Drools5 Integration
Helper 1.3.0 user guide

Mathieu Boretti <mathieu.boretti@gmail.com>

Drools5 Integration Helper 1.3.0 user guide

Mathieu Boretti <mathieu.boretti@gmail.com>

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

Presentation

This user guide defines what is Drools5-integration-helper and how to use it.

More information regarding the project can be found at http://www.javaforge.com/project/DroolsHelp. Also, javadoc and maven site are part of the various downloadable files.

Drools5-Integration-Helper is available under the GPL-3 license.

License

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, 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 them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without

permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any

non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium

customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of

that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a)

provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered

work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If 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 convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU 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 Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

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

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM 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 PROGRAM (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 PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state 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 program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

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

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

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

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.
```

Drools5 Integration Helper

Drools5 Integration Helper is a set of java classes, maven plugin and maven archetype to deal with Drools5 (and specifically with Drools Expert [http://www.jboss.org/drools/drools-expert.html] and Drools Guvnor [http://www.jboss.org/drools/drools-guvnor.html]).

This project provide three main artifacts:

- One java library to provide various integration to interface from java to drools.

    - Support of drools call thru interface in stateless mode.
    - Support of drools call thru interface in statefull mode.
    - Support of drools interface integration into spring [http://www.springsource.org/documentation].
    - Support of drools annotation on classes.
        - Support of field updated based on annotations.
        - Support of usage of condition on method.
- One Maven [http://maven.apache.org/] plugin to validate and compile drools files and support additional annotations..
    - Support of validation of drools source file.
    - Support of compilation of drools source file.
    - Support of instrumentalization of class file.
- One Maven [http://maven.apache.org/] archetype to produce ready to use Maven 2 project.
    - Support generation of a Maven 2 project with both classes and interfaces approach.
    - Support generation of a Maven 2 project that use JUnit 4 [http://www.junit.org/] tests as example.

Drools5 Integration Helper is provided as ready to use Maven artifact. No IDE integration is provided as no IDE integration is needed for **Drools5 Integration Helper** itself. IDE integration regarding Drools can be provided by the IDE related to drools.

Drools5 Integration Helper require Java 6.

This library uses log4j library [http://logging.apache.org/log4j/1.2/index.html] to log and provides access to log4j from drools rules.

The main idea behind **Drools5 Integration Helper** is to provide tools that allow access to the power of rule engine, without having to care about how to interface with this type of engine. To do so, **Drools5 Integration Helper** focus on providing a way to provide this interface by only say "here I like to be interfaced with drools" and not "here I like to do so and so to access to drools". So, almost all usage of **Drools5 Integration Helper** are to be done in declarative way (using annotation for example).

Installation

The various artifacts of Drools5 Integration helper are integrated into the maven central repository, starting from version 1.3.0. No specific configuration is needed to use them.

Note that potentially some dependencies may not be available in central. In this case, the jboss repository [http://repository.jboss.org/maven2/] may be add to your list of repositories.

It is possible to download them manual and install them.

This chapter describe how to download and then install all the required file into maven local or remote repository. For readers with good Maven skill, this chapter is potentially useless. Also it is not strictly required to use this installation procedure, if you can have access to the central repository.

All artifacts have **org.boretti.drools.integration** as groupId, except for the examples that have groupId **org.boretti.drools.integration.examples**. Examples can't be get from the Maven central repo. There are available from the SVN and in the documents part of the site.

Download

All artifacts can be downloaded from http://www.javaforge.com/proj/doc.do?doc_id=73466.

All in one download

All artifacts are available in a single zip file.

Per files download

It is possible to download separately all file.

Installation in local maven repository

All maven artifacts can be install in the local repository. The various file are named like 'artifactId'-'version'.'extension' or 'artifactId'-'version'-'classifier'.'extension'

The plugin maven-install-plugin can be used to install each artifact into your local repository:

```
mvn install:install-file  -Dfile=path-to-your-artifact-file \
                          -DgroupId=org.boretti.drools.integration \
                          -DartifactId=your-artifactId \
                          -Dversion=version \
                          -Dpackaging=extension
```

or

```
mvn install:install-file  -Dfile=path-to-your-artifact-file \
                          -DgroupId=org.boretti.drools.integration \
                          -DartifactId=your-artifactId \
                          -Dversion=version \
                          -Dclassifier=classifier
                          -Dpackaging=extension
```

Installation in remote maven repository

All maven artifacts can be deploy in a remote maven repository. The various file are named like 'artifactId'-'version'.'extension' or 'artifactId'-'version'-'classifier'.'extension'

The plugin maven-deploy-plugin can be used to install each artifact into a remote repository:

```
mvn deploy:deploy-file -Durl=file://C:\m2-repo \
                       -DrepositoryId=some.id \
                       -Dfile=your-artifact \
                       [-DgroupId=org.boretti.drools.integration] \
                       [-DartifactId=your-artifact] \
                       [-Dversion=version] \
                       [-Dpackaging=extension] \
                       [-Dclassifier=classifier] \
```

Chapter 2. Java Library - Drools5 Integration Helper Library

Introduction

The artifact **org.boretti.drools.integration**:**drools5-integration-helper-library** provides the following support:

- A set of annotations and enumeration to make an interface as **drools interface**.
- A set of classes that provide a way to instantiate these interfaces, directly or from Spring.
- A set of annotations and enumeration to make a class as **drools class**.

All the Classes, Annotations, Enumerations and Interfaces are part of the following package (or sub-packages) : org.boretti.drools.integration.drools5. The sub-package annotations contains all the annotations that can be used. The sub-package exceptions contains all the exceptions that can be thrown by this library. Finally the sub-package implementation contains all the stuff that provide the core implementation for this library. Classes and interfaces of this sub-package must not be used as they are reserved for internal implementation of the library and the definition of these classes, interfaces and methods may change without notice.

This chapter describe how to use this library. The javadoc is available as part of the downloadable files.

This library can be add as a maven dependency by using:

