
proobjectlink-jpx-jpl-swi
v. 1.0-SNAPSHOT
User Guide

Table of Contents

1. Table of Contents	i
2. What Is	1
3. Getting Started	3
4. Prolog Programming	5
5. Bidirectional Interface	7
6. Development Tools	9
7. Contribution	11
8. Related Works	13
9. FAQ	15

1 What Is

1.1 What is

1.1.1 Introduction

Java Prolog Interface (JPI) is an Application Provider Interface (API) for interaction between Java and Prolog programming languages. Is a bidirectional interface that communicate Java applications with Prolog program or database and Prolog procedures with Java class and methods.

JPI is an abstraction layer over concrete prolog drivers over Prolog Engines. This API define all mechanism to interact with any Prolog Engine and maintain the application independent to a specific underlying engine. JPI have several connectors to open source prolog engines like SWI, YAP, XSB native engines and tuProlog, jTrolog, jLog Java based prolog engines.

JPI study all related Java-Prolog integration libraries and take the better features from each solution with the propose to achieve a common integration interface. The last feature allows switch the underlying Prolog Engine driver and the application code still be the same.

JPI run over any Java Virtual Machine that support Java SE 5 or above. The project was tested over HotSpot, Open J9 and JRockit Virtual Machines over Operating Systems like Windows (7,8,10), Linux (Debian, Ubuntu) and Mac OS X. Can be deployed on Servlets Containers like Jetty, Tomcat or Glassfish Application Server. JPI can be include in any Java Project using the commonest Java Integration Development Enviroment (IDE) like Eclipse, Netbeans, IntelliJIDEA and so on.

JPI is developed and maintained by Prolobjectlink Project an open source initiative for build logic based applications using Prolog like fundamental Logic Programming Language in the persistence layer and application programming.

The selected license for JPI is Simplified BSD License a permissive license allowing to concrete implementations can use some possibilities like GPL, Apache 2.0 and others in the interface implementation. We suggest adopt the same license from prolog java driver if it is possible. In this way the java prolog driver and your JPI implementation share the same license and can be combined with JPI interface that is less restrictive licensed. Finally, license is the most restrictive licensed, being in many occasions the java prolog driver licenses the most restrictive.

1.1.2 Copyright and License Information

JPI is release under Simplified BSD License:

Copyright © 2019 Prolobjectlink Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.1.3 Release Notes

Version 1.0.0: Initial release.

1.1.4 Acknowledgments

Blah, Blah, ...

2 Getting Started

2.1 Getting Started

2.1.1 Install

Java Prolog Interface API is distributed with implementation adapter and concrete prolog driver library until it is possible according to related libraries licenses. The distributions are named normally such that **proobjectlink-jpi-jpl7-swi7-x.y.z-dist.zip** meaning that this distribution is a JPI implementation over JPL version 7 or above and SWI-Prolog version 7 or above. The x.y.z is the distribution version. The distribution can be downloaded in zip or tar.gz compresses format. To install you need perform the following steps:

- Install Java Runtime Environment (JRE) 1.8 or above.
- Install Native Prolog Engine compatible to Operating System and your architecture. If the Prolog Engine to use is Java-based this step is omitted.
- Configure System Path with Prolog Engine routes. If the Prolog Engine to use is Java-based this step is omitted.
- Download Java Prolog Interface compatible to related prolog engine and unzip the distribution over Operating File System.
- Configure System Path with JPI unzip folder route.
- Open a new System console and type `plink -i` to see the product information.

For the JPI beginners we recommended start with a Pure Java-Prolog Engine because have less configuration aspects and native engine are more difficult to link.

2.1.2 Directories

After download and unzip JPI distribution in the final JPI folder you will see the following structure:

Folder/File	Description
bin	Binaries scripts
doc	Documentation
lang	Prolog programs files
lib	Library jars files
obj	Programs to link native engine procedures
src	Adapter source folder
CONTRIBUTING	Binaries scripts
LICENSE	Binaries scripts
NOTICE	Binaries scripts
README	Binaries scripts

2.1.3 Architecture

In general way and in bottom-up order the JPI architecture is composed by the guest Operating System at low level. Over this level we find compatible with guest Operating System and Native Prolog Engines implementations. Over this level we find Pure Java Prolog Engine implementations

and Java Driver libraries to Native Prolog Engine. Over this layer is the JPI interface adapter implementation for your correspondent Java Prolog Driver. In the top level we find a User Application that use the JPI interface.



