/InventoryParser.hpp>
00031
00032 namespace RMOL {
00033
00034   // ////////////////////////////////////////////////////////////////////
00035   bool InventoryParser::
00036   parseInputFileAndBuildBom (const std::string& iInputFileName,
00037                              stdair::BomRoot& ioBomRoot) {
00038     bool hasReadBeenSuccessful = false;
00039
```

```
00040     // Check that the file path given as input corresponds to an actual file
00041     const bool doesExistAndIsReadable =
00042       stdair::BasFileMgr::doesExistAndIsReadable (iInputFileName);
00043     if (doesExistAndIsReadable == false) {
00044       std::ostringstream oMessage;
00045       oMessage << "The input file, '" << iInputFileName
00046                << "', can not be retrieved on the file-system";
00047       throw stdair::FileNotFoundException (oMessage.str());
00048     }
00049
00050     // Retrieve the (sample) leg-cabin
00051     stdair::LegCabin& lLegCabin =
00052       stdair::BomRetriever::retrieveDummyLegCabin (ioBomRoot);
00053
00054     // Retrieve the (sample) segment-cabin
00055     stdair::SegmentCabin& lSegmentCabin =
00056       stdair::BomRetriever::retrieveDummySegmentCabin (ioBomRoot);
00057
00058     // Open the input file
00059     std::ifstream inputFile (iInputFileName.c_str());
00060     if (! inputFile) {
00061       STDAIR_LOG_ERROR ("Can not open input file '" << iInputFileName << "'");
00062       throw new stdair::FileNotFoundException ("Can not open input file '"
00063                                                + iInputFileName + "'");
00064     }
00065
00066     char buffer[80];
00067     double dval;
00068     short i = 1;
00069     bool hasAllPArams = true;
00070     stdair::Yield_T lYield;
00071     stdair::MeanValue_T lMean;
00072     stdair::StdDevValue_T lStdDev;
00073     stdair::BookingClassKey lBCKey (stdair::DEFAULT_CLASS_CODE);
00074
00075     while (inputFile.getline (buffer, sizeof (buffer), ';')) {
00076       std::istringstream iStringStr (buffer);
00077
00078       if (i == 1) {
00079         hasAllPArams = true;
00080       }
00081
00082       if (iStringStr >> dval) {
00083         if (i == 1) {
00084           lYield = dval;
00085           // std::cout << "Yield[" << i << "] = '" << dval << "'" << std::endl;
00086
00087         } else if (i == 2) {
00088           lMean = dval;
00089           // std::cout << "Mean[" << i << "] = '" << dval << "'" << std::endl;
00090
00091         } else if (i == 3) {
00092           lStdDev = dval;
00093           //std::cout << "stdDev[" << i << "] = '" << dval << "'" << std::endl;
00094           i = 0;
00095         }
00096         i++;
00097
00098       } else {
00099         hasAllPArams = false;
00100       }
00101
00102       if (hasAllPArams && i == 1) {
00103         stdair::BookingClass& lBookingClass =
00104           stdair::FacBom<stdair::BookingClass>::instance().create (lBCKey);
00105         stdair::FacBomManager::addToList (lSegmentCabin, lBookingClass);
00106         lBookingClass.setYield (lYield);
00107         lBookingClass.setMean (lMean);
00108         lBookingClass.setStdDev (lStdDev);
00109         stdair::BookingClassList_T lBookingClassList;
00110         lBookingClassList.push_back(&lBookingClass);
00111         stdair::VirtualClassStruct lVirtualClass (lBookingClassList);
00112         lVirtualClass.setYield (lYield);
00113         lVirtualClass.setMean (lMean);
00114         lVirtualClass.setStdDev (lStdDev);
00115         lLegCabin.addVirtualClass (lVirtualClass);
00116       }
00117     }
00118
00119     //
00120     if (!inputFile.eof()) {
00121       STDAIR_LOG_ERROR ("Problem when reading input file '" << iInputFileName
00122                         << "'");
00123       return hasReadBeenSuccessful;
00124     }
00125
00126     //
```

```
00127     hasReadBeenSuccessful = true;
00128     return hasReadBeenSuccessful;
00129   }
00130
00131 }
```

26.97 rmol/command/InventoryParser.hpp File Reference

```
#include <string>
#include <stdair/command/CmdAbstract.hpp>
```

Classes

- class RMOL::InventoryParser

  Class filling the virtual class list (representing a list of classes/buckets) from a given input inventory.

Namespaces

- stdair

  Forward declarations.

- RMOL

26.98 InventoryParser.hpp

```
00001 #ifndef __RMOL_CMD_INVENTORYPARSER_HPP
00002 #define __RMOL_CMD_INVENTORYPARSER_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <string>
00009 // StdAir
00010 #include <stdair/command/CmdAbstract.hpp>
00011
00013 namespace stdair {
00014   class BomRoot;
00015   class LegCabin;
00016   class SegmentCabin;
00017 }
00018
00019 namespace RMOL {
00020
00025   class InventoryParser : public stdair::CmdAbstract {
00026   public:
00027
00035     static bool parseInputFileAndBuildBom (const std::string& iInputFileName,
00036                                            stdair::BomRoot&);
00037   };
00038 }
00039 #endif // __RMOL_CMD_INVENTORYPARSER_HPP
```

26.99 rmol/command/MarginalRevenueTransformation.cpp File Reference

```
#include <cassert>
```

```
#include <sstream>
#include <cmath>
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/bom/SimpleNestingStructure.hpp>
#include <stdair/bom/NestingNode.hpp>
#include <stdair/bom/Policy.hpp>
#include <stdair/factory/FacBomManager.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/PolicyHelper.hpp>
#include <rmol/bom/Utilities.hpp>
#include <rmol/command/MarginalRevenueTransformation.hpp>
```

Namespaces

- RMOL

26.100 MarginalRevenueTransformation.cpp

```
00001 // //////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 #include <cmath>
00008 // StdAir
00009 #include <stdair/basic/BasConst_General.hpp>
00010 #include <stdair/basic/BasConst_Inventory.hpp>
00011 #include <stdair/bom/BomManager.hpp>
00012 #include <stdair/bom/SegmentCabin.hpp>
00013 #include <stdair/bom/BookingClass.hpp>
00014 #include <stdair/bom/SimpleNestingStructure.hpp>
00015 #include <stdair/bom/NestingNode.hpp>
00016 #include <stdair/bom/Policy.hpp>
00017 #include <stdair/factory/FacBomManager.hpp>
00018 #include <stdair/service/Logger.hpp>
00019 // RMOL
00020 #include <rmol/bom/PolicyHelper.hpp>
00021 #include <rmol/bom/Utilities.hpp>
00022 #include <rmol/command/MarginalRevenueTransformation.hpp>
00023
00024 namespace RMOL {
00025
00026   // //////////////////////////////////////////////////////////////////////
00027   bool MarginalRevenueTransformation::
00028   prepareDemandInput (stdair::SegmentCabin& ioSegmentCabin) {
00029     // Build the convex hull, then adjust the yield and demand of all
00030     // classes based on the hull.
00031
00032     buildNestedConvexHull (ioSegmentCabin);
00033     bool isSucceeded = adjustYieldAndDemand (ioSegmentCabin);
00034
00035     return isSucceeded;
00036   }
00037
00038   // //////////////////////////////////////////////////////////////////////
00039   void MarginalRevenueTransformation::
00040   buildConvexHull (stdair::SegmentCabin& ioSegmentCabin) {
00041     // Reset the convex hull of the segment.
00042     ioSegmentCabin.resetConvexHull();
00043
00044     // The first (from the left side) point of the convex hull is the "empty"
00045     // policy, i.e. the one with all fare families closed.
00046     const stdair::PolicyList_T& lPolicyList =
00047       stdair::BomManager::getList<stdair::Policy> (ioSegmentCabin);
00048
00049     // By construction, the empty policy is the first one on the list of
00050     // eligible policies.
00051     stdair::PolicyList_T::const_iterator itPolicy=lPolicyList.begin();
```

```
00052     stdair::Policy* lEmptyPolicy_ptr = *itPolicy;
00053     assert (lEmptyPolicy_ptr != NULL);
00054     ioSegmentCabin.addPolicy (*lEmptyPolicy_ptr);
00055
00056     // Pointer on the current policy of the convex hull.
00057     stdair::Policy* lCurrentPolicy_ptr = lEmptyPolicy_ptr;
00058     bool lEndOfHull = false;
00059
00060     // The end of hull is reached when from the current policy, we cannot
00061     // find an other one with greater demand and total revenue.
00062     while (lEndOfHull == false) {
00063       // Demand and total revenue of the current policy.
00064       const double& lCurrentDem = lCurrentPolicy_ptr->getDemand();
00065       const double lCurrentTR = lCurrentPolicy_ptr->getTotalRevenue();
00066
00067       // Search for the next policy.
00068       double lGradient = 0.0;
00069       stdair::Policy* lNextPolicy_ptr = NULL;
00070       for (stdair::PolicyList_T::const_iterator itPol = lPolicyList.begin();
00071            itPol != lPolicyList.end(); ++itPol) {
00072         stdair::Policy* lPolicy_ptr = *itPol;
00073         assert (lPolicy_ptr != NULL);
00074
00075         const double& lDem = lPolicy_ptr->getDemand();
00076         const double lTR = lPolicy_ptr->getTotalRevenue();
00077         if (lDem > lCurrentDem && lTR > lCurrentTR) {
00078           const double lNewGradient = (lTR-lCurrentTR)/(lDem-lCurrentDem);
00079           if (lNewGradient > lGradient) {
00080             lGradient = lNewGradient;
00081             lNextPolicy_ptr = lPolicy_ptr;
00082           }
00083         }
00084       }
00085
00086       // Check if we have found the next policy
00087       if (lNextPolicy_ptr == NULL) {
00088         lEndOfHull = true;
00089       } else {
00090         ioSegmentCabin.addPolicy (*lNextPolicy_ptr);
00091         lCurrentPolicy_ptr = lNextPolicy_ptr;
00092       }
00093     }
00094   }
00095
00096   // ////////////////////////////////////////////////////////////////////
00097   void MarginalRevenueTransformation::
00098   buildNestedConvexHull (stdair::SegmentCabin& ioSegmentCabin) {
00099     // Reset the convex hull of the segment.
00100     ioSegmentCabin.resetConvexHull();
00101
00102     // The first (from the left side) point of the convex hull is the "empty"
00103     // policy, i.e. the one with all fare families closed.
00104     const stdair::PolicyList_T& lPolicyList =
00105       stdair::BomManager::getList<stdair::Policy> (ioSegmentCabin);
00106
00107     // By construction, the empty policy is the first one on the list of
00108     // eligible policies.
00109     stdair::PolicyList_T::const_iterator itPolicy=lPolicyList.begin();
00110     stdair::Policy* lEmptyPolicy_ptr = *itPolicy;
00111     assert (lEmptyPolicy_ptr != NULL);
00112     ioSegmentCabin.addPolicy (*lEmptyPolicy_ptr);
00113
00114     // Pointer on the current policy of the convex hull.
00115     stdair::Policy* lCurrentPolicy_ptr = lEmptyPolicy_ptr;
00116     bool lEndOfHull = false;
00117
00118     // The end of hull is reached when from the current policy, we cannot
00119     // find an other one with greater demand and total revenue.
00120     while (lEndOfHull == false) {
00121       // Demand and total revenue of the current policy.
00122       const double& lCurrentDem = lCurrentPolicy_ptr->getDemand();
00123       const double lCurrentTR = lCurrentPolicy_ptr->getTotalRevenue();
00124
00125       // Search for the next policy.
00126       double lGradient = 0.0;
00127       stdair::Policy* lNextPolicy_ptr = NULL;
00128       for (stdair::PolicyList_T::const_iterator itPol = lPolicyList.begin();
00129            itPol != lPolicyList.end(); ++itPol) {
00130         stdair::Policy* lPolicy_ptr = *itPol;
00131         assert (lPolicy_ptr != NULL);
00132
00133         const double& lDem = lPolicy_ptr->getDemand();
00134         const double lTR = lPolicy_ptr->getTotalRevenue();
00135         if (lDem > lCurrentDem && lTR > lCurrentTR
00136             && PolicyHelper::isNested (*lCurrentPolicy_ptr, *lPolicy_ptr)) {
00137           const double lNewGradient = (lTR-lCurrentTR)/(lDem-lCurrentDem);
00138           if (lNewGradient > lGradient) {
```

```
00139             lGradient = lNewGradient;
00140             lNextPolicy_ptr = lPolicy_ptr;
00141           }
00142         }
00143       }
00144
00145       // Check if we have found the next policy
00146       if (lNextPolicy_ptr == NULL) {
00147         lEndOfHull = true;
00148       } else {
00149         ioSegmentCabin.addPolicy (*lNextPolicy_ptr);
00150         lCurrentPolicy_ptr = lNextPolicy_ptr;
00151       }
00152     }
00153   }
00154
00155   // ////////////////////////////////////////////////////////////////////
00156   bool MarginalRevenueTransformation::
00157   adjustYieldAndDemand (stdair::SegmentCabin& ioSegmentCabin) {
00158     bool isSucceeded = false;
00159     stdair::NbOfClasses_T lBookingClassCounter = 0;
00160     // Browse the list of policies on the convex hull, compute the differences
00161     // between pairs of consecutive policies.
00162     const stdair::PolicyList_T& lConvexHull = ioSegmentCabin.getConvexHull();
00163     stdair::PolicyList_T::const_iterator itCurrentPolicy = lConvexHull.begin();
00164     assert (itCurrentPolicy != lConvexHull.end());
00165     stdair::PolicyList_T::const_iterator itNextPolicy = itCurrentPolicy;
00166     ++itNextPolicy;
00167     // If the nesting has only one element (the empty policy),
00168     // there is no optimisation and no pre-optimisation.
00169     if (itNextPolicy == lConvexHull.end()) {
00170       return isSucceeded;
00171     }
00172
00173     // Reset the yield-based nesting structure
00174     stdair::FacBomManager::resetYieldBasedNestingStructure (ioSegmentCabin);
00175
00176     // Retrieve the yield-based nesting structure.
00177     stdair::SimpleNestingStructure& lYieldBasedNS =
00178       stdair::BomManager::getObject<stdair::SimpleNestingStructure> (ioSegmentCabin,
      stdair::YIELD_BASED_NESTING_STRUCTURE_CODE);
00179     const stdair::NestingNodeList_T& lNodeList =
00180       stdair::BomManager::getList<stdair::NestingNode> (lYieldBasedNS);
00181     stdair::NestingNodeList_T::const_iterator itNode = lNodeList.begin();
00182
00183     for (; itNextPolicy != lConvexHull.end();
00184          ++itCurrentPolicy, ++itNextPolicy, ++itNode){
00185       const stdair::Policy* lCurrentPolicy_ptr = *itCurrentPolicy;
00186       assert (lCurrentPolicy_ptr != NULL);
00187       const stdair::Policy* lNextPolicy_ptr = *itNextPolicy;
00188       assert (lNextPolicy_ptr != NULL);
00189
00190       // Retrieve the node. If there isn't any node left, create new one.
00191       stdair::NestingNode* lNode_ptr = NULL;
00192       if (itNode == lNodeList.end()) {
00193         // Create a nesting node
00194         stdair::NestingNodeCode_T lNodeCode ("XXX");
00195         stdair::NestingNodeKey lNodeKey (lNodeCode);
00196         stdair::NestingNode& lNestingNode =
00197           stdair::FacBom<stdair::NestingNode>::instance().create (lNodeKey);
00198         stdair::FacBomManager::addToList (lYieldBasedNS, lNestingNode);
00199         stdair::FacBomManager::linkWithParent (lYieldBasedNS, lNestingNode);
00200         lNode_ptr = &lNestingNode;
00201       } else {
00202         lNode_ptr = *itNode;
00203       }
00204       assert (lNode_ptr != NULL);
00205       PolicyHelper::diffBetweenTwoPolicies (*lNode_ptr, *
      lNextPolicy_ptr,
00206                                             *lCurrentPolicy_ptr);
00207
00208       // Compute the adjusted yield, demand mean and demand standard deviation.
00209       // Note: because of the nature of the convex hull, in the adjusted
00210       // standard deviation computation, we can take the difference between
00211       // the squares of the standard deviations of the two policies instead of
00212       // the sum of the squares.
00213       const stdair::MeanValue_T lAdjustedDemMean =
00214         lNextPolicy_ptr->getDemand()-lCurrentPolicy_ptr->getDemand();
00215       assert (lAdjustedDemMean > 0.0);
00216       const stdair::StdDevValue_T& lCurrentStdDev =
00217         lCurrentPolicy_ptr->getStdDev();
00218       const stdair::StdDevValue_T& lNextStdDev = lNextPolicy_ptr->getStdDev();
00219       assert (lNextStdDev > lCurrentStdDev);
00220       const stdair::StdDevValue_T lAdjustedDemStdDev =
00221         std::sqrt (lNextStdDev*lNextStdDev - lCurrentStdDev*lCurrentStdDev);
00222       const stdair::Yield_T lAdjustedYield =
00223         (lNextPolicy_ptr->getTotalRevenue()-lCurrentPolicy_ptr->getTotalRevenue())/(lAdjustedDemMean);
```

```
00224       assert (lAdjustedYield > 0.0);
00225       lNode_ptr->setYield (lAdjustedYield);
00226
00227       // Browse the list of booking classes in the node. Set the adjusted yield
00228       // for each class. However, the adjusted demand forecast will be
00229       // distributed only to the first class of the list.
00230       const stdair::BookingClassList_T lBCList =
00231         stdair::BomManager::getList<stdair::BookingClass> (*lNode_ptr);
00232       stdair::BookingClassList_T::const_iterator itBC = lBCList.begin();
00233       assert (itBC != lBCList.end());
00234       stdair::BookingClass* lFirstClass = *itBC;
00235       assert (lFirstClass != NULL);
00236       lFirstClass->setMean (lAdjustedDemMean);
00237       lFirstClass->setStdDev (lAdjustedDemStdDev);
00238       for (; itBC != lBCList.end(); ++itBC) {
00239         stdair::BookingClass* lClass = *itBC;
00240         assert (lClass != NULL);
00241         lClass->setAdjustedYield (lAdjustedYield);
00242         ++lBookingClassCounter;
00243       }
00244     }
00245
00246     const stdair::BookingClassList_T& lSCBookingClassList =
00247        stdair::BomManager::getList<stdair::BookingClass> (ioSegmentCabin);
00248     const stdair::NbOfClasses_T lNbOfBookingClass = lSCBookingClassList.size();
00249     assert (lNbOfBookingClass >= lBookingClassCounter);
00250     if (lBookingClassCounter < lNbOfBookingClass) {
00251       // At the last node. All the classes which haven't been added to the
00252       // nesting structure will be added to the next nesting node, with
00253       // an adjusted yield of zero.
00254       // Retrieve the node. If there isn't any node left, create new one.
00255       stdair::NestingNode* lLastNode_ptr = NULL;
00256       if (itNode == lNodeList.end()) {
00257         // Create a nesting node
00258         stdair::NestingNodeCode_T lNodeCode ("XXX");
00259         stdair::NestingNodeKey lNodeKey (lNodeCode);
00260         stdair::NestingNode& lNestingNode =
00261           stdair::FacBom<stdair::NestingNode>::instance().create (lNodeKey);
00262         stdair::FacBomManager::addToList (lYieldBasedNS, lNestingNode);
00263         stdair::FacBomManager::linkWithParent (lYieldBasedNS, lNestingNode);
00264         lLastNode_ptr =
00265           stdair::BomManager::getObjectPtr<stdair::NestingNode>(lYieldBasedNS,
00266                                                                 lNodeKey.toString());
00267       } else {
00268         lLastNode_ptr = *itNode;
00269       }
00270       assert (lLastNode_ptr != NULL);
00271       const stdair::Policy* lLastPolicy_ptr = *itCurrentPolicy;
00272       assert (lLastPolicy_ptr != NULL);
00273       PolicyHelper::computeLastNode (*lLastNode_ptr, *lLastPolicy_ptr,
00274                                      ioSegmentCabin);
00275     }
00276
00277     isSucceeded = true;
00278     return isSucceeded;
00279   }
00280
00281 }
```

26.101 rmol/command/MarginalRevenueTransformation.hpp File Reference

```
#include <map>
#include <stdair/stdair_inventory_types.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::MarginalRevenueTransformation

Namespaces

- stdair

  Forward declarations.
- RMOL

26.102 MarginalRevenueTransformation.hpp

```
00001 #ifndef __RMOL_COMMAND_MARGINALREVENUETRANSFORMATION_HPP
00002 #define __RMOL_COMMAND_MARGINALREVENUETRANSFORMATION_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <map>
00009 // StdAir
00010 #include <stdair/stdair_inventory_types.hpp>
00011 // RMOL
00012 #include <rmol/RMOL_Types.hpp>
00013
00014 // Forward declarations
00015 namespace stdair {
00016   class SegmentCabin;
00017 }
00018
00019 namespace RMOL {
00021   class MarginalRevenueTransformation {
00022   public:
00026     static bool prepareDemandInput (stdair::SegmentCabin&);
00027
00028   private:
00032     static void buildNestedConvexHull (stdair::SegmentCabin&);
00033
00037     static void buildConvexHull (stdair::SegmentCabin&);
00038
00042     static bool adjustYieldAndDemand (stdair::SegmentCabin&);
00043   };
00044 }
00045 #endif // __RMOL_COMMAND_MARGINALREVENUETRANSFORMATION_HPP
```

26.103 rmol/command/NewQFF.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <cmath>
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/SegmentSnapshotTable.hpp>
#include <stdair/bom/FareFamily.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/bom/Policy.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/Utilities.hpp>
#include <rmol/bom/SegmentSnapshotTableHelper.hpp>
#include <rmol/bom/HistoricalBookingHolder.hpp>
#include <rmol/bom/HistoricalBooking.hpp>
#include <rmol/bom/EMDetruncator.hpp>
#include <rmol/command/NewQFF.hpp>
#include <rmol/command/Detruncator.hpp>
```

Namespaces

- RMOL

26.104 NewQFF.cpp

```
00001 // //////////////////////////////////////////////////////////////////
00002 // Import section
```

```
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 #include <cmath>
00008 // StdAir
00009 #include <stdair/basic/BasConst_General.hpp>
00010 #include <stdair/basic/BasConst_Inventory.hpp>
00011 #include <stdair/bom/BomManager.hpp>
00012 #include <stdair/bom/SegmentDate.hpp>
00013 #include <stdair/bom/SegmentCabin.hpp>
00014 #include <stdair/bom/SegmentSnapshotTable.hpp>
00015 #include <stdair/bom/FareFamily.hpp>
00016 #include <stdair/bom/BookingClass.hpp>
00017 #include <stdair/bom/Policy.hpp>
00018 #include <stdair/service/Logger.hpp>
00019 // RMOL
00020 #include <rmol/bom/Utilities.hpp>
00021 #include <rmol/bom/SegmentSnapshotTableHelper.hpp>
00022 #include <rmol/bom/HistoricalBookingHolder.hpp>
00023 #include <rmol/bom/HistoricalBooking.hpp>
00024 #include <rmol/bom/EMDetruncator.hpp>
00025 #include <rmol/command/NewQFF.hpp>
00026 #include <rmol/command/Detruncator.hpp>
00027
00028 namespace RMOL {
00029   // ////////////////////////////////////////////////////////////////////
00030   bool NewQFF::
00031   forecast (stdair::SegmentCabin& ioSegmentCabin,
00032             const stdair::Date_T& iCurrentDate,
00033             const stdair::DTD_T& iCurrentDTD,
00034             const stdair::UnconstrainingMethod& iUnconstrainingMethod,
00035             const stdair::NbOfSegments_T& iNbOfDepartedSegments) {
00036     // Retrieve the snapshot table.
00037     const stdair::SegmentSnapshotTable& lSegmentSnapshotTable =
00038       ioSegmentCabin.getSegmentSnapshotTable();
00039
00040     // Browse the list of fare families and execute "Q-forecasting" within
00041     // each fare family.
00042     const stdair::FareFamilyList_T& lFFList =
00043       stdair::BomManager::getList<stdair::FareFamily>(ioSegmentCabin);
00044     for (stdair::FareFamilyList_T::const_iterator itFF = lFFList.begin();
00045          itFF != lFFList.end(); ++itFF) {
00046       stdair::FareFamily* lFF_ptr = *itFF;
00047       assert (lFF_ptr != NULL);
00048
00049       forecast (*lFF_ptr,
00050                 iCurrentDate,
00051                 iCurrentDTD,
00052                 iUnconstrainingMethod,
00053                 iNbOfDepartedSegments,
00054                 lSegmentSnapshotTable);
00055     }
00056
00057     // Dispatch the demand forecast to the policies.
00058     dispatchDemandForecastToPolicies (ioSegmentCabin);
00059
00060     return true;
00061   }
00062
00063   // ////////////////////////////////////////////////////////////////////
00064   void NewQFF::
00065   forecast (stdair::FareFamily& ioFareFamily,
00066             const stdair::Date_T& iCurrentDate,
00067             const stdair::DTD_T& iCurrentDTD,
00068             const stdair::UnconstrainingMethod& iUnconstrainingMethod,
00069             const stdair::NbOfSegments_T& iNbOfDepartedSegments,
00070             const stdair::SegmentSnapshotTable& iSegmentSnapshotTable) {
00071     // Retrieve the FRAT5Curve.
00072     const stdair::FRAT5Curve_T& lFRAT5Curve = ioFareFamily.getFrat5Curve();
00073
00074     // Retrieve the booking class list and compute the sell up curves
00075     // and the dispatching curves.
00076     const stdair::BookingClassList_T& lBCList =
00077       stdair::BomManager::getList<stdair::BookingClass>(ioFareFamily);
00078     const stdair::BookingClassSellUpCurveMap_T lBCSellUpCurveMap =
00079       Utilities::computeSellUpFactorCurves (lFRAT5Curve, lBCList);
00080     const stdair::BookingClassDispatchingCurveMap_T lBCDispatchingCurveMap =
00081       Utilities::computeDispatchingFactorCurves (lFRAT5Curve,
      lBCList);
00082
00083     // Browse all remaining DCP's and do unconstraining, forecasting
00084     // and dispatching.
00085     const stdair::DCPList_T lWholeDCPList = stdair::DEFAULT_DCP_LIST;
00086     stdair::DCPList_T::const_iterator itDCP = lWholeDCPList.begin();
00087     stdair::DCPList_T::const_iterator itNextDCP = itDCP; ++itNextDCP;
00088     for (; itNextDCP != lWholeDCPList.end(); ++itDCP, ++itNextDCP) {
```

```
00089       const stdair::DCP_T& lCurrentDCP = *itDCP;
00090       const stdair::DCP_T& lNextDCP = *itNextDCP;
00091
00092       // The end of the interval is after the current DTD.
00093       if (lNextDCP < iCurrentDTD) {
00094         // Get the number of similar segments which has already passed the
00095         // (lNextDCP+1)
00096         const stdair::NbOfSegments_T& lNbOfUsableSegments =
00097           SegmentSnapshotTableHelper::
00098           getNbOfSegmentAlreadyPassedThisDTD (
      iSegmentSnapshotTable,
00099                                               lNextDCP+1,
00100                                               iCurrentDate);
00101         stdair::NbOfSegments_T lSegmentBegin = 0;
00102         const stdair::NbOfSegments_T lSegmentEnd = lNbOfUsableSegments-1;
00103         if (iNbOfDepartedSegments > 52) {
00104           lSegmentBegin = iNbOfDepartedSegments - 52;
00105         }
00106
00107         // Retrieve the historical bookings and convert them to
00108         // Q-equivalent bookings.
00109         HistoricalBookingHolder lHBHolder;
00110         preparePriceOrientedHistoricalBooking (ioFareFamily,
00111                                                iSegmentSnapshotTable,
00112                                                lHBHolder,
00113                                                lCurrentDCP, lNextDCP,
00114                                                lSegmentBegin, lSegmentEnd,
00115                                                lBCSellUpCurveMap);
00116
00117         // Unconstrain the historical bookings.
00118         Detruncator::unconstrain (lHBHolder, iUnconstrainingMethod);
00119
00120         // Retrieve the historical unconstrained demand and perform the
00121         // forecasting.
00122         stdair::UncDemVector_T lUncDemVector;
00123         // Be careful, the getter returns the vector size,
00124         // so there is no reference.
00125         const short lNbOfHistoricalFlights = lHBHolder.getNbOfFlights();
00126         for (short i = 0; i < lNbOfHistoricalFlights; ++i) {
00127           const stdair::NbOfBookings_T& lUncDemand =
00128             lHBHolder.getUnconstrainedDemand (i);
00129           lUncDemVector.push_back (lUncDemand);
00130         }
00131         stdair::MeanValue_T lMean = 0.0;
00132         stdair::StdDevValue_T lStdDev = 0.0;
00133         Utilities::computeDistributionParameters (lUncDemVector,
00134                                                   lMean, lStdDev);
00135
00136         // Dispatch the forecast to all the classes.
00137         Utilities::dispatchDemandForecast (lBCDispatchingCurveMap,
00138                                            lMean, lStdDev, lCurrentDCP);
00139
00140         // Dispatch the forecast to all classes for Fare Adjustment or MRT.
00141         // The sell-up probability will be used in this case.
00142         Utilities::dispatchDemandForecastForFA (lBCSellUpCurveMap,
00143                                                 lMean, lStdDev, lCurrentDCP);
00144
00145         // Add the demand forecast to the fare family.
00146         const stdair::MeanValue_T& lCurrentMean = ioFareFamily.getMean();
00147         const stdair::StdDevValue_T& lCurrentStdDev = ioFareFamily.getStdDev();
00148
00149         const stdair::MeanValue_T lNewMean = lCurrentMean + lMean;
00150         const stdair::StdDevValue_T lNewStdDev =
00151           std::sqrt (lCurrentStdDev * lCurrentStdDev + lStdDev * lStdDev);
00152
00153         ioFareFamily.setMean (lNewMean);
00154         ioFareFamily.setStdDev (lNewStdDev);
00155       }
00156     }
00157
00158   }
00159
00160   // ////////////////////////////////////////////////////////////////////
00161   void NewQFF::preparePriceOrientedHistoricalBooking
00162     (const stdair::FareFamily& iFareFamily,
00163      const stdair::SegmentSnapshotTable& iSegmentSnapshotTable,
00164      HistoricalBookingHolder& ioHBHolder,
00165      const stdair::DCP_T& iDCPBegin, const stdair::DCP_T& iDCPEnd,
00166      const stdair::NbOfSegments_T& iSegmentBegin,
00167      const stdair::NbOfSegments_T& iSegmentEnd,
00168      const stdair::BookingClassSellUpCurveMap_T& iBCSellUpCurveMap) {
00169
00170     // Retrieve the gross daily booking and availability snapshots.
00171     const stdair::ConstSegmentCabinDTDRangeSnapshotView_T lPriceBookingView =
00172       iSegmentSnapshotTable.getConstSegmentCabinDTDRangePriceOrientedGrossBookingSnapshotView (
      iSegmentBegin, iSegmentEnd, iDCPEnd, iDCPBegin);
00173     const stdair::ConstSegmentCabinDTDRangeSnapshotView_T lProductBookingView =
```