```
<dependency>
   <groupId>org.boretti.drools.integration</groupId>
   <artifactId>drools5-integration-helper-library</artifactId>
   <version>1.3.0</version>
</dependency>
```

This dependency will indirectly reference the various drools dependencies.

Two various approaches are provided to interfaces with Drools. They both are complementary way of defining this interface.

- The first one is based on usage of an annotated interface. In this case, we can say it is a service oriented way of thinking. The drools file is exposed as classical java method, by an interface.
- The second one is based on usage of an annotated class. This can be done in two different ways.
  - By using annotations on field. In this case it is possible to have field auto-computed by drools.
  - By using annotations in method. In this case, these annotations can define pre and post-condition that will be evaluated by using drools.

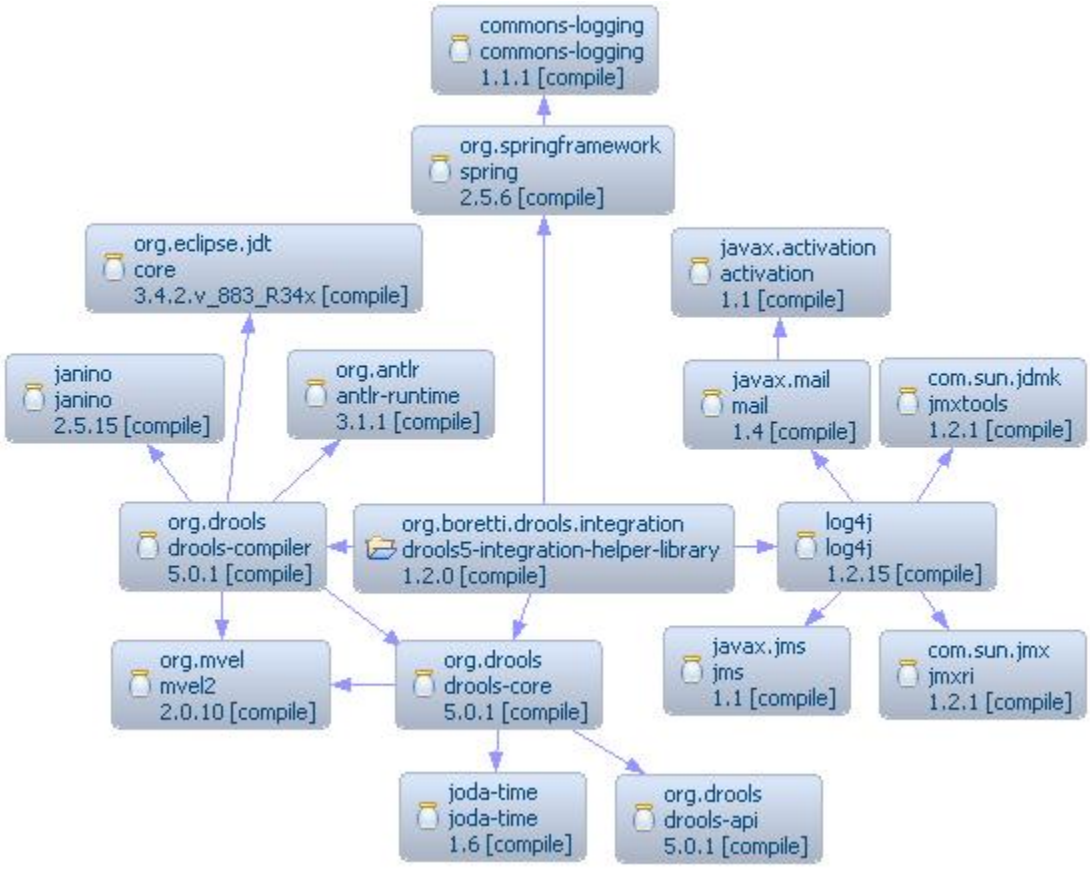
  Theses both ways will require class instrumentalization.

These both approaches cover different types of use case and can be use in the same java program without problem.

Drools resources can be get from the resource available by using the class loader, but also from an URL. These URLs can point to drools resources exposed by Drools Guvnor. Drools Guvnor usage is out of scope of this document.

Dependencies

Figure 2.1. View of the runtime dependencies



There are four main dependencies for the library:

log4j	This is the classical *log4j* library.
spring	This is a dependency to the *spring* library to support the spring integration.
drools-core	This is a dependency to the *drools-core* library. It is the main dependency for drools.
drools-compiler	This is a dependency to the *drools-compiler* library. This dependency provides the feature to compile Drools file.

DroolsInterface

Interface(s)

DroolsInterface is the first main concept behind this library. The goal of a DroolsInterface is to provides a way to call Drools rules from java, without having to care about the Drools interfacing.

To achieve this goal, the developer can define an *interface* and annotate it with the annotation @DroolsService:

```
package foo;

import org.boretti.drools.integration.drools5.annotations.DroolsService;

@DroolsService
public interface MyName {
        /*... */
}
```

This annotation make this interface as a DroolsInterface. This annotation support two attributes:

type — can be SOURCE or COMPILED or XML.

- *SOURCE* is to be used to define that this class will be based on a drools source file.
- *COMPILED* (default) is to be used to define that this class will be based on a drools compiled file (using the maven plugin).
- *XML* is to be used to define that this interface will be based on a drools XML file.

resourceName — is to be used to define the resource name to be used to load the rules. If this is unset or null or empty, the default resource name used will be the interface with .drl extension for source and .cdrl for compiled.
The resourceName must point a valid resource. Resource are found using the classic getResourceAsStream from the Class object that is this interface or by reading an url (http://XXX or file://XXX).

Both type and resourceName can be override at execution, by passing others value with creating the concrete instance of the interface.

Method(s)

Developer must of course add several method definition to this interface. Every method will be use to call the rules that are behind this interface, except for the special case defined later in this document. The parameter of the method will be pass (depending of the annotation @DroolsParameter) to the rule engine. if the method define a return type other that void, the return value will be the first fact that match this type or all if the type is an array.

By default, methods are execution method. A special case is for method that are setter and that have an annotation @DroolsVariable. These methods can be used to inject additional object to the drools context.

@DroolsVariable and @DroolsMethod are incompatible and can't be used on the same method. This constraint is not checked at compile time, but only at execution time.

Session

Optionally, method can be annotated with the annotation @DroolsMethod. This annotation supports several attributes, one is session. This attribute can have the following value:

NONE — (default also when both @DroolsMethod and @DroolsVariable are not used). In this case, the execution of the drools is done in purely stateless mode.

In case a session exists at time this method is call, the session is not impacted by this call. This type of call are fully thread safe.

NEW — In this case, the execution will first discard any existing drools session, and then create a new one. Then a fireAll will be trigger.
The generated concrete class doesn't provide in security regarding threading. It is the responsability of the caller of the concrete interface.

END — In this case, a session must existing before. A fireAll will be trigger and then the session will be discard. In case no session exists, an IllegalArgumentException exception is thrown.
The generated concrete class doesn't provide in security regarding threading. It is the responsability of the caller of the concrete interface. In case no session exists at this time, an error is raised.

JOIN — In this case, if a session exists, this session will be used ; if not, the execution will be done in stateless mode.
The generated concrete class doesn't provide in security regarding threading. It is the responsability of the caller of the concrete interface.

REQUIRE — In this case, a session must exists before execution. In case no session exists, an IllegalArgumentException exception is thrown.

The generated concrete class doesn't provide any security regarding threading. It is the responsability of the caller of the concrete interface. In case no session exists at this time when a session is required, an error is raised.

Activation Group

Activation Group is a new attribute of the @DroolsMethod annotation. Currently this attribute is not used.

Setter Methods

The setter method must be named setXXX, must have exactly one parameter and a void return type. Also, the annotation @DroolsVariable is mandatory. This annotation support one attribute, global. By default this attribute is false. The logic of passing the field to drools is the following:

global=false — In this case, the variable is stored internally in the interface when the setter is called. When called with null, the variable is removed. At time a new stateless session or a new statefull session is created, this variable is added to the fact.

global=true — In this case, the variable is stored internally in the interface when the setter is called. When called with null, the variable is removed. At time a new stateless session or a new statefull session is created, this variable is added to the global list. The field name is used as the name for the global variable in drools.
Drools may produce error in case a global is defined on java side and not on drools side.

Calling a setter method when a session is already existing doesn't impact the existing session. The change of field will only be applyed for the new session.

Asynchronous methods

A method (not a setter) can be asynchronous. To do so, it is required to use a Future as a return value. The call will then be asynchronous (executed in a separated Thread) and the Future will be available to have access to the response.

Parameter(s)

Parameters of the various methods may be annotated with the annotation @DroolsParameter. This annotation support one attribute: handling. This attribute can have the following value:

NONE	(default, also when annotation is not used). In this case, the parameter is simply pass to the drools.
IGNORED	In this case, the parameter is ignored and not pass to drools.
TOFLAT1	In this case, if the parameter is an array or a Collection, all items of the array or the Collection will be pass to drools and not the parameter itself.
TOFLATALL	In this case, if the parameter is an array or a Collection, all items of the array or the Collection will be pass to drools and not the parameter itself, and recursively for all items.
GENERICDEFINITION	In this case, it is assumed the parameter is a Class parameter to be used for the return type. The goal is to support generic.