2.1.4 Getting started Java to Prolog

After installation and architecture compression you can use the hello world sample for test the system integration. This hello world sample show how interacts with JPI from Java programming language with Abstracted Prolog Engine. For the first experience we suggesting use a Java-based Prolog engine like tuProlog because have less configuration aspects.

Create in your preferred development environment an empty project. Set in the project build path the JPI downloaded libraries located at lib folder. Create a Main Java class that look like below code:

```

public class Main {

    public static void main(String[] args) {
        PrologProvider provider = Prolog.
            getProvider(XsbProlog.class);
        PrologEngine engine = provider.newEngine();
        engine.asserta("sample('hello wolrd')");
        PrologQuery query=engine.query("sample(X)");
        System.out.println(query.one());
    }

}

```

2.1.5 Getting started Prolog to Java

Blah, Blah, ...

3 Prolog Programming

Paragraph 1, line 1. Paragraph 1, line 2.

Paragraph 2, line 1. Paragraph 2, line 2.

3.1 Section title

3.1.1 Sub-section title

3.1.1.1 Sub-sub-section title

3.Sub-sub-sub-section title

3.Sub-sub-sub-sub-section title

- List item 1.
- List item 2.
Paragraph contained in list item 2.
 - Sub-list item 1.
 - Sub-list item 2.
- List item 3. Force end of list:

Verbatim text not contained in list item 3

1. Numbered item 1.

A.Numbered item A.

B.Numbered item B.

2. Numbered item 2.

List numbering schemes: [[1]], [[a]], [[A]], [[i]], [[I]].

Defined term 1

of definition list.

Defined term 2

of definition list.

Verbatim text
in a box

--- instead of +- suppresses the box around verbatim text.

Figure caption

Centered cell 1,1	Left-aligned cell 1,2	Right-aligned cell 1,3
cell 2,1	cell 2,2	cell 2,3

Table caption

No grid, no caption:

cell	cell
cell	cell

Horizontal line:

3.2 ^L New page.

Italic font. **Bold** font. Monospaced font.

Anchor. Link to [anchor](#). Link to <http://www.pixware.fr>. Link to [showing alternate text](#). Link to [Pixware home page](#).

Force line
break.

Non breaking space.

Escaped special characters: ~, =, -, +, *, [,], <, >, {, }, \.

Copyright symbol: ©, ©, ©.

4 Bidirectional Interface

Paragraph 1, line 1. Paragraph 1, line 2.

Paragraph 2, line 1. Paragraph 2, line 2.

4.1 Section title

4.1.1 Sub-section title

4.1.1.1 Sub-sub-section title

4.Sub-sub-sub-section title

4.Sub-sub-sub-sub-section title

- List item 1.
- List item 2.
Paragraph contained in list item 2.
 - Sub-list item 1.
 - Sub-list item 2.
- List item 3. Force end of list:

Verbatim text not contained in list item 3

1. Numbered item 1.

A.Numbered item A.

B.Numbered item B.

2. Numbered item 2.

List numbering schemes: [[1]], [[a]], [[A]], [[i]], [[I]].

Defined term 1

of definition list.

Defined term 2

of definition list.

Verbatim text
in a box

--- instead of +-- suppresses the box around verbatim text.

Figure caption

Centered cell 1,1	Left-aligned cell 1,2	Right-aligned cell 1,3
cell 2,1	cell 2,2	cell 2,3

Table caption

No grid, no caption:

cell	cell
cell	cell

Horizontal line:

4.2 ^L New page.

Italic font. **Bold** font. Monospaced font.

Anchor. Link to [anchor](#). Link to <http://www.pixware.fr>. Link to [showing alternate text](#). Link to [Pixware home page](#).

Force line
break.

Non breaking space.

Escaped special characters: ~, =, -, +, *, [,], <, >, {, }, \.

Copyright