```
00174       iSegmentSnapshotTable.getConstSegmentCabinDTDRangeProductOrientedGrossBookingSnapshotView (
      iSegmentBegin, iSegmentEnd, iDCPEnd, iDCPBegin);
00175     const stdair::ConstSegmentCabinDTDRangeSnapshotView_T lAvlView =
00176       iSegmentSnapshotTable.getConstSegmentCabinDTDRangeAvailabilitySnapshotView (iSegmentBegin,
      iSegmentEnd, iDCPEnd, iDCPBegin);
00177
00178     // Browse the list of segments and build the historical booking holder.
00179     const stdair::ClassIndexMap_T& lVTIdxMap =
00180       iSegmentSnapshotTable.getClassIndexMap();
00181     const stdair::NbOfClasses_T lNbOfClasses = lVTIdxMap.size();
00182
00183     for (short i = 0; i <= iSegmentEnd-iSegmentBegin; ++i) {
00184       stdair::Flag_T lCensorshipFlag = false;
00185       const short lNbOfDTDs = iDCPBegin - iDCPEnd + 1;
00186
00187       // Parse the DTDs during the period and compute the censorship flag
00188       for (short j = 0; j < lNbOfDTDs; ++j) {
00189         // Check if the data has been censored during this day.
00190         // STDAIR_LOG_DEBUG ("i: " << i << ", NbOfClasses: " << lNbOfClasses
00191         //                   << ", ClassIdx: " << iClassIdx << ", j: " << j);
00192         bool tempCensorship = true;
00193         for (stdair::BookingClassSellUpCurveMap_T::const_iterator itBCSUC =
00194              iBCSellUpCurveMap.begin();
00195             itBCSUC != iBCSellUpCurveMap.end(); ++itBCSUC) {
00196           const stdair::BookingClass* lBookingClass_ptr = itBCSUC->first;
00197           assert (lBookingClass_ptr != NULL);
00198           const stdair::ClassIndex_T& lClassIdx =
00199             iSegmentSnapshotTable.getClassIndex(lBookingClass_ptr->describeKey());
00200           const stdair::UnsignedIndex_T lAvlIdx = i*lNbOfClasses + lClassIdx;
00201           if (lAvlView[lAvlIdx][j] >= 1.0) {
00202             tempCensorship = false;
00203             break;
00204           }
00205         }
00206         if (tempCensorship == true) {
00207           lCensorshipFlag = true;
00208           break;
00209         }
00210       }
00211
00212       // Compute the Q-equivalent bookings
00213       stdair::NbOfBookings_T lNbOfHistoricalBkgs = 0.0;
00214       for (stdair::BookingClassSellUpCurveMap_T::const_iterator itBCSUC =
00215              iBCSellUpCurveMap.begin();
00216            itBCSUC != iBCSellUpCurveMap.end(); ++itBCSUC) {
00217         const stdair::BookingClass* lBookingClass_ptr = itBCSUC->first;
00218         assert (lBookingClass_ptr != NULL);
00219         const stdair::SellUpCurve_T& lSellUpCurve = itBCSUC->second;
00220         stdair::SellUpCurve_T::const_iterator itSellUp =
00221           lSellUpCurve.find (iDCPBegin);
00222         assert (itSellUp != lSellUpCurve.end());
00223         const stdair::SellupProbability_T& lSellUp = itSellUp->second;
00224         assert (lSellUp != 0);
00225
00226         // Retrieve the number of bookings
00227         const stdair::ClassIndex_T& lClassIdx =
00228           iSegmentSnapshotTable.getClassIndex(lBookingClass_ptr->describeKey());
00229         const stdair::UnsignedIndex_T lIdx = i*lNbOfClasses + lClassIdx;
00230
00231         stdair::NbOfBookings_T lNbOfBookings = 0.0;
00232         for (short j = 0; j < lNbOfDTDs; ++j) {
00233           lNbOfBookings +=
00234             lPriceBookingView[lIdx][j] + lProductBookingView[lIdx][j];
00235         }
00236         const stdair::NbOfBookings_T lNbOfQEquivalentBkgs=lNbOfBookings/lSellUp;
00237
00238         lNbOfHistoricalBkgs += lNbOfQEquivalentBkgs;
00239       }
00240
00241       HistoricalBooking lHistoricalBkg (lNbOfHistoricalBkgs, lCensorshipFlag);
00242       ioHBHolder.addHistoricalBooking (lHistoricalBkg);
00243     }
00244   }
00245
00246   // ////////////////////////////////////////////////////////////////////
00247   void NewQFF::
00248   dispatchDemandForecastToPolicies (const stdair::SegmentCabin& iSegmentCabin){
00249     // Retrieve the list of policies.
00250     const stdair::PolicyList_T& lPolicyList =
00251       stdair::BomManager::getList<stdair::Policy> (iSegmentCabin);
00252
00253     for (stdair::PolicyList_T::const_iterator itPolicy = lPolicyList.begin();
00254          itPolicy != lPolicyList.end(); ++itPolicy) {
00255       stdair::Policy* lPolicy_ptr = *itPolicy;
00256       assert (lPolicy_ptr != NULL);
00257       dispatchDemandForecastToPolicy(*lPolicy_ptr);
00258     }
```

```
00259   }
00260
00261   // ////////////////////////////////////////////////////////////////////
00262   void NewQFF::
00263   dispatchDemandForecastToPolicy (stdair::Policy& ioPolicy){
00264     // Reset the demand forecast of the policy
00265     ioPolicy.resetDemandForecast();
00266
00267     const stdair::MeanValue_T& lPolicyDemand = ioPolicy.getDemand();
00268     const stdair::StdDevValue_T& lPolicyStdDev = ioPolicy.getStdDev();
00269     stdair::MeanValue_T lNewPolicyDemand = lPolicyDemand;
00270     stdair::MeanValue_T lNewPolicyStdDev = lPolicyStdDev;
00271
00272     // Browse the list of booking classes of the policy and use the
00273     // cumulative price-oriented demand forecast of each class.
00274     const bool hasAListOfBC =
00275       stdair::BomManager::hasList<stdair::BookingClass> (ioPolicy);
00276     if (hasAListOfBC == true) {
00277       const stdair::BookingClassList_T& lBCList =
00278         stdair::BomManager::getList<stdair::BookingClass> (ioPolicy);
00279       for (stdair::BookingClassList_T::const_iterator itBC = lBCList.begin();
00280            itBC != lBCList.end(); ++itBC) {
00281         const stdair::BookingClass* lBC_ptr = *itBC;
00282         assert (lBC_ptr != NULL);
00283         const stdair::Yield_T& lYield = lBC_ptr->getYield();
00284         const stdair::MeanValue_T& lDemand = lBC_ptr->getCumuPriceDemMean();
00285         const stdair::StdDevValue_T& lStdDev =
00286           lBC_ptr->getCumuPriceDemStdDev();
00287
00288         ioPolicy.addYieldDemand (lYield, lDemand);
00289         lNewPolicyDemand += lDemand;
00290         const stdair::StdDevValue_T lSquareNewPolicyStdDev =
00291           lNewPolicyStdDev*lNewPolicyStdDev + lStdDev*lStdDev;
00292         lNewPolicyStdDev =
00293           std::sqrt (lSquareNewPolicyStdDev);
00294       }
00295       ioPolicy.setDemand(lNewPolicyDemand);
00296       ioPolicy.setStdDev(lNewPolicyStdDev);
00297     }
00298   }
00299 }
```

26.105 rmol/command/NewQFF.hpp File Reference

```
#include <map>
#include <stdair/stdair_inventory_types.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::NewQFF

Namespaces

- stdair

  Forward declarations.
- RMOL

26.106 NewQFF.hpp

```
00001 #ifndef __RMOL_COMMAND_NEWQFF_HPP
00002 #define __RMOL_COMMAND_NEWQFF_HPP
00003
00004 // ////////////////////////////////////////////////////////////////////
00005 // Import section
00006 // ////////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <map>
00009 // StdAir
00010 #include <stdair/stdair_inventory_types.hpp>
00011 // RMOL
```

```
00012 #include <rmol/RMOL_Types.hpp>
00013
00014 // Forward declarations
00015 namespace stdair {
00016   class SegmentCabin;
00017   class FareFamily;
00018   class SegmentSnapshotTable;
00019 }
00020
00021 namespace RMOL {
00023   class NewQFF {
00024   public:
00034     static bool forecast (stdair::SegmentCabin&, const stdair::Date_T&,
00035                           const stdair::DTD_T&,
00036                           const stdair::UnconstrainingMethod&,
00037                           const stdair::NbOfSegments_T&);
00038
00039   private:
00043     static void forecast (stdair::FareFamily&,
00044                           const stdair::Date_T&,
00045                           const stdair::DTD_T&,
00046                           const stdair::UnconstrainingMethod&,
00047                           const stdair::NbOfSegments_T&,
00048                           const stdair::SegmentSnapshotTable&);
00049
00058     static void preparePriceOrientedHistoricalBooking
00059     (const stdair::FareFamily&, const stdair::SegmentSnapshotTable&,
00060      HistoricalBookingHolder&, const stdair::DCP_T&, const stdair::DCP_T&,
00061      const stdair::NbOfSegments_T&, const stdair::NbOfSegments_T&,
00062      const stdair::BookingClassSellUpCurveMap_T&);
00063
00067     static void dispatchDemandForecastToPolicies (const stdair::SegmentCabin&);
00068
00072     static void dispatchDemandForecastToPolicy (stdair::Policy&);
00073   };
00074 }
00075 #endif // __RMOL_COMMAND_NEWQFF_HPP
```

26.107 rmol/command/OldQFF.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <cmath>
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/SegmentSnapshotTable.hpp>
#include <stdair/bom/FareFamily.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/bom/Policy.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/Utilities.hpp>
#include <rmol/bom/SegmentSnapshotTableHelper.hpp>
#include <rmol/bom/HistoricalBookingHolder.hpp>
#include <rmol/bom/HistoricalBooking.hpp>
#include <rmol/bom/EMDetruncator.hpp>
#include <rmol/command/OldQFF.hpp>
#include <rmol/command/Detruncator.hpp>
```

Namespaces

- RMOL

26.108 OldQFF.cpp

```
00001 // //////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 #include <cmath>
00008 // StdAir
00009 #include <stdair/basic/BasConst_General.hpp>
00010 #include <stdair/basic/BasConst_Inventory.hpp>
00011 #include <stdair/bom/BomManager.hpp>
00012 #include <stdair/bom/SegmentDate.hpp>
00013 #include <stdair/bom/SegmentCabin.hpp>
00014 #include <stdair/bom/SegmentSnapshotTable.hpp>
00015 #include <stdair/bom/FareFamily.hpp>
00016 #include <stdair/bom/BookingClass.hpp>
00017 #include <stdair/bom/Policy.hpp>
00018 #include <stdair/service/Logger.hpp>
00019 // RMOL
00020 #include <rmol/bom/Utilities.hpp>
00021 #include <rmol/bom/SegmentSnapshotTableHelper.hpp>
00022 #include <rmol/bom/HistoricalBookingHolder.hpp>
00023 #include <rmol/bom/HistoricalBooking.hpp>
00024 #include <rmol/bom/EMDetruncator.hpp>
00025 #include <rmol/command/OldQFF.hpp>
00026 #include <rmol/command/Detruncator.hpp>
00027
00028 namespace RMOL {
00029   // //////////////////////////////////////////////////////////////////////
00030   bool OldQFF::
00031   forecast (stdair::SegmentCabin& ioSegmentCabin,
00032             const stdair::Date_T& iCurrentDate,
00033             const stdair::DTD_T& iCurrentDTD,
00034             const stdair::UnconstrainingMethod& iUnconstrainingMethod,
00035             const stdair::NbOfSegments_T& iNbOfDepartedSegments) {
00036     // Retrieve the snapshot table.
00037     const stdair::SegmentSnapshotTable& lSegmentSnapshotTable =
00038       ioSegmentCabin.getSegmentSnapshotTable();
00039
00040     // Retrieve the FRAT5Curve.
00041     const stdair::FareFamilyList_T& lFFList =
00042       stdair::BomManager::getList<stdair::FareFamily>(ioSegmentCabin);
00043     stdair::FareFamilyList_T::const_reverse_iterator itFF = lFFList.rbegin();
00044     assert (itFF != lFFList.rend());
00045     stdair::FareFamily* lFF_ptr = *itFF;
00046     assert (lFF_ptr != NULL);
00047     const stdair::FRAT5Curve_T lFRAT5Curve = lFF_ptr->getFrat5Curve();
00048
00049     // Retrieve the booking class list and compute the sell up curves
00050     // and the dispatching curves.
00051     const stdair::BookingClassList_T& lBCList =
00052       stdair::BomManager::getList<stdair::BookingClass>(ioSegmentCabin);
00053     const stdair::BookingClassSellUpCurveMap_T lBCSellUpCurveMap =
00054       Utilities::computeSellUpFactorCurves (lFRAT5Curve, lBCList);
00055
00056     // Retrieve the list of all policies and reset the demand forecast
00057     // for each one.
00058     const stdair::PolicyList_T& lPolicyList =
00059       stdair::BomManager::getList<stdair::Policy> (ioSegmentCabin);
00060     for (stdair::PolicyList_T::const_iterator itPolicy = lPolicyList.begin();
00061          itPolicy != lPolicyList.end(); ++itPolicy) {
00062       stdair::Policy* lPolicy_ptr = *itPolicy;
00063       assert (lPolicy_ptr != NULL);
00064       lPolicy_ptr->resetDemandForecast();
00065     }
00066
00067     // Browse all remaining DCP's and do unconstraining, forecasting
00068     // and dispatching.
00069     const stdair::DCPList_T lWholeDCPList = stdair::DEFAULT_DCP_LIST;
00070     stdair::DCPList_T::const_iterator itDCP = lWholeDCPList.begin();
00071     stdair::DCPList_T::const_iterator itNextDCP = itDCP; ++itNextDCP;
00072     for (; itNextDCP != lWholeDCPList.end(); ++itDCP, ++itNextDCP) {
00073       const stdair::DCP_T& lCurrentDCP = *itDCP;
00074       const stdair::DCP_T& lNextDCP = *itNextDCP;
00075
00076       // The end of the interval is after the current DTD.
00077       if (lNextDCP < iCurrentDTD) {
00078         // Get the number of similar segments which has already passed the
00079         // (lNextDCP+1)
00080         const stdair::NbOfSegments_T& lNbOfUsableSegments =
00081           SegmentSnapshotTableHelper::
00082           getNbOfSegmentAlreadyPassedThisDTD (
      lSegmentSnapshotTable,
00083                                               lNextDCP+1,
00084                                               iCurrentDate);
```

```
00085         stdair::NbOfSegments_T lSegmentBegin = 0;
00086         const stdair::NbOfSegments_T lSegmentEnd = lNbOfUsableSegments-1;
00087         if (iNbOfDepartedSegments > 52) {
00088           lSegmentBegin = iNbOfDepartedSegments - 52;
00089         }
00090
00091         // Retrieve the historical bookings and convert them to
00092         // Q-equivalent bookings.
00093         HistoricalBookingHolder lHBHolder;
00094         prepareHistoricalBooking (ioSegmentCabin, lSegmentSnapshotTable,
00095                                   lHBHolder, lCurrentDCP, lNextDCP,
00096                                   lSegmentBegin, lSegmentEnd,
00097                                   lBCSellUpCurveMap);
00098
00099         // Unconstrain the historical bookings.
00100         Detruncator::unconstrain (lHBHolder, iUnconstrainingMethod);
00101
00102         // Retrieve the historical unconstrained demand and perform the
00103         // forecasting.
00104         stdair::UncDemVector_T lUncDemVector;
00105         const short lNbOfHistoricalFlights = lHBHolder.getNbOfFlights();
00106         for (short i = 0; i < lNbOfHistoricalFlights; ++i) {
00107           const stdair::NbOfBookings_T& lUncDemand =
00108             lHBHolder.getUnconstrainedDemand (i);
00109           lUncDemVector.push_back (lUncDemand);
00110         }
00111         stdair::MeanValue_T lMean = 0.0;
00112         stdair::StdDevValue_T lStdDev = 0.0;
00113         Utilities::computeDistributionParameters (lUncDemVector,
00114                                                   lMean, lStdDev);
00115
00116         // Add the demand forecast to the fare family.
00117         const stdair::MeanValue_T& lCurrentMean = lFF_ptr->getMean();
00118         const stdair::StdDevValue_T& lCurrentStdDev = lFF_ptr->getStdDev();
00119
00120         const stdair::MeanValue_T lNewMean = lCurrentMean + lMean;
00121         const stdair::StdDevValue_T lNewStdDev =
00122           std::sqrt (lCurrentStdDev * lCurrentStdDev + lStdDev * lStdDev);
00123
00124         lFF_ptr->setMean (lNewMean);
00125         lFF_ptr->setStdDev (lNewStdDev);
00126
00127         // Dispatch the demand forecast to the policies.
00128         dispatchDemandForecastToPolicies (lPolicyList, lCurrentDCP, lMean,
00129                                           lStdDev, lBCSellUpCurveMap);
00130       }
00131     }
00132
00133     return true;
00134   }
00135
00136   // ////////////////////////////////////////////////////////////////////
00137   void OldQFF::prepareHistoricalBooking
00138   (const stdair::SegmentCabin& iSegmentCabin,
00139    const stdair::SegmentSnapshotTable& iSegmentSnapshotTable,
00140    HistoricalBookingHolder& ioHBHolder,
00141    const stdair::DCP_T& iDCPBegin, const stdair::DCP_T& iDCPEnd,
00142    const stdair::NbOfSegments_T& iSegmentBegin,
00143    const stdair::NbOfSegments_T& iSegmentEnd,
00144    const stdair::BookingClassSellUpCurveMap_T& iBCSellUpCurveMap) {
00145
00146     // Retrieve the segment-cabin index within the snapshot table
00147     std::ostringstream lSCMapKey;
00148     lSCMapKey << stdair::DEFAULT_SEGMENT_CABIN_VALUE_TYPE
00149               << iSegmentCabin.describeKey();
00150     const stdair::ClassIndex_T& lCabinIdx =
00151       iSegmentSnapshotTable.getClassIndex (lSCMapKey.str());
00152
00153     // Retrieve the gross daily booking and availability snapshots.
00154     const stdair::ConstSegmentCabinDTDRangeSnapshotView_T lPriceBookingView =
00155       iSegmentSnapshotTable.getConstSegmentCabinDTDRangePriceOrientedGrossBookingSnapshotView (
      iSegmentBegin, iSegmentEnd, iDCPEnd, iDCPBegin);
00156     const stdair::ConstSegmentCabinDTDRangeSnapshotView_T lProductBookingView =
00157       iSegmentSnapshotTable.getConstSegmentCabinDTDRangeProductOrientedGrossBookingSnapshotView (
      iSegmentBegin, iSegmentEnd, iDCPEnd, iDCPBegin);
00158     const stdair::ConstSegmentCabinDTDRangeSnapshotView_T lAvlView =
00159       iSegmentSnapshotTable.getConstSegmentCabinDTDRangeAvailabilitySnapshotView (iSegmentBegin,
      iSegmentEnd, iDCPEnd, iDCPBegin);
00160
00161     // Browse the list of segments and build the historical booking holder.
00162     const stdair::ClassIndexMap_T& lVTIdxMap =
00163       iSegmentSnapshotTable.getClassIndexMap();
00164     const stdair::NbOfClasses_T lNbOfClasses = lVTIdxMap.size();
00165
00166     for (short i = 0; i <= iSegmentEnd-iSegmentBegin; ++i) {
00167       stdair::Flag_T lCensorshipFlag = false;
00168       const short lNbOfDTDs = iDCPBegin - iDCPEnd + 1;
```

```
00169       const stdair::UnsignedIndex_T lAvlIdx = i*lNbOfClasses + lCabinIdx;
00170
00171       // Parse the DTDs during the period and compute the censorship flag
00172       for (short j = 0; j < lNbOfDTDs; ++j) {
00173         // Check if the data has been censored during this day.
00174         // STDAIR_LOG_DEBUG ("i: " << i << ", NbOfClasses: " << lNbOfClasses
00175         //                       << ", ClassIdx: " << iClassIdx << ", j: " << j);
00176         if (lAvlView[lAvlIdx][j] < 1.0) {
00177           lCensorshipFlag = true;
00178           break;
00179         }
00180       }
00181
00182       // Compute the Q-equivalent bookings
00183       stdair::NbOfBookings_T lNbOfHistoricalBkgs = 0.0;
00184       const stdair::BookingClassList_T& lBCList =
00185         stdair::BomManager::getList<stdair::BookingClass> (iSegmentCabin);
00186       for (short j = 0; j < lNbOfDTDs; ++j) {
00187         stdair::BookingClass* lLowestBC_ptr = NULL;
00188         stdair::NbOfBookings_T lNbOfBksOfTheDay = 0.0;
00189         for (stdair::BookingClassList_T::const_iterator itBC = lBCList.begin();
00190              itBC != lBCList.end(); ++itBC) {
00191           stdair::BookingClass* lBC_ptr = *itBC;
00192           assert (lBC_ptr != NULL);
00193
00194           // Retrieve the number of bookings
00195           const stdair::ClassIndex_T& lClassIdx =
00196             iSegmentSnapshotTable.getClassIndex(lBC_ptr->describeKey());
00197           const stdair::UnsignedIndex_T lIdx = i*lNbOfClasses + lClassIdx;
00198           const stdair::NbOfBookings_T lNbOfBookings =
00199             lPriceBookingView[lIdx][j] + lProductBookingView[lIdx][j];
00200           lNbOfBksOfTheDay += lNbOfBookings;
00201
00202           if (lAvlView[lIdx][j] >= 1.0) {
00203             lLowestBC_ptr = lBC_ptr;
00204           }
00205         }
00206
00207         // Convert the number of bookings of the day to Q-equivalent
00208         // bookings using the sell-up probability of the lowest class
00209         // available of the day.
00210         if (lLowestBC_ptr != NULL) {
00211           stdair::BookingClassSellUpCurveMap_T::const_iterator itBCSUC =
00212             iBCSellUpCurveMap.find (lLowestBC_ptr);
00213           assert (itBCSUC != iBCSellUpCurveMap.end());
00214           const stdair::SellUpCurve_T& lSellUpCurve = itBCSUC->second;
00215           stdair::SellUpCurve_T::const_iterator itSellUp =
00216             lSellUpCurve.find (iDCPBegin);
00217           assert (itSellUp != lSellUpCurve.end());
00218           const stdair::SellupProbability_T& lSellUp = itSellUp->second;
00219           assert (lSellUp != 0);
00220
00221           lNbOfHistoricalBkgs += lNbOfBksOfTheDay/lSellUp;
00222         }
00223       }
00224
00225       HistoricalBooking lHistoricalBkg (lNbOfHistoricalBkgs, lCensorshipFlag);
00226       ioHBHolder.addHistoricalBooking (lHistoricalBkg);
00227     }
00228   }
00229
00230   // ////////////////////////////////////////////////////////////////////
00231   void OldQFF::
00232   dispatchDemandForecastToPolicies (const stdair::PolicyList_T& iPolicyList,
00233                                     const stdair::DCP_T& iCurrentDCP,
00234                                     const stdair::MeanValue_T& iMean,
00235                                     const stdair::StdDevValue_T& iStdDev,
00236                                     const stdair::BookingClassSellUpCurveMap_T& iBCSellUpCurveMap) {
00237     for (stdair::PolicyList_T::const_iterator itPolicy = iPolicyList.begin();
00238          itPolicy != iPolicyList.end(); ++itPolicy) {
00239       stdair::Policy* lPolicy_ptr = *itPolicy;
00240       assert (lPolicy_ptr != NULL);
00241       dispatchDemandForecastToPolicy (*lPolicy_ptr,
00242                                       iCurrentDCP,
00243                                       iMean,
00244                                       iStdDev,
00245                                       iBCSellUpCurveMap);
00246     }
00247   }
00248
00249   // ////////////////////////////////////////////////////////////////////
00250   void OldQFF::
00251   dispatchDemandForecastToPolicy (stdair::Policy& ioPolicy,
00252                                   const stdair::DCP_T& iCurrentDCP,
00253                                   const stdair::MeanValue_T& iMean,
00254                                   const stdair::StdDevValue_T& iStdDev,
00255                                   const stdair::BookingClassSellUpCurveMap_T& iBCSellUpCurveMap) {
```

```
00256     const stdair::MeanValue_T& lPolicyDemand = ioPolicy.getDemand();
00257     const stdair::StdDevValue_T& lPolicyStdDev = ioPolicy.getStdDev();
00258
00259     // Browse the list of booking classes of the policy and use the
00260     // cumulative price-oriented demand forecast of each class.
00261     const bool hasAListOfBC =
00262       stdair::BomManager::hasList<stdair::BookingClass> (ioPolicy);
00263     if (hasAListOfBC == true) {
00264       const stdair::BookingClassList_T& lBCList =
00265         stdair::BomManager::getList<stdair::BookingClass> (ioPolicy);
00266       stdair::BookingClassList_T::const_reverse_iterator itCurrentBC =
00267         lBCList.rbegin();
00268       assert(itCurrentBC != lBCList.rend());
00269       stdair::BookingClass* lLowestBC_ptr = *itCurrentBC;
00270       assert (lLowestBC_ptr != NULL);
00271       const stdair::Yield_T& lLowestBCYield = lLowestBC_ptr->getYield();
00272       // Retrieve the sell-up factor for the lowest class.
00273       stdair::BookingClassSellUpCurveMap_T::const_iterator itBCSU =
00274         iBCSellUpCurveMap.find (lLowestBC_ptr);
00275       assert (itBCSU != iBCSellUpCurveMap.end());
00276       const stdair::SellUpCurve_T& lSellUpCurve = itBCSU->second;
00277       stdair::SellUpCurve_T::const_iterator itSellUpFactor =
00278         lSellUpCurve.find (iCurrentDCP);
00279       assert (itSellUpFactor != lSellUpCurve.end());
00280       const stdair::SellupProbability_T& lSUToLowestClass = itSellUpFactor->second;
00281
00282       const stdair::MeanValue_T lAdditinalPolicyDemandMean =
00283         iMean * lSUToLowestClass;
00284       const stdair::StdDevValue_T lAdditinalPolicyDemandStdDev =
00285         iStdDev * std::sqrt (lSUToLowestClass);
00286
00287       const stdair::MeanValue_T lNewPolicyDemandMean =
00288         lPolicyDemand + lAdditinalPolicyDemandMean;
00289       const stdair::StdDevValue_T lNewPolicyDemandStdDev =
00290         std::sqrt (lPolicyStdDev*lPolicyStdDev
00291         + lAdditinalPolicyDemandStdDev * lAdditinalPolicyDemandStdDev);
00292       //
00293       ioPolicy.setDemand (lNewPolicyDemandMean);
00294       ioPolicy.setStdDev (lNewPolicyDemandStdDev);
00295
00296       ioPolicy.addYieldDemand (lLowestBCYield,
00297                                lAdditinalPolicyDemandMean);
00298
00299       // Iterate other classes.
00300       stdair::BookingClassList_T::const_reverse_iterator itNextBC=itCurrentBC;
00301       ++itNextBC;
00302       for (; itNextBC != lBCList.rend(); ++itNextBC, ++itCurrentBC) {
00303         stdair::BookingClass* lCurrentBC_ptr = *itCurrentBC;
00304         assert (lCurrentBC_ptr != NULL);
00305         stdair::BookingClass* lNextBC_ptr = *itNextBC;
00306         assert (lNextBC_ptr != NULL);
00307
00308         // Retrieve the disutility for the current policy to the next one.
00309         const stdair::FareFamily& lCurrentFF =
00310           stdair::BomManager::getParent<stdair::FareFamily> (*lCurrentBC_ptr);
00311         const stdair::FFDisutilityCurve_T& lDisutilityCurve =
00312           lCurrentFF.getDisutilityCurve();
00313         stdair::FFDisutilityCurve_T::const_iterator itDU =
00314           lDisutilityCurve.find (iCurrentDCP);
00315         assert (itDU != lDisutilityCurve.end());
00316         const double& lDU = itDU->second;
00317
00318         // Retrieve the sell-up factor for the next class.
00319         stdair::BookingClassSellUpCurveMap_T::const_iterator itBCSUN =
00320           iBCSellUpCurveMap.find (lNextBC_ptr);
00321         assert (itBCSUN != iBCSellUpCurveMap.end());
00322         const stdair::SellUpCurve_T& lSellUpCurveN = itBCSUN->second;
00323         stdair::SellUpCurve_T::const_iterator itSellUpFactorN =
00324           lSellUpCurveN.find (iCurrentDCP);
00325         assert (itSellUpFactorN != lSellUpCurveN.end());
00326         const stdair::SellupProbability_T& lSUToNextClass = itSellUpFactorN->second;
00327         assert (lSUToNextClass > 0.0);
00328         assert(lSUToNextClass < lSUToLowestClass);
00329
00330         // Retrieve the yields of the two classes
00331         const stdair::Yield_T& lCurrentYield = lCurrentBC_ptr->getYield();
00332         const stdair::Yield_T& lNextYield = lNextBC_ptr->getYield();
00333         const double lBuyUpFactor = exp ((lCurrentYield-lNextYield)*lDU);
00334
00335         // Withdraw an amount demand forecast from the current class. This
00336         // amount of forecast will be added to the next class.
00337         const stdair::MeanValue_T lDemandForNextClass =
00338           iMean * lSUToNextClass * lBuyUpFactor;
00339         ioPolicy.addYieldDemand (lNextYield, lDemandForNextClass);
00340         ioPolicy.addYieldDemand (lCurrentYield, -lDemandForNextClass);
00341       }
00342     }
```

```
00343   }
00344 }
```

26.109 rmol/command/OldQFF.hpp File Reference

```
#include <map>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/bom/PolicyTypes.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::OldQFF

Namespaces

- stdair

  Forward declarations.
- RMOL

26.110 OldQFF.hpp

```
00001 #ifndef __RMOL_COMMAND_OLDQFF_HPP
00002 #define __RMOL_COMMAND_OLDQFF_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <map>
00009 // StdAir
00010 #include <stdair/stdair_inventory_types.hpp>
00011 #include <stdair/bom/PolicyTypes.hpp>
00012 // RMOL
00013 #include <rmol/RMOL_Types.hpp>
00014
00015 // Forward declarations
00016 namespace stdair {
00017   class SegmentCabin;
00018   class SegmentSnapshotTable;
00019 }
00020
00021 namespace RMOL {
00023   class OldQFF {
00024   public:
00034     static bool forecast (stdair::SegmentCabin&, const stdair::Date_T&,
00035                           const stdair::DTD_T&,
00036                           const stdair::UnconstrainingMethod&,
00037                           const stdair::NbOfSegments_T&);
00038
00039   private:
00048     static void prepareHistoricalBooking (const stdair::SegmentCabin&,
00049                                           const stdair::SegmentSnapshotTable&,
00050                                           HistoricalBookingHolder&,
00051                                           const stdair::DCP_T&,
00052                                           const stdair::DCP_T&,
00053                                           const stdair::NbOfSegments_T&,
00054                                           const stdair::NbOfSegments_T&,
00055                                           const stdair::BookingClassSellUpCurveMap_T&);
00056
00060     static void
00061     dispatchDemandForecastToPolicies (const stdair::PolicyList_T&,
00062                                       const stdair::DCP_T&,
00063                                       const stdair::MeanValue_T&,
00064                                       const stdair::StdDevValue_T&,
00065                                       const stdair::BookingClassSellUpCurveMap_T&);
00066
00070     static void
00071     dispatchDemandForecastToPolicy (stdair::Policy&,
00072                                     const stdair::DCP_T&,
```

```
00073                                     const stdair::MeanValue_T&,
00074                                     const stdair::StdDevValue_T&,
00075                                     const stdair::BookingClassSellUpCurveMap_T&);
00076   };
00077 }
00078 #endif // __RMOL_COMMAND_OLDQFF_HPP
```