```
public <T> T myMethod(
   @DroolsParameter(handling=GENERICDEFINITION)
   Class<T> clazz,
   /*...*/
);
```

This will ensure the method returns the right type of object at runtime. This parameter is not pass to drools. It is only used to support generic.

Drools5-Integration-Helper supports null parameters, but not for a GENERICDEFINITION parameter.

Return value

Methods that are not setter can have a return type. If the return type is void, nothing is returned. If the returned value is an array, then all objects in drools that match this type (that are the same class or a sub class) are returned. It is also possible to use a Future as a return value. In this case, the execution of the method is asynchronous.

In case the return type is not an array and several object match the expected type, only the first one found will be returned. As the order of fact is not determined, in this case, there is no way to know which object will be returned.

In case the return type is an array, the order of the object in this array can be different between two consecutive calls.

Generic

Return type can be generic, but in this case the GENERICDEFINITION logic must be used to inform drools5-Integration-Helper about the real type. In any case, before using generic type in rule, it is better to make an analysis before using generic in drools, as the result can be very difficult to predict.

Future

Using a return type of Future (java.util.concurrent.Future) marks the method as asynchronous method. For readability it is better to add a specific text at the end of the name of the method, for example like Async.

For the Future usage, there are two use cases:

```
public Future<String> myMethod(/*...*/);
```

In this case, the generic part of the Future is clearly defined. Drools5-Integration-Helper will return a Future that will return a String from drools.

```
public <T> Future<T> myMethod(
   @DroolsParameter(handling=GENERICDEFINITION) Class<T> clazz,
   /*...*/);
```

In this case, the generic part of the Future is defined by the clazz parameter.

Primitive types

A special note on primitive type is required. For example, if we have the following interface:

```
@DroolsService(type=SOURCE,resourceName="/drools.drl")
private interface PrimitifInterface {
        public boolean testBoolean1(boolean in);

        public Boolean testBoolean2(boolean in);

        public boolean testBoolean3(Boolean in);

        public Boolean testBoolean4(Boolean in);
}
```

In this case, if the rule file do nothing, what is asserted as parameter will be used for the return value. Even if both Boolean and boolean are used, both Boolean and boolean are accepted as matching for each others.

DroolsClass

Class(s)

Any class can be transform into automated drools class.

To achieve this goal, the developer can define a *class* and annotate it with the annotation @DroolsService:

```
package foo;

import org.boretti.drools.integration.drools5.annotations.DroolsService;

@DroolsService
public class MyName {
        /*... */
}
```

This annotation make this class as a DroolsClass. This annotation support two attributes:

type can be SOURCE or COMPILED or XML.

- *SOURCE* is to be used to define that this class will be based on a drools source file.

- *COMPILED* (default) is to be used to define that this class will be based on a drools compiled file (using the maven plugin).
- *XML* is to be used to define that this interface will be based on a drools XML file.

resourceName — is to be used to define the resource name to be used to load the rules. If this is unset or null or empty, the default resource name used will be the class with .drl extension for source and .cdrl for compiled.
The resourceName must point a valid resource. Resource are found using the classic getResourceAsStream from the Class object that is this interface or by reading an URL (http://XXX or file://XXX).

The class must be instrumented using the maven plugin. Missing this step will produce classes that are not integrating the drools interfacing.

Both resourceName and type can be override by use special annotated parameter in the constructor.

Field(s)

The class will contains several fields. The field can have tree states:

@DroolsGenerated — With this annotation, the field is marked as generated thru drools. In this case, the Maven plugin will add a call to the drools engine in the getter of this field, when needed. The call will pass one single object to drools : this. It is the responsability of the drools rule to set the field.

@DroolsIgnored — With this annotation, the field is marked as not generated thru drools and that setting this field will not trigger a rerun of the drools rule.

Not annotated — Without annotation, the field is marked as not generated thru drools but that changing this field must trigger drools execution. In this case, the Maven plugin will add a piece of code to mark the execution of drools as required.

```
package foo;

import org.boretti.drools.integration.drools5.annotations.DroolsService;
import org.boretti.drools.integration.drools5.annotations.DroolsIgnored;
import org.boretti.drools.integration.drools5.annotations.DroolsGenerated;

@DroolsService
public class MyName {
        @DroolsIgnored
        private String ignoredField;

        private String sourceField;

        @DroolsGenerated
        private String generatedField;

        /* ... put here the various getter and setter */
}
```

Only access to the field thru the getter and setter are instrumented. Direct access to the field will not trigger or use recomputation.

Constructor(s)

By default, all constructors are instrumented at compile time to have additional code to initialize the rule base. The parameter of the constructor can use two special annotations:

@DroolsResourceName	Must be use zero or one time in the constructor and must be use on a String parameter. It marks the parameter as an injection of the resourceName.
@DroolsResourceType	Must be use zero or one time in the constructor and must be use on a DroolsServiceType parameter. It marks the parameter as an injection of the serviceType.

When at least one of these two annotations are used, the resourceName and/or resourceType will be override by using the received value. Only overriding at construction time is supported.

Wrong usage of this annotations (for instance @DroolsResourceName on a Boolean parameter) will be reported as a warning in the logs of the maven plugin execution.

It is not required to have a default constructor or a constructor without any override parameter.

Multiple constructors usage.

A class can have several constructors. All constructors will be instrumented by the maven plugin. But what happen if a constructor use an other one (classical approach) ?

Two different cases are interesting to be study. The first one is with constructor that doesn't override the drools resource(s). and the second one with override of the drools location.

Multiple constructors without Drools override

Let's take an example:

```
public constructor() {
   /* ... */
}

public constructor(String arg) {
   this();
   /* ... */
}

public constructor(String arg0,boolean arg1) {
   this(arg);
   /* ... */
}
```

We have several constructors, with zero, one, or two parameters. The zero-parameter constructor use the implicit call to super(). All others constructors first call the constructor with parameter-1 parameters. None of the various constructors use resource override.

In this case, all constructor will have additional code to configure the drools Rule. This code will be added just after the call to the parent constructor (for the first method) or just after the call to the other constructor (for the two last methods). The side effect is that the drools Rule object will be constructed several time, event if only the one from the most outside constructor will be used.

Multiple constructors with Drools override.

Let's take an example:

```
public constructor() {
   /* ... */
}

public constructor(@DroolsResourceName String resourceName) {
   this();
}
```

In this case, we have one default constructor and one additional constructor that just call the default constructor, but use a resource override.

In this case, all constructor will have additional code to configure the drools Rule. This code will be added just after the call to the parent constructor (for the first method) or just after the call to the other constructor (for the last method). The side effect is that the drools Rule object will be constructed several time, event if only the one from the most outside constructor will be used. It will ensure that the used Rule object will be the one with the override parameter. But, in all this, with this approach, the Rule object with default value will be constructed.

The fact that with this approach the object with default will be constructed in any case can be a potential issue, for example when the resource with default value is not available, because this will make fail the inner constructor and indirectly the outer constructor, before construction of the Rule object with overrided value.

How to avoid the unnecessary construction of the rule data

As we can see in the previous section, depending on how the constructor are used, we may have several construction of the rules object, when only one is required. It can have performance issues and also unwanted errors.

It exists a workaround to avoid (or limit) this issue. To do so, the call to this must be removed and a internal private initialization method is defined. This method is call and not the other constructor:

```
public constructor(/*...*/) {
   init(/*...*/);
}

public constructor(@DroolsResourceName String resourceName) {
   init(/*...*/);
}

private final void init(/*...*/) {
   /*...*/
}
```

Drools Execution

Drools is called :

- The first time a field annotated with @DroolsGenerated is getted.
- Every time a field annotated with @DroolsGenerated is getted, just after a none annotated field is set.

All executions are done in purely stateless mode. Drools5-integration-helper doesn't care about multi-threading access. Developer must use synchronized getter and setter if needed.

DroolsCondition

Idea

In some cases it can be interesting to define conditions related to a method. This is an approach similar to the Programming by Contract model. The goal is to annotated a method to define pre-condition and post-condition.

The class must be instrumented using the maven plugin. Missing this step will produce classes that are not integrating the drools interfacing.

Condition definition

PreCondition

PreCondition can be defined by using the @DroolsPreCondition annotation. This annotation support several attributes:

type — can be SOURCE or COMPILED or XML.

- *SOURCE* is to be used to define that this class will be based on a drools source file.
- *COMPILED* (default) is to be used to define that this class will be based on a drools compiled file (using the maven plugin).
- *XML* is to be used to define that this interface will be based on a drools XML file.

resourceName — is to be used to define the resource name to be used to load the rules.
The resourceName must point a valid resource. Resource are found using the classic getResourceAsStream from the Class object that is this interface or by reading an url (http://XXX or file://XXX).

error — is to be used to define the error that can be throw in case of error. This must be a Class object that extends Error.

```
Class<? extends Error> error();
```

Methods with this annotations will contains, at the start of the method, an evaluation of the drools rules defined in the annotation. this and all parameters of the method will be passed as fact. The rule must insert an concrete instance of an Error (based on the attribute error) in case the precondition are not meet.

PostCondition

PostCondition can be defined by using the @DroolsPostCondition annotation. This annotation support several attributes:

type — can be SOURCE or COMPILED or XML.

- *SOURCE* is to be used to define that this class will be based on a drools source file.
- *COMPILED* (default) is to be used to define that this class will be based on a drools compiled file (using the maven plugin).

- *XML* is to be used to define that this interface will be based on a drools XML file.

resourceName — is to be used to define the resource name to be used to load the rules.
The resourceName must point a valid resource. Resource are found using the classic getResourceAsStream from the Class object that is this interface or by reading an url (http://XXX or file://XXX).

error — is to be used to define the error that can be throw in case of error. This must be a Class object that extends Error.

```
Class<? extends Error> error();
```

onException — is to be used to mark that throw instruction must be considered as to be marked for post condition check. Default is false

Methods with this annotations will contains, before each (X)return, an evaluation of the drools rules defined in the annotation. this and all parameters of the method will be passed as fact. The rule must insert an concrete instance of an Error (based on the attribute error) in case the postcondition are not meet.

If onException is set to true, all athrow (throw instruction at java level) are also preceded by the postcondition.

In the current implementation, post condition can only have access to the state of the argument and this at time of the evaluation. No access to old value (before execution of the method) are not available.

Exception throwing

In case error is inserted into the drools fact, the error will be thrown by the system. As it is required to extends Error, it is not required to explicitly use throws feature on the method.

Static methods

Static methods are not supported.

DroolsProvider

Once the developer has define the drools interface, a way to create concrete instance is required.

Only **Drools Interface** requires direct usage of the DroolsProvider. **Drools Class** doesn't need special instantiation but these classes will only internally the DroolsProvider.

This can be achieve using the DroolsProvider class. This class provides several methods.

Getting concrete instance from interface

```
public <T> T getService(Class<T> clazz)
```

or

```
public <T> T getService(Class<T> clazz,
                        String resourceName)
```

or

```
public <T> T getService(Class<T> clazz,
```

```
                                String resourceName,
                                DroolsServiceType serviceType)
```

It is possible to pass the interface to this method to create a new concrete instance. This method return a new interface for all usage.

In case a Class, a not annotated (or wrongly annotated) Interface is used it will generate an error.

All concrete instance of the interface returned by this method will implements T and also the interface DroolsInterface. When casting explicitly to DroolsInterface this concrete class, it is possible to get manually the current session (for statefull execution) or the RuleBase related to this interface.

The two additional parameters can be used to override the resourceName and serviceType that are defined at interface level.

Getting RuleBase from interface or class

```
public <T> RuleBase getRuleBase(Class<T> clazz)
```

or

```
public <T> RuleBase getRuleBaseOverride(Class<T> clazz,
                                        String resourceName)
```

or

```
public <T> RuleBase getRuleBaseOverride(Class<T> clazz,
                                        DroolsServiceType resourceType)
```

or

```
public <T> RuleBase getRuleBaseOverride(Class<T> clazz,
                                        String resourceName,
                                        DroolsServiceType resourceType)
```

This method returns a new RuleBase object, based on the annotation of the received interface or class. Usage of this method is normally not needed for end-user. This method is used by the classes that have been annotated @DroolsService to get the rules.

Getting RuleBase from type and resource

```
public <T> RuleBase getRuleBase(
   Class<T> clazz,
   DroolsServiceType type,
   String resourceName)
```

This method returns a cachable RuleBase object, based on the received parameter. Usage of this method is normally not needed for end-user. This method is used by the classes that have been annotated to support pre-condition.

Running pre-condition

```
public static <T> void runPreCondition(
   Class<T> clazz,
   DroolsServiceType type,
   String resourceName,
```

```
   Class<? extends Error> error,
   Object reference,
   Object args[])
```

This method run a precondition and produce an error if needed. Usage of this method is normally not needed for end-user. This method is used by the classes that have been annotated to support pre-condition.

Running post-condition

```
public static <T> void runPostCondition(
   Class<T> clazz,
   DroolsServiceType type,
   String resourceName,
   Class<? extends Error> error,
   Object reference,
   Object args[])
```

This method run a postcondition and produce an error if needed. Usage of this method is normally not needed for end-user. This method is used by the classes that have been annotated to support post-condition.

Spring Integration

It is possible to have an integration with Spring using the DroolsBeanFactory class. To do so, the developer must:

1. Define the annotated interface. To do so, it is just required to have a standard interface that is annotated with @DroolsService.

2. Register the bean in spring bean configuration:

```
<bean
   id="Drools1"
   class="org.boretti.drools.integration.drools5.DroolsBeanFactory">
      <property
         name="droolsInterface"
         value="foo.MyPackage"
          />
</bean>
```

This will provide a Bean that is a DroolsInterface and implements the provided interface. This BeanFactory use the DroolsProvider to create the concrete instance of the interface.

In case some setter (method named setXXX, with exactly one parameter, annotated with @DroolsVariable) are also annotated with @Autowired (from Spring), the DroolsBeanFactory will try to inject into this setter a Bean with the same type that is the parameter. The required attribute is not checked.

The object created by this BeanFactory are not singleton. Spring will take care of creating singleton if needed.

Exceptions

The Drools5-Integration-Helper-Library classes can throw exception. All specific exceptions are in the package org.boretti.drools.integration.drools5.exceptions. The parent Exception is DroolsError which extends Error. These exceptions are related to the following main categories of problem:

- Invalid objects. For example, not annotated interface passed to the DroolsProvider class.
- Access to drools file error. For example, resource with the drools rule not found.
- Compilation error. For example, the source drools file is not valid.

Also, the IllegalArgumentException may be thrown for null parameter or invalid session state.

Logging

The log4j library [http://logging.apache.org/log4j/1.2/index.html] is used to provide logging functionalities for the various classes. All the logger name are based on the class name.

Some very low level informations are logged in debug mode. Errors are logged before been thrown.

The global droolsLogger is added to every working memory. This is an instance of org.apache.log4j.Logger that can be used for logging from drools. The name of this logger is based on the name class of the Interface or Class annotated with @DroolsService.

This field must be declared in drools file :