26.111 rmol/command/Optimiser.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/basic/RandomGeneration.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/FlightDate.hpp>
#include <stdair/bom/LegDate.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/FareFamily.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/basic/BasConst_General.hpp>
#include <rmol/bom/MCOptimiser.hpp>
#include <rmol/bom/Emsr.hpp>
#include <rmol/bom/DPOptimiser.hpp>
#include <rmol/command/Optimiser.hpp>
```

Namespaces

- RMOL

26.112 Optimiser.cpp

```
00001 // //////////////////////////////////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 // StdAir
00008 #include <stdair/basic/BasConst_General.hpp>
00009 #include <stdair/basic/RandomGeneration.hpp>
00010 #include <stdair/bom/BomManager.hpp>
00011 #include <stdair/bom/FlightDate.hpp>
00012 #include <stdair/bom/LegDate.hpp>
00013 #include <stdair/bom/SegmentDate.hpp>
00014 #include <stdair/bom/LegCabin.hpp>
00015 #include <stdair/bom/SegmentCabin.hpp>
00016 #include <stdair/bom/FareFamily.hpp>
00017 #include <stdair/bom/BookingClass.hpp>
00018 #include <stdair/service/Logger.hpp>
00019 // RMOL
00020 #include <rmol/basic/BasConst_General.hpp>
00021 #include <rmol/bom/MCOptimiser.hpp>
00022 #include <rmol/bom/Emsr.hpp>
00023 #include <rmol/bom/DPOptimiser.hpp>
00024 #include <rmol/command/Optimiser.hpp>
00025
00026 namespace RMOL {
00027
00028   // //////////////////////////////////////////////////////////////////
00029   void Optimiser::
00030   optimalOptimisationByMCIntegration (const stdair::NbOfSamples_T& K,
00031                                       stdair::LegCabin& ioLegCabin) {
00032     // Retrieve the segment-cabin
00033     const stdair::SegmentCabinList_T lSegmentCabinList =
00034       stdair::BomManager::getList<stdair::SegmentCabin> (ioLegCabin);
```

```
00035     stdair::SegmentCabinList_T::const_iterator itSC = lSegmentCabinList.begin();
00036     assert (itSC != lSegmentCabinList.end());
00037     const stdair::SegmentCabin* lSegmentCabin_ptr = *itSC;
00038     assert (lSegmentCabin_ptr != NULL);
00039
00040     // Retrieve the class list.
00041     const stdair::BookingClassList_T lBookingClassList =
00042       stdair::BomManager::getList<stdair::BookingClass> (*lSegmentCabin_ptr);
00043     stdair::RandomGeneration lSeedGenerator (stdair::DEFAULT_RANDOM_SEED);
00044
00045     // Generate the demand samples for the booking classes.
00046     for (stdair::BookingClassList_T::const_iterator itBC =
00047            lBookingClassList.begin(); itBC != lBookingClassList.end(); ++itBC) {
00048       stdair::RandomSeed_T lRandomSeed =
00049         lSeedGenerator.generateUniform01 () * 1e9;
00050       stdair::BookingClass* lBookingClass_ptr = *itBC;
00051       assert (lBookingClass_ptr != NULL);
00052       lBookingClass_ptr->generateDemandSamples (K, lRandomSeed);
00053
00054       // DEBUG
00055       //STDAIR_LOG_DEBUG ("Generating " << K << " demand samples for the class "
00056       //                  << lBookingClass_ptr->describeKey());
00057     }
00058
00059     // Call the class performing the actual algorithm
00060     MCOptimiser::optimalOptimisationByMCIntegration (
      ioLegCabin);
00061   }
00062
00063   // ////////////////////////////////////////////////////////////////////
00064   void Optimiser::optimalOptimisationByDP (stdair::LegCabin& ioLegCabin)
      {
00065     DPOptimiser::optimalOptimisationByDP (ioLegCabin);
00066   }
00067
00068   // ////////////////////////////////////////////////////////////////////
00069   void Optimiser::heuristicOptimisationByEmsr (stdair::LegCabin&
      ioLegCabin) {
00070     Emsr::heuristicOptimisationByEmsr (ioLegCabin);
00071   }
00072
00073   // ////////////////////////////////////////////////////////////////////
00074   void Optimiser::heuristicOptimisationByEmsrA (stdair::LegCabin&
      ioLegCabin) {
00075     Emsr::heuristicOptimisationByEmsrA (ioLegCabin);
00076   }
00077
00078   // ////////////////////////////////////////////////////////////////////
00079   void Optimiser::heuristicOptimisationByEmsrB (stdair::LegCabin&
      ioLegCabin) {
00080     Emsr::heuristicOptimisationByEmsrB (ioLegCabin);
00081   }
00082
00083   // ////////////////////////////////////////////////////////////////////
00084   bool Optimiser::optimise (stdair::FlightDate& ioFlightDate,
00085                             const stdair::OptimisationMethod& iOptimisationMethod) {
00086     bool optimiseSucceeded = false;
00087     // Browse the leg-cabin list and build the virtual class list for
00088     // each cabin.
00089     const stdair::LegDateList_T& lLDList =
00090       stdair::BomManager::getList<stdair::LegDate> (ioFlightDate);
00091     for (stdair::LegDateList_T::const_iterator itLD = lLDList.begin();
00092          itLD != lLDList.end(); ++itLD) {
00093       stdair::LegDate* lLD_ptr = *itLD;
00094       assert (lLD_ptr != NULL);
00095       const bool isSucceeded = optimise(*lLD_ptr, iOptimisationMethod);
00096       // If at least one leg date is optimised, the optimisation is succeeded.
00097       if (isSucceeded == true) {
00098         optimiseSucceeded = true;
00099         // Do not return now because all leg dates need to be optimised.
00100       }
00101     }
00102     return optimiseSucceeded;
00103   }
00104
00105   // ////////////////////////////////////////////////////////////////////
00106   bool Optimiser::
00107   optimise (stdair::LegDate& ioLegDate,
00108             const stdair::OptimisationMethod& iOptimisationMethod) {
00109     bool optimiseSucceeded = false;
00110     // Browse the leg-cabin list
00111     const stdair::LegCabinList_T& lLCList =
00112       stdair::BomManager::getList<stdair::LegCabin> (ioLegDate);
00113     for (stdair::LegCabinList_T::const_iterator itLC = lLCList.begin();
00114          itLC != lLCList.end(); ++itLC) {
00115       stdair::LegCabin* lLC_ptr = *itLC;
00116       assert (lLC_ptr != NULL);
```

```
00117       const bool isSucceeded = optimise(*lLC_ptr, iOptimisationMethod);
00118       // If at least one leg cabin is optimised, the optimisation is succeeded.
00119       if (isSucceeded == true) {
00120         optimiseSucceeded = true;
00121         // Do not return now because all leg cabins need to be optimised.
00122       }
00123     }
00124     return optimiseSucceeded;
00125   }
00126
00127   // ////////////////////////////////////////////////////////////////////
00128   bool Optimiser::
00129   optimise (stdair::LegCabin& ioLegCabin,
00130             const stdair::OptimisationMethod& iOptimisationMethod) {
00131     bool optimiseSucceeded = false;
00132     //
00133     // Build the virtual class list.
00134     bool hasVirtualClass =
00135       buildVirtualClassListForLegBasedOptimisation (ioLegCabin)
    ;
00136     if (hasVirtualClass == true) {
00137       switch (iOptimisationMethod.getMethod()) {
00138       case stdair::OptimisationMethod::LEG_BASED_MC: {
00139         // Number of samples generated for the Monte Carlo integration.
00140         // It is important that number is greater than 100 (=10000 here).
00141         const stdair::NbOfSamples_T lNbOfSamples =
00142           DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION;
00143         optimalOptimisationByMCIntegration (lNbOfSamples, ioLegCabin);
00144         optimiseSucceeded = true;
00145         break;
00146       }
00147       case stdair::OptimisationMethod::LEG_BASED_EMSR_B: {
00148         heuristicOptimisationByEmsrB (ioLegCabin);
00149         optimiseSucceeded = true;
00150         break;
00151       }
00152       default: {
00153         assert (false);
00154           break;
00155       }
00156       }
00157     }
00158
00159     return optimiseSucceeded;
00160   }
00161
00162   // ////////////////////////////////////////////////////////////////////
00163   bool Optimiser::
00164   buildVirtualClassListForLegBasedOptimisation (
      stdair::LegCabin& ioLegCabin) {
00165     // The map holding all virtual classes to be created.
00166     stdair::VirtualClassMap_T lVirtualClassMap;
00167     bool isNotEmpty = false;
00168
00169     // Retrieve the segment-cabin
00170     const stdair::SegmentCabinList_T& lSegmentCabinList =
00171       stdair::BomManager::getList<stdair::SegmentCabin> (ioLegCabin);
00172     stdair::SegmentCabinList_T::const_iterator itSC = lSegmentCabinList.begin();
00173     assert (itSC != lSegmentCabinList.end());
00174     const stdair::SegmentCabin* lSegmentCabin_ptr = *itSC;
00175     assert (lSegmentCabin_ptr != NULL);
00176
00177     // Retrieve the class list.
00178     const stdair::BookingClassList_T lBookingClassList =
00179       stdair::BomManager::getList<stdair::BookingClass> (*lSegmentCabin_ptr);
00180
00181     // Generate the demand samples for the booking classes.
00182     for (stdair::BookingClassList_T::const_iterator itBC =
00183            lBookingClassList.begin(); itBC != lBookingClassList.end(); ++itBC) {
00184       stdair::BookingClass* lBookingClass_ptr = *itBC;
00185       assert (lBookingClass_ptr != NULL);
00186
00187       // If the demand forecast of the class is zero, there no need to create
00188       // a virtual class.
00189       // TODO: use float utils
00190       const stdair::NbOfRequests_T& lMean = lBookingClass_ptr->getMean();
00191       const stdair::StdDevValue_T& lStdDev = lBookingClass_ptr->getStdDev();
00192       if (lMean > 0.0) {
00193         const stdair::Yield_T& lYield = lBookingClass_ptr->getAdjustedYield();
00194         // TODO: use float utils
00195         assert (lYield >= 0.0);
00196         const stdair::Yield_T lRoundedYieldDouble = std::floor(lYield +0.5);
00197         const stdair::YieldLevel_T lRoundedYieldLevel =
00198           static_cast<stdair::YieldLevel_T>(lRoundedYieldDouble);
00199         if (lRoundedYieldLevel > 0) {
00200           // If there is already a virtual class with this yield, add the current
00201           // booking class to its list and sum the two demand distributions.
```

```
00202           // Otherwise, create a new virtual class.
00203           stdair::VirtualClassMap_T::iterator itVCMap =
00204             lVirtualClassMap.find(lRoundedYieldLevel);
00205           if (itVCMap == lVirtualClassMap.end()) {
00206             stdair::BookingClassList_T lBookingClassList;
00207             lBookingClassList.push_back(lBookingClass_ptr);
00208             stdair::VirtualClassStruct lVirtualClass (lBookingClassList);
00209             lVirtualClass.setYield (lRoundedYieldLevel);
00210             lVirtualClass.setMean (lMean);
00211             lVirtualClass.setStdDev (lStdDev);
00212
00213             lVirtualClassMap.insert (stdair::VirtualClassMap_T::
00214                                      value_type (lRoundedYieldLevel, lVirtualClass));
00215           } else {
00216             stdair::VirtualClassStruct& lVirtualClass = itVCMap->second;
00217             const stdair::MeanValue_T& lVCMean = lVirtualClass.getMean();
00218             const stdair::StdDevValue_T& lVCStdDev = lVirtualClass.getStdDev();
00219             const stdair::MeanValue_T lNewMean = lVCMean + lMean;
00220             const stdair::StdDevValue_T lNewStdDev =
00221               std::sqrt(lVCStdDev * lVCStdDev + lStdDev * lStdDev);
00222             lVirtualClass.setMean (lNewMean);
00223             lVirtualClass.setStdDev (lNewStdDev);
00224
00225             lVirtualClass.addBookingClass(*lBookingClass_ptr);
00226           }
00227         }
00228       }
00229     }
00230
00231     // Browse the virtual class map from high to low yield.
00232     ioLegCabin.emptyVirtualClassList();
00233     for (stdair::VirtualClassMap_T::reverse_iterator itVC =
00234            lVirtualClassMap.rbegin(); itVC != lVirtualClassMap.rend(); ++itVC) {
00235       stdair::VirtualClassStruct& lVC = itVC->second;
00236
00237       ioLegCabin.addVirtualClass (lVC);
00238       if (isNotEmpty == false) {
00239         isNotEmpty = true;
00240       }
00241     }
00242     return isNotEmpty;
00243   }
00244
00245   // ////////////////////////////////////////////////////////////////////
00246   double Optimiser::
00247   optimiseUsingOnDForecast (stdair::FlightDate& ioFlightDate,
00248                             const bool& iReduceFluctuations) {
00249     double lMaxBPVariation = 0.0;
00250     // Check if the flight date holds a list of leg dates.
00251     // If so, retieve it and optimise the cabins.
00252     const bool hasLegDateList =
00253       stdair::BomManager::hasList<stdair::LegDate> (ioFlightDate);
00254     if (hasLegDateList == true) {
00255       STDAIR_LOG_DEBUG ("Optimisation for the flight date: "
00256                         << ioFlightDate.toString());
00257       const stdair::LegDateList_T& lLDList =
00258         stdair::BomManager::getList<stdair::LegDate> (ioFlightDate);
00259       for (stdair::LegDateList_T::const_iterator itLD = lLDList.begin();
00260            itLD != lLDList.end(); ++itLD) {
00261         stdair::LegDate* lLD_ptr = *itLD;
00262         assert (lLD_ptr != NULL);
00263
00264         //
00265         const stdair::LegCabinList_T& lLCList =
00266           stdair::BomManager::getList<stdair::LegCabin> (*lLD_ptr);
00267         for (stdair::LegCabinList_T::const_iterator itLC = lLCList.begin();
00268              itLC != lLCList.end(); ++itLC) {
00269           stdair::LegCabin* lLC_ptr = *itLC;
00270           assert (lLC_ptr != NULL);
00271           MCOptimiser::optimisationByMCIntegration (*lLC_ptr);
00272           const stdair::BidPrice_T& lCurrentBidPrice =
00273             lLC_ptr->getCurrentBidPrice();
00274           const stdair::BidPrice_T& lPreviousBidPrice =
00275             lLC_ptr->getPreviousBidPrice();
00276           assert (lPreviousBidPrice != 0);
00277           const double lBPVariation =
00278             std::abs((lCurrentBidPrice - lPreviousBidPrice)/lPreviousBidPrice);
00279           lMaxBPVariation = std::max(lMaxBPVariation, lBPVariation);
00280         }
00281       }
00282     }
00283     return lMaxBPVariation;
00284   }
00285
00286 }
```

26.113 rmol/command/Optimiser.hpp File Reference

```
#include <stdair/basic/OptimisationMethod.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::Optimiser

Namespaces

- stdair

  Forward declarations.
- RMOL

26.114 Optimiser.hpp

```
00001 #ifndef __RMOL_COMMAND_OPTIMISER_HPP
00002 #define __RMOL_COMMAND_OPTIMISER_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STDAIR
00008 #include <stdair/basic/OptimisationMethod.hpp>
00009 // RMOL
00010 #include <rmol/RMOL_Types.hpp>
00011
00012 // Forward declarations
00013 namespace stdair {
00014   class FlightDate;
00015   class LegCabin;
00016 }
00017
00018 namespace RMOL {
00020   class Optimiser {
00021   public:
00022
00034     static void optimalOptimisationByMCIntegration (const
      stdair::NbOfSamples_T&,
00035                                                     stdair::LegCabin&);
00036
00040     static void optimalOptimisationByDP (stdair::LegCabin&);
00041
00045     static void heuristicOptimisationByEmsr (stdair::LegCabin&);
00046
00050     static void heuristicOptimisationByEmsrA (stdair::LegCabin&);
00051
00055     static void heuristicOptimisationByEmsrB (stdair::LegCabin&);
00056
00060     static bool optimise (stdair::FlightDate&,
00061                           const stdair::OptimisationMethod&);
00062
00066     static bool buildVirtualClassListForLegBasedOptimisation(
      stdair::LegCabin&);
00067
00069     static double optimiseUsingOnDForecast (stdair::FlightDate&,
00070                                             const bool& iReduceFluctuations = false);
00071
00072   private:
00076     static bool optimise (stdair::LegDate&,
00077                           const stdair::OptimisationMethod&);
00081     static bool optimise (stdair::LegCabin&,
00082                           const stdair::OptimisationMethod&);
00083
00084
00085   };
00086 }
00087 #endif // __RMOL_COMMAND_OPTIMISER_HPP
```

26.115 rmol/command/PreOptimiser.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <cmath>
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/basic/RandomGeneration.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/FlightDate.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/SegmentSnapshotTable.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/Utilities.hpp>
#include <rmol/command/PreOptimiser.hpp>
#include <rmol/command/DemandInputPreparation.hpp>
#include <rmol/command/FareAdjustment.hpp>
#include <rmol/command/MarginalRevenueTransformation.hpp>
```

Namespaces

- RMOL

26.116 PreOptimiser.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 #include <cmath>
00008 // StdAir
00009 #include <stdair/basic/BasConst_General.hpp>
00010 #include <stdair/basic/BasConst_Inventory.hpp>
00011 #include <stdair/basic/RandomGeneration.hpp>
00012 #include <stdair/bom/BomManager.hpp>
00013 #include <stdair/bom/FlightDate.hpp>
00014 #include <stdair/bom/SegmentDate.hpp>
00015 #include <stdair/bom/SegmentCabin.hpp>
00016 #include <stdair/bom/SegmentSnapshotTable.hpp>
00017 #include <stdair/bom/BookingClass.hpp>
00018 #include <stdair/service/Logger.hpp>
00019 // RMOL
00020 #include <rmol/bom/Utilities.hpp>
00021 #include <rmol/command/PreOptimiser.hpp>
00022 #include <rmol/command/DemandInputPreparation.hpp>
00023 #include <rmol/command/FareAdjustment.hpp>
00024 #include <rmol/command/MarginalRevenueTransformation.hpp>
00025
00026 namespace RMOL {
00027
00028   // ////////////////////////////////////////////////////////////////////
00029   bool PreOptimiser::
00030   preOptimise (stdair::FlightDate& ioFlightDate,
00031                const stdair::PreOptimisationMethod& iPreOptimisationMethod) {
00032     bool isSucceeded = true;
00033     const stdair::SegmentDateList_T& lSDList =
00034       stdair::BomManager::getList<stdair::SegmentDate> (ioFlightDate);
00035     for (stdair::SegmentDateList_T::const_iterator itSD = lSDList.begin();
00036          itSD != lSDList.end(); ++itSD) {
00037       stdair::SegmentDate* lSD_ptr = *itSD;
00038       assert (lSD_ptr != NULL);
00039
00040       //
00041       const stdair::SegmentCabinList_T& lSCList =
00042         stdair::BomManager::getList<stdair::SegmentCabin> (*lSD_ptr);
00043       for (stdair::SegmentCabinList_T::const_iterator itSC = lSCList.begin();
```

```
00044           itSC != lSCList.end(); ++itSC) {
00045         stdair::SegmentCabin* lSC_ptr = *itSC;
00046         assert (lSC_ptr != NULL);
00047
00048         //
00049         // STDAIR_LOG_NOTIFICATION (ioFlightDate.getDepartureDate()
00050         //                          << ";" << lSegmentDTD);
00051         bool isPreOptimised = preOptimise (*lSC_ptr, iPreOptimisationMethod);
00052         if (isPreOptimised == false) {
00053           isSucceeded = false;
00054         }
00055       }
00056     }
00057
00058     return isSucceeded;
00059   }
00060
00061   // ////////////////////////////////////////////////////////////////////
00062   bool PreOptimiser::
00063   preOptimise (stdair::SegmentCabin& ioSegmentCabin,
00064                const stdair::PreOptimisationMethod& iPreOptimisationMethod) {
00065     const stdair::PreOptimisationMethod::EN_PreOptimisationMethod& lPreOptimisationMethod =
00066       iPreOptimisationMethod.getMethod();
00067     switch (lPreOptimisationMethod) {
00068     case stdair::PreOptimisationMethod::NONE: {
00069       return DemandInputPreparation::prepareDemandInput (
      ioSegmentCabin);
00070     }
00071     case stdair::PreOptimisationMethod::FA: {
00072       return FareAdjustment::adjustYield (ioSegmentCabin);
00073     }
00074     case stdair::PreOptimisationMethod::MRT: {
00075       if (ioSegmentCabin.getFareFamilyStatus()==false) {
00076         return FareAdjustment::adjustYield (ioSegmentCabin);
00077       } else {
00078         return MarginalRevenueTransformation::
00079           prepareDemandInput (ioSegmentCabin);
00080       }
00081     }
00082     default:{
00083       assert (false);
00084       break;
00085     }
00086     }
00087     return false;
00088   }
00089
00090   // ////////////////////////////////////////////////////////////////////
00091   // void PreOptimiser::
00092   // setDemandForecastsToZero(const stdair::SegmentCabin& iSegmentCabin) {
00093   // }
00094 }
```

26.117 rmol/command/PreOptimiser.hpp File Reference

```
#include <map>
#include <stdair/stdair_inventory_types.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::PreOptimiser

Namespaces

- stdair

  Forward declarations.
- RMOL

26.118 PreOptimiser.hpp

```
00001 #ifndef __RMOL_COMMAND_PREOPTIMISER_HPP
```

```
00002 #define __RMOL_COMMAND_PREOPTIMISER_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <map>
00009 // StdAir
00010 #include <stdair/stdair_inventory_types.hpp>
00011 // RMOL
00012 #include <rmol/RMOL_Types.hpp>
00013
00014 // Forward declarations
00015 namespace stdair {
00016   class FlightDate;
00017   class SegmentCabin;
00018 }
00019
00020 namespace RMOL {
00022   class PreOptimiser {
00023   public:
00027     static bool preOptimise (stdair::FlightDate&,
00028                              const stdair::PreOptimisationMethod&);
00029
00030   private:
00034     static bool preOptimise (stdair::SegmentCabin&,
00035                              const stdair::PreOptimisationMethod&);
00036
00040     //static void setDemandForecastsToZero (const stdair::SegmentCabin&);
00041
00042   };
00043 }
00044 #endif // __RMOL_COMMAND_PREOPTIMISER_HPP
```

26.119 rmol/command/QForecasting.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <cmath>
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/LegDate.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/SegmentSnapshotTable.hpp>
#include <stdair/bom/FareFamily.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/Utilities.hpp>
#include <rmol/bom/SegmentSnapshotTableHelper.hpp>
#include <rmol/bom/HistoricalBookingHolder.hpp>
#include <rmol/bom/HistoricalBooking.hpp>
#include <rmol/command/QForecasting.hpp>
#include <rmol/command/Detruncator.hpp>
```

Namespaces

- RMOL

26.120 QForecasting.cpp

```
00001 // //////////////////////////////////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////////////////////////////////
00004 // STL
```

```
00005 #include <cassert>
00006 #include <sstream>
00007 #include <cmath>
00008 // StdAir
00009 #include <stdair/basic/BasConst_General.hpp>
00010 #include <stdair/basic/BasConst_Inventory.hpp>
00011 #include <stdair/bom/BomManager.hpp>
00012 #include <stdair/bom/LegDate.hpp>
00013 #include <stdair/bom/SegmentDate.hpp>
00014 #include <stdair/bom/LegCabin.hpp>
00015 #include <stdair/bom/SegmentCabin.hpp>
00016 #include <stdair/bom/SegmentSnapshotTable.hpp>
00017 #include <stdair/bom/FareFamily.hpp>
00018 #include <stdair/bom/BookingClass.hpp>
00019 #include <stdair/service/Logger.hpp>
00020 // RMOL
00021 #include <rmol/bom/Utilities.hpp>
00022 #include <rmol/bom/SegmentSnapshotTableHelper.hpp>
00023 #include <rmol/bom/HistoricalBookingHolder.hpp>
00024 #include <rmol/bom/HistoricalBooking.hpp>
00025 #include <rmol/command/QForecasting.hpp>
00026 #include <rmol/command/Detruncator.hpp>
00027
00028 namespace RMOL {
00029   // ////////////////////////////////////////////////////////////////////
00030   bool QForecasting::
00031   forecast (stdair::SegmentCabin& ioSegmentCabin,
00032             const stdair::Date_T& iCurrentDate,
00033             const stdair::DTD_T& iCurrentDTD,
00034             const stdair::UnconstrainingMethod& iUnconstrainingMethod,
00035             const stdair::NbOfSegments_T& iNbOfDepartedSegments) {
00036     // Retrieve the snapshot table.
00037     const stdair::SegmentSnapshotTable& lSegmentSnapshotTable =
00038       ioSegmentCabin.getSegmentSnapshotTable();
00039
00040     // Retrieve the FRAT5Curve.
00041     const stdair::FareFamilyList_T& lFFList =
00042       stdair::BomManager::getList<stdair::FareFamily>(ioSegmentCabin);
00043     stdair::FareFamilyList_T::const_reverse_iterator itFF = lFFList.rbegin();
00044     assert (itFF != lFFList.rend());
00045     stdair::FareFamily* lFF_ptr = *itFF;
00046     assert (lFF_ptr != NULL);
00047     const stdair::FRAT5Curve_T lFRAT5Curve = lFF_ptr->getFrat5Curve();
00048
00049     // Retrieve the booking class list and compute the sell up curves
00050     // and the dispatching curves.
00051     const stdair::BookingClassList_T& lBCList =
00052       stdair::BomManager::getList<stdair::BookingClass>(ioSegmentCabin);
00053     const stdair::BookingClassSellUpCurveMap_T lBCSellUpCurveMap =
00054       Utilities::computeSellUpFactorCurves (lFRAT5Curve, lBCList);
00055     const stdair::BookingClassDispatchingCurveMap_T lBCDispatchingCurveMap =
00056       Utilities::computeDispatchingFactorCurves (lFRAT5Curve,
      lBCList);
00057
00058
00059     // Browse all remaining DCP's and do unconstraining, forecasting
00060     // and dispatching.
00061     const stdair::DCPList_T lWholeDCPList = stdair::DEFAULT_DCP_LIST;
00062     stdair::DCPList_T::const_iterator itDCP = lWholeDCPList.begin();
00063     stdair::DCPList_T::const_iterator itNextDCP = itDCP; ++itNextDCP;
00064     for (; itNextDCP != lWholeDCPList.end(); ++itDCP, ++itNextDCP) {
00065       const stdair::DCP_T& lCurrentDCP = *itDCP;
00066       const stdair::DCP_T& lNextDCP = *itNextDCP;
00067
00068       // The end of the interval is after the current DTD.
00069       if (lNextDCP < iCurrentDTD) {
00070         // Get the number of similar segments which has already passed the
00071         // (lNextDCP+1)
00072         const stdair::NbOfSegments_T& lNbOfUsableSegments =
00073           SegmentSnapshotTableHelper::
00074           getNbOfSegmentAlreadyPassedThisDTD (
      lSegmentSnapshotTable,
00075                                               lNextDCP+1,
00076                                               iCurrentDate);
00077         stdair::NbOfSegments_T lSegmentBegin = 0;
00078         const stdair::NbOfSegments_T lSegmentEnd = lNbOfUsableSegments-1;
00079         if (iNbOfDepartedSegments > 52) {
00080           lSegmentBegin = iNbOfDepartedSegments - 52;
00081         }
00082
00083         // Retrieve the historical bookings and convert them to
00084         // Q-equivalent bookings.
00085         HistoricalBookingHolder lHBHolder;
00086         preparePriceOrientedHistoricalBooking (ioSegmentCabin,
00087                                                lSegmentSnapshotTable, lHBHolder,
00088                                                lCurrentDCP, lNextDCP,
00089                                                lSegmentBegin, lSegmentEnd,
```

```
00090                                             lBCSellUpCurveMap);
00091
00092         // Unconstrain the historical bookings.
00093         Detruncator::unconstrain (lHBHolder, iUnconstrainingMethod);
00094
00095         // Retrieve the historical unconstrained demand and perform the
00096         // forecasting.
00097         stdair::UncDemVector_T lUncDemVector;
00098         const short lNbOfHistoricalFlights = lHBHolder.getNbOfFlights();
00099         for (short i = 0; i < lNbOfHistoricalFlights; ++i) {
00100           const stdair::NbOfBookings_T& lUncDemand =
00101             lHBHolder.getUnconstrainedDemand (i);
00102           lUncDemVector.push_back (lUncDemand);
00103         }
00104         stdair::MeanValue_T lMean = 0.0;
00105         stdair::StdDevValue_T lStdDev = 0.0;
00106         Utilities::computeDistributionParameters (lUncDemVector,
00107                                                   lMean, lStdDev);
00108
00109         // Dispatch the forecast to all the classes.
00110         Utilities::dispatchDemandForecast (lBCDispatchingCurveMap,
00111                                            lMean, lStdDev, lCurrentDCP);
00112
00113         // Dispatch the forecast to all classes for Fare Adjustment or MRT.
00114         // The sell-up probability will be used in this case.
00115         Utilities::dispatchDemandForecastForFA (
      lBCDispatchingCurveMap,
00116                                                 lMean, lStdDev, lCurrentDCP);
00117
00118         // Add the demand forecast to the fare family.
00119         const stdair::MeanValue_T& lCurrentMean = lFF_ptr->getMean();
00120         const stdair::StdDevValue_T& lCurrentStdDev = lFF_ptr->getStdDev();
00121
00122         const stdair::MeanValue_T lNewMean = lCurrentMean + lMean;
00123         const stdair::StdDevValue_T lNewStdDev =
00124           std::sqrt (lCurrentStdDev * lCurrentStdDev + lStdDev * lStdDev);
00125
00126         lFF_ptr->setMean (lNewMean);
00127         lFF_ptr->setStdDev (lNewStdDev);
00128       }
00129     }
00130
00131     return true;
00132   }
00133
00134   // ////////////////////////////////////////////////////////////////////
00135   void QForecasting::preparePriceOrientedHistoricalBooking
00136     (const stdair::SegmentCabin& iSegmentCabin,
00137      const stdair::SegmentSnapshotTable& iSegmentSnapshotTable,
00138      HistoricalBookingHolder& ioHBHolder,
00139      const stdair::DCP_T& iDCPBegin, const stdair::DCP_T& iDCPEnd,
00140      const stdair::NbOfSegments_T& iSegmentBegin,
00141      const stdair::NbOfSegments_T& iSegmentEnd,
00142      const stdair::BookingClassSellUpCurveMap_T& iBCSellUpCurveMap) {
00143
00144     // Retrieve the segment-cabin index within the snapshot table
00145     std::ostringstream lSCMapKey;
00146     lSCMapKey << stdair::DEFAULT_SEGMENT_CABIN_VALUE_TYPE
00147               << iSegmentCabin.describeKey();
00148     const stdair::ClassIndex_T& lCabinIdx =
00149       iSegmentSnapshotTable.getClassIndex (lSCMapKey.str());
00150
00151     // Retrieve the gross daily booking and availability snapshots.
00152     const stdair::ConstSegmentCabinDTDRangeSnapshotView_T lBookingView =
00153       iSegmentSnapshotTable.getConstSegmentCabinDTDRangePriceOrientedGrossBookingSnapshotView (
      iSegmentBegin, iSegmentEnd, iDCPEnd, iDCPBegin);
00154     const stdair::ConstSegmentCabinDTDRangeSnapshotView_T lAvlView =
00155       iSegmentSnapshotTable.getConstSegmentCabinDTDRangeAvailabilitySnapshotView (iSegmentBegin,
      iSegmentEnd, iDCPEnd, iDCPBegin);
00156
00157     // Browse the list of segments and build the historical booking holder.
00158     const stdair::ClassIndexMap_T& lVTIdxMap =
00159       iSegmentSnapshotTable.getClassIndexMap();
00160     const stdair::NbOfClasses_T lNbOfClasses = lVTIdxMap.size();
00161
00162     for (short i = 0; i <= iSegmentEnd-iSegmentBegin; ++i) {
00163       stdair::Flag_T lCensorshipFlag = false;
00164       const short lNbOfDTDs = iDCPBegin - iDCPEnd + 1;
00165       const stdair::UnsignedIndex_T lIdx = i*lNbOfClasses + lCabinIdx;
00166
00167       // Parse the DTDs during the period and compute the censorship flag
00168       for (short j = 0; j < lNbOfDTDs; ++j) {
00169         // Check if the data has been censored during this day.
00170         // STDAIR_LOG_DEBUG ("i: " << i << ", NbOfClasses: " << lNbOfClasses
00171         //                   << ", ClassIdx: " << iClassIdx << ", j: " << j);
00172         if (lAvlView[lIdx][j] < 1.0) {
00173           lCensorshipFlag = true;
```

```
00174           break;
00175         }
00176       }
00177
00178       // Compute the Q-equivalent bookings
00179       stdair::NbOfBookings_T lNbOfHistoricalBkgs = 0.0;
00180       for (stdair::BookingClassSellUpCurveMap_T::const_iterator itBCSUC =
00181              iBCSellUpCurveMap.begin();
00182            itBCSUC != iBCSellUpCurveMap.end(); ++itBCSUC) {
00183         const stdair::BookingClass* lBookingClass_ptr = itBCSUC->first;
00184         assert (lBookingClass_ptr != NULL);
00185         const stdair::SellUpCurve_T& lSellUpCurve = itBCSUC->second;
00186         stdair::SellUpCurve_T::const_iterator itSellUp =
00187           lSellUpCurve.find (iDCPBegin);
00188         assert (itSellUp != lSellUpCurve.end());
00189         const stdair::SellupProbability_T& lSellUp = itSellUp->second;
00190         assert (lSellUp != 0);
00191
00192         // Retrieve the number of bookings
00193         const stdair::ClassIndex_T& lClassIdx =
00194           iSegmentSnapshotTable.getClassIndex(lBookingClass_ptr->describeKey());
00195         stdair::NbOfBookings_T lNbOfBookings = 0.0;
00196         for (short j = 0; j < lNbOfDTDs; ++j) {
00197           lNbOfBookings += lBookingView[i*lNbOfClasses + lClassIdx][j];
00198         }
00199
00200         const stdair::NbOfBookings_T lNbOfQEquivalentBkgs=lNbOfBookings/lSellUp;
00201         lNbOfHistoricalBkgs += lNbOfQEquivalentBkgs;
00202       }
00203
00204       HistoricalBooking lHistoricalBkg (lNbOfHistoricalBkgs, lCensorshipFlag);
00205       ioHBHolder.addHistoricalBooking (lHistoricalBkg);
00206     }
00207   }
00208 }
```