```
global org.apache.log4j.Logger droolsLogger;
```

Configuration of the log4j library is out of scope of Drools5-Integration-Helper.

Summaries

Annotations

Table 2.1. Summary of the annotations

Annotation	Scope	Description	Attributes
@DroolsGenerated	on field	Mark this field as auto-generated.	N/A
@DroolsIgnored	on field	Mark this field as not part ofdrools source field.	N/A
@DroolsMethod	on method(interface)	Mark this method as drools method.(optional usage)	session
@DroolsParameter	on parameter(interface)	Add additional information for the parameter(optional usage)	handling
@DroolsPostCondition	on method	Mark this method as using post-condition.	type, resourceName, error, onException
@DroolsPreCondition	on method(class)	Mark this method as using pre-condition.	type,resourceName,error
@DroolsResourceName	on parameter(constructor)	Mark this parameter as resource Name injector	

@DroolsResourceType	on parameter(constructor)	Mark this parameter as resource type injector	
@DroolsService	on interface	Mark this interface as Drools interface.	type,resourceName
@DroolsVariable	on method(interface)	Mark this method as variable injectionmethod	global

Enumerations

Table 2.2. Summary of the enumerations

Enumeration	Description	Values
DroolsParameterHandling	Definition of the parameter handling for the method of the interface.	GENERICDEFINITION, IGNORED, NONE, TOFLAT1, TOFLATALL
DroolsServiceType	Definition of the type of the drools source	COMPILED, SOURCE, XML
DroolsSessionType	Definition of the session handling for the method of the interface.	END, JOIN, NEW, NONE, REQUIRE

Classes

Exceptions and Errors are listed in a separated table

Table 2.3. Summary of the classes

Name	Description
DroolsBeanFactory	The Spring BeanFactory to support dynamic creation of Proxy from Spring.
DroolsProvider	The class to be used to get Proxy for any interface.

Interfaces

Table 2.4. Summary of the interface

Name	Description
DroolsInterface	This interface is automatically implemented by all Proxies generated bythe classes DroolsBeanFactory and DroolsProvider

Errors and Exceptions

- ClassCastDroolsError
- CompilationDroolsError
- DroolsError

- IncompatibleAnnotationDroolsError
- IODroolsError
- MalformedURLDroolsError
- NotAClassDroolsError
- NotAnInterfaceDroolsError
- NotAnnotatedClassDroolsError
- NotAnnotatedInterfaceDroolsError
- XMLDroolsError

Performance

Performance of drools itself is out of scope. Drools5-Integration-Helper add a small footprint to a direct usage of drools. Initialization of Proxy and classes are the most consuming operation (because of the need to load resource, parse resource, etc). So it is a very good idea to avoid recreating object if it is not needed.

Also, pre-condition and post-condition usage require passing all the arguments of the method to drools. This is an operation that can consume time.

In case log4j logging for the drools5-Integration-Helper classes are set to debug, it can have a severe impact on the performance. So, debug log level for these classes should be avoid in production environment. Depending of the type of appender used, usage of the debug level can be 5 time slower that without.

Chapter 3. Maven plugin - Drools5 Integration Helper Maven Plugin

Introduction

A maven plugin **drools5-integration-helper-maven-plugin** is provided. Plugin prefix is drools5. The plugin can be used by adding this section in the plugin section:

```
<plugin>
 <groupId>org.boretti.drools.integration</groupId>
 <artifactId>drools5-integration-helper-maven-plugin</artifactId>
 <version>1.3.0</version>
 <extension>true</extension>
</plugin>
```

The extension section is needed because of the additional packaging provided by the plugin.

The plugin is used when

- One of the provided packaging is used.
- One or several goal(s) are listed to be used.

Please refer to the site (part of the artefacts) for the standard plugin documentation. Also, the plugin exposes the help goal.

Plugin has been tested with Maven 2.0.10 and Maven 2.2.1.

Drools Packaging

This maven 2 plugin provides several packaging that extends standard packaging:

drools	This packaging is a synonym of the jar-drools packaging.
jar-drools	This packaging is the same one that the packaging jar, but in addition, the various goals of this plugin are executed during the compile, process-class, test-compile and process-test-class phase.
ejb-drools	This packaging is the same one that the packaging ejb, but in addition, the various goals of this plugin are executed during the compile, process-class, test-compile and process-test-class phase.
ejb3-drools	This packaging is the same one that the packaging ejb3, but in addition, the various goals of this plugin are executed during the compile, process-class, test-compile and process-test-class phase.
rar-drools	This packaging is the same one that the packaging rar, but in addition, the various goals of this plugin are executed during the compile, process-class, test-compile and process-test-class phase.
par-drools	This packaging is the same one that the packaging par, but in addition, the various goals of this plugin are executed during the compile, process-class, test-compile and process-test-class phase.

war-drools	This packaging is the same one that the packaging war, but in addition, the various goals of this plugin are executed during the compile, process-class, test-compile and process-test-class phase.

The packaging can be used in the following way:

```
<project
   xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                       http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>${groupId}</groupId>
        <artifactId>${artifactId}</artifactId>
        <packaging>jar-drools</packaging>
        <version>${version}</version>
```

Is is not required to use these packaging to use the maven plugin. The various goals can be called manually.

All the previous packaging add the following goals:

compile	drools-copy-validate, drools-compile.
process-classes	drools-postprocessor.
test-compile	drools-copy-validate-test, drools-compile-test.
process-test-classes	drools-postprocessor-test.

Report goal are not part of the executed goal. It must be call manually.

Validation Drools Goals

Theses two goals are related to validation of the Drools files. Drools files can be used for the main code of the artifact or for the tests. Is it why there is two goals.

drools-copy-validate

This goal is executed by default in phase compile. It work in the following way:

1. For each file with extension .drl in folder src/main/drools:
    1. Validate that the file is valid regarding drools compiler.
    2. Copy the file into target/classes. The folder hierarchy is the same in the target/classes folder.
2. For each file with extension .xml in folder src/main/drools:
    1. Validate that the file is valid regarding drools compiler.
    2. Copy the file into target/classes. The folder hierarchy is the same in the target/classes folder.

configuration

This goal supports the following configuration items:

outputDirectory	Where to copy files. Default is into target/classes

inputDirectory	From where to read source files. Default is from src/main/drools
extension	Extension for source files. Default is .drl
xmlExtension	Extension for xml source files. Default is .xml
reportDirectory	The folder into which reports of execution must be wrote.
reportFile	The file name for the report file.

Notes

This goal handle both Drools Source files and XML sources file at the same time. It is necessary to ensure there is no none Drools XML file in the drools source file.

drools-copy-validate-test

This goal is executed by default in phase test-compile. It work in the following way:

1. For each file with extension .drl in folder src/test/drools:
    1. Validate that the file is valid regarding drools compiler.
    2. Copy the file into target/test-classes. The folder hierarchy is the same in the target/test-classes folder.
2. For each file with extension .xml in folder src/test/drools:
    1. Validate that the file is valid regarding drools compiler.
    2. Copy the file into target/test-classes. The folder hierarchy is the same in the target/test-classes folder.

configuration

This goal supports the following configuration items:

outputDirectory	Where to copy files. Default is into target/test-classes
inputDirectory	From where to read source files. Default is from src/test/drools
extension	Extension for source files. Default is .drl
xmlExtension	Extension for xml source files. Default is .xml
reportDirectory	The folder into which reports of execution must be wrote.
reportFile	The file name for the report file.

Notes

This goal handle both Drools Source files and XML sources file at the same time. It is necessary to ensure there is no none Drools XML file in the drools source file.

Compilation Drools Goals

Theses two goals are related to compilation of the Drools files. Drools files can be used for the main code of the artifact or for the tests. Is it why there is two goals. Both drools source files and drools XML source files will be compiled.

drools-compile

This goal is executed by default in phase compile. It work in the following way:

1. For each file with extension .drl in folder src/main/drools:
    1. Compile the file using drools compiler.
    2. Write the resulting compiled file into target/classes. The folder hierarchy is the same in the target/classes folder. The file will have .cdrl extension.
2. For each file with extension .xml in folder src/main/drools:
    1. Compile the file using drools compiler.
    2. Write the resulting compiled file into target/classes. The folder hierarchy is the same in the target/classes folder. The file will have .cdrl extension.

configuration

This goal supports the following configuration items:

outputDirectory	Where to copy compiled files. Default is into target/classes
inputDirectory	From where to read source files. Default is from src/main/drools
extension	Extension for source files. Default is .drl
xmlExtension	Extension for xml source files. Default is .xml
compiledExtension	Extension for compiled files. Default is .cdrl
reportDirectory	The folder into which reports of execution must be wrote.
reportFile	The file name for the report file.

Notes

This goal handle both Drools Source files and XML sources file at the same time. It is necessary to ensure there is no none Drools XML file in the drools source file.

drools-compile-test

This goal is executed by default in phase test-compile. It work in the following way:

1. For each file with extension .drl in folder src/test/drools:
    1. Compile the file using drools compiler.
    2. Write the resulting compiled file into target/test-classes. The folder hierarchy is the same in the target/test-classes folder. The file will have .cdrl extension.
2. For each file with extension .xml in folder src/test/drools:
    1. Compile the file using drools compiler.

2. Write the resulting compiled file into target/test-classes. The folder hierarchy is the same in the target/test-classes folder. The file will have .cdrl extension.

configuration

This goal supports the following configuration items:

outputDirectory	Where to copy compiled files. Default is into target/test-classes
inputDirectory	From where to read source files. Default is from src/test/drools
extension	Extension for source files. Default is .drl
xmlExtension	Extension for xml source files. Default is .xml
compiledExtension	Extension for compiled files. Default is .cdrl
reportDirectory	The folder into which reports of execution must be wrote.
reportFile	The file name for the report file.

Notes

This goal handle both Drools Source files and XML sources file at the same time. It is necessary to ensure there is no none Drools XML file in the drools source file.

PostProcessing Drools Goals

Theses two goals are related to instrumentalization of the Drools classes. Drools files can be used for the main code of the artifact or for the tests. Is it why there is two goals.

In case these goals are runned twice, on the same classes, the second time, the classes will not be instrumented as they already are.

The code instrumentalization is based on ASM [http://asm.ow2.org/].

drools-postprocessor

This goal is executed by default in phase process-classes. It work in the following way:

1. For each class files that match patterns in folder target/classes:
    1. Check if the class is candidate to instrumentalization.
    2. If the class is candidate, instrumentalizate it.
    3. Replace the file of the class with the instrumentalized one.

configuration

This goal supports the following configuration items:

inputDirectory	From where to read class files. Default is from target/classes.
extension	Extension for source files. Default is .class

includes	List of file inclusion patterns
excludes	List of file exclusion patterns
reportDirectory	The folder into which reports of execution must be wrote.
reportFile	The file name for the report file.

Notes

Filtering classes can be usefull to avoid this plugin inspecting all the classes (which can take time).

drools-postprocessor-test

This goal is executed by default in phase process-test-classes. It work in the following way:

1. For each class files that match patterns in folder target/test-classes:
    1. Check if the class is candidate to instrumentalization.
    2. If the class is candidate, instrumentalizate it.
    3. Replace the file of the class with the instrumentalized one.

configuration

This goal supports the following configuration items:

inputDirectory	From where to read class files. Default is from target/test-classes.
extension	Extension for source files. Default is .class
includes	List of file inclusion patterns
excludes	List of file exclusion patterns
reportDirectory	The folder into which reports of execution must be wrote.
reportFile	The file name for the report file.

Notes

Filtering classes can be usefull to avoid this plugin inspecting all the classes (which can take time).

Report Goal

A special report goal is available to integrate the reporting of goals execution in the generated site.

drools-report

This goal will inspect each report file and add it to the site. To do so, it will:

1. For each reports files:
    1. Load it into memory.

2. Write recursively the comments from the report file into the site.

configuration

This goal supports the following configuration items:

reportInputDirectory	This is the source folder that contains the generated logs from the execution of the various goals.
reportOutputDirectory	This is the destination folder that will contains the resulted site.

Results

This plugin will list all the report file and display all the log of each file. If an entry contains other entries, they will be displayed also.

Figure 3.1. Example of report

Logging of maven plugin execution

Every goal of this plugin have two special parameters:

reportDirectory	This is the folder where the logging of the execution are stored.
reportFile	This is the file name that will be used by the plugin to store his logging information.

These logging information are based on an XML logging.

Here is an example of this log:

```
<droolsGoalExecutionLogs>
 <logs>
  <action>postprocessor</action>
  <comments>
   Class defined by file XXXXX has been rewritten by the plugin
  </comments>
  <fileName>XXXXX</fileName>
  <logs>
   <action>postprocessor</action>
```

```
 <comments>
  DroolsService annotation found. Field annotation will be used.
 </comments>
 <fileName>XXXXX</fileName>
 <timestamp>1258403938343</timestamp>
</logs>
<logs>
 <action>postprocessor</action>
 <comments>
  Constructor instrumentalization ()V
 </comments>
 <fileName>XXXXX</fileName>
 <timestamp>1258403938359</timestamp>
</logs>
<logs>
 <action>postprocessor</action>
 <comments>
  Method instrumentalization getField1/()Ljava/lang/String;
 </comments>
 <fileName>XXXXX</fileName>
 <timestamp>1258403938375</timestamp>
</logs>
...
```

This can be usefull to check what has been modified in the class files.

Chapter 4. Maven Archetype - Drools5 Integration Helper Archetype

Introduction

It is possible to automatically generated a basic Maven project that integrate all the Drools5-Integration-Helper features. To do so, a Maven archetype is provided. This archetype provides the following stuff:

- Maven 2 project, with references to the required libraries and plugin.
- Example of Annotated interfaces.
- Example of Annotated classes.
  - Using Field oriented annotations.
  - Using PreCondition oriented annotations.
- Example of JUnit for the two previous entries.

Presentation of what is an archetype is out of the scope of this document. Please refer to Maven Archetype plugin [http://maven.apache.org/plugins/maven-archetype-plugin/] for more information on the archetype plugin.

Usage

The creation of the Maven 2 project is based on the maven-archetype-plugin. It is possible to use the following call, from a terminal:

```
mvn \
   archetype:generate \
   -DarchetypeGroupId=org.boretti.drools.integration \
   -DarchetypeArtifactId=drools5-integration-helper-archetype \
   -DarchetypeVersion=1.3.0
```

The system will then ask you several questions (like package name for example). You must reply to the various questions and then a default Maven project, based on the archetype, will be created. This archetype is directly usable.

Example of usage

For example, it is possible to use the previous command:

```
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'archetype'.
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Default Project
[INFO]    task-segment: [archetype:generate] (aggregator-style)
[INFO] ------------------------------------------------------------------------
[INFO] Preparing archetype:generate
[INFO] No goals needed for project - skipping
```

```
[INFO] Setting property: classpath.resource.loader.class => 'org.codehaus.plexus.v
[INFO] Setting property: velocimacro.messages.on => 'false'.
[INFO] Setting property: resource.loader => 'classpath'.
[INFO] Setting property: resource.manager.logwhenfound => 'false'.
[INFO] [archetype:generate]
[INFO] Generating project in Interactive mode
[WARNING] No archetype repository found. Falling back to central repository (http:
[WARNING] Use -DarchetypeRepository=<your repository> if archetype's repository is
Define value for groupId: :
```

Here input grpid

```
Define value for artifactId: :
```

Here input artifactid

```
Define value for version:  1.0-SNAPSHOT: :
```

Here input 1.0-SNAPSHOT

```
Define value for package:  grpid: :
```

Here input package

```
Confirm properties configuration:
groupId: grpid
artifactId: artifactid
version: 1.0-SNAPSHOT
package: package
 Y: :
```

Finally confirm the data you input

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 25 seconds
[INFO] Finished at: Sun Nov 15 12:46:56 CET 2009
[INFO] Final Memory: 8M/15M
[INFO] ------------------------------------------------------------------------
```

The resulting folder will be named *artifactid*. It will contains a single pom.xml file. A *src* folder will be created. It will contains the *main* and *test* folder. You will then have a *drools* folder for the drools file, a *java* folder for the java source.

Chapter 5. Examples

Presentation

Drools5-Integration-Helper can be use in various ways to achieve similar goals. In the following section we will discuss how to achieve this goal (interfacing with drools business rules) from java, using the various approaches .

Additional examples are available as part of the special group *org.boretti.drools.integration.examples*.

We will use the same problem as a basis for all the examples:

The problem

We must develop a logic, based on business rule, to allow (or disallow) access to credit for customer of a bank.

Basically, in all solutions, we will have the following entities:

- A drools file defining how to compute the fact a customer is allowed to have or not access to credit.
- A class defining the customer.

The computation is based on the sex (a boolean value), the age (an int value) and the fact the customer is unemployed (boolean).

Examples using interface

General idea

The general idea is to have a interface that directly delegate computation of the credit status. To do so, the Customer class, a dedicated interface, the drools file and usage of the Maven plugin are required.

Implementation

Java Source

Customer.java

The class Customer defines the various field and getter/setter like in the standard way. Only fields that are not computed are defined.

```
package example1;

public class Customer {
        private String name;

        private String prenom;

        private boolean sexe;

        private int age;
```

```
private boolean chomeur;

/**
 * @return the name
 */
public String getName() {
        return name;
}

public Customer(String name, String prenom, boolean sexe, int age,
                boolean chomeur) {
        super();
        this.name = name;
        this.prenom = prenom;
        this.sexe = sexe;
        this.age = age;
        this.chomeur = chomeur;
}

/**
 * @param name the name to set
 */
public void setName(String name) {
        this.name = name;
}

/**
 * @return the prenom
 */
public String getPrenom() {
        return prenom;
}

/**
 * @param prenom the prenom to set
 */
public void setPrenom(String prenom) {
        this.prenom = prenom;
}

/**
 * @return the sexe
 */
public boolean isSexe() {
        return sexe;
}

/**
 * @param sexe the sexe to set
 */
public void setSexe(boolean sexe) {
        this.sexe = sexe;
}
```

```
        /**
         * @return the age
         */
        public int getAge() {
                return age;
        }

        /**
         * @param age the age to set
         */
        public void setAge(int age) {
                this.age = age;
        }

        /**
         * @return the chomeur
         */
        public boolean isChomeur() {
                return chomeur;
        }

        /**
         * @param chomeur the chomeur to set
         */
        public void setChomeur(boolean chomeur) {
                this.chomeur = chomeur;
        }

}
```

CustomerEvaluator.java

This interface define that it is possible to compute the credit check regarding a Customer. The annotation is used to define the drools file.

```
package example1;

import org.boretti.drools.integration.drools5.annotations.DroolsService;

@DroolsService(resourceName="/example1/Customer.cdrl")

public interface CustomerEvaluator {
        public boolean checkCredit(Customer customer);
}
```

Drools file

The drools file must use the various information regarding the Customer to compute the value. A boolean value is asserted into the workingMemory.

```
package example1;

global org.apache.log4j.Logger droolsLogger;
```

```
rule "Good sexe"
when
        c:Customer(sexe == true,chomeur == false)
then
        insert(true);
end

rule "Wrong sexe good age"
when
        c:Customer(sexe == false,chomeur == false,age>45)
then
        insert(true);
end

rule "Wrong customer 1"
when
        c:Customer(chomeur == true)
then
        insert(false);
end

rule "Wrong customer 2"
when
        c:Customer(sexe == false, age<=45)
then
        insert(false);
end
```

Maven integration

The pom.xml file must reference the drools helper library and the plugin. Using the packaging jar-drools ensure the drools goals to be executed automatically.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                            http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>
           org.boretti.drools.integration.test
        </groupId>
        <artifactId>
           drools5-integration-helper-maven-plugin-test9
        </artifactId>
        <packaging>jar-drools</packaging>
        <version>1.3.0</version>

        <dependencies>
                <dependency>
                        <groupId>
                           org.boretti.drools.integration
                        </groupId>
                        <artifactId>
```

```
                        drools5-integration-helper-library
                    </artifactId>
                    <version>1.3.0</version>
                </dependency>
                <dependency>
                    <groupId>junit</groupId>
                    <artifactId>junit</artifactId>
                    <version>4.4</version>
                    <scope>test</scope>
                </dependency>
            </dependencies>

            <build>
                <plugins>
                    <plugin>
                        <groupId>
                            org.apache.maven.plugins
                        </groupId>
                        <artifactId>
                            maven-compiler-plugin
                        </artifactId>
                        <configuration>
                            <source>1.6</source>
                            <target>1.6</target>
                        </configuration>
                    </plugin>
                    <plugin>
                        <artifactId>
                            drools5-integration-helper-maven-plugin
                        </artifactId>
                        <groupId>
                            org.boretti.drools.integration
                        </groupId>
                        <version>1.3.0</version>
                        <extensions>true</extensions>
                    </plugin>
                </plugins>
            </build>
        </project>
```

Usage

The user must create an instance of the Subscriber class and then get a reference to the CustomerEvaluator interface. To do so, it is possible to use the following call for example new DroolsProvider().getService(CustomerEvaluator.class). Then it is possible to just call the method exposed by the concrete implementation of the interface.

Examples using class

General idea

The general idea is to have a class that directly delegate computation of the field containing the credit status. To do so, only the Customer class, the drools file and usage of the Maven plugin are required.

Implementation

Java Source

The class Customer defines the various field and getter/setter like in the standard way. Fields that are not part of the credit check are annotated with @DroolsIgnored. The field generated by the drools is annotated with @DroolsGenerated. The annotation @DroolsService references the drools file.

```
package example1;

import org.boretti.drools.integration.drools5.annotations.DroolsService;
import org.boretti.drools.integration.drools5.annotations.DroolsGenerated;
import org.boretti.drools.integration.drools5.annotations.DroolsIgnored;

@DroolsService(resourceName="/example1/Customer.cdrl")
public class Customer {

	//This field must not trigger field recomputation
	@DroolsIgnored
	private String name;

	//This field must not trigger field recomputation
	@DroolsIgnored
	private String prenom;

	private boolean sexe;

	private int age;

	private boolean chomeur;

	//This field is generated using drools
	@DroolsGenerated
	private boolean creditAllowed;

	/**
	 * @return the name
	 */
	public String getName() {
		return name;
	}

	public Customer(String name, String prenom, boolean sexe, int age,
			boolean chomeur) {
		super();
		this.name = name;
		this.prenom = prenom;
		this.sexe = sexe;
		this.age = age;
		this.chomeur = chomeur;
	}

	/**
```

```
 * @param name the name to set
 */
public void setName(String name) {
        this.name = name;
}

/**
 * @return the prenom
 */
public String getPrenom() {
        return prenom;
}