26.121 rmol/command/QForecasting.hpp File Reference

```
#include <stdair/stdair_inventory_types.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::QForecasting

Namespaces

- stdair

  Forward declarations.
- RMOL

26.122 QForecasting.hpp

```
00001 #ifndef __RMOL_COMMAND_QFORECASTING_HPP
00002 #define __RMOL_COMMAND_QFORECASTING_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // StdAir
00008 #include <stdair/stdair_inventory_types.hpp>
00009 // RMOL
00010 #include <rmol/RMOL_Types.hpp>
00011
00012 // Forward declarations
00013 namespace stdair {
00014   class SegmentCabin;
00015   class SegmentSnapshotTable;
00016 }
00017
00018 namespace RMOL {
```

```
00019   // Forward declarations
00020   struct HistoricalBookingHolder;
00021
00023   class QForecasting {
00024   public:
00031     static bool forecast (stdair::SegmentCabin&,
00032                           const stdair::Date_T&, const stdair::DTD_T&,
00033                           const stdair::UnconstrainingMethod&,
00034                           const stdair::NbOfSegments_T&);
00035
00044     static void preparePriceOrientedHistoricalBooking
00045     (const stdair::SegmentCabin&, const stdair::SegmentSnapshotTable&,
00046      HistoricalBookingHolder&, const stdair::DCP_T&, const stdair::DCP_T&,
00047      const stdair::NbOfSegments_T&, const stdair::NbOfSegments_T&,
00048      const stdair::BookingClassSellUpCurveMap_T&);
00049   };
00050 }
00051 #endif // __RMOL_COMMAND_QFORECASTING_HPP
```

26.123 rmol/factory/FacRmolServiceContext.cpp File Reference

```
#include <cassert>
#include <stdair/service/FacSupervisor.hpp>
#include <rmol/factory/FacRmolServiceContext.hpp>
#include <rmol/service/RMOL_ServiceContext.hpp>
```

Namespaces

- RMOL

26.124 FacRmolServiceContext.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 // StdAir
00007 #include <stdair/service/FacSupervisor.hpp>
00008 // RMOL
00009 #include <rmol/factory/FacRmolServiceContext.hpp>
00010 #include <rmol/service/RMOL_ServiceContext.hpp>
00011
00012 namespace RMOL {
00013
00014   FacRmolServiceContext* FacRmolServiceContext::_instance = NULL;
00015
00016   // ////////////////////////////////////////////////////////////////////
00017   FacRmolServiceContext::~FacRmolServiceContext() {
00018     _instance = NULL;
00019   }
00020
00021   // ////////////////////////////////////////////////////////////////////
00022   FacRmolServiceContext& FacRmolServiceContext::instance
      () {
00023
00024     if (_instance == NULL) {
00025       _instance = new FacRmolServiceContext();
00026       assert (_instance != NULL);
00027
00028       stdair::FacSupervisor::instance().registerServiceFactory (_instance);
00029     }
00030     return *_instance;
00031   }
00032
00033   // ////////////////////////////////////////////////////////////////////
00034   RMOL_ServiceContext& FacRmolServiceContext::create() {
00035     RMOL_ServiceContext* aServiceContext_ptr = NULL;
00036
00037     aServiceContext_ptr = new RMOL_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 }
```

26.125 rmol/factory/FacRmolServiceContext.hpp File Reference

```
#include <string>
#include <stdair/stdair_basic_types.hpp>
#include <stdair/service/FacServiceAbstract.hpp>
```

Classes

- class RMOL::FacRmolServiceContext

  Factory for the service context.

Namespaces

- RMOL

26.126 FacRmolServiceContext.hpp

```
00001 #ifndef __RMOL_FAC_FACRMOLSERVICECONTEXT_HPP
00002 #define __RMOL_FAC_FACRMOLSERVICECONTEXT_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 RMOL {
00014
00016   class RMOL_ServiceContext;
00017
00018
00022   class FacRmolServiceContext : public stdair::FacServiceAbstract {
00023   public:
00024
00031     static FacRmolServiceContext& instance();
00032
00039     ~FacRmolServiceContext();
00040
00048     RMOL_ServiceContext& create();
00049
00050
00051   protected:
00057     FacRmolServiceContext() {}
00058
00059
00060   private:
00064     static FacRmolServiceContext* _instance;
00065   };
00066
00067 }
00068 #endif // __RMOL_FAC_FACRMOLSERVICECONTEXT_HPP
```

26.127 rmol/RMOL_Service.hpp File Reference

```
#include <string>
```

```
#include <stdair/stdair_basic_types.hpp>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/stdair_service_types.hpp>
#include <stdair/stdair_maths_types.hpp>
#include <stdair/basic/UnconstrainingMethod.hpp>
#include <stdair/basic/ForecastingMethod.hpp>
#include <stdair/basic/PreOptimisationMethod.hpp>
#include <stdair/basic/OptimisationMethod.hpp>
#include <stdair/basic/PartnershipTechnique.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::RMOL_Service

  Interface for the RMOL Services.

Namespaces

- stdair

  Forward declarations.
- RMOL

26.128 RMOL_Service.hpp

```
00001 #ifndef __RMOL_SVC_RMOL_SERVICE_HPP
00002 #define __RMOL_SVC_RMOL_SERVICE_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <string>
00009 // StdAir
00010 #include <stdair/stdair_basic_types.hpp>
00011 #include <stdair/stdair_inventory_types.hpp>
00012 #include <stdair/stdair_service_types.hpp>
00013 #include <stdair/stdair_maths_types.hpp>
00014 #include <stdair/basic/UnconstrainingMethod.hpp>
00015 #include <stdair/basic/ForecastingMethod.hpp>
00016 #include <stdair/basic/PreOptimisationMethod.hpp>
00017 #include <stdair/basic/OptimisationMethod.hpp>
00018 #include <stdair/basic/PartnershipTechnique.hpp>
00019 // RMOL
00020 #include <rmol/RMOL_Types.hpp>
00021
00023 namespace stdair {
00024   class FlightDate;
00025   struct BasLogParams;
00026   struct BasDBParams;
00027   class BomRoot;
00028   class AirlineClassList;
00029   class YieldFeatures;
00030   class Inventory;
00031   class OnDDate;
00032   class SegmentCabin;
00033 }
00034
00035 namespace RMOL {
00036
00038   class RMOL_ServiceContext;
00039
00043   class RMOL_Service {
00044   public:
00045     // ////////// Constructors and destructors //////////
00061     RMOL_Service (const stdair::BasLogParams&, const stdair::BasDBParams&);
00062
00074     RMOL_Service (const stdair::BasLogParams&);
00075
00091     RMOL_Service (stdair::STDAIR_ServicePtr_T);
00092
```

```
00116     void parseAndLoad (const stdair::CabinCapacity_T& iCabinCapacity,
00117                        const stdair::Filename_T& iDemandAndClassDataFile);
00118
00122     void setUpStudyStatManager();
00123
00127     ~RMOL_Service();
00128
00129
00130   public:
00131     // /////////////// Business Methods /////////////////
00137     void buildSampleBom();
00138
00142     void clonePersistentBom ();
00143
00148     void buildComplementaryLinks (stdair::BomRoot&);
00149
00153     void optimalOptimisationByMCIntegration (const int K);
00154
00158     void optimalOptimisationByDP();
00159
00163     void heuristicOptimisationByEmsr();
00164
00168     void heuristicOptimisationByEmsrA();
00169
00173     void heuristicOptimisationByEmsrB();
00174
00178     void heuristicOptimisationByMCIntegrationForQFF();
00179
00183     void heuristicOptimisationByEmsrBForQFF();
00184
00188     void MRTForNewQFF();
00189
00197     const stdair::SegmentCabin&
00198     retrieveDummySegmentCabin(const bool isForFareFamilies = false);
00199
00203     bool optimise (stdair::FlightDate&, const stdair::DateTime_T&,
00204                    const stdair::UnconstrainingMethod&,
00205                    const stdair::ForecastingMethod&,
00206                    const stdair::PreOptimisationMethod&,
00207                    const stdair::OptimisationMethod&,
00208                    const stdair::PartnershipTechnique&);
00209
00214     // O&D based forecast
00215     void forecastOnD (const stdair::DateTime_T&);
00216
00217     stdair::YieldFeatures* getYieldFeatures(const stdair::OnDDate&, const
      stdair::CabinCode_T&,
00218                                             stdair::BomRoot&);
00219
00220     void forecastOnD (const stdair::YieldFeatures&, stdair::OnDDate&,
00221                       const stdair::CabinCode_T&, const stdair::DTD_T&,
00222                       stdair::BomRoot&);
00223
00224     void setOnDForecast (const stdair::AirlineClassList&, const stdair::MeanValue_T&,
00225                          const stdair::StdDevValue_T&, stdair::OnDDate&, const stdair::CabinCode_T&,
00226                          stdair::BomRoot&);
00227
00228     // Single segment O&D
00229     void setOnDForecast (const stdair::AirlineCode_T&, const stdair::Date_T&, const
      stdair::AirportCode_T&,
00230                          const stdair::AirportCode_T&, const stdair::CabinCode_T&, const
      stdair::ClassCode_T&,
00231                          const stdair::MeanValue_T&, const stdair::StdDevValue_T&, const stdair::Yield_T&,
      stdair::BomRoot&);
00232
00233     // Multiple segment O&D
00234     void setOnDForecast (const stdair::AirlineCodeList_T&, const stdair::AirlineCode_T&,const
       stdair::Date_T&,
00235                          const stdair::AirportCode_T&, const stdair::AirportCode_T&, const
      stdair::CabinCode_T&,
00236                          const stdair::ClassCodeList_T&, const stdair::MeanValue_T&, const
      stdair::StdDevValue_T&,
00237                          const stdair::Yield_T&, stdair::BomRoot&);
00238
00239     // Initialise (or re-initialise) the demand projections in all leg cabins
00240     void resetDemandInformation (const stdair::DateTime_T&);
00241
00242     void resetDemandInformation (const stdair::DateTime_T&, const stdair::Inventory&)
      ;
00243
00244     /* Projection of demand */
00245
00246     // Aggregated demand at booking class level.
00247     void projectAggregatedDemandOnLegCabins(const stdair::DateTime_T&);
00248
00249     // Static rule prorated yield
00250     void projectOnDDemandOnLegCabinsUsingYP(const stdair::DateTime_T&);
```

```
00251
00252     // Displacement-adjusted yield
00253     void projectOnDDemandOnLegCabinsUsingDA(const stdair::DateTime_T&);
00254
00255     // Dynamic yield proration (PF = BP_i/BP_{total}, where BP_{total} = sum(BP_i))
00256     void projectOnDDemandOnLegCabinsUsingDYP(const stdair::DateTime_T&);
00257
00258     void projectOnDDemandOnLegCabinsUsingDYP(const stdair::DateTime_T&,
    const stdair::Inventory&);
00259
00261     // O&D-based optimisation (using demand aggregation or demand aggregation).
00262     void optimiseOnD (const stdair::DateTime_T&);
00263
00264     // O&D-based optimisation using displacement-adjusted yield.
00265     void optimiseOnDUsingRMCooperation (const stdair::DateTime_T&);
00266
00267     // Advanced version of O&D-based optimisation using displacement-adjusted yield.
00268     // Network optimisation instead of separate inventory optimisation.
00269     void optimiseOnDUsingAdvancedRMCooperation (const
    stdair::DateTime_T&);
00270
00271     // Update bid priceand send to partners
00272     void updateBidPrice (const stdair::DateTime_T&);
00273     void updateBidPrice (const stdair::FlightDate&, stdair::BomRoot&);
00274
00275   public:
00276     // ///////////////// Export support methods /////////////////
00287     std::string jsonExport (const stdair::AirlineCode_T&,
00288                             const stdair::FlightNumber_T&,
00289                             const stdair::Date_T& iDepartureDate) const;
00290
00291
00292   public:
00293     // ///////////////// Display support methods /////////////////
00301     std::string csvDisplay() const;
00302
00303
00304   private:
00305     // /////// Construction and Destruction helper methods ///////
00309     RMOL_Service();
00310
00314     RMOL_Service (const RMOL_Service&);
00315
00325     stdair::STDAIR_ServicePtr_T initStdAirService (const stdair::BasLogParams&,
00326                                                    const stdair::BasDBParams&);
00327
00336     stdair::STDAIR_ServicePtr_T initStdAirService (const stdair::BasLogParams&);
00337
00346     void addStdAirService (stdair::STDAIR_ServicePtr_T,
00347                            const bool iOwnStdairService);
00348
00353     void initServiceContext();
00354
00361     void initRmolService();
00362
00366     void finalise();
00367
00368
00369   private:
00370     // ////////// Service Context //////////
00374     RMOL_ServiceContext* _rmolServiceContext;
00375
00377     stdair::Date_T _previousForecastDate;
00378   };
00379 }
00380 #endif // __RMOL_SVC_RMOL_SERVICE_HPP
```

26.129 rmol/RMOL_Types.hpp File Reference

```
#include <map>
#include <vector>
#include <boost/shared_ptr.hpp>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/stdair_rm_types.hpp>
#include <stdair/stdair_exceptions.hpp>
```

Classes

- class RMOL::OverbookingException

  Overbooking-related exception.
- class RMOL::UnconstrainingException

  Unconstraining-related exception.
- class RMOL::EmptyNestingStructException

  Empty nesting structure in unconstrainer exception.
- class RMOL::MissingDCPException

  Missing a DCP in unconstrainer exception.
- class RMOL::OptimisationException

  Optimisation-related exception.
- class RMOL::PolicyException

  Policy-related exception.
- class RMOL::ConvexHullException

  Convex Hull-related exception.
- class RMOL::EmptyConvexHullException

  Empty convex hull exception.
- class RMOL::FirstPolicyNotNullException

  Missing policy NULL in convex hull exception.
- class RMOL::YieldConvexHullException

  Yield convex hull exception.
- class RMOL::FareFamilyException

  Fare Family-related exception.
- class RMOL::EmptyBookingClassListException

  Empty Booking Class List of Fare Family exception.
- class RMOL::MissingBookingClassInFareFamilyException

  Missing Booking Class in Fare Family exception.
- class RMOL::FareFamilyDemandVectorSizeException

  Fare Family demand exception.

Namespaces

- stdair

  Forward declarations.
- RMOL

Typedefs

- typedef boost::shared_ptr< RMOL_Service > RMOL::RMOL_ServicePtr_T
- typedef std::vector< stdair::Flag_T > RMOL::FlagVector_T
- typedef std::map< stdair::BookingClass *, stdair::MeanStdDevPair_T > RMOL::BookingClassMeanStd↩ DevPairMap_T

26.130 RMOL_Types.hpp

```
00001 #ifndef __RMOL_RMOL_TYPES_HPP
00002 #define __RMOL_RMOL_TYPES_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <map>
00009 #include <vector>
00010 // Boost
00011 #include <boost/shared_ptr.hpp>
00012 // StdAir
00013 #include <stdair/stdair_inventory_types.hpp>
00014 #include <stdair/stdair_rm_types.hpp>
00015 #include <stdair/stdair_exceptions.hpp>
00016
00017 // Forward declarations.
00018 namespace stdair {
00019   class BookingClass;
00020 }
00021
00022
00023 namespace RMOL {
00024
00025   // Forward declarations
00026   class RMOL_Service;
00027
00028   // ///////// Exceptions ////////////
00032   class OverbookingException : public stdair::RootException {
00033   public:
00035     OverbookingException (const std::string& iWhat)
00036       : stdair::RootException (iWhat) {}
00037   };
00038
00042   class UnconstrainingException : public stdair::RootException {
00043   public:
00045     UnconstrainingException (const std::string& iWhat)
00046       : stdair::RootException (iWhat) {}
00047   };
00048
00052   class EmptyNestingStructException : public
      UnconstrainingException {
00053   public:
00055     EmptyNestingStructException (const std::string& iWhat)
00056       : UnconstrainingException (iWhat) {}
00057   };
00058
00062   class MissingDCPException : public UnconstrainingException {
00063   public:
00065     MissingDCPException (const std::string& iWhat)
00066       : UnconstrainingException (iWhat) {}
00067   };
00068
00072   class OptimisationException : public stdair::RootException {
00073   public:
00075     OptimisationException (const std::string& iWhat)
00076       : stdair::RootException (iWhat) {}
00077   };
00078
00082   class PolicyException : public stdair::RootException {
00083   public:
00085     PolicyException (const std::string& iWhat)
00086       : stdair::RootException (iWhat) {}
00087   };
00088
00089
00093   class ConvexHullException : public PolicyException {
00094   public:
00096     ConvexHullException (const std::string& iWhat)
00097       : PolicyException (iWhat) {}
00098   };
00099
00103   class EmptyConvexHullException : public
      ConvexHullException {
00104   public:
00106     EmptyConvexHullException (const std::string& iWhat)
00107       : ConvexHullException (iWhat) {}
00108   };
00109
00113   class FirstPolicyNotNullException : public
      ConvexHullException {
00114   public:
00116     FirstPolicyNotNullException (const std::string& iWhat)
00117       : ConvexHullException (iWhat) {}
00118   };
```

```
00119
00123   class YieldConvexHullException : public
      ConvexHullException {
00124   public:
00126     YieldConvexHullException (const std::string& iWhat)
00127       : ConvexHullException (iWhat) {}
00128   };
00129
00130
00134   class FareFamilyException : public stdair::RootException {
00135   public:
00137     FareFamilyException (const std::string& iWhat)
00138       : stdair::RootException (iWhat) {}
00139   };
00140
00144   class EmptyBookingClassListException : public
      FareFamilyException {
00145   public:
00147     EmptyBookingClassListException (const std::string& iWhat)
00148       : FareFamilyException (iWhat) {}
00149   };
00150
00154   class MissingBookingClassInFareFamilyException : public
      FareFamilyException {
00155   public:
00157     MissingBookingClassInFareFamilyException (const std::string&
      iWhat)
00158       : FareFamilyException (iWhat) {}
00159   };
00160
00164   class FareFamilyDemandVectorSizeException : public
      FareFamilyException {
00165   public:
00167     FareFamilyDemandVectorSizeException (const std::string& iWhat)
00168       : FareFamilyException (iWhat) {}
00169   };
00170
00171
00172   // //////// Type definitions /////////
00176   typedef boost::shared_ptr<RMOL_Service> RMOL_ServicePtr_T;
00177
00179   typedef std::vector<stdair::Flag_T> FlagVector_T;
00180
00182   typedef std::map<stdair::BookingClass*, stdair::MeanStdDevPair_T>
      BookingClassMeanStdDevPairMap_T;
00183 }
00184 #endif // __RMOL_RMOL_TYPES_HPP
```

26.131 rmol/service/RMOL_Service.cpp File Reference

```
#include <cassert>
```

```
#include <boost/make_shared.hpp>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/basic/BasChronometer.hpp>
#include <stdair/basic/ContinuousAttributeLite.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/BomRetriever.hpp>
#include <stdair/bom/BomRoot.hpp>
#include <stdair/bom/Inventory.hpp>
#include <stdair/bom/FlightDate.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/LegDate.hpp>
#include <stdair/bom/YieldFeatures.hpp>
#include <stdair/bom/AirportPair.hpp>
#include <stdair/bom/PosChannel.hpp>
#include <stdair/bom/DatePeriod.hpp>
#include <stdair/bom/TimePeriod.hpp>
#include <stdair/bom/AirlineClassList.hpp>
#include <stdair/basic/BasConst_Request.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/bom/OnDDate.hpp>
#include <stdair/bom/OnDDateTypes.hpp>
#include <stdair/command/CmdBomManager.hpp>
#include <stdair/service/Logger.hpp>
#include <stdair/STDAIR_Service.hpp>
#include <rmol/basic/BasConst_RMOL_Service.hpp>
#include <rmol/factory/FacRmolServiceContext.hpp>
#include <rmol/command/InventoryParser.hpp>
#include <rmol/command/Optimiser.hpp>
#include <rmol/command/PreOptimiser.hpp>
#include <rmol/command/Forecaster.hpp>
#include <rmol/service/RMOL_ServiceContext.hpp>
#include <rmol/RMOL_Service.hpp>
```

Namespaces

- RMOL

26.132 RMOL_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/stdair_inventory_types.hpp>
00010 #include <stdair/basic/BasChronometer.hpp>
00011 #include <stdair/basic/ContinuousAttributeLite.hpp>
00012 #include <stdair/bom/BomManager.hpp>
00013 #include <stdair/bom/BomRetriever.hpp>
00014 #include <stdair/bom/BomRoot.hpp>
00015 #include <stdair/bom/Inventory.hpp>
00016 #include <stdair/bom/FlightDate.hpp>
00017 #include <stdair/bom/LegCabin.hpp>
00018 #include <stdair/bom/LegDate.hpp>
00019 #include <stdair/bom/YieldFeatures.hpp>
00020 #include <stdair/bom/AirportPair.hpp>
00021 #include <stdair/bom/PosChannel.hpp>
```

```
00022 #include <stdair/bom/DatePeriod.hpp>
00023 #include <stdair/bom/TimePeriod.hpp>
00024 #include <stdair/bom/AirlineClassList.hpp>
00025 #include <stdair/basic/BasConst_Request.hpp>
00026 #include <stdair/basic/BasConst_Inventory.hpp>
00027 #include <stdair/bom/Inventory.hpp>
00028 #include <stdair/bom/FlightDate.hpp>
00029 #include <stdair/bom/SegmentDate.hpp>
00030 #include <stdair/bom/SegmentCabin.hpp>
00031 #include <stdair/bom/BookingClass.hpp>
00032 #include <stdair/bom/OnDDate.hpp>
00033 #include <stdair/bom/OnDDateTypes.hpp>
00034 #include <stdair/command/CmdBomManager.hpp>
00035 #include <stdair/service/Logger.hpp>
00036 #include <stdair/STDAIR_Service.hpp>
00037 // RMOL
00038 #include <rmol/basic/BasConst_RMOL_Service.hpp>
00039 #include <rmol/factory/FacRmolServiceContext.hpp>
00040 #include <rmol/command/InventoryParser.hpp>
00041 #include <rmol/command/Optimiser.hpp>
00042 #include <rmol/command/PreOptimiser.hpp>
00043 #include <rmol/command/Forecaster.hpp>
00044 #include <rmol/service/RMOL_ServiceContext.hpp>
00045 #include <rmol/RMOL_Service.hpp>
00046
00047 namespace RMOL {
00048
00049   // ////////////////////////////////////////////////////////////////////
00050   RMOL_Service::RMOL_Service()
00051   : _rmolServiceContext (NULL),
00052     _previousForecastDate (stdair::Date_T (2000, 1, 1)) {
00053     assert (false);
00054   }
00055
00056   // ////////////////////////////////////////////////////////////////////
00057   RMOL_Service::RMOL_Service (const RMOL_Service& iService) :
00058     _rmolServiceContext (NULL),
00059     _previousForecastDate (stdair::Date_T (2000, 1, 1)) {
00060     assert (false);
00061   }
00062
00063   // ////////////////////////////////////////////////////////////////////
00064   RMOL_Service::RMOL_Service (const stdair::BasLogParams& iLogParams) :
00065     _rmolServiceContext (NULL),
00066     _previousForecastDate (stdair::Date_T (2000, 1, 1)) {
00067
00068     // Initialise the STDAIR service handler
00069     stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
00070       initStdAirService (iLogParams);
00071
00072     // Initialise the service context
00073     initServiceContext();
00074
00075     // Add the StdAir service context to the RMOL service context
00076     // \note RMOL owns the STDAIR service resources here.
00077     const bool ownStdairService = true;
00078     addStdAirService (lSTDAIR_Service_ptr, ownStdairService);
00079
00080     // Initialise the (remaining of the) context
00081     initRmolService();
00082   }
00083
00084   // ////////////////////////////////////////////////////////////////////
00085   RMOL_Service::RMOL_Service (const stdair::BasLogParams& iLogParams,
00086                               const stdair::BasDBParams& iDBParams) :
00087     _rmolServiceContext (NULL),
00088     _previousForecastDate (stdair::Date_T (2000, 1, 1)) {
00089
00090     // Initialise the STDAIR service handler
00091     stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
00092       initStdAirService (iLogParams, iDBParams);
00093
00094     // Initialise the service context
00095     initServiceContext();
00096
00097     // Add the StdAir service context to the RMOL service context
00098     // \note RMOL owns the STDAIR service resources here.
00099     const bool ownStdairService = true;
00100     addStdAirService (lSTDAIR_Service_ptr, ownStdairService);
00101
00102     // Initialise the (remaining of the) context
00103     initRmolService();
00104   }
00105
00106   // ////////////////////////////////////////////////////////////////////
00107   RMOL_Service::RMOL_Service (stdair::STDAIR_ServicePtr_T ioSTDAIRServicePtr)
00108     : _rmolServiceContext (NULL),
```

```
00109       _previousForecastDate (stdair::Date_T (2000, 1, 1)) {
00110
00111     // Initialise the context
00112     initServiceContext();
00113
00114     // Add the StdAir service context to the RMOL service context.
00115     // \note RMOL does not own the STDAIR service resources here.
00116     const bool doesNotOwnStdairService = false;
00117     addStdAirService (ioSTDAIRServicePtr, doesNotOwnStdairService);
00118
00119     // Initialise the (remaining of the) context
00120     initRmolService();
00121   }
00122
00123   // ////////////////////////////////////////////////////////////////////
00124   RMOL_Service::~RMOL_Service() {
00125     // Delete/Clean all the objects from memory
00126     finalise();
00127   }
00128
00129   // ////////////////////////////////////////////////////////////////////
00130   void RMOL_Service::finalise() {
00131     assert (_rmolServiceContext != NULL);
00132     // Reset the (Boost.)Smart pointer pointing on the STDAIR_Service object.
00133     _rmolServiceContext->reset();
00134   }
00135
00136   // ////////////////////////////////////////////////////////////////////
00137   void RMOL_Service::initServiceContext() {
00138     // Initialise the service context
00139     RMOL_ServiceContext& lRMOL_ServiceContext =
00140       FacRmolServiceContext::instance().create();
00141     _rmolServiceContext = &lRMOL_ServiceContext;
00142   }
00143
00144   // ////////////////////////////////////////////////////////////////////
00145   void RMOL_Service::
00146   addStdAirService (stdair::STDAIR_ServicePtr_T ioSTDAIR_Service_ptr,
00147                     const bool iOwnStdairService) {
00148
00149     // Retrieve the RMOL service context
00150     assert (_rmolServiceContext != NULL);
00151     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
00152
00153     // Store the STDAIR service object within the (AIRINV) service context
00154     lRMOL_ServiceContext.setSTDAIR_Service (ioSTDAIR_Service_ptr,
00155                                             iOwnStdairService);
00156   }
00157
00158   // ////////////////////////////////////////////////////////////////////
00159   stdair::STDAIR_ServicePtr_T RMOL_Service::
00160   initStdAirService (const stdair::BasLogParams& iLogParams) {
00161
00169     stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
00170       boost::make_shared<stdair::STDAIR_Service> (iLogParams);
00171
00172     return lSTDAIR_Service_ptr;
00173   }
00174
00175   // ////////////////////////////////////////////////////////////////////////
00176   stdair::STDAIR_ServicePtr_T RMOL_Service::
00177   initStdAirService (const stdair::BasLogParams& iLogParams,
00178                      const stdair::BasDBParams& iDBParams) {
00179
00187     stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
00188       boost::make_shared<stdair::STDAIR_Service> (iLogParams, iDBParams);
00189
00190     return lSTDAIR_Service_ptr;
00191   }
00192
00193   // ////////////////////////////////////////////////////////////////////
00194   void RMOL_Service::initRmolService() {
00195     // Do nothing at this stage. A sample BOM tree may be built by
00196     // calling the buildSampleBom() method
00197   }
00198
00199   // ////////////////////////////////////////////////////////////////////
00200   void RMOL_Service::
00201   parseAndLoad (const stdair::CabinCapacity_T& iCabinCapacity,
00202                 const stdair::Filename_T& iInputFileName) {
00203
00204     // Retrieve the RMOL service context
00205     if (_rmolServiceContext == NULL) {
00206       throw stdair::NonInitialisedServiceException ("The RMOL service has not"
00207                                                     " been initialised");
00208     }
00209     assert (_rmolServiceContext != NULL);
```

```
00210     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
00211     const bool doesOwnStdairService =
00212       lRMOL_ServiceContext.getOwnStdairServiceFlag();
00213
00214     // Retrieve the StdAir service object from the (RMOL) service context
00215     stdair::STDAIR_Service& lSTDAIR_Service =
00216       lRMOL_ServiceContext.getSTDAIR_Service();
00217     stdair::BomRoot& lPersistentBomRoot =
00218       lSTDAIR_Service.getPersistentBomRoot();
00219
00223     lSTDAIR_Service.buildDummyInventory (iCabinCapacity);
00224
00228     InventoryParser::parseInputFileAndBuildBom (iInputFileName,
00229                                                 lPersistentBomRoot);
00230
00246     buildComplementaryLinks (lPersistentBomRoot);
00247
00252     if (doesOwnStdairService == true) {
00253
00254       //
00255       clonePersistentBom ();
00256     }
00257   }
00258
00259   // ////////////////////////////////////////////////////////////////////
00260   void RMOL_Service::buildSampleBom() {
00261
00262     // Retrieve the RMOL service context
00263     if (_rmolServiceContext == NULL) {
00264       throw stdair::NonInitialisedServiceException ("The RMOL service has not"
00265                                                     " been initialised");
00266     }
00267     assert (_rmolServiceContext != NULL);
00268
00269     // Retrieve the RMOL service context and whether it owns the Stdair
00270     // service
00271     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
00272     const bool doesOwnStdairService =
00273       lRMOL_ServiceContext.getOwnStdairServiceFlag();
00274
00275     // Retrieve the StdAir service object from the (RMOL) service context
00276     stdair::STDAIR_Service& lSTDAIR_Service =
00277       lRMOL_ServiceContext.getSTDAIR_Service();
00278     stdair::BomRoot& lPersistentBomRoot =
00279       lSTDAIR_Service.getPersistentBomRoot();
00280
00285     if (doesOwnStdairService == true) {
00286       //
00287       lSTDAIR_Service.buildSampleBom();
00288     }
00289
00305     buildComplementaryLinks (lPersistentBomRoot);
00306
00311     if (doesOwnStdairService == true) {
00312
00313       //
00314       clonePersistentBom ();
00315     }
00316   }
00317
00318   // ////////////////////////////////////////////////////////////////////
00319   void RMOL_Service::clonePersistentBom () {
00320
00321     // Retrieve the RMOL service context
00322     if (_rmolServiceContext == NULL) {
00323       throw stdair::NonInitialisedServiceException("The RMOL service has not "
00324                                                    "been initialised");
00325     }
00326     assert (_rmolServiceContext != NULL);
00327
00328     // Retrieve the RMOL service context and whether it owns the Stdair
00329     // service
00330     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
00331     const bool doesOwnStdairService =
00332       lRMOL_ServiceContext.getOwnStdairServiceFlag();
00333
00334     // Retrieve the StdAir service object from the (RMOL) service context
00335     stdair::STDAIR_Service& lSTDAIR_Service =
00336       lRMOL_ServiceContext.getSTDAIR_Service();
00337
00342     if (doesOwnStdairService == true) {
00343
00344       //
00345       lSTDAIR_Service.clonePersistentBom ();
00346     }
00347
00351     stdair::BomRoot& lBomRoot =
```

```
00352       lSTDAIR_Service.getBomRoot();
00353     buildComplementaryLinks (lBomRoot);
00354   }
00355
00356   // ////////////////////////////////////////////////////////////////////
00357   void RMOL_Service::buildComplementaryLinks (stdair::BomRoot&
      ioBomRoot) {
00358
00359     // Retrieve the RMOL service context
00360     if (_rmolServiceContext == NULL) {
00361       throw stdair::NonInitialisedServiceException("The RMOL service has not "
00362                                                    "been initialised");
00363     }
00364     assert (_rmolServiceContext != NULL);
00365
00366     // Retrieve the RMOL service context and whether it owns the Stdair
00367     // service
00368     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
00369
00370     // Retrieve the StdAir service object from the (RMOL) service context
00371     stdair::STDAIR_Service& lSTDAIR_Service =
00372       lRMOL_ServiceContext.getSTDAIR_Service();
00373
00378     lSTDAIR_Service.buildDummyLegSegmentAccesses (ioBomRoot);
00379   }
00380
00381   // ////////////////////////////////////////////////////////////////////
00382   void RMOL_Service::optimalOptimisationByMCIntegration (
      const int K) {
00383     assert (_rmolServiceContext != NULL);
00384     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
00385
00386     // Retrieve the StdAir service
00387     stdair::STDAIR_Service& lSTDAIR_Service =
00388       lRMOL_ServiceContext.getSTDAIR_Service();
00389     // TODO: gsabatier
00390     // Replace the getPersistentBomRoot method by the getBomRoot method,
00391     // in order to work on the clone Bom root instead of the persistent one.
00392     // Does not work for now because virtual classes are not cloned.
00393     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getPersistentBomRoot();
00394
00395     //
00396     stdair::LegCabin& lLegCabin =
00397       stdair::BomRetriever::retrieveDummyLegCabin (lBomRoot);
00398
00399     stdair::BasChronometer lOptimisationChronometer;
00400     lOptimisationChronometer.start();
00401
00402     Optimiser::optimalOptimisationByMCIntegration (K,
      lLegCabin);
00403
00404     const double lOptimisationMeasure = lOptimisationChronometer.elapsed();
00405
00406     // DEBUG
00407     STDAIR_LOG_DEBUG ("Optimisation by Monte-Carlo performed in "
00408                       << lOptimisationMeasure);
00409     STDAIR_LOG_DEBUG ("Result: " << lLegCabin.displayVirtualClassList());
00410
00411     std::ostringstream logStream;
00412     stdair::BidPriceVector_T lBidPriceVector = lLegCabin.getBidPriceVector();
00413     logStream << "Bid-Price Vector (BPV): ";
00414     const unsigned int size = lBidPriceVector.size();
00415
00416     for (unsigned int i = 0; i < size - 1; ++i) {
00417       const double bidPrice = lBidPriceVector.at(i);
00418       logStream << std::fixed << std::setprecision (2) << bidPrice << ", ";
00419     }
00420     const double bidPrice = lBidPriceVector.at(size -1);
00421     logStream << std::fixed << std::setprecision (2) << bidPrice;
00422     STDAIR_LOG_DEBUG (logStream.str());
00423   }
00424
00425   // ////////////////////////////////////////////////////////////////////
00426   void RMOL_Service::optimalOptimisationByDP() {
00427   }
00428
00429   // ////////////////////////////////////////////////////////////////////
00430   void RMOL_Service::heuristicOptimisationByEmsr() {
00431     assert (_rmolServiceContext != NULL);
00432     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
00433
00434     // Retrieve the StdAir service
00435     stdair::STDAIR_Service& lSTDAIR_Service =
00436       lRMOL_ServiceContext.getSTDAIR_Service();
00437     // TODO: gsabatier
00438     // Replace the getPersistentBomRoot method by the getBomRoot method,
00439     // in order to work on the clone Bom root instead of the persistent one.
```

```
00440     // Does not work for now because virtual classes are not cloned.
00441     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getPersistentBomRoot();
00442
00443     //
00444     stdair::LegCabin& lLegCabin =
00445       stdair::BomRetriever::retrieveDummyLegCabin (lBomRoot);
00446
00447     stdair::BasChronometer lOptimisationChronometer;
00448     lOptimisationChronometer.start();
00449
00450     Optimiser::heuristicOptimisationByEmsr (lLegCabin);
00451
00452     const double lOptimisationMeasure = lOptimisationChronometer.elapsed();
00453     // DEBUG
00454     STDAIR_LOG_DEBUG ("Optimisation EMSR performed in "
00455                       << lOptimisationMeasure);
00456     STDAIR_LOG_DEBUG ("Result: " << lLegCabin.displayVirtualClassList());
00457
00458     stdair::BidPriceVector_T lBidPriceVector = lLegCabin.getBidPriceVector();
00459     std::ostringstream logStream;
00460     logStream << "Bid-Price Vector (BPV): ";
00461     stdair::UnsignedIndex_T idx = 0;
00462     for (stdair::BidPriceVector_T::const_iterator itBP = lBidPriceVector.begin();
00463          itBP != lBidPriceVector.end(); ++itBP) {
00464       if (idx != 0) {
00465         logStream << ", ";
00466       }
00467       const stdair::BidPrice_T& lBidPrice = *itBP;
00468       logStream << std::fixed << std::setprecision (2) << lBidPrice;
00469     }
00470     // DEBUG
00471     STDAIR_LOG_DEBUG (logStream.str());
00472   }
00473
00474   // ////////////////////////////////////////////////////////////////////
00475   void RMOL_Service::heuristicOptimisationByEmsrA() {
00476     assert (_rmolServiceContext != NULL);
00477     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
00478
00479     // Retrieve the StdAir service
00480     stdair::STDAIR_Service& lSTDAIR_Service =
00481       lRMOL_ServiceContext.getSTDAIR_Service();
00482     // TODO: gsabatier
00483     // Replace the getPersistentBomRoot method by the getBomRoot method,
00484     // in order to work on the clone Bom root instead of the persistent one.
00485     // Does not work for now because virtual classes are not cloned.
00486     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getPersistentBomRoot();
00487
00488     //
00489     stdair::LegCabin& lLegCabin =
00490       stdair::BomRetriever::retrieveDummyLegCabin (lBomRoot);
00491
00492     Optimiser::heuristicOptimisationByEmsrA (lLegCabin);
00493
00494     // DEBUG
00495     STDAIR_LOG_DEBUG ("Result: " << lLegCabin.displayVirtualClassList());
00496
00497   }
00498
00499   // ////////////////////////////////////////////////////////////////////
00500   void RMOL_Service::heuristicOptimisationByEmsrB() {
00501     assert (_rmolServiceContext != NULL);
00502     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
00503
00504     // Retrieve the StdAir service
00505     stdair::STDAIR_Service& lSTDAIR_Service =
00506       lRMOL_ServiceContext.getSTDAIR_Service();
00507     // TODO: gsabatier
00508     // Replace the getPersistentBomRoot method by the getBomRoot method,
00509     // in order to work on the clone Bom root instead of the persistent one.
00510     // Does not work for now because virtual classes are not cloned.
00511     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getPersistentBomRoot();
00512
00513     //
00514     stdair::LegCabin& lLegCabin =
00515       stdair::BomRetriever::retrieveDummyLegCabin (lBomRoot);
00516
00517     Optimiser::heuristicOptimisationByEmsrB (lLegCabin);
00518
00519     // DEBUG
00520     STDAIR_LOG_DEBUG ("Result: " << lLegCabin.displayVirtualClassList());
00521   }
00522
00523   // ////////////////////////////////////////////////////////////////////
00524   const stdair::SegmentCabin& RMOL_Service::
00525   retrieveDummySegmentCabin(const bool isForFareFamilies) {
00526     assert (_rmolServiceContext != NULL);
```

```
00527     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
00528
00529     // Retrieve the StdAir service
00530     stdair::STDAIR_Service& lSTDAIR_Service =
00531       lRMOL_ServiceContext.getSTDAIR_Service();
00532     // TODO: gsabatier
00533     // Replace the getPersistentBomRoot method by the getBomRoot method,
00534     // in order to work on the clone Bom root instead of the persistent one.
00535     // Does not work for now because virtual classes are not cloned.
00536     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getPersistentBomRoot();
00537
00538     const stdair::SegmentCabin& lSegmentCabin =
00539       stdair::BomRetriever::retrieveDummySegmentCabin(lBomRoot,
00540                                                       isForFareFamilies);
00541     return lSegmentCabin;
00542
00543   }
00544
00545   // ////////////////////////////////////////////////////////////////////
00546   bool RMOL_Service::
00547   optimise (stdair::FlightDate& ioFlightDate,
00548             const stdair::DateTime_T& iRMEventTime,
00549             const stdair::UnconstrainingMethod& iUnconstrainingMethod,
00550             const stdair::ForecastingMethod& iForecastingMethod,
00551             const stdair::PreOptimisationMethod& iPreOptimisationMethod,
00552             const stdair::OptimisationMethod& iOptimisationMethod,
00553             const stdair::PartnershipTechnique& iPartnershipTechnique) {
00554
00555
00556     STDAIR_LOG_DEBUG ("Forecast & Optimisation");
00557
00558     const stdair::PartnershipTechnique::EN_PartnershipTechnique& lPartnershipTechnique =
00559       iPartnershipTechnique.getTechnique();
00560
00561     switch (lPartnershipTechnique) {
00562     case stdair::PartnershipTechnique::NONE:{
00563       // DEBUG
00564       STDAIR_LOG_DEBUG ("Forecast");
00565
00566       // 1. Forecasting
00567       const bool isForecasted = Forecaster::forecast (ioFlightDate,
00568                                                       iRMEventTime,
00569                                                       iUnconstrainingMethod,
00570                                                       iForecastingMethod);
00571       // DEBUG
00572       STDAIR_LOG_DEBUG ("Forecast successful: " << isForecasted);
00573
00574       if (isForecasted == true) {
00575         // 2a. MRT or FA
00576         // DEBUG
00577         STDAIR_LOG_DEBUG ("Pre-optimise");
00578
00579         const bool isPreOptimised =
00580           PreOptimiser::preOptimise (ioFlightDate, iPreOptimisationMethod);
00581
00582         // DEBUG
00583         STDAIR_LOG_DEBUG ("Pre-Optimise successful: " << isPreOptimised);
00584
00585         if (isPreOptimised == true) {
00586           // 2b. Optimisation
00587           // DEBUG
00588           STDAIR_LOG_DEBUG ("Optimise");
00589           const bool optimiseSucceeded =
00590             Optimiser::optimise (ioFlightDate, iOptimisationMethod);
00591           // DEBUG
00592           STDAIR_LOG_DEBUG ("Optimise successful: " << optimiseSucceeded);
00593           return optimiseSucceeded ;
00594         }
00595       }
00596       break;
00597     }
00598     case stdair::PartnershipTechnique::RAE_DA:
00599     case stdair::PartnershipTechnique::IBP_DA:{
00600       if (_previousForecastDate < iRMEventTime.date()) {
00601         forecastOnD (iRMEventTime);
00602         resetDemandInformation (iRMEventTime);
00603         projectAggregatedDemandOnLegCabins (iRMEventTime);
00604         optimiseOnD (iRMEventTime);
00605       }
00606       break;
00607     }
00608     case stdair::PartnershipTechnique::RAE_YP:
00609     case stdair::PartnershipTechnique::IBP_YP:
00610     case stdair::PartnershipTechnique::IBP_YP_U:{
00611       if (_previousForecastDate < iRMEventTime.date()) {
00612         forecastOnD (iRMEventTime);
00613         resetDemandInformation (iRMEventTime);
```

```
00614         projectOnDDemandOnLegCabinsUsingYP (iRMEventTime);
00615         optimiseOnD (iRMEventTime);
00616       }
00617       break;
00618     }
00619     case stdair::PartnershipTechnique::RMC:{
00620       if (_previousForecastDate < iRMEventTime.date()) {
00621         forecastOnD (iRMEventTime);
00622         resetDemandInformation (iRMEventTime);
00623         updateBidPrice (iRMEventTime);
00624         projectOnDDemandOnLegCabinsUsingDYP (iRMEventTime);
00625         optimiseOnDUsingRMCooperation (iRMEventTime);
00626       }
00627       break;
00628     }
00629     case stdair::PartnershipTechnique::A_RMC:{
00630       if (_previousForecastDate < iRMEventTime.date()) {
00631         forecastOnD (iRMEventTime);
00632         resetDemandInformation (iRMEventTime);
00633         updateBidPrice (iRMEventTime);
00634         projectOnDDemandOnLegCabinsUsingDYP (iRMEventTime);
00635         optimiseOnDUsingAdvancedRMCooperation (iRMEventTime);
00636       }
00637       break;
00638     }
00639     default:{
00640       assert (false);
00641       break;
00642       }
00643     }
00644     return false;
00645   }
00646
00647   // ////////////////////////////////////////////////////////////////////
00648   void RMOL_Service::forecastOnD (const stdair::DateTime_T& iRMEventTime) {
00649
00650     if (_rmolServiceContext == NULL) {
00651       throw stdair::NonInitialisedServiceException ("The Rmol service "
00652                                                     "has not been initialised");
00653     }
00654     assert (_rmolServiceContext != NULL);
00655     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
00656
00657     // Retrieve the bom root
00658     stdair::STDAIR_Service& lSTDAIR_Service =
00659       lRMOL_ServiceContext.getSTDAIR_Service();
00660     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
00661
00662     // Retrieve the date from the RM event
00663     const stdair::Date_T lDate = iRMEventTime.date();
00664
00665     _previousForecastDate = lDate;
00666
00667     const stdair::InventoryList_T& lInventoryList =
00668       stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
00669     assert (!lInventoryList.empty());
00670     for (stdair::InventoryList_T::const_iterator itInv = lInventoryList.begin();
00671          itInv != lInventoryList.end(); ++itInv) {
00672       const stdair::Inventory* lInventory_ptr = *itInv;
00673       assert (lInventory_ptr != NULL);
00674       const bool hasOnDDateList =
00675         stdair::BomManager::hasList<stdair::OnDDate> (*lInventory_ptr);
00676       if (hasOnDDateList == true) {
00677         const stdair::OnDDateList_T lOnDDateList =
00678           stdair::BomManager::getList<stdair::OnDDate> (*lInventory_ptr);
00679
00680         for (stdair::OnDDateList_T::const_iterator itOD = lOnDDateList.begin();
00681              itOD != lOnDDateList.end(); ++itOD) {
00682           stdair::OnDDate* lOnDDate_ptr = *itOD;
00683           assert (lOnDDate_ptr != NULL);
00684
00685           const stdair::Date_T& lDepartureDate = lOnDDate_ptr->getDate();
00686           stdair::DateOffset_T lDateOffset = lDepartureDate - lDate;
00687           stdair::DTD_T lDTD = short (lDateOffset.days());
00688
00689           stdair::DCPList_T::const_iterator itDCP =
00690             std::find (stdair::DEFAULT_DCP_LIST.begin(),
00691                        stdair::DEFAULT_DCP_LIST.end(), lDTD);
00692           // Check if the forecast for this O&D date needs to be forecasted.
00693           if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
00694             // Retrieve the total forecast map.
00695             const stdair::CabinForecastMap_T& lTotalForecastMap =
00696               lOnDDate_ptr->getTotalForecastMap();
00697
00698             // Browse the map and make a forecast for every cabin.
00699             for (stdair::CabinForecastMap_T::const_iterator itCF =
00700                    lTotalForecastMap.begin();
```

```
00701                 itCF != lTotalForecastMap.end(); ++itCF) {
00702               const stdair::CabinCode_T lCabinCode = itCF->first;
00703               stdair::YieldFeatures* lYieldFeatures_ptr =
00704                 getYieldFeatures(*lOnDDate_ptr, lCabinCode, lBomRoot);
00705               if (lYieldFeatures_ptr == NULL) {
00706                 STDAIR_LOG_ERROR ("Cannot find yield corresponding to "
00707                                   << "the O&D date"
00708                                   << lOnDDate_ptr->toString()
00709                                   << " Cabin " << lCabinCode);
00710                 assert (false);
00711               }
00712               forecastOnD (*lYieldFeatures_ptr, *lOnDDate_ptr, lCabinCode, lDTD,
00713                            lBomRoot);
00714             }
00715           }
00716         }
00717       }
00718     }
00719   }
00720
00721   // ////////////////////////////////////////////////////////////////////
00722   stdair::YieldFeatures* RMOL_Service::
00723   getYieldFeatures(const stdair::OnDDate& iOnDDate,
00724                    const stdair::CabinCode_T& iCabinCode,
00725                    stdair::BomRoot& iBomRoot) {
00726
00727     const stdair::AirportCode_T& lOrigin = iOnDDate.getOrigin();
00728     const stdair::AirportCode_T& lDestination = iOnDDate.getDestination();
00729
00730     const stdair::Date_T& lDepartureDate = iOnDDate.getDate();
00731
00732     // Build the airport pair key out of O&D and get the airport pair object
00733     const stdair::AirportPairKey lAirportPairKey(lOrigin, lDestination);
00734     stdair::AirportPair* lAirportPair_ptr = stdair::BomManager::
00735       getObjectPtr<stdair::AirportPair> (iBomRoot,
00736                                          lAirportPairKey.toString());
00737     if (lAirportPair_ptr == NULL) {
00738       STDAIR_LOG_ERROR ("Cannot find yield corresponding to the airport "
00739                         << "pair: " << lAirportPairKey.toString());
00740       assert (false);
00741     }
00742
00743     // Retrieve the corresponding date period to lDepartureDate.
00744     const stdair::DatePeriodList_T lDatePeriodList =
00745       stdair::BomManager::getList<stdair::DatePeriod> (*lAirportPair_ptr);
00746     for (stdair::DatePeriodList_T::const_iterator itDatePeriod =
00747            lDatePeriodList.begin();
00748          itDatePeriod != lDatePeriodList.end(); ++itDatePeriod) {
00749       const stdair::DatePeriod* lDatePeriod_ptr = *itDatePeriod;
00750       assert (lDatePeriod_ptr != NULL);
00751
00752       const bool isDepartureDateValid =
00753         lDatePeriod_ptr->isDepartureDateValid (lDepartureDate);
00754
00755       if (isDepartureDateValid == true) {
00756         // Retrieve the PoS-Channel.
00757         // TODO: Use POS and Channel from demand instead of default
00758         const stdair::PosChannelKey lPosChannelKey (stdair::DEFAULT_POS,
00759                                                     stdair::DEFAULT_CHANNEL);
00760         stdair::PosChannel* lPosChannel_ptr = stdair::BomManager::
00761           getObjectPtr<stdair::PosChannel> (*lDatePeriod_ptr,
00762                                             lPosChannelKey.toString());
00763         if (lPosChannel_ptr == NULL) {
00764           STDAIR_LOG_ERROR ("Cannot find yield corresponding to the PoS-"
00765                             << "Channel: " << lPosChannelKey.toString());
00766           assert (false);
00767         }
00768         // Retrieve the yield features.
00769         const stdair::TimePeriodList_T lTimePeriodList = stdair::
00770           BomManager::getList<stdair::TimePeriod> (*lPosChannel_ptr);
00771         for (stdair::TimePeriodList_T::const_iterator itTimePeriod =
00772                lTimePeriodList.begin();
00773              itTimePeriod != lTimePeriodList.end(); ++itTimePeriod) {
00774           const stdair::TimePeriod* lTimePeriod_ptr = *itTimePeriod;
00775           assert (lTimePeriod_ptr != NULL);
00776
00777           // TODO: Use trip type from demand instead of default value.
00778           const stdair::YieldFeaturesKey lYieldFeaturesKey (stdair::TRIP_TYPE_ONE_WAY,
00779                                                             iCabinCode);
00780           stdair::YieldFeatures* oYieldFeatures_ptr = stdair::BomManager::
00781             getObjectPtr<stdair::YieldFeatures>(*lTimePeriod_ptr,
00782                                                 lYieldFeaturesKey.toString());
00783           if (oYieldFeatures_ptr != NULL) {
00784             return oYieldFeatures_ptr;
00785           }
00786         }
00787       }
```

```
00788     }
00789     return NULL;
00790
00791   }
00792
00793
00794   // ////////////////////////////////////////////////////////////////////
00795   void RMOL_Service::
00796   forecastOnD (const stdair::YieldFeatures& iYieldFeatures,
00797                stdair::OnDDate& iOnDDate,
00798                const stdair::CabinCode_T& iCabinCode,
00799                const stdair::DTD_T& iDTD,
00800                stdair::BomRoot& iBomRoot) {
00801
00802     const stdair::AirlineClassListList_T lAirlineClassListList =
00803       stdair::BomManager::getList<stdair::AirlineClassList> (iYieldFeatures);
00804     assert (lAirlineClassListList.begin() != lAirlineClassListList.end());
00805
00806     // Yield order check
00807     stdair::AirlineClassListList_T::const_iterator  itACL =
00808       lAirlineClassListList.begin();
00809     stdair::Yield_T lPreviousYield((*itACL)->getYield());
00810     ++itACL;
00811     for (; itACL != lAirlineClassListList.end(); ++itACL) {
00812       const stdair::AirlineClassList* lAirlineClassList = *itACL;
00813       const stdair::Yield_T& lYield = lAirlineClassList->getYield();
00814       if (lYield <= lPreviousYield) {
00815         lPreviousYield = lYield;
00816       }
00817       else{
00818         STDAIR_LOG_ERROR ("Yields should be given in a descendant order"
00819                           << " in the yield input file") ;
00820         assert (false);
00821       }
00822     }
00823     // Proportion factor list initialisation
00824     // Each element corresponds to a yield rule
00825     stdair::ProportionFactorList_T lProportionFactorList;
00826     stdair::ProportionFactor_T lPreviousProportionFactor = 0;
00827
00828     // Retrieve the minimal willingness to pay associated to the demand
00829     const stdair::WTPDemandPair_T& lTotalForecast =
00830       iOnDDate.getTotalForecast (iCabinCode);
00831     const stdair::WTP_T& lMinWTP = lTotalForecast.first;
00832
00833     // Retrieve the remaining percentage of booking requests
00834     const stdair::ContinuousAttributeLite<stdair::FloatDuration_T>
00835       lArrivalPattern (stdair::DEFAULT_DTD_PROB_MAP);
00836
00837     STDAIR_LOG_DEBUG (lArrivalPattern.displayCumulativeDistribution());
00838     const stdair::Probability_T lRemainingProportion =
00839       lArrivalPattern.getRemainingProportion(-float(iDTD));
00840
00841     // Compute the characteristics (mean and std dev) of the total
00842     // forecast demand to come
00843     const stdair::MeanStdDevPair_T lForecatsMeanStdDevPair =
00844       lTotalForecast.second;
00845     const stdair::MeanValue_T& lMeanValue =
00846       lForecatsMeanStdDevPair.first;
00847     const stdair::MeanValue_T& lRemainingMeanValue =
00848       lRemainingProportion*lMeanValue;
00849     const stdair::StdDevValue_T& lStdDevValue =
00850       lForecatsMeanStdDevPair.second;
00851     const stdair::StdDevValue_T& lRemainingStdDevValue =
00852       lRemainingProportion*lStdDevValue;
00853
00854     // Retrieve the frat5 coef corresponding to the input dtd
00855     stdair::DTDFratMap_T::const_iterator itDFC =
00856       stdair::DEFAULT_DTD_FRAT5COEF_MAP.find(iDTD);
00857     if (itDFC == stdair::DEFAULT_DTD_FRAT5COEF_MAP.end()) {
00858       STDAIR_LOG_ERROR ("Cannot find frat5 coef for DTD = " << iDTD );
00859       assert (false);
00860     }
00861     stdair::RealNumber_T lFrat5Coef =
00862       stdair::DEFAULT_DTD_FRAT5COEF_MAP.at(iDTD);
00863
00864     STDAIR_LOG_DEBUG ("Remaining proportion " << lRemainingProportion
00865                       << " Total " << lMeanValue
00866                       << " StdDev " << lStdDevValue
00867                       << "Frat5 Coef " << lFrat5Coef);
00868
00869     std::ostringstream oStr;
00870     // Compute the "forecast demand to come" proportion by class
00871     itACL = lAirlineClassListList.begin();
00872     for (; itACL != lAirlineClassListList.end(); ++itACL) {
00873       const stdair::AirlineClassList* lAirlineClassList_ptr = *itACL;
00874       const stdair::Yield_T& lYield = lAirlineClassList_ptr->getYield();
```

```
00875       stdair::ProportionFactor_T lProportionFactor =
00876         exp ((lYield - lMinWTP)*log(0.5)/(lMinWTP*(lFrat5Coef-1.0)));
00877       // If the yield is smaller than minimal WTP, the factor is greater than 1.
00878       // In that case it should be modified and put to 1.
00879       lProportionFactor = std::min (lProportionFactor, 1.0);
00880       lProportionFactorList.push_back(lProportionFactor - lPreviousProportionFactor);
00881       lPreviousProportionFactor = lProportionFactor;
00882       oStr << lAirlineClassList_ptr->toString() << lProportionFactor << " ";
00883     }
00884
00885     STDAIR_LOG_DEBUG (oStr.str());
00886
00887     // Sanity check
00888     assert (lAirlineClassListList.size() == lProportionFactorList.size());
00889
00890     STDAIR_LOG_DEBUG ("Forecast for " << iOnDDate.describeKey()
00891                       << " " << iDTD << " days to departure");
00892
00893     // store the forecast demand to come characteristics in the booking classes
00894     stdair::ProportionFactorList_T::const_iterator itPF =
00895       lProportionFactorList.begin();
00896     itACL = lAirlineClassListList.begin();
00897     for (; itACL != lAirlineClassListList.end(); ++itACL, ++itPF) {
00898       const stdair::AirlineClassList* lAirlineClassList_ptr = *itACL;
00899       const stdair::ProportionFactor_T& lProportionFactor = *itPF;
00900       stdair::MeanValue_T lMeanValue = lProportionFactor*lRemainingMeanValue;
00901       stdair::StdDevValue_T lStdDevValue =
00902         lProportionFactor*lRemainingStdDevValue;
00903       setOnDForecast(*lAirlineClassList_ptr, lMeanValue, lStdDevValue,
00904                      iOnDDate, iCabinCode, iBomRoot);
00905     }
00906
00907   }
00908
00909   // ////////////////////////////////////////////////////////////////////
00910   void RMOL_Service::
00911   setOnDForecast (const stdair::AirlineClassList& iAirlineClassList,
00912                   const stdair::MeanValue_T& iMeanValue,
00913                   const stdair::StdDevValue_T& iStdDevValue,
00914                   stdair::OnDDate& iOnDDate,
00915                   const stdair::CabinCode_T& iCabinCode,
00916                   stdair::BomRoot& iBomRoot) {
00917
00918     const stdair::AirportCode_T& lOrigin = iOnDDate.getOrigin();
00919     const stdair::AirportCode_T& lDestination = iOnDDate.getDestination();
00920
00921     const stdair::Date_T& lDepartureDate = iOnDDate.getDate();
00922
00923     const stdair::AirlineCodeList_T& lAirlineCodeList =
00924       iAirlineClassList.getAirlineCodeList();
00925
00926     // Retrieve the class list (one class per airline)
00927     const stdair::ClassList_StringList_T& lClassList_StringList =
00928       iAirlineClassList.getClassCodeList();
00929     assert (!lClassList_StringList.empty());
00930     stdair::ClassCodeList_T lClassCodeList;
00931     for (stdair::ClassList_StringList_T::const_iterator itCL =
00932            lClassList_StringList.begin();
00933          itCL != lClassList_StringList.end(); ++itCL){
00934       const stdair::ClassList_String_T& lClassList_String = *itCL;
00935       assert (lClassList_String.size() > 0);
00936       stdair::ClassCode_T lFirstClass;
00937       lFirstClass.append (lClassList_String, 0, 1);
00938       lClassCodeList.push_back(lFirstClass);
00939     }
00940
00941     // Sanity check
00942     assert (lAirlineCodeList.size() == lClassCodeList.size());
00943     assert (!lAirlineCodeList.empty());
00944
00945     if (lAirlineCodeList.size() == 1) {
00946       // Store the forecast information in the case of a single segment
00947       stdair::AirlineCode_T lAirlineCode = lAirlineCodeList.front();
00948       stdair::ClassCode_T lClassCode = lClassCodeList.front();
00949       stdair::Yield_T lYield = iAirlineClassList.getYield();
00950       setOnDForecast(lAirlineCode, lDepartureDate, lOrigin,
00951                      lDestination, iCabinCode, lClassCode,
00952                      iMeanValue, iStdDevValue, lYield, iBomRoot);
00953     } else {
00954       // Store the forecast information in the case of a multiple segment
00955
00956       stdair::Yield_T lYield = iAirlineClassList.getYield();
00957       for (stdair::AirlineCodeList_T::const_iterator itAC =
00958              lAirlineCodeList.begin();
00959            itAC != lAirlineCodeList.end(); ++itAC) {
00960         const stdair::AirlineCode_T& lAirlineCode = *itAC;
00961         setOnDForecast(lAirlineCodeList, lAirlineCode, lDepartureDate, lOrigin,
```

```
00962                       lDestination, iCabinCode, lClassCodeList,
00963                       iMeanValue, iStdDevValue, lYield, iBomRoot);
00964       }
00965     }
00966   }
00967
00968   // ////////////////////////////////////////////////////////////////////
00969   void RMOL_Service::
00970   setOnDForecast (const stdair::AirlineCode_T& iAirlineCode,
00971                   const stdair::Date_T& iDepartureDate,
00972                   const stdair::AirportCode_T& iOrigin,
00973                   const stdair::AirportCode_T& iDestination,
00974                   const stdair::CabinCode_T& iCabinCode,
00975                   const stdair::ClassCode_T& iClassCode,
00976                   const stdair::MeanValue_T& iMeanValue,
00977                   const stdair::StdDevValue_T& iStdDevValue,
00978                   const stdair::Yield_T& iYield,
00979                   stdair::BomRoot& iBomRoot) {
00980     stdair::Inventory* lInventory_ptr = iBomRoot.getInventory(iAirlineCode);
00981     if (lInventory_ptr == NULL) {
00982       STDAIR_LOG_ERROR ("Cannot find the inventory corresponding"
00983                         << " to the airline" << iAirlineCode) ;
00984       assert(false);
00985     }
00986     const stdair::OnDDateList_T lOnDDateList =
00987       stdair::BomManager::getList<stdair::OnDDate> (*lInventory_ptr);
00988     assert (!lOnDDateList.empty());
00989     bool lFoundOnDDate = false;
00990     for (stdair::OnDDateList_T::const_iterator itOD = lOnDDateList.begin();
00991          itOD != lOnDDateList.end(); ++itOD) {
00992       stdair::OnDDate* lOnDDate_ptr = *itOD;
00993       assert (lOnDDate_ptr != NULL);
00994       const stdair::Date_T& lDepartureDate = lOnDDate_ptr->getDate();
00995       const stdair::AirportCode_T& lOrigin = lOnDDate_ptr->getOrigin();
00996       const stdair::AirportCode_T& lDestination = lOnDDate_ptr->getDestination();
00997       const bool hasSegmentDateList =
00998         stdair::BomManager::hasList<stdair::SegmentDate> (*lOnDDate_ptr);
00999       if (hasSegmentDateList == false) {
01000         STDAIR_LOG_ERROR ("The O&D date " << lOnDDate_ptr->describeKey()
01001                           << "has not been correctly initialized : SegmentDate list is missing");
01002         assert (false);
01003       }
01004       const stdair::SegmentDateList_T& lSegmentDateList =
01005         stdair::BomManager::getList<stdair::SegmentDate> (*lOnDDate_ptr);
01006       // Check if the the O&D date is the one we are looking for
01007       if (lDepartureDate == iDepartureDate && lOrigin == iOrigin &&
01008           lDestination == iDestination && lSegmentDateList.size() == 1) {
01009         stdair::CabinClassPair_T lCabinClassPair (iCabinCode, iClassCode);
01010         stdair::CabinClassPairList_T lCabinClassPairList;
01011         lCabinClassPairList.push_back(lCabinClassPair);
01012         const stdair::MeanStdDevPair_T lMeanStdDevPair (iMeanValue, iStdDevValue);
01013         const stdair::WTPDemandPair_T lWTPDemandPair (iYield, lMeanStdDevPair);
01014         lOnDDate_ptr->setDemandInformation(lCabinClassPairList, lWTPDemandPair);
01015         lFoundOnDDate = true;
01016         STDAIR_LOG_DEBUG (iAirlineCode << " Class " << iClassCode
01017                           << " Mean " << iMeanValue
01018                           << " Std Dev " << iStdDevValue);
01019         break;
01020       }
01021     }
01022
01023     if (!lFoundOnDDate) {
01024       STDAIR_LOG_ERROR ("Cannot find class " << iClassCode << " in cabin "
01025                         << iCabinCode << " for the segment "
01026                         << iOrigin << "-" << iDestination << " with"
01027                         << " the airline " << iAirlineCode);
01028       assert(false);
01029     }
01030   }
01031
01032   // ////////////////////////////////////////////////////////////////////
01033   void RMOL_Service::
01034   setOnDForecast (const stdair::AirlineCodeList_T& iAirlineCodeList,
01035                   const stdair::AirlineCode_T& iAirlineCode,
01036                   const stdair::Date_T& iDepartureDate,
01037                   const stdair::AirportCode_T& iOrigin,
01038                   const stdair::AirportCode_T& iDestination,
01039                   const stdair::CabinCode_T& iCabinCode,
01040                   const stdair::ClassCodeList_T& iClassCodeList,
01041                   const stdair::MeanValue_T& iMeanValue,
01042                   const stdair::StdDevValue_T& iStdDevValue,
01043                   const stdair::Yield_T& iYield,
01044                   stdair::BomRoot& iBomRoot) {
01045     stdair::Inventory* lInventory_ptr = iBomRoot.getInventory(iAirlineCode);
01046     if (lInventory_ptr == NULL) {
01047       STDAIR_LOG_ERROR ("Cannot find the inventory corresponding"
01048                         << " to the airline" << iAirlineCode) ;
```

```
01049       assert(false);
01050     }
01051     const stdair::OnDDateList_T lOnDDateList =
01052       stdair::BomManager::getList<stdair::OnDDate> (*lInventory_ptr);
01053     assert (!lOnDDateList.empty());
01054     bool lFoundOnDDate = false;
01055     for (stdair::OnDDateList_T::const_iterator itOD = lOnDDateList.begin();
01056          itOD != lOnDDateList.end(); ++itOD) {
01057       stdair::OnDDate* lOnDDate_ptr = *itOD;
01058       assert (lOnDDate_ptr != NULL);
01059       const stdair::Date_T& lDepartureDate = lOnDDate_ptr->getDate();
01060       const stdair::AirportCode_T& lOrigin = lOnDDate_ptr->getOrigin();
01061       const stdair::AirportCode_T& lDestination = lOnDDate_ptr->getDestination();
01062       const bool hasSegmentDateList =
01063         stdair::BomManager::hasList<stdair::SegmentDate> (*lOnDDate_ptr);
01064       if (hasSegmentDateList == false) {
01065         STDAIR_LOG_ERROR ("The O&D date " << lOnDDate_ptr->describeKey()
01066                           << "has not been correctly initialized : SegmentDate list is missing");
01067         assert (false);
01068       }
01069       const stdair::SegmentDateList_T& lSegmentDateList =
01070         stdair::BomManager::getList<stdair::SegmentDate> (*lOnDDate_ptr);
01071       // Check if the O&D date might be the one we are looking for.
01072       // There still is a test to go through to see if the combination of airlines is right.
01073       if (lDepartureDate == iDepartureDate && lOrigin == iOrigin &&
01074           lDestination == iDestination && lSegmentDateList.size() == iAirlineCodeList.size()) {
01075         const stdair::SegmentDateList_T& lSegmentDateList =
01076           stdair::BomManager::getList<stdair::SegmentDate> (*lOnDDate_ptr);
01077         stdair::AirlineCodeList_T::const_iterator itAC = iAirlineCodeList.begin();
01078         stdair::SegmentDateList_T::const_iterator itSD = lSegmentDateList.begin();
01079         for (;itAC != iAirlineCodeList.end(); ++itAC, ++itSD) {
01080           const stdair::AirlineCode_T lForecastAirlineCode = *itAC;
01081           const stdair::SegmentDate* lSegmentDate_ptr = *itSD;
01082           // Check if the operating airline is a different one and check if it
01083           // is the airline that we are looking for.
01084           const stdair::SegmentDate* lOperatingSegmentDate_ptr =
01085             lSegmentDate_ptr->getOperatingSegmentDate ();
01086           if (lOperatingSegmentDate_ptr != NULL) {
01087             const stdair::FlightDate* lOperatingFD_ptr =
01088               stdair::BomManager::getParentPtr<stdair::FlightDate>(*lOperatingSegmentDate_ptr);
01089             const stdair::AirlineCode_T lOperatingAirlineCode =
01090               lOperatingFD_ptr->getAirlineCode();
01091             if (lOperatingAirlineCode != lForecastAirlineCode) {
01092               break;
01093             }
01094           } else {
01095             const stdair::AirlineCode_T lOperatingAirlineCode =
01096               lOnDDate_ptr->getAirlineCode();
01097             if (lOperatingAirlineCode != lForecastAirlineCode) {
01098               break;
01099             }
01100           }
01101         }
01102         if (itAC == iAirlineCodeList.end()) {lFoundOnDDate = true;}
01103       }
01104       if (lFoundOnDDate) {
01105         stdair::CabinClassPairList_T lCabinClassPairList;
01106         for (stdair::ClassCodeList_T::const_iterator itCC = iClassCodeList.begin();
01107              itCC != iClassCodeList.end(); ++itCC) {
01108           const stdair::ClassCode_T lClassCode = *itCC;
01109           stdair::CabinClassPair_T lCabinClassPair (iCabinCode, lClassCode);
01110           lCabinClassPairList.push_back(lCabinClassPair);
01111         }
01112         const stdair::MeanStdDevPair_T lMeanStdDevPair (iMeanValue, iStdDevValue);
01113         const stdair::YieldDemandPair_T lYieldDemandPair (iYield, lMeanStdDevPair);
01114         lOnDDate_ptr->setDemandInformation(lCabinClassPairList, lYieldDemandPair);
01115         lFoundOnDDate = true;
01116         std::ostringstream oACStr;
01117         for (stdair::AirlineCodeList_T::const_iterator itAC = iAirlineCodeList.begin();
01118              itAC != iAirlineCodeList.end(); ++itAC) {
01119           if (itAC == iAirlineCodeList.begin()) {
01120             oACStr << *itAC;
01121           }
01122           else {
01123             oACStr << "-" << *itAC;
01124           }
01125         }
01126         std::ostringstream oCCStr;
01127         for (stdair::ClassCodeList_T::const_iterator itCC = iClassCodeList.begin();
01128              itCC != iClassCodeList.end(); ++itCC) {