/**
 * @param prenom the prenom to set
 */
public void setPrenom(String prenom) {
        this.prenom = prenom;
}

/**
 * @return the sexe
 */
public boolean isSexe() {
        return sexe;
}

/**
 * @param sexe the sexe to set
 */
public void setSexe(boolean sexe) {
        this.sexe = sexe;
}

/**
 * @return the age
 */
public int getAge() {
        return age;
}

/**
 * @param age the age to set
 */
public void setAge(int age) {
        this.age = age;
}

/**
 * @return the chomeur
 */
public boolean isChomeur() {
        return chomeur;
}
```

```
        /**
         * @param chomeur the chomeur to set
         */
        public void setChomeur(boolean chomeur) {
                this.chomeur = chomeur;
        }

        /**
         * @return the creditAllowed
         */
        public boolean isCreditAllowed() {
                return creditAllowed;
        }

        /**
         * @param creditAllowed the creditAllowed to set
         */
        public void setCreditAllowed(boolean creditAllowed) {
                this.creditAllowed = creditAllowed;
        }
}
```

Drools file

The drools file contains the logic to compute the creditAllowed value. This file is implicitly call to compute the creditAllowed. This drools file must set the creditAllowed.

```
package example1;

global org.apache.log4j.Logger droolsLogger;

rule "Reset"
        salience 2
when
        c:Customer()
then
        c.setCreditAllowed(false);
end

rule "Good sexe"
        salience 1
when
        c:Customer(sexe == true,chomeur == false)
then
        c.setCreditAllowed(true);
end

rule "Wrong sexe good age"
        salience 1
when
        c:Customer(sexe == false,chomeur == false,age>45)
then
```

```
        c.setCreditAllowed(true);
end
```

Maven integration

The pom.xml file must reference the drools helper library and the plugin. Using the packaging jar-drools ensure the drools goals to be executed automatically.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                       http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>
           org.boretti.drools.integration.test
        </groupId>
        <artifactId>
           drools5-integration-helper-maven-plugin-test8
        </artifactId>
        <packaging>jar-drools</packaging>
        <version>1.3.0</version>

        <dependencies>
                <dependency>
                        <groupId>
                           org.boretti.drools.integration
                        </groupId>
                        <artifactId>
                           drools5-integration-helper-library
                        </artifactId>
                        <version>1.3.0</version>
                </dependency>
                <dependency>
                        <groupId>junit</groupId>
                        <artifactId>junit</artifactId>
                        <version>4.4</version>
                        <scope>test</scope>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <groupId>
                                   org.apache.maven.plugins
                                </groupId>
                                <artifactId>
                                   maven-compiler-plugin
                                </artifactId>
                                <configuration>
                                        <source>1.6</source>
                                        <target>1.6</target>
                                </configuration>
                        </plugin>
```

```
                        <plugin>
                                <artifactId>
                                   drools5-integration-helper-maven-plugin
                                </artifactId>
                                <groupId>
                                   org.boretti.drools.integration
                                </groupId>
                                <version>1.3.0</version>
                                <extensions>true</extensions>
                    </plugin>
                </plugins>
        </build>
</project>
```

Usage

With this approach, it is only necessary to instantiate the object. Every time a field that is use to compute the credit access is modified, the system will ensure the credit access field is recomputed at next access (in getter method). Everything is transparent at runtime.

Examples using condition

General idea

In this approach the check to know if the customer is allowed to have credit is not done in a dedicated method or thru a field, but but defining that there are some preconditions to be checked before granting (and giving) the credit.

So, only a Customer class and rules (and maven usage) are required. For the completude of the solution, a dedicated Error is also defined.

Implementation

Java Source

Customer class

This class have several field and one method annotated with @DroolsPreCondition.

```
package conditions;

import org.boretti.drools.integration.drools5.annotations.DroolsPreCondition;

public class Customer {
        private String name;

        private String prenom;

        private boolean sexe;

        private int age;
```

```
private boolean chomeur;

/**
 * @return the name
 */
public String getName() {
        return name;
}

public Customer(String name, String prenom, boolean sexe, int age,
                boolean chomeur) {
        super();
        this.name = name;
        this.prenom = prenom;
        this.sexe = sexe;
        this.age = age;
        this.chomeur = chomeur;
}

/**
 * @param name the name to set
 */
public void setName(String name) {
        this.name = name;
}

/**
 * @return the prenom
 */
public String getPrenom() {
        return prenom;
}

/**
 * @param prenom the prenom to set
 */
public void setPrenom(String prenom) {
        this.prenom = prenom;
}

/**
 * @return the sexe
 */
public boolean isSexe() {
        return sexe;
}

/**
 * @param sexe the sexe to set
 */
public void setSexe(boolean sexe) {
        this.sexe = sexe;
}
```

```
		/**
		 * @return the age
		 */
		public int getAge() {
			return age;
		}

		/**
		 * @param age the age to set
		 */
		public void setAge(int age) {
			this.age = age;
		}

		/**
		 * @return the chomeur
		 */
		public boolean isChomeur() {
			return chomeur;
		}

		/**
		 * @param chomeur the chomeur to set
		 */
		public void setChomeur(boolean chomeur) {
			this.chomeur = chomeur;
		}

		@DroolsPreCondition(resourceName="/Customer-conditions-pre.cdrl",error=UnA
		public void doCredit(float howMany) {
			System.err.println("Credit granted :"+howMany);
		}

	}
```

Error class

```
	package conditions;

	public class UnAuthorizedError extends Error {
		public UnAuthorizedError() {}

		public UnAuthorizedError(String msg) {
			super(msg);
		}

		public UnAuthorizedError(String msg,Throwable cause) {
			super(msg,cause);
		}
	}
```

Drools file

This file define the business rule. In case credit can't be allowed, an Error is inserted.

```
package conditions;

global org.apache.log4j.Logger droolsLogger;


rule "Wrong customer 1"
when
        c:Customer(chomeur == true)
then
        insert(new UnAuthorizedError("Chomeur are not allowed"));
end

rule "Wrong customer 2"
when
        c:Customer(sexe == false, age<=45)
then
        insert(new UnAuthorizedError("Bad sex with age under 45 are not allowed"))
end
```

Maven integration

The pom.xml file must reference the drools helper library and the plugin. Using the packaging jar-drools ensure the drools goals to be executed automatically.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                            http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>
           org.boretti.drools.integration.test
        </groupId>
        <artifactId>
           drools5-integration-helper-maven-plugin-testXXX
        </artifactId>
        <packaging>jar-drools</packaging>
        <version>1.3.0</version>

        <dependencies>
                <dependency>
                        <groupId>
                           org.boretti.drools.integration
                        </groupId>
                        <artifactId>
                           drools5-integration-helper-library
                        </artifactId>
                        <version>1.3.0</version>
                </dependency>
                <dependency>
                        <groupId>junit</groupId>
                        <artifactId>junit</artifactId>
                        <version>4.4</version>
                        <scope>test</scope>
                </dependency>
```

```
        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <groupId>
                                   org.apache.maven.plugins
                                </groupId>
                                <artifactId>
                                   maven-compiler-plugin
                                </artifactId>
                                <configuration>
                                        <source>1.6</source>
                                        <target>1.6</target>
                                </configuration>
                        </plugin>
                        <plugin>
                                <artifactId>
                                   drools5-integration-helper-maven-plugin
                                </artifactId>
                                <groupId>
                                   org.boretti.drools.integration
                                </groupId>
                                <version>1.3.0</version>
                                <extensions>true</extensions>
                    </plugin>
                </plugins>
        </build>
</project>
```

Usage

The class Customer can be used directly. When the method doCredit is call, the rules will be run priori to the execution of the method itself.