01129           if (itCC == iClassCodeList.begin()) {
01130             oCCStr << *itCC;
01131           }
01132           else {
01133             oCCStr << "-" << *itCC;
01134           }
01135         }
```

```
01136
01137         STDAIR_LOG_DEBUG (oACStr.str() << " Classes " << oCCStr.str()
01138                           << " Mean " << iMeanValue << " Std Dev " << iStdDevValue);
01139         break;
01140       }
01141     }
01142     if (!lFoundOnDDate) {
01143       STDAIR_LOG_ERROR ("Cannot find the required multi-segment O&D date:  "
01144                         << iOrigin << "-" << iDestination << " " << iDepartureDate);
01145       assert(false);
01146     }
01147   }
01148
01149   // ////////////////////////////////////////////////////////////////////
01150   void RMOL_Service::
01151   resetDemandInformation (const stdair::DateTime_T& iRMEventTime) {
01152     if (_rmolServiceContext == NULL) {
01153       throw stdair::NonInitialisedServiceException ("The Rmol service "
01154                                                     "has not been initialised");
01155     }
01156     assert (_rmolServiceContext != NULL);
01157     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
01158
01159     // Retrieve the bom root
01160     stdair::STDAIR_Service& lSTDAIR_Service =
01161       lRMOL_ServiceContext.getSTDAIR_Service();
01162     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01163
01164     const stdair::InventoryList_T lInventoryList =
01165       stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01166     assert (!lInventoryList.empty());
01167     for (stdair::InventoryList_T::const_iterator itInv = lInventoryList.begin();
01168          itInv != lInventoryList.end(); ++itInv) {
01169       const stdair::Inventory* lInventory_ptr = *itInv;
01170       assert (lInventory_ptr != NULL);
01171       resetDemandInformation (iRMEventTime, *lInventory_ptr);
01172     }
01173   }
01174
01175   // ////////////////////////////////////////////////////////////////////
01176   void RMOL_Service::
01177   resetDemandInformation (const stdair::DateTime_T& iRMEventTime,
01178                           const stdair::Inventory& iInventory) {
01179
01180     const stdair::FlightDateList_T lFlightDateList =
01181       stdair::BomManager::getList<stdair::FlightDate> (iInventory);
01182     assert (!lFlightDateList.empty());
01183     for (stdair::FlightDateList_T::const_iterator itFD = lFlightDateList.begin();
01184          itFD != lFlightDateList.end(); ++itFD) {
01185       const stdair::FlightDate* lFlightDate_ptr = *itFD;
01186       assert (lFlightDate_ptr != NULL);
01187
01188       // Retrieve the date from the RM event
01189       const stdair::Date_T lDate = iRMEventTime.date();
01190
01191       const stdair::Date_T& lDepartureDate = lFlightDate_ptr->getDepartureDate();
01192       stdair::DateOffset_T lDateOffset = lDepartureDate - lDate;
01193       stdair::DTD_T lDTD = short (lDateOffset.days());
01194
01195       stdair::DCPList_T::const_iterator itDCP =
01196         std::find (stdair::DEFAULT_DCP_LIST.begin(), stdair::DEFAULT_DCP_LIST.end(), lDTD);
01197       // Check if the demand forecast info corresponding to this flight date needs to be reset.
01198       if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01199         // Check if the flight date holds a list of leg dates.
01200         // If so, find all leg cabin and reset the forecast they are holding.
01201         const bool hasLegDateList =
01202           stdair::BomManager::hasList<stdair::LegDate> (*lFlightDate_ptr);
01203         if (hasLegDateList == true) {
01204           const stdair::LegDateList_T lLegDateList =
01205             stdair::BomManager::getList<stdair::LegDate> (*lFlightDate_ptr);
01206           assert (!lLegDateList.empty());
01207           for (stdair::LegDateList_T::const_iterator itLD = lLegDateList.begin();
01208                itLD != lLegDateList.end(); ++itLD) {
01209             const stdair::LegDate* lLegDate_ptr = *itLD;
01210             assert (lLegDate_ptr != NULL);
01211             const stdair::LegCabinList_T lLegCabinList =
01212               stdair::BomManager::getList<stdair::LegCabin> (*lLegDate_ptr);
01213             assert (!lLegCabinList.empty());
01214             for (stdair::LegCabinList_T::const_iterator itLC = lLegCabinList.begin();
01215                  itLC != lLegCabinList.end(); ++itLC) {
01216               stdair::LegCabin* lLegCabin_ptr = *itLC;
01217               assert (lLegCabin_ptr != NULL);
01218               lLegCabin_ptr->emptyYieldLevelDemandMap();
01219             }
01220           }
01221         }
01222       }
```

```
01223     }
01224   }
01225
01226   // ////////////////////////////////////////////////////////////////////
01227   void RMOL_Service::projectAggregatedDemandOnLegCabins(
      const stdair::DateTime_T& iRMEventTime) {
01228
01229     if (_rmolServiceContext == NULL) {
01230       throw stdair::NonInitialisedServiceException ("The Rmol service "
01231                                                     "has not been initialised");
01232     }
01233     assert (_rmolServiceContext != NULL);
01234     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
01235
01236     // Retrieve the bom root
01237     stdair::STDAIR_Service& lSTDAIR_Service =
01238       lRMOL_ServiceContext.getSTDAIR_Service();
01239     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01240
01241     // Retrieve the date from the RM event
01242     const stdair::Date_T lDate = iRMEventTime.date();
01243
01244     const stdair::InventoryList_T lInventoryList =
01245       stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01246     assert (!lInventoryList.empty());
01247     for (stdair::InventoryList_T::const_iterator itInv = lInventoryList.begin();
01248          itInv != lInventoryList.end(); ++itInv) {
01249       const stdair::Inventory* lInventory_ptr = *itInv;
01250       assert (lInventory_ptr != NULL);
01251       const stdair::OnDDateList_T lOnDDateList =
01252         stdair::BomManager::getList<stdair::OnDDate> (*lInventory_ptr);
01253       assert (!lOnDDateList.empty());
01254       for (stdair::OnDDateList_T::const_iterator itOD = lOnDDateList.begin();
01255            itOD != lOnDDateList.end(); ++itOD) {
01256         stdair::OnDDate* lOnDDate_ptr = *itOD;
01257         assert (lOnDDate_ptr != NULL);
01258
01259         const stdair::Date_T& lDepartureDate = lOnDDate_ptr->getDate();
01260         stdair::DateOffset_T lDateOffset = lDepartureDate - lDate;
01261         stdair::DTD_T lDTD = short (lDateOffset.days());
01262
01263         stdair::DCPList_T::const_iterator itDCP =
01264           std::find (stdair::DEFAULT_DCP_LIST.begin(), stdair::DEFAULT_DCP_LIST.end(), lDTD);
01265         // Check if the forecast for this O&D date needs to be projected.
01266         if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01267
01268           // Browse the demand info map.
01269           const stdair::StringDemandStructMap_T& lStringDemandStructMap =
01270             lOnDDate_ptr->getDemandInfoMap ();
01271           for (stdair::StringDemandStructMap_T::const_iterator itStrDS = lStringDemandStructMap.begin();
01272                itStrDS != lStringDemandStructMap.end(); ++itStrDS) {
01273             std::string lCabinClassPath = itStrDS->first;
01274             const stdair::YieldDemandPair_T& lYieldDemandPair =
01275               itStrDS->second;
01276             const stdair::CabinClassPairList_T& lCabinClassPairList =
01277               lOnDDate_ptr->getCabinClassPairList(lCabinClassPath);
01278             const stdair::NbOfSegments_T& lNbOfSegments = lOnDDate_ptr->getNbOfSegments();
01279             // Sanity check
01280             assert (lCabinClassPairList.size() == lNbOfSegments);
01281
01282             const stdair::SegmentDateList_T lOnDSegmentDateList =
01283               stdair::BomManager::getList<stdair::SegmentDate> (*lOnDDate_ptr);
01284             // Sanity check
01285             assert (lOnDSegmentDateList.size() == lNbOfSegments);
01286             stdair::CabinClassPairList_T::const_iterator itCCP = lCabinClassPairList.begin();
01287             stdair::SegmentDateList_T::const_iterator itSD = lOnDSegmentDateList.begin();
01288             for (; itSD != lOnDSegmentDateList.end(); ++itCCP, ++itSD) {
01289               const stdair::SegmentDate* lSegmentDate_ptr = *itSD;
01290               const stdair::SegmentDate* lOperatingSegmentDate_ptr =
01291                 lSegmentDate_ptr->getOperatingSegmentDate ();
01292               assert (lSegmentDate_ptr != NULL);
01293               // Only operated legs receive the demand information.
01294               if (lOperatingSegmentDate_ptr == NULL) {
01295                 const stdair::CabinCode_T lCabinCode = itCCP->first;
01296                 const stdair::ClassCode_T lClassCode = itCCP->second;
01297                 const stdair::SegmentCabin* lSegmentCabin_ptr =
01298                   stdair::BomManager::getObjectPtr<stdair::SegmentCabin> (*lSegmentDate_ptr,
01299                                                                           lCabinCode);
01300                 assert (lSegmentCabin_ptr != NULL);
01301                 // Retrieve the booking class (level of aggregation of demand).
01302                 // The yield of the class is assigned to all types of demand for it.
01303                 const stdair::BookingClass* lBookingClass_ptr =
01304                   stdair::BomManager::getObjectPtr<stdair::BookingClass> (*lSegmentCabin_ptr,
01305                                                                           lClassCode);
01306                 assert (lBookingClass_ptr != NULL);
01307                 const stdair::LegCabinList_T lLegCabinList =
01308                   stdair::BomManager::getList<stdair::LegCabin> (*lSegmentCabin_ptr);
```

```
01309                 assert (!lLegCabinList.empty());
01310                 const int lNbOfLegs = lLegCabinList.size();
01311                 // Determine the yield (equally distributed over legs).
01312                 const stdair::Yield_T& lYield = lBookingClass_ptr->getYield()/lNbOfLegs;
01313                 const stdair::MeanStdDevPair_T& lMeanStdDevPair =
01314                   lYieldDemandPair.second;
01315                 const stdair::MeanValue_T& lMeanValue = lMeanStdDevPair.first;
01316                 const stdair::StdDevValue_T& lStdDevValue = lMeanStdDevPair.second;
01317                 for (stdair::LegCabinList_T::const_iterator itLC = lLegCabinList.begin();
01318                      itLC != lLegCabinList.end(); ++itLC) {
01319                   stdair::LegCabin* lLegCabin_ptr = *itLC;
01320                   assert (lLegCabin_ptr != NULL);
01321                   lLegCabin_ptr->addDemandInformation (lYield, lMeanValue, lStdDevValue);
01322                 }
01323               }
01324             }
01325           }
01326         }
01327       }
01328     }
01329   }
01330
01331   // ////////////////////////////////////////////////////////////////////
01332   void RMOL_Service::projectOnDDemandOnLegCabinsUsingYP(
      const stdair::DateTime_T& iRMEventTime) {
01333
01334     if (_rmolServiceContext == NULL) {
01335       throw stdair::NonInitialisedServiceException ("The Rmol service "
01336                                                     "has not been initialised");
01337     }
01338     assert (_rmolServiceContext != NULL);
01339     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
01340
01341     // Retrieve the bom root
01342     stdair::STDAIR_Service& lSTDAIR_Service =
01343       lRMOL_ServiceContext.getSTDAIR_Service();
01344     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01345
01346     // Retrieve the date from the RM event
01347     const stdair::Date_T lDate = iRMEventTime.date();
01348
01349     const stdair::InventoryList_T lInventoryList =
01350       stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01351     assert (!lInventoryList.empty());
01352     for (stdair::InventoryList_T::const_iterator itInv = lInventoryList.begin();
01353          itInv != lInventoryList.end(); ++itInv) {
01354       const stdair::Inventory* lInventory_ptr = *itInv;
01355       assert (lInventory_ptr != NULL);
01356       const stdair::OnDDateList_T lOnDDateList =
01357         stdair::BomManager::getList<stdair::OnDDate> (*lInventory_ptr);
01358       assert (!lOnDDateList.empty());
01359       for (stdair::OnDDateList_T::const_iterator itOD = lOnDDateList.begin();
01360            itOD != lOnDDateList.end(); ++itOD) {
01361         stdair::OnDDate* lOnDDate_ptr = *itOD;
01362         assert (lOnDDate_ptr != NULL);
01363
01364         const stdair::Date_T& lDepartureDate = lOnDDate_ptr->getDate();
01365         stdair::DateOffset_T lDateOffset = lDepartureDate - lDate;
01366         stdair::DTD_T lDTD = short (lDateOffset.days());
01367
01368         stdair::DCPList_T::const_iterator itDCP =
01369           std::find (stdair::DEFAULT_DCP_LIST.begin(), stdair::DEFAULT_DCP_LIST.end(), lDTD);
01370         // Check if the forecast for this O&D date needs to be projected.
01371         if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01372
01373           // Browse the demand info map.
01374           const stdair::StringDemandStructMap_T& lStringDemandStructMap =
01375             lOnDDate_ptr->getDemandInfoMap ();
01376           for (stdair::StringDemandStructMap_T::const_iterator itStrDS = lStringDemandStructMap.begin();
01377                itStrDS != lStringDemandStructMap.end(); ++itStrDS) {
01378             std::string lCabinClassPath = itStrDS->first;
01379             const stdair::YieldDemandPair_T& lYieldDemandPair =
01380               itStrDS->second;
01381             const stdair::CabinClassPairList_T& lCabinClassPairList =
01382               lOnDDate_ptr->getCabinClassPairList(lCabinClassPath);
01383             const stdair::NbOfSegments_T& lNbOfSegments = lOnDDate_ptr->getNbOfSegments();
01384             // Sanity check
01385             assert (lCabinClassPairList.size() == lNbOfSegments);
01386
01387             const stdair::SegmentDateList_T lOnDSegmentDateList =
01388               stdair::BomManager::getList<stdair::SegmentDate> (*lOnDDate_ptr);
01389             // Sanity check
01390             assert (lOnDSegmentDateList.size() == lNbOfSegments);
01391             stdair::CabinClassPairList_T::const_iterator itCCP = lCabinClassPairList.begin();
01392             stdair::SegmentDateList_T::const_iterator itSD = lOnDSegmentDateList.begin();
01393             for (; itSD != lOnDSegmentDateList.end(); ++itCCP, ++itSD) {
01394               const stdair::SegmentDate* lSegmentDate_ptr = *itSD;
```

```
01395               assert (lSegmentDate_ptr != NULL);
01396               const stdair::SegmentDate* lOperatingSegmentDate_ptr =
01397                 lSegmentDate_ptr->getOperatingSegmentDate ();
01398               // Only operated legs receive the demand information.
01399               if (lOperatingSegmentDate_ptr == NULL) {
01400                 const stdair::CabinCode_T lCabinCode = itCCP->first;
01401                 const stdair::ClassCode_T lClassCode = itCCP->second;
01402                 const stdair::SegmentCabin* lSegmentCabin_ptr =
01403                   stdair::BomManager::getObjectPtr<stdair::SegmentCabin> (*lSegmentDate_ptr,
01404                                                                           lCabinCode);
01405                 assert (lSegmentCabin_ptr != NULL);
01406                 const stdair::LegCabinList_T lLegCabinList =
01407                   stdair::BomManager::getList<stdair::LegCabin> (*lSegmentCabin_ptr);
01408                 assert (!lLegCabinList.empty());
01409                 const int lNbOfLegs = lLegCabinList.size();
01410                 // Determine the yield (equally distributed over segments and then legs).
01411                 const stdair::MeanStdDevPair_T& lMeanStdDevPair =
01412                   lYieldDemandPair.second;
01413                 const stdair::Yield_T& lYield = lYieldDemandPair.first/(lNbOfLegs*lNbOfSegments);
01414                 const stdair::MeanValue_T& lMeanValue = lMeanStdDevPair.first;
01415                 const stdair::StdDevValue_T& lStdDevValue = lMeanStdDevPair.second;
01416                 for (stdair::LegCabinList_T::const_iterator itLC = lLegCabinList.begin();
01417                      itLC != lLegCabinList.end(); ++itLC) {
01418                   stdair::LegCabin* lLegCabin_ptr = *itLC;
01419                   assert (lLegCabin_ptr != NULL);
01420                   lLegCabin_ptr->addDemandInformation (lYield, lMeanValue, lStdDevValue);
01421                 }
01422               }
01423             }
01424           }
01425         }
01426       }
01427     }
01428   }
01429
01430   // ////////////////////////////////////////////////////////////////////
01431   void RMOL_Service::optimiseOnD (const stdair::DateTime_T& iRMEventTime) {
01432
01433     if (_rmolServiceContext == NULL) {
01434       throw stdair::NonInitialisedServiceException ("The Rmol service "
01435                                                     "has not been initialised");
01436     }
01437     assert (_rmolServiceContext != NULL);
01438     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
01439
01440     // Retrieve the bom root
01441     stdair::STDAIR_Service& lSTDAIR_Service =
01442       lRMOL_ServiceContext.getSTDAIR_Service();
01443     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01444
01445     // Retrieve the date from the RM event
01446     const stdair::Date_T lDate = iRMEventTime.date();
01447
01448     const stdair::InventoryList_T& lInvList =
01449       stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01450     for (stdair::InventoryList_T::const_iterator itInv = lInvList.begin();
01451          itInv != lInvList.end(); ++itInv) {
01452       stdair::Inventory* lCurrentInv_ptr = *itInv;
01453       assert (lCurrentInv_ptr != NULL);
01454
01455       const stdair::FlightDateList_T& lFlightDateList =
01456         stdair::BomManager::getList<stdair::FlightDate> (*lCurrentInv_ptr);
01457       for (stdair::FlightDateList_T::const_iterator itFlightDate =
01458              lFlightDateList.begin();
01459            itFlightDate != lFlightDateList.end(); ++itFlightDate) {
01460         stdair::FlightDate* lCurrentFlightDate_ptr = *itFlightDate;
01461         assert (lCurrentFlightDate_ptr != NULL);
01462
01463         const stdair::Date_T& lCurrentDepartureDate = lCurrentFlightDate_ptr->getDepartureDate();
01464         stdair::DateOffset_T lDateOffset = lCurrentDepartureDate - lDate;
01465         stdair::DTD_T lDTD = short (lDateOffset.days());
01466
01467         stdair::DCPList_T::const_iterator itDCP =
01468           std::find (stdair::DEFAULT_DCP_LIST.begin(), stdair::DEFAULT_DCP_LIST.end(), lDTD);
01469         // Check if the optimisation is needed.
01470         if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01471           STDAIR_LOG_DEBUG ("Optimisation using O&D forecast: " << lCurrentInv_ptr->getAirlineCode()
01472                             << " Departure " << lCurrentDepartureDate << " DTD " << lDTD);
01473           Optimiser::optimiseUsingOnDForecast (*lCurrentFlightDate_ptr);
01474         }
01475       }
01476     }
01477   }
01478
01479   // ////////////////////////////////////////////////////////////////////
01480   void RMOL_Service::updateBidPrice (const stdair::DateTime_T& iRMEventTime) {
01481
```

```
01482     if (_rmolServiceContext == NULL) {
01483       throw stdair::NonInitialisedServiceException ("The Rmol service "
01484                                                     "has not been initialised");
01485     }
01486     assert (_rmolServiceContext != NULL);
01487     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
01488
01489     // Retrieve the bom root
01490     stdair::STDAIR_Service& lSTDAIR_Service =
01491       lRMOL_ServiceContext.getSTDAIR_Service();
01492     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01493
01494     // Retrieve the date from the RM event
01495     const stdair::Date_T lDate = iRMEventTime.date();
01496
01497     const stdair::InventoryList_T& lInvList =
01498       stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01499
01500     for (stdair::InventoryList_T::const_iterator itInv = lInvList.begin();
01501          itInv != lInvList.end(); ++itInv) {
01502       stdair::Inventory* lCurrentInv_ptr = *itInv;
01503       assert (lCurrentInv_ptr != NULL);
01504
01505       const stdair::FlightDateList_T& lFlightDateList =
01506         stdair::BomManager::getList<stdair::FlightDate> (*lCurrentInv_ptr);
01507       for (stdair::FlightDateList_T::const_iterator itFlightDate =
01508              lFlightDateList.begin();
01509            itFlightDate != lFlightDateList.end(); ++itFlightDate) {
01510         stdair::FlightDate* lCurrentFlightDate_ptr = *itFlightDate;
01511         assert (lCurrentFlightDate_ptr != NULL);
01512
01513         const stdair::Date_T& lCurrentDepartureDate = lCurrentFlightDate_ptr->getDepartureDate();
01514         stdair::DateOffset_T lDateOffset = lCurrentDepartureDate - lDate;
01515         stdair::DTD_T lDTD = short (lDateOffset.days());
01516
01517         stdair::DCPList_T::const_iterator itDCP =
01518           std::find (stdair::DEFAULT_DCP_LIST.begin(), stdair::DEFAULT_DCP_LIST.end(), lDTD);
01519         // Check if the operation is needed.
01520         if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01521           updateBidPrice (*lCurrentFlightDate_ptr, lBomRoot);
01522         }
01523       }
01524     }
01525   }
01526
01527   // ////////////////////////////////////////////////////////////////////
01528   void RMOL_Service::updateBidPrice (const stdair::FlightDate& iFlightDate,
01529                                      stdair::BomRoot& iBomRoot) {
01530     const stdair::SegmentDateList_T& lSegmentDateList =
01531       stdair::BomManager::getList<stdair::SegmentDate> (iFlightDate);
01532     const stdair::AirlineCode_T& lOptAC = iFlightDate.getAirlineCode();
01533     const std::string lFDKeyStr = iFlightDate.describeKey();
01534
01535     for (stdair::SegmentDateList_T::const_iterator itSegmentDate = lSegmentDateList.begin();
01536          itSegmentDate != lSegmentDateList.end(); ++itSegmentDate) {
01537       stdair::SegmentDate* lSegmentDate_ptr = *itSegmentDate;
01538       assert (lSegmentDate_ptr != NULL);
01539       const bool hasSegmentDateList =
01540         stdair::BomManager::hasList<stdair::SegmentDate>(*lSegmentDate_ptr);
01541       if (hasSegmentDateList == true) {
01542         const stdair::LegDateList_T& lLegDateList =
01543           stdair::BomManager::getList<stdair::LegDate>(*lSegmentDate_ptr);
01544         // Get the list of marketing carriers segments.
01545         // These are part of maketing partners inventories images held by the operating airline.
01546         const stdair::SegmentDateList_T& lMktSegmentDateList =
01547           stdair::BomManager::getList<stdair::SegmentDate>(*lSegmentDate_ptr);
01548         for (stdair::SegmentDateList_T::const_iterator itMktSD = lMktSegmentDateList.begin();
01549              itMktSD != lMktSegmentDateList.end(); ++itMktSD) {
01550           // Get the marketing airline code.
01551           stdair::SegmentDate* lMktSD_ptr = *itMktSD;
01552           assert (lMktSD_ptr != NULL);
01553           stdair::FlightDate* lMktFD_ptr =
01554             stdair::BomManager::getParentPtr<stdair::FlightDate>(*lMktSD_ptr);
01555           assert (lMktFD_ptr != NULL);
01556           const stdair::AirlineCode_T& lMktAC = lMktFD_ptr->getAirlineCode();
01557           // Get the (real) marketer inventory.
01558           const stdair::Inventory* lMktInv_ptr =
01559             stdair::BomManager::getObjectPtr<stdair::Inventory>(iBomRoot,lMktAC);
01560           assert (lMktInv_ptr != NULL);
01561           // Get the image of the operating airline inventory held by the marketer.
01562           const stdair::Inventory* lOptInv_ptr =
01563             stdair::BomManager::getObjectPtr<stdair::Inventory>(*lMktInv_ptr,lOptAC);
01564           assert (lOptInv_ptr != NULL);
01565           // Find the image of the concerned flight date.
01566           const stdair::FlightDate* lOptFD_ptr =
01567             stdair::BomManager::getObjectPtr<stdair::FlightDate>(*lOptInv_ptr,lFDKeyStr);
01568           assert (lOptFD_ptr != NULL);
```

```
01569           // Browse the list of leg dates in the real operating inventory.
01570           // Retrieve the image of each leg date.
01571           for (stdair::LegDateList_T::const_iterator itLD = lLegDateList.begin();
01572                itLD != lLegDateList.end(); ++itLD) {
01573             const stdair::LegDate* lLD_ptr = *itLD;
01574             assert (lLD_ptr != NULL);
01575             const std::string lLDKeyStr = lLD_ptr->describeKey();
01576             stdair::LegDate* lOptLD_ptr =
01577               stdair::BomManager::getObjectPtr<stdair::LegDate>(*lOptFD_ptr,lLDKeyStr);
01578             assert (lOptLD_ptr != NULL);
01579             const stdair::LegCabinList_T& lLegCabinList_T =
01580               stdair::BomManager::getList<stdair::LegCabin>(*lLD_ptr);
01581             // Browse the list of leg cabins in the real operating inventory.
01582             // Retrieve the image of each leg cabin and update the bid price of the real and send it to the
    image.
01583             for (stdair::LegCabinList_T::const_iterator itLC = lLegCabinList_T.begin();
01584                  itLC != lLegCabinList_T.end(); ++itLC) {
01585               stdair::LegCabin* lLC_ptr = *itLC;
01586               assert (lLC_ptr != NULL);
01587               const std::string lLCKeyStr = lLC_ptr->describeKey();
01588               stdair::LegCabin* lOptLC_ptr =
01589                 stdair::BomManager::getObjectPtr<stdair::LegCabin>(*lOptLD_ptr, lLCKeyStr);
01590               assert (lOptLC_ptr != NULL);
01591               // Update the current bid price of the real leg.
01592               lLC_ptr->updateCurrentBidPrice();
01593               // Update the previous bid price (store the current).
01594               lOptLC_ptr->updatePreviousBidPrice();
01595               // Update the current bid price.
01596               lOptLC_ptr->setCurrentBidPrice (lLC_ptr->getCurrentBidPrice());
01597
01598               STDAIR_LOG_DEBUG ("Update bid price of " << lLC_ptr->getFullerKey()
01599                                 << " : " << lOptLC_ptr->getCurrentBidPrice()
01600                                 << " Availability pool " << lLC_ptr->getAvailabilityPool());
01601             }
01602           }
01603         }
01604       }
01605     }
01606   }
01607
01608   // ////////////////////////////////////////////////////////////////////
01609   void RMOL_Service::projectOnDDemandOnLegCabinsUsingDA(
    const stdair::DateTime_T& iRMEventTime) {
01610
01611     if (_rmolServiceContext == NULL) {
01612       throw stdair::NonInitialisedServiceException ("The Rmol service "
01613                                                     "has not been initialised");
01614     }
01615     assert (_rmolServiceContext != NULL);
01616     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
01617
01618     // Retrieve the bom root
01619     stdair::STDAIR_Service& lSTDAIR_Service =
01620       lRMOL_ServiceContext.getSTDAIR_Service();
01621     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01622
01623     // Retrieve the date from the RM event
01624     const stdair::Date_T lDate = iRMEventTime.date();
01625
01626     const stdair::InventoryList_T lInventoryList =
01627       stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01628     assert (!lInventoryList.empty());
01629     for (stdair::InventoryList_T::const_iterator itInv = lInventoryList.begin();
01630          itInv != lInventoryList.end(); ++itInv) {
01631       const stdair::Inventory* lInventory_ptr = *itInv;
01632       assert (lInventory_ptr != NULL);
01633       const stdair::OnDDateList_T lOnDDateList =
01634         stdair::BomManager::getList<stdair::OnDDate> (*lInventory_ptr);
01635       assert (!lOnDDateList.empty());
01636       for (stdair::OnDDateList_T::const_iterator itOD = lOnDDateList.begin();
01637            itOD != lOnDDateList.end(); ++itOD) {
01638         stdair::OnDDate* lOnDDate_ptr = *itOD;
01639         assert (lOnDDate_ptr != NULL);
01640
01641         const stdair::Date_T& lDepartureDate = lOnDDate_ptr->getDate();
01642         stdair::DateOffset_T lDateOffset = lDepartureDate - lDate;
01643         stdair::DTD_T lDTD = short (lDateOffset.days());
01644
01645         stdair::DCPList_T::const_iterator itDCP =
01646           std::find (stdair::DEFAULT_DCP_LIST.begin(), stdair::DEFAULT_DCP_LIST.end(), lDTD);
01647         // Check if the forecast for this O&D date needs to be projected.
01648         if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01649
01650           // Browse the demand info map.
01651           const stdair::StringDemandStructMap_T& lStringDemandStructMap =
01652             lOnDDate_ptr->getDemandInfoMap ();
01653           for (stdair::StringDemandStructMap_T::const_iterator itStrDS = lStringDemandStructMap.begin();
```

```
01654               itStrDS != lStringDemandStructMap.end(); ++itStrDS) {
01655           std::string lCabinClassPath = itStrDS->first;
01656           const stdair::YieldDemandPair_T& lYieldDemandPair = itStrDS->second;
01657           const stdair::CabinClassPairList_T& lCabinClassPairList =
01658             lOnDDate_ptr->getCabinClassPairList(lCabinClassPath);
01659           const stdair::NbOfSegments_T& lNbOfSegments = lOnDDate_ptr->getNbOfSegments();
01660           // Sanity check
01661           assert (lCabinClassPairList.size() == lNbOfSegments);
01662
01663           //
01664           const stdair::SegmentDateList_T lOnDSegmentDateList =
01665             stdair::BomManager::getList<stdair::SegmentDate> (*lOnDDate_ptr);
01666           // Sanity check
01667           assert (lOnDSegmentDateList.size() == lNbOfSegments);
01668           stdair::CabinClassPairList_T::const_iterator itCCP = lCabinClassPairList.begin();
01669           stdair::SegmentDateList_T::const_iterator itSD = lOnDSegmentDateList.begin();
01670           // List of bid prices that will be used to easily compute displacement-ajusted yields.
01671           std::list<stdair::BidPrice_T> lBidPriceList;
01672           // The sum of bid prices that will be stored in the list above.
01673           stdair::BidPrice_T lTotalBidPrice = 0;
01674           // Retrieve the bid prices
01675           for (; itSD != lOnDSegmentDateList.end(); ++itCCP, ++itSD) {
01676             // Get the operating segment cabin (it holds the bid price information).
01677             const stdair::SegmentDate* lSegmentDate_ptr = *itSD;
01678             assert (lSegmentDate_ptr != NULL);
01679             // Get the operating airline code and check if it is the airline we are looking for.
01680             const stdair::SegmentDate* lOperatingSegmentDate_ptr =
01681               lSegmentDate_ptr->getOperatingSegmentDate ();
01682             if (lOperatingSegmentDate_ptr != NULL) {
01683               lSegmentDate_ptr = lOperatingSegmentDate_ptr;
01684             }
01685             const stdair::CabinCode_T lCabinCode = itCCP->first;
01686             const stdair::SegmentCabin* lSegmentCabin_ptr =
01687               stdair::BomManager::getObjectPtr<stdair::SegmentCabin> (*lSegmentDate_ptr,
01688                                                                       lCabinCode);
01689             assert (lSegmentCabin_ptr != NULL);
01690             stdair::BidPrice_T lBidPrice = 0;
01691             const stdair::LegCabinList_T lLegCabinList =
01692               stdair::BomManager::getList<stdair::LegCabin>(*lSegmentCabin_ptr);
01693             for (stdair::LegCabinList_T::const_iterator itLC = lLegCabinList.begin();
01694                  itLC != lLegCabinList.end(); ++itLC) {
01695               const stdair::LegCabin* lLegCabin_ptr = *itLC;
01696               assert (lLegCabin_ptr != NULL);
01697               lBidPrice += lLegCabin_ptr->getCurrentBidPrice();
01698             }
01699             lBidPriceList.push_back (lBidPrice);
01700             lTotalBidPrice += lBidPrice;
01701           }
01702
01703
01704           itCCP = lCabinClassPairList.begin();
01705           itSD = lOnDSegmentDateList.begin();
01706           std::list<stdair::BidPrice_T>::const_iterator itBP = lBidPriceList.begin();
01707           for (; itSD != lOnDSegmentDateList.end(); ++itCCP, ++itSD, ++itBP) {
01708             stdair::BidPrice_T lBidPrice = *itBP;
01709             stdair::BidPrice_T lComplementaryBidPrice = lTotalBidPrice - lBidPrice;
01710             const stdair::SegmentDate* lSegmentDate_ptr = *itSD;
01711             assert (lSegmentDate_ptr != NULL);
01712             const stdair::SegmentDate* lOperatingSegmentDate_ptr =
01713               lSegmentDate_ptr->getOperatingSegmentDate ();
01714             // Only operated legs receive the demand information.
01715             if (lOperatingSegmentDate_ptr == NULL) {
01716               const stdair::CabinCode_T lCabinCode = itCCP->first;
01717               const stdair::ClassCode_T lClassCode = itCCP->second;
01718               const stdair::SegmentCabin* lSegmentCabin_ptr =
01719                 stdair::BomManager::getObjectPtr<stdair::SegmentCabin> (*lSegmentDate_ptr,
01720                                                                         lCabinCode);
01721               assert (lSegmentCabin_ptr != NULL);
01722               const stdair::LegCabinList_T lLegCabinList =
01723                 stdair::BomManager::getList<stdair::LegCabin> (*lSegmentCabin_ptr);
01724               assert (!lLegCabinList.empty());
01725               // Determine the displacement-adjusted yield.
01726               // It is set to 100 (positive small value), if the computed value is negative.

01727               const stdair::Yield_T& lDAYield =
01728                 std::max(100., lYieldDemandPair.first - lComplementaryBidPrice);
01729
01730
01731               stdair::Yield_T lYield = lDAYield;
01732               // In order to be protected against important variations of partners' bid price,
01733               // the displacement adjusted yield is noy allowed to get out of a certain range.
01734               // This range is here chosen to be from 80% to 100% of the (static rule) prorated yield.
01735               /*
01736               const int lNbOfLegs = lLegCabinList.size();
01737               const stdair::Yield_T& lStaticProrationYield =
01738                 lDemandStruct.getYield()/(lNbOfLegs*lNbOfSegments);
01739               if (lDAYield < 0.8*lStaticProrationYield){
```

```
01740                   lYield = 0.8*lStaticProrationYield;
01741                 }
01742                 if (lDAYield > lStaticProrationYield) {
01743                   lYield = lStaticProrationYield;
01744                 }
01745                 */
01746                 const stdair::MeanStdDevPair_T& lMeanStdDevPair =
01747                   lYieldDemandPair.second;
01748                 const stdair::MeanValue_T& lMeanValue = lMeanStdDevPair.first;
01749                 const stdair::StdDevValue_T& lStdDevValue = lMeanStdDevPair.second;
01750                 for (stdair::LegCabinList_T::const_iterator itLC = lLegCabinList.begin();
01751                      itLC != lLegCabinList.end(); ++itLC) {
01752                   stdair::LegCabin* lLegCabin_ptr = *itLC;
01753                   assert (lLegCabin_ptr != NULL);
01754                   lLegCabin_ptr->addDemandInformation (lYield, lMeanValue, lStdDevValue);
01755                 }
01756               }
01757             }
01758           }
01759         }
01760       }
01761     }
01762   }
01763
01764   // ////////////////////////////////////////////////////////////////////
01765   void RMOL_Service::projectOnDDemandOnLegCabinsUsingDYP(
      const stdair::DateTime_T& iRMEventTime) {
01766
01767     if (_rmolServiceContext == NULL) {
01768       throw stdair::NonInitialisedServiceException ("The Rmol service "
01769                                                     "has not been initialised");
01770     }
01771     assert (_rmolServiceContext != NULL);
01772     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
01773
01774     // Retrieve the bom root
01775     stdair::STDAIR_Service& lSTDAIR_Service =
01776       lRMOL_ServiceContext.getSTDAIR_Service();
01777     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01778
01779     const stdair::InventoryList_T lInventoryList =
01780       stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01781     assert (!lInventoryList.empty());
01782     for (stdair::InventoryList_T::const_iterator itInv = lInventoryList.begin();
01783          itInv != lInventoryList.end(); ++itInv) {
01784       const stdair::Inventory* lInventory_ptr = *itInv;
01785       assert (lInventory_ptr != NULL);
01786       projectOnDDemandOnLegCabinsUsingDYP (iRMEventTime, *lInventory_ptr
      );
01787     }
01788   }
01789
01790   // ////////////////////////////////////////////////////////////////////
01791   void RMOL_Service::projectOnDDemandOnLegCabinsUsingDYP(
      const stdair::DateTime_T& iRMEventTime,
01792                                                          const stdair::Inventory& iInventory) {
01793
01794     const stdair::OnDDateList_T lOnDDateList =
01795       stdair::BomManager::getList<stdair::OnDDate> (iInventory);
01796     assert (!lOnDDateList.empty());
01797     for (stdair::OnDDateList_T::const_iterator itOD = lOnDDateList.begin();
01798          itOD != lOnDDateList.end(); ++itOD) {
01799       stdair::OnDDate* lOnDDate_ptr = *itOD;
01800       assert (lOnDDate_ptr != NULL);
01801
01802       // Retrieve the date from the RM event
01803       const stdair::Date_T lDate = iRMEventTime.date();
01804
01805       const stdair::Date_T& lDepartureDate = lOnDDate_ptr->getDate();
01806       stdair::DateOffset_T lDateOffset = lDepartureDate - lDate;
01807       stdair::DTD_T lDTD = short (lDateOffset.days());
01808
01809       stdair::DCPList_T::const_iterator itDCP =
01810         std::find (stdair::DEFAULT_DCP_LIST.begin(), stdair::DEFAULT_DCP_LIST.end(), lDTD);
01811       // Check if the forecast for this O&D date needs to be projected.
01812       if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01813
01814         // Browse the demand info map.
01815         const stdair::StringDemandStructMap_T& lStringDemandStructMap =
01816           lOnDDate_ptr->getDemandInfoMap ();
01817         for (stdair::StringDemandStructMap_T::const_iterator itStrDS = lStringDemandStructMap.begin();
01818              itStrDS != lStringDemandStructMap.end(); ++itStrDS) {
01819           std::string lCabinClassPath = itStrDS->first;
01820           const stdair::YieldDemandPair_T& lYieldDemandPair = itStrDS->second;
01821           const stdair::CabinClassPairList_T& lCabinClassPairList =
01822             lOnDDate_ptr->getCabinClassPairList(lCabinClassPath);
01823           const stdair::NbOfSegments_T& lNbOfSegments = lOnDDate_ptr->getNbOfSegments();
```

```
01824           // Sanity check
01825           assert (lCabinClassPairList.size() == lNbOfSegments);
01826
01827           //
01828           const stdair::SegmentDateList_T lOnDSegmentDateList =
01829               stdair::BomManager::getList<stdair::SegmentDate> (*lOnDDate_ptr);
01830           // Sanity check
01831           assert (lOnDSegmentDateList.size() == lNbOfSegments);
01832           stdair::CabinClassPairList_T::const_iterator itCCP = lCabinClassPairList.begin();
01833           stdair::SegmentDateList_T::const_iterator itSD = lOnDSegmentDateList.begin();
01834           // The sum of bid prices of all cabins.
01835           stdair::BidPrice_T lTotalBidPrice = 0;
01836           for (; itSD != lOnDSegmentDateList.end(); ++itCCP, ++itSD) {
01837             // Get the operating segment cabin (it holds the bid price information).
01838             const stdair::SegmentDate* lSegmentDate_ptr = *itSD;
01839             assert (lSegmentDate_ptr != NULL);
01840             // Get the operating airline code and check if it is the airline we are looking for.
01841             const stdair::SegmentDate* lOperatingSegmentDate_ptr =
01842               lSegmentDate_ptr->getOperatingSegmentDate ();
01843             if (lOperatingSegmentDate_ptr != NULL) {
01844               lSegmentDate_ptr = lOperatingSegmentDate_ptr;
01845             }
01846             const stdair::CabinCode_T lCabinCode = itCCP->first;
01847             const stdair::SegmentCabin* lSegmentCabin_ptr =
01848               stdair::BomManager::getObjectPtr<stdair::SegmentCabin> (*lSegmentDate_ptr,
01849                                                                       lCabinCode);
01850             assert (lSegmentCabin_ptr != NULL);
01851             const stdair::LegCabinList_T lLegCabinList =
01852               stdair::BomManager::getList<stdair::LegCabin>(*lSegmentCabin_ptr);
01853             for (stdair::LegCabinList_T::const_iterator itLC = lLegCabinList.begin();
01854                  itLC != lLegCabinList.end(); ++itLC) {
01855               const stdair::LegCabin* lLegCabin_ptr = *itLC;
01856               assert (lLegCabin_ptr != NULL);
01857               lTotalBidPrice += lLegCabin_ptr->getCurrentBidPrice();
01858             }
01859           }
01860
01861
01862           itCCP = lCabinClassPairList.begin();
01863           itSD = lOnDSegmentDateList.begin();
01864           for (; itSD != lOnDSegmentDateList.end(); ++itCCP, ++itSD) {
01865             const stdair::SegmentDate* lSegmentDate_ptr = *itSD;
01866             assert (lSegmentDate_ptr != NULL);
01867             const stdair::SegmentDate* lOperatingSegmentDate_ptr =
01868               lSegmentDate_ptr->getOperatingSegmentDate ();
01869             // Only operated legs receive the demand information.
01870             if (lOperatingSegmentDate_ptr == NULL) {
01871               const stdair::CabinCode_T lCabinCode = itCCP->first;
01872               const stdair::ClassCode_T lClassCode = itCCP->second;
01873               const stdair::SegmentCabin* lSegmentCabin_ptr =
01874                 stdair::BomManager::getObjectPtr<stdair::SegmentCabin> (*lSegmentDate_ptr,
01875                                                                         lCabinCode);
01876               assert (lSegmentCabin_ptr != NULL);
01877               const stdair::LegCabinList_T lLegCabinList =
01878                 stdair::BomManager::getList<stdair::LegCabin> (*lSegmentCabin_ptr);
01879               assert (!lLegCabinList.empty());
01880               const stdair::Yield_T& lYield = lYieldDemandPair.first;
01881               const stdair::MeanStdDevPair_T& lMeanStdDevPair =
01882                 lYieldDemandPair.second;
01883               const stdair::MeanValue_T& lMeanValue = lMeanStdDevPair.first;
01884               const stdair::StdDevValue_T& lStdDevValue = lMeanStdDevPair.second;
01885               for (stdair::LegCabinList_T::const_iterator itLC = lLegCabinList.begin();
01886                    itLC != lLegCabinList.end(); ++itLC) {
01887                 stdair::LegCabin* lLegCabin_ptr = *itLC;
01888                 assert (lLegCabin_ptr != NULL);
01889                 const stdair::BidPrice_T& lBidPrice = lLegCabin_ptr->getCurrentBidPrice();
01890                 const stdair::RealNumber_T lDynamicYieldProrationFactor = lBidPrice / lTotalBidPrice;
01891                 const stdair::Yield_T lProratedYield = lDynamicYieldProrationFactor*lYield;
01892                 lLegCabin_ptr->addDemandInformation (lProratedYield, lMeanValue, lStdDevValue);
01893
01894                 // STDAIR_LOG_DEBUG ("Addding demand information to leg-cabin " <<
      lLegCabin_ptr->getFullerKey()
01895                 //                   << " Total yield " << lYield << " Proration factor "
01896                 //                   << lDynamicYieldProrationFactor << " Prorated yield " <<
      lProratedYield
01897                 //                   << " Mean demand " << lMeanValue << " StdDev " << lStdDevValue);
01898               }
01899             }
01900           }
01901         }
01902       }
01903     }
01904   }
01905
01906   // ////////////////////////////////////////////////////////////////////
01907   void RMOL_Service::optimiseOnDUsingRMCooperation (const
      stdair::DateTime_T& iRMEventTime) {
```

```
01908
01909     if (_rmolServiceContext == NULL) {
01910       throw stdair::NonInitialisedServiceException ("The Rmol service "
01911                                                     "has not been initialised");
01912     }
01913     assert (_rmolServiceContext != NULL);
01914     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
01915
01916     // Retrieve the bom root
01917     stdair::STDAIR_Service& lSTDAIR_Service =
01918       lRMOL_ServiceContext.getSTDAIR_Service();
01919     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01920
01921     // Retrieve the date from the RM event
01922     const stdair::Date_T lDate = iRMEventTime.date();
01923
01924     // Browse the list of inventories and optimise within each one independently.
01925     const stdair::InventoryList_T& lInvList =
01926       stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01927     for (stdair::InventoryList_T::const_iterator itInv = lInvList.begin();
01928          itInv != lInvList.end(); ++itInv) {
01929       stdair::Inventory* lCurrentInv_ptr = *itInv;
01930       assert (lCurrentInv_ptr != NULL);
01931
01932       double lMaxBPVariation = 1.0;
01933       short lIterationCounter = 0;
01934       // Iterate until the variation is under the wanted level or the maximal number of iterations is
      reached.
01935       while (lMaxBPVariation > 0.01 && lIterationCounter < 10) {
01936         lMaxBPVariation = 0.0;
01937         lIterationCounter++;
01938         const stdair::FlightDateList_T& lFlightDateList =
01939           stdair::BomManager::getList<stdair::FlightDate> (*lCurrentInv_ptr);
01940         for (stdair::FlightDateList_T::const_iterator itFlightDate =
01941                lFlightDateList.begin();
01942              itFlightDate != lFlightDateList.end(); ++itFlightDate) {
01943           stdair::FlightDate* lCurrentFlightDate_ptr = *itFlightDate;
01944           assert (lCurrentFlightDate_ptr != NULL);
01945
01946           const stdair::Date_T& lCurrentDepartureDate = lCurrentFlightDate_ptr->getDepartureDate();
01947           stdair::DateOffset_T lDateOffset = lCurrentDepartureDate - lDate;
01948           stdair::DTD_T lDTD = short (lDateOffset.days());
01949
01950           stdair::DCPList_T::const_iterator itDCP =
01951             std::find (stdair::DEFAULT_DCP_LIST.begin(), stdair::DEFAULT_DCP_LIST.end(), lDTD);
01952           // Check if the optimisation is needed.
01953           if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01954             const double lBPVariation = Optimiser::optimiseUsingOnDForecast
      (*lCurrentFlightDate_ptr);
01955             lMaxBPVariation = std::max(lMaxBPVariation, lBPVariation);
01956           }
01957         }
01958         // Update the prorated yields for the current inventory.
01959         resetDemandInformation (iRMEventTime, *lCurrentInv_ptr);
01960         projectOnDDemandOnLegCabinsUsingDYP (iRMEventTime, *
      lCurrentInv_ptr);
01961       }
01962     }
01963   }
01964
01965
01966   // ////////////////////////////////////////////////////////////////////
01967   void RMOL_Service::optimiseOnDUsingAdvancedRMCooperation
      (const stdair::DateTime_T& iRMEventTime) {
01968
01969     if (_rmolServiceContext == NULL) {
01970       throw stdair::NonInitialisedServiceException ("The Rmol service "
01971                                                     "has not been initialised");
01972     }
01973     assert (_rmolServiceContext != NULL);
01974     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
01975
01976     // Retrieve the bom root
01977     stdair::STDAIR_Service& lSTDAIR_Service =
01978       lRMOL_ServiceContext.getSTDAIR_Service();
01979     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01980
01981     // Retrieve the date from the RM event
01982     const stdair::Date_T lDate = iRMEventTime.date();
01983
01984     double lMaxBPVariation = 1.0;
01985     short lIterationCounter = 0;
01986     // Iterate until the variation is under the wanted level or the maximal number of iterations is
      reached.
01987     // Every iteration corresponds to the optimisation of the whole network. Bid prices are communicated
01988     // between partners at the end of each iteration.
01989     while (lMaxBPVariation > 0.01 && lIterationCounter < 50) {
```

```
01990       lMaxBPVariation = 0.0;
01991       lIterationCounter++;
01992
01993       const stdair::InventoryList_T& lInvList =
01994         stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01995       for (stdair::InventoryList_T::const_iterator itInv = lInvList.begin();
01996            itInv != lInvList.end(); ++itInv) {
01997         stdair::Inventory* lCurrentInv_ptr = *itInv;
01998         assert (lCurrentInv_ptr != NULL);
01999         const stdair::FlightDateList_T& lFlightDateList =
02000           stdair::BomManager::getList<stdair::FlightDate> (*lCurrentInv_ptr);
02001         for (stdair::FlightDateList_T::const_iterator itFlightDate =
02002                lFlightDateList.begin();
02003              itFlightDate != lFlightDateList.end(); ++itFlightDate) {
02004           stdair::FlightDate* lCurrentFlightDate_ptr = *itFlightDate;
02005           assert (lCurrentFlightDate_ptr != NULL);
02006
02007           const stdair::Date_T& lCurrentDepartureDate = lCurrentFlightDate_ptr->getDepartureDate();
02008           stdair::DateOffset_T lDateOffset = lCurrentDepartureDate - lDate;
02009           stdair::DTD_T lDTD = short (lDateOffset.days());
02010
02011           stdair::DCPList_T::const_iterator itDCP =
02012             std::find (stdair::DEFAULT_DCP_LIST.begin(), stdair::DEFAULT_DCP_LIST.end(), lDTD);
02013           if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
02014             const double lBPVariation = Optimiser::optimiseUsingOnDForecast
     (*lCurrentFlightDate_ptr);
02015             lMaxBPVariation = std::max(lMaxBPVariation, lBPVariation);
02016           }
02017         }
02018       }
02019       // At the end of each iteration, communicate bid prices and compute displacement adjusted yields.
02020       updateBidPrice (iRMEventTime);
02021       resetDemandInformation (iRMEventTime);
02022       projectOnDDemandOnLegCabinsUsingDYP (iRMEventTime);
02023     }
02024   }
02025
02026 }
```

26.133 rmol/service/RMOL_ServiceContext.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <stdair/STDAIR_Service.hpp>
#include <rmol/basic/BasConst_RMOL_Service.hpp>
#include <rmol/service/RMOL_ServiceContext.hpp>
```

Namespaces

- RMOL

26.134 RMOL_ServiceContext.cpp

```
00001 // ////////////////////////////////////////////////////////////////////
00002 // Import section
00003 // ////////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 // StdAir
00008 #include <stdair/STDAIR_Service.hpp>
00009 // RMOL
00010 #include <rmol/basic/BasConst_RMOL_Service.hpp>
00011 #include <rmol/service/RMOL_ServiceContext.hpp>
00012
00013 namespace RMOL {
00014
00015   // ////////////////////////////////////////////////////////////////////
00016   RMOL_ServiceContext::RMOL_ServiceContext() : _ownStdairService (false) {
00017   }
00018
00019   // ////////////////////////////////////////////////////////////////////
00020   RMOL_ServiceContext::RMOL_ServiceContext (const RMOL_ServiceContext&) {
00021     assert (false);
```

```
00022   }
00023
00024   // ////////////////////////////////////////////////////////////////////
00025   RMOL_ServiceContext::~RMOL_ServiceContext() {
00026   }
00027
00028   // ////////////////////////////////////////////////////////////////////
00029   stdair::STDAIR_Service& RMOL_ServiceContext::getSTDAIR_Service() const {
00030     assert (_stdairService != NULL);
00031     return *_stdairService;
00032   }
00033
00034   // ////////////////////////////////////////////////////////////////////
00035   const std::string RMOL_ServiceContext::shortDisplay() const {
00036     std::ostringstream oStr;
00037     oStr << "RMOL_ServiceContext -- Owns StdAir service: " << _ownStdairService;
00038     return oStr.str();
00039   }
00040
00041   // ////////////////////////////////////////////////////////////////////
00042   const std::string RMOL_ServiceContext::display() const {
00043     std::ostringstream oStr;
00044     oStr << shortDisplay();
00045     return oStr.str();
00046   }
00047
00048   // ////////////////////////////////////////////////////////////////////
00049   const std::string RMOL_ServiceContext::describe() const {
00050     return shortDisplay();
00051   }
00052
00053   // ////////////////////////////////////////////////////////////////////
00054   void RMOL_ServiceContext::reset() {
00055
00056     // The shared_ptr<>::reset() method drops the refcount by one.
00057     // If the count result is dropping to zero, the resource pointed to
00058     // by the shared_ptr<> will be freed.
00059
00060     // Reset the stdair shared pointer
00061     _stdairService.reset();
00062   }
00063
00064 }
```

26.135 rmol/service/RMOL_ServiceContext.hpp File Reference

```
#include <string>
#include <stdair/stdair_basic_types.hpp>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/stdair_maths_types.hpp>
#include <stdair/stdair_service_types.hpp>
#include <stdair/service/ServiceAbstract.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class RMOL::RMOL_ServiceContext

  Inner class holding the context for the RMOL Service object.

Namespaces

- stdair

  Forward declarations.
- RMOL

26.136 RMOL_ServiceContext.hpp

```
00001 #ifndef __RMOL_SVC_RMOL_SERVICE_CONTEXT_HPP
```

```
00002 #define __RMOL_SVC_RMOL_SERVICE_CONTEXT_HPP
00003
00004 // //////////////////////////////////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////////////////////////////////
00007 // STL
00008 #include <string>
00009 // StdAir
00010 #include <stdair/stdair_basic_types.hpp>
00011 #include <stdair/stdair_inventory_types.hpp>
00012 #include <stdair/stdair_maths_types.hpp>
00013 #include <stdair/stdair_service_types.hpp>
00014 #include <stdair/service/ServiceAbstract.hpp>
00015 // RMOL
00016 #include <rmol/RMOL_Types.hpp>
00017
00019 namespace stdair {
00020   class STDAIR_Service;
00021   class LegCabin;
00022 }
00023
00024 namespace RMOL {
00025
00029   class RMOL_ServiceContext : public stdair::ServiceAbstract {
00035     friend class RMOL_Service;
00036     friend class FacRmolServiceContext;
00037
00038   private:
00039     // ///////// Getters ///////////
00043     stdair::STDAIR_ServicePtr_T getSTDAIR_ServicePtr() const {
00044       return _stdairService;
00045     }
00046
00050     stdair::STDAIR_Service& getSTDAIR_Service() const;
00051
00055     const bool getOwnStdairServiceFlag() const {
00056       return _ownStdairService;
00057     }
00058
00059
00060   private:
00061     // ///////// Setters ///////////
00065     void setSTDAIR_Service (stdair::STDAIR_ServicePtr_T ioSTDAIR_ServicePtr,
00066                             const bool iOwnStdairService) {
00067       _stdairService = ioSTDAIR_ServicePtr;
00068       _ownStdairService = iOwnStdairService;
00069     }
00070
00074     void reset();
00075
00076
00077   private:
00078     // ///////// Display Methods ///////////
00082     const std::string shortDisplay() const;
00083
00087     const std::string display() const;
00088
00092     const std::string describe() const;
00093
00094
00095   private:
00096     // /////// Construction / initialisation ////////
00100     RMOL_ServiceContext();
00104     RMOL_ServiceContext (const RMOL_ServiceContext&);
00105
00109     ~RMOL_ServiceContext();
00110
00111
00112   private:
00113     // ////////////// Children ////////////
00117     stdair::STDAIR_ServicePtr_T _stdairService;
00118
00122     bool _ownStdairService;
00123   };
00124
00125 }
00126 #endif // __RMOL_SVC_RMOL_SERVICE_CONTEXT_HPP
```

26.137 test/rmol/bomsforforecaster.cpp File Reference

26.138 bomsforforecaster.cpp

```
00001
```

```
00005 // ////////////////////////////////////////////////////////////////////
00006 // Import section
00007 // ////////////////////////////////////////////////////////////////////
00008 // STL
00009 #include <cassert>
00010 #include <limits>
00011 #include <sstream>
00012 #include <fstream>
00013 #include <string>
00014 // Boost Unit Test Framework (UTF)
00015 #define BOOST_TEST_DYN_LINK
00016 #define BOOST_TEST_MAIN
00017 #define BOOST_TEST_MODULE OptimiseTestSuite
00018 #include <boost/test/unit_test.hpp>
00019 // StdAir
00020 #include <stdair/basic/BasLogParams.hpp>
00021 #include <stdair/basic/BasDBParams.hpp>
00022 #include <stdair/service/Logger.hpp>
00023 // RMOL
00024 #include <rmol/RMOL_Service.hpp>
00025 #include <rmol/config/rmol-paths.hpp>
00026
00027 namespace boost_utf = boost::unit_test;
00028
00029 // (Boost) Unit Test XML Report
00030 std::ofstream utfReportStream ("bomsforforecaster_utfresults.xml");
00031
00035 struct UnitTestConfig {
00037   UnitTestConfig() {
00038     boost_utf::unit_test_log.set_stream (utfReportStream);
00039     boost_utf::unit_test_log.set_format (boost_utf::XML);
00040     boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
00041     //boost_utf::unit_test_log.set_threshold_level (boost_utf::log_successful_tests);
00042   }
00043
00045   ~UnitTestConfig() {
00046   }
00047 };
00048
00049 namespace RMOL {
00050
00052   struct BookingClassData {
00053
00054     // Attributes
00055     double _bookingCount;
00056     double _fare;
00057     double _sellupFactor;
00058     bool _censorshipFlag;
00059
00060     // Constructer
00061     BookingClassData (const double iBookingCount, const double iFare,
00062                       const double iSellupFactor, const bool iCensorshipFlag)
00063       : _bookingCount(iBookingCount), _fare(iFare),
00064         _sellupFactor(iSellupFactor), _censorshipFlag(iCensorshipFlag) {
00065     }
00066
00067     // Getters
00068     double getFare () const {
00069       return _fare;
00070     }
00071
00072     bool getCensorshipFlag () const {
00073       return _censorshipFlag;
00074     }
00075
00076     // Display
00077     std::string toString() const {
00078       std::ostringstream oStr;
00079       oStr <<  std::endl
00080            << "[Booking class data information]" << std::endl
00081            << "Booking counter: " << _bookingCount << std::endl
00082            << "Fare: " << _fare << std::endl
00083            << "Sell-up Factor: " << _sellupFactor << std::endl
00084            << "censorshipFlag: " << _censorshipFlag << std::endl;
00085       return oStr.str();
00086     }
00087
00088   };
00089
00091   struct BookingClassDataSet {
00092
00093     typedef std::vector<BookingClassData*> BookingClassDataList_T;
00094
00095     // Attributes
00096     int _numberOfClass;
00097     double _minimumFare;
00098     bool _censorshipFlag; // true if any of the classes is censored
```

```
00099     BookingClassDataList_T _bookingClassDataList;
00100
00101     // Constructor
00102     BookingClassDataSet ()
00103       : _numberOfClass(0), _minimumFare(0),
00104         _censorshipFlag(false) {
00105     }
00106
00107     // Add BookingClassData
00108     void addBookingClassData (BookingClassData& ioBookingClassData) {
00109       _bookingClassDataList.push_back (&ioBookingClassData);
00110     }
00111
00112     // Getters
00113     stdair::NbOfClasses_T getNumberOfClass () const {
00114       return _bookingClassDataList.size();
00115     }
00116
00117     double getMinimumFare () const {
00118       return _minimumFare;
00119     }
00120
00121     bool getCensorshipFlag () const {
00122       return _censorshipFlag;
00123     }
00124
00125     // Setters
00126     void setMinimumFare (const double iMinFare) {
00127       _minimumFare = iMinFare;
00128     }
00129
00130     void setCensorshipFlag (const bool iCensorshipFlag) {
00131       _censorshipFlag = iCensorshipFlag;
00132     }
00133
00134     // compute minimum fare
00135     void updateMinimumFare() {
00136       double minFare = std::numeric_limits<double>::max();
00137       BookingClassDataList_T::iterator itBookingClassDataList;
00138       for (itBookingClassDataList = _bookingClassDataList.begin();
00139            itBookingClassDataList != _bookingClassDataList.end();
00140            ++itBookingClassDataList) {
00141         BookingClassData* lBookingClassData = *itBookingClassDataList;
00142         assert (lBookingClassData != NULL);
00143
00144         const double lFare = lBookingClassData->getFare();
00145         if (lFare < minFare) {
00146           minFare = lFare;
00147         }
00148       }
00149       //
00150       setMinimumFare(minFare);
00151     }
00152
00153     // compute censorship flag for the data set
00154     void updateCensorshipFlag () {
00155       bool censorshipFlag = false;
00156       BookingClassDataList_T::iterator itBookingClassDataList;
00157       for (itBookingClassDataList = _bookingClassDataList.begin();
00158            itBookingClassDataList != _bookingClassDataList.end();
00159            ++itBookingClassDataList) {
00160         BookingClassData* lBookingClassData = *itBookingClassDataList;
00161         assert (lBookingClassData != NULL);
00162
00163         const bool lCensorshipFlagOfAClass =
00164           lBookingClassData->getCensorshipFlag();
00165         if (lCensorshipFlagOfAClass) {
00166           censorshipFlag = true;
00167           break;
00168         }
00169       }
00170       //
00171       setCensorshipFlag(censorshipFlag);
00172     }
00173
00174     // Display
00175     std::string toString() const {
00176       std::ostringstream oStr;
00177       oStr << std::endl
00178            << "[Booking class data set information]" << std::endl
00179            << "Number of classes: " << _numberOfClass << std::endl
00180            << "Minimum fare: " << _minimumFare << std::endl
00181            << "The data of the class set are sensored: " << _censorshipFlag
00182            << std::endl;
00183       return oStr.str();
00184     }
00185
```

```
00186   };
00187
00188   // /**-------------- BOM : Q-Forecaster ---------------------- */
00189   // struct QForecaster {
00190
00191   //   // Function focused BOM
00192
00193   //   // 1. calculate sell up probability for Q-eq
00194
00195   //   // 2. calculate Q-Equivalent Booking
00196   //   double calculateQEqBooking (BookingClassDataSet& iBookingClassDataSet) {
00197   //     double lQEqBooking = 0.0;
00198   //     double lMinFare = iBookingClassDataSet.getMinimumFare();
00199
00200
00201   //     return lQEqBooking;
00202   //   }
00203
00204   //   /* Calculate Q-equivalent demand
00205   //      [<- performed by unconstrainer if necessary (Using ExpMax BOM)]
00206   //   */
00207
00208
00209   //   // 3. Partition to each class
00210
00211   //   //
00212
00213   // };
00214
00215 }
00216
00217 // ///////////////// Main: Unit Test Suite ///////////////
00218
00219 // Set the UTF configuration (re-direct the output to a specific file)
00220 BOOST_GLOBAL_FIXTURE (UnitTestConfig);
00221
00225 BOOST_AUTO_TEST_SUITE (master_test_suite)
00226
00227
00230 BOOST_AUTO_TEST_CASE (rmol_forecaster) {
00231
00232   // Output log File
00233   std::string lLogFilename ("bomsforforecaster.log");
00234   std::ofstream logOutputFile;
00235
00236   // Open and clean the log outputfile
00237   logOutputFile.open (lLogFilename.c_str());
00238   logOutputFile.clear();
00239
00240   // Initialise the RMOL service
00241   const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);
00242
00243   // Initialise the RMOL service
00244   RMOL::RMOL_Service rmolService (lLogParams);
00245
00246   // Build a sample BOM tree
00247   rmolService.buildSampleBom();
00248
00249   // Register BCDataSet
00250   RMOL::BookingClassDataSet lBookingClassDataSet;
00251
00252   // Register BookingClassData
00253   RMOL::BookingClassData QClassData (10, 100, 1, false);
00254   RMOL::BookingClassData MClassData (5, 150, 0.8, true);
00255   RMOL::BookingClassData BClassData (0, 200, 0.6, false);
00256   RMOL::BookingClassData YClassData (0, 300, 0.3, false);
00257
00258   // Display
00259   STDAIR_LOG_DEBUG (QClassData.toString());
00260   STDAIR_LOG_DEBUG (MClassData.toString());
00261   STDAIR_LOG_DEBUG (BClassData.toString());
00262   STDAIR_LOG_DEBUG (YClassData.toString());
00263
00264   // Add BookingClassData into the BCDataSet
00265   lBookingClassDataSet.addBookingClassData (QClassData);
00266   lBookingClassDataSet.addBookingClassData (MClassData);
00267   lBookingClassDataSet.addBookingClassData (BClassData);
00268   lBookingClassDataSet.addBookingClassData (YClassData);
00269
00270   // DEBUG
00271   STDAIR_LOG_DEBUG (lBookingClassDataSet.toString());
00272
00273   // Number of classes
00274   const stdair::NbOfClasses_T lNbOfClass = lBookingClassDataSet.getNumberOfClass();
00275
00276   // DEBUG
00277   STDAIR_LOG_DEBUG ("Number of Classes: " << lNbOfClass);
```

```
00278
00279   // Minimum fare
00280   BOOST_CHECK_NO_THROW (lBookingClassDataSet.updateMinimumFare());
00281   const double lMinFare = lBookingClassDataSet.getMinimumFare();
00282
00283   // DEBUG
00284   STDAIR_LOG_DEBUG ("Minimum fare: " << lMinFare);
00285
00286   // Censorship flag
00287   BOOST_CHECK_NO_THROW (lBookingClassDataSet.updateCensorshipFlag());
00288   const bool lCensorshipFlag = lBookingClassDataSet.getCensorshipFlag();
00289
00290   // DEBUG
00291   STDAIR_LOG_DEBUG ("Censorship Flag: " << lCensorshipFlag);
00292
00293   // Close the log output file
00294   logOutputFile.close();
00295 }
00296
00297 // End the test suite
00298 BOOST_AUTO_TEST_SUITE_END()
00299
00300
```

26.139 test/rmol/ForecasterTestSuite.cpp File Reference

26.140 ForecasterTestSuite.cpp

```
00001
00005 // ////////////////////////////////////////////////////////////////////
00006 // Import section
00007 // ////////////////////////////////////////////////////////////////////
00008 // STL
00009 #include <sstream>
00010 #include <fstream>
00011 #include <string>
00012 #include <vector>
00013 #include <cmath>
00014 // Boost Unit Test Framework (UTF)
00015 #define BOOST_TEST_DYN_LINK
00016 #define BOOST_TEST_MAIN
00017 #define BOOST_TEST_MODULE ForecasterTestSuite
00018 #include <boost/test/unit_test.hpp>
00019 // StdAir
00020 #include <stdair/basic/BasLogParams.hpp>
00021 #include <stdair/basic/BasDBParams.hpp>
00022 #include <stdair/basic/BasFileMgr.hpp>
00023 #include <stdair/service/Logger.hpp>
00024 // RMOL
00025 #include <rmol/RMOL_Service.hpp>
00026
00027 namespace boost_utf = boost::unit_test;
00028
00029 // (Boost) Unit Test XML Report
00030 std::ofstream utfReportStream ("ForecasterTestSuite_utfresults.xml");
00031
00035 struct UnitTestConfig {
00037   UnitTestConfig() {
00038     boost_utf::unit_test_log.set_stream (utfReportStream);
00039     boost_utf::unit_test_log.set_format (boost_utf::XML);
00040     boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
00041     //boost_utf::unit_test_log.set_threshold_level (boost_utf::log_successful_tests);
00042   }
00043
00045   ~UnitTestConfig() {
00046   }
00047 };
00048
00049
00050 // ///////////////// Main: Unit Test Suite //////////////
00051
00052 // Set the UTF configuration (re-direct the output to a specific file)
00053 BOOST_GLOBAL_FIXTURE (UnitTestConfig);
00054
00059 BOOST_AUTO_TEST_SUITE (master_test_suite)
00060
00061
00064 BOOST_AUTO_TEST_CASE (rmol_forecaster_q_forecasting) {
00065   const bool lTestFlag = true; //testForecasterHelper(0);
00066   BOOST_CHECK_EQUAL (lTestFlag, true);
00067   BOOST_CHECK_MESSAGE (lTestFlag == true,
00068                        "The test has failed. Please see the log file for "
00069                        << "more details");
```

```
00070 }
00071
00072 // End the test suite
00073 BOOST_AUTO_TEST_SUITE_END()
00074
00075
```

26.141 test/rmol/ForecasterTestSuite.hpp File Reference

```
#include <sstream>
#include <cppunit/extensions/HelperMacros.h>
```

Classes

- class ForecasterTestSuite

Functions

- CPPUNIT_TEST_SUITE_REGISTRATION (ForecasterTestSuite)

26.141.1 Function Documentation

26.141.1.1 CPPUNIT_TEST_SUITE_REGISTRATION (ForecasterTestSuite)

26.142 ForecasterTestSuite.hpp

```
00001 // STL
00002 #include <sstream>
00003 // CPPUNIT
00004 #include <cppunit/extensions/HelperMacros.h>
00005
00006 class ForecasterTestSuite : public CppUnit::TestFixture {
00007   CPPUNIT_TEST_SUITE (ForecasterTestSuite);
00008   CPPUNIT_TEST (testQForecaster);
00009   CPPUNIT_TEST_SUITE_END ();
00010 public:
00011
00013   void testQForecaster();
00014
00016   ForecasterTestSuite ();
00017
00018 protected:
00019   std::stringstream _describeKey;
00020 };
00021
00022 CPPUNIT_TEST_SUITE_REGISTRATION (
      ForecasterTestSuite);
```

26.143 test/rmol/OptimiseTestSuite.cpp File Reference

26.144 OptimiseTestSuite.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 OptimiseTestSuite
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 // RMOL
00023 #include <rmol/basic/BasConst_General.hpp>
00024 #include <rmol/RMOL_Service.hpp>
00025 #include <rmol/config/rmol-paths.hpp>
00026
00027 namespace boost_utf = boost::unit_test;
00028
00029 // (Boost) Unit Test XML Report
00030 std::ofstream utfReportStream ("OptimiseTestSuite_utfresults.xml");
00031
00035 struct UnitTestConfig {
00037   UnitTestConfig() {
00038     boost_utf::unit_test_log.set_stream (utfReportStream);
00039     boost_utf::unit_test_log.set_format (boost_utf::XML);
00040     boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
00041     //boost_utf::unit_test_log.set_threshold_level (boost_utf::log_successful_tests);
00042   }
00043
00045   ~UnitTestConfig() {
00046   }
00047 };
00048
00049
00050 // ////////////////////////////////////////////////////////////////////
00051 int testOptimiseHelper (const unsigned short optimisationMethodFlag,
00052                         const bool isBuiltin) {
00053
00054   // Return value
00055   int oExpectedBookingLimit = 0;
00056
00057   // Output log File
00058   std::ostringstream oStr;
00059   oStr << "OptimiseTestSuite_" << optimisationMethodFlag << "_" << isBuiltin << ".log";
00060   const stdair::Filename_T lLogFilename (oStr.str());
00061
00062   // Number of random draws to be generated (best if greater than 100)
00063   const int K = RMOL::DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION
      ;
00064
00065   // Methods of optimisation (0 = Monte-Carlo, 1 = Dynamic Programming,
00066   // 2 = EMSR, 3 = EMSR-a, 4 = EMSR-b, 5 = EMSR-a with sellup prob.)
00067   const unsigned short METHOD_FLAG = optimisationMethodFlag;
00068
00069   // Cabin Capacity (it must be greater then 100 here)
00070   const double cabinCapacity = 100.0;
00071
00072   // Set the log parameters
00073   std::ofstream logOutputFile;
00074   // Open and clean the log outputfile
00075   logOutputFile.open (lLogFilename.c_str());
00076   logOutputFile.clear();
00077
00078   // Initialise the RMOL service
00079   const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);
00080   RMOL::RMOL_Service rmolService (lLogParams);
00081
00082   // Check wether or not a (CSV) input file should be read
00083   if (isBuiltin == true) {
00084
00085     // Build the default sample BOM tree and build a dummy BOM tree.
00086     rmolService.buildSampleBom();
00087
00088   } else {
00089
00090     // Parse the optimisation data and build a dummy BOM tree
00091     const stdair::Filename_T lRMInputFileName (STDAIR_SAMPLE_DIR "/rm02.csv");
00092     rmolService.parseAndLoad (cabinCapacity, lRMInputFileName);
00093   }
00094
00095   switch (METHOD_FLAG) {
00096   case 0: {
00097     // DEBUG
00098     STDAIR_LOG_DEBUG ("Optimisation by Monte-Carlo (MC)");
00099
00100     // Calculate the optimal protections by the Monte Carlo
00101     // Integration approach
00102     rmolService.optimalOptimisationByMCIntegration (K);
00103     break;
00104   }
00105
00106   case 1: {
00107     // DEBUG
00108     STDAIR_LOG_DEBUG ("Optimisation by Dynamic Programming (DP)");
00109
```

```
00110     // Calculate the optimal protections by DP.
00111     rmolService.optimalOptimisationByDP ();
00112     break;
00113   }
00114
00115   case 2: {
00116     // DEBUG
00117     STDAIR_LOG_DEBUG ("Calculate the Bid-Price Vectors (BPV) by EMSR");
00118
00119     // Calculate the Bid-Price Vector by EMSR
00120     rmolService.heuristicOptimisationByEmsr ();
00121     break;
00122   }
00123
00124   case 3: {
00125     // DEBUG
00126     STDAIR_LOG_DEBUG ("Calculate the Authorisation Levels (AUs) by EMSRa");
00127
00128     // Calculate the protections by EMSR-a
00129     // Test the EMSR-a algorithm implementation
00130     rmolService.heuristicOptimisationByEmsrA ();
00131
00132     // Return a cumulated booking limit value to test
00133     // oExpectedBookingLimit = static_cast<int> (lBookingLimitVector.at(2));
00134     break;
00135   }
00136
00137   case 4: {
00138     // DEBUG
00139     STDAIR_LOG_DEBUG ("Calculate the Authorisation Levels (AUs) by EMSRb");
00140
00141     // Calculate the protections by EMSR-b
00142     rmolService.heuristicOptimisationByEmsrB ();
00143     break;
00144   }
00145
00146   default: rmolService.optimalOptimisationByMCIntegration (K);
00147   }
00148
00149   // Close the log file
00150   logOutputFile.close();
00151
00152   return oExpectedBookingLimit;
00153 }
00154
00155
00156 // ///////////////// Main: Unit Test Suite //////////////
00157
00158 // Set the UTF configuration (re-direct the output to a specific file)
00159 BOOST_GLOBAL_FIXTURE (UnitTestConfig);
00160
00161 // ////////////////////////////////////////////////////////////////////
00162 // Tests are based on the following input values
00163 // price; mean; standard deviation;
00164 // 1050; 17.3; 5.8;
00165 // 567; 45.1; 15.0;
00166 // 534; 39.6; 13.2;
00167 // 520; 34.0; 11.3;
00168 // ////////////////////////////////////////////////////////////////////
00169
00174 BOOST_AUTO_TEST_SUITE (master_test_suite)
00175
00176
00179 BOOST_AUTO_TEST_CASE (rmol_optimisation_monte_carlo) {
00180
00181   // State whether the BOM tree should be built-in or parsed from an input file
00182   const bool isBuiltin = false;
00183
00184   BOOST_CHECK_NO_THROW (testOptimiseHelper(0, isBuiltin););
00185 }
00186
00190 BOOST_AUTO_TEST_CASE (rmol_optimisation_dynamic_programming) {
00191
00192   // State whether the BOM tree should be built-in or parsed from an input file
00193   const bool isBuiltin = false;
00194
00195   BOOST_CHECK_NO_THROW (testOptimiseHelper(1, isBuiltin););
00196 }
00197
00202 BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_bpv) {
00203
00204   // State whether the BOM tree should be built-in or parsed from an input file
00205   const bool isBuiltin = false;
00206
00207   BOOST_CHECK_NO_THROW (testOptimiseHelper(2, isBuiltin););
00208 }
00209
```

```
00214 BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_a) {
00215
00216   // State whether the BOM tree should be built-in or parsed from an input file
00217   const bool isBuiltin = false;
00218
00219   BOOST_CHECK_NO_THROW (testOptimiseHelper(3, isBuiltin););
00220   // const int lBookingLimit = testOptimiseHelper(3);
00221   // const int lExpectedBookingLimit = 61;
00222   // BOOST_CHECK_EQUAL (lBookingLimit, lExpectedBookingLimit);
00223   // BOOST_CHECK_MESSAGE (lBookingLimit == lExpectedBookingLimit,
00224   //                      "The booking limit is " << lBookingLimit
00225   //                      << ", but it is expected to be "
00226   //                      << lExpectedBookingLimit);
00227 }
00228
00233 BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_b) {
00234
00235   // State whether the BOM tree should be built-in or parsed from an input file
00236   const bool isBuiltin = false;
00237
00238   BOOST_CHECK_NO_THROW (testOptimiseHelper(4, isBuiltin););
00239 }
00240
00244 BOOST_AUTO_TEST_CASE (rmol_optimisation_monte_carlo_built_in) {
00245
00246   // State whether the BOM tree should be built-in or parsed from an input file
00247   const bool isBuiltin = true;
00248
00249   BOOST_CHECK_NO_THROW (testOptimiseHelper(0, isBuiltin););
00250 }
00251
00255 BOOST_AUTO_TEST_CASE (rmol_optimisation_dynamic_programming_built_in) {
00256
00257   // State whether the BOM tree should be built-in or parsed from an input file
00258   const bool isBuiltin = true;
00259
00260   BOOST_CHECK_NO_THROW (testOptimiseHelper(1, isBuiltin););
00261 }
00262
00267 BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_bpv_built_in) {
00268
00269   // State whether the BOM tree should be built-in or parsed from an input file
00270   const bool isBuiltin = true;
00271
00272   BOOST_CHECK_NO_THROW (testOptimiseHelper(2, isBuiltin););
00273 }
00274
00279 BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_a_built_in) {
00280
00281   // State whether the BOM tree should be built-in or parsed from an input file
00282   const bool isBuiltin = true;
00283
00284   BOOST_CHECK_NO_THROW (testOptimiseHelper(3, isBuiltin););
00285 }
00286
00291 BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_b_built_in) {
00292
00293   // State whether the BOM tree should be built-in or parsed from an input file
00294   const bool isBuiltin = true;
00295
00296   BOOST_CHECK_NO_THROW (testOptimiseHelper(4, isBuiltin););
00297 }
00298
00299 // End the test suite
00300 BOOST_AUTO_TEST_SUITE_END()
00301
00302
```

26.145 test/rmol/OptimiseTestSuite.hpp File Reference

```
#include <sstream>
#include <cppunit/extensions/HelperMacros.h>
```

Classes

- class OptimiseTestSuite

Functions

- CPPUNIT_TEST_SUITE_REGISTRATION (OptimiseTestSuite)

26.145.1 Function Documentation

26.145.1.1 CPPUNIT_TEST_SUITE_REGISTRATION (OptimiseTestSuite)

26.146 OptimiseTestSuite.hpp

```
00001 // STL
00002 #include <sstream>
00003 // CPPUNIT
00004 #include <cppunit/extensions/HelperMacros.h>
00005
00006 class OptimiseTestSuite : public CppUnit::TestFixture {
00007   CPPUNIT_TEST_SUITE (OptimiseTestSuite);
00008   CPPUNIT_TEST (testOptimiseMC);
00009   CPPUNIT_TEST (testOptimiseDP);
00010   CPPUNIT_TEST (testOptimiseEMSR);
00011   CPPUNIT_TEST (testOptimiseEMSRa);
00012   CPPUNIT_TEST (testOptimiseEMSRb);
00013   CPPUNIT_TEST (testOptimiseEMSRaWithSU);
00014   //  CPPUNIT_TEST (errorCase);
00015   CPPUNIT_TEST_SUITE_END ();
00016 public:
00017
00019   void testOptimiseMC();
00020
00022   void testOptimiseDP();
00023
00026   void testOptimiseEMSR();
00027
00030   void testOptimiseEMSRa();
00031
00034   void testOptimiseEMSRb();
00035
00037   //  void errorCase ();
00038
00040   OptimiseTestSuite ();
00041
00042 protected:
00043   std::stringstream _describeKey;
00044 };
00045
00046 CPPUNIT_TEST_SUITE_REGISTRATION (
      OptimiseTestSuite);
```

26.147 test/rmol/UnconstrainerTestSuite.cpp File Reference

26.148 UnconstrainerTestSuite.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 UnconstrainerTestSuite
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 // RMOL
00023 #include <rmol/RMOL_Service.hpp>
00024
00025 namespace boost_utf = boost::unit_test;
00026
00027 // (Boost) Unit Test XML Report
00028 std::ofstream utfReportStream ("UnconstrainerTestSuite_utfresults.xml");
```

```
00029
00033 struct UnitTestConfig {
00035   UnitTestConfig() {
00036     boost_utf::unit_test_log.set_stream (utfReportStream);
00037     boost_utf::unit_test_log.set_format (boost_utf::XML);
00038     boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
00039     //boost_utf::unit_test_log.set_threshold_level (boost_utf::log_successful_tests);
00040   }
00041
00043   ~UnitTestConfig() {
00044   }
00045 };
00046
00047
00048 // ///////////////// Main: Unit Test Suite //////////////
00049
00050 // Set the UTF configuration (re-direct the output to a specific file)
00051 BOOST_GLOBAL_FIXTURE (UnitTestConfig);
00052
00057 BOOST_AUTO_TEST_SUITE (master_test_suite)
00058
00059
00062 BOOST_AUTO_TEST_CASE (rmol_unconstraining_em) {
00063   const bool lTestFlag = true;// testUnconstrainerHelper(0);
00064   BOOST_CHECK_EQUAL (lTestFlag, true);
00065   BOOST_CHECK_MESSAGE (lTestFlag == true,
00066                        "The test has failed. Please see the log file for "
00067                        << "more details");
00068 }
00069
00070 // End the test suite
00071 BOOST_AUTO_TEST_SUITE_END()
00072
00073
```

26.149 test/rmol/UnconstrainerTestSuite.hpp File Reference

```
#include <sstream>
#include <cppunit/extensions/HelperMacros.h>
```

Classes

- class UnconstrainerTestSuite

Functions

- CPPUNIT_TEST_SUITE_REGISTRATION (UnconstrainerTestSuite)

26.149.1 Function Documentation

26.149.1.1 CPPUNIT_TEST_SUITE_REGISTRATION (UnconstrainerTestSuite)

26.150 UnconstrainerTestSuite.hpp

```
00001 // STL
00002 #include <sstream>
00003 // CPPUNIT
00004 #include <cppunit/extensions/HelperMacros.h>
00005
00006 class UnconstrainerTestSuite : public CppUnit::TestFixture {
00007   CPPUNIT_TEST_SUITE (UnconstrainerTestSuite);
00008   CPPUNIT_TEST (testUnconstrainingByEM);
00009   CPPUNIT_TEST_SUITE_END ();
00010 public:
00011
00013   void testUnconstrainingByEM();
00014
00016   UnconstrainerTestSuite ();
00017
00018 protected:
00019   std::stringstream _describeKey;
```

```
00020 };
00021
00022 CPPUNIT_TEST_SUITE_REGISTRATION (
      UnconstrainerTestSuite);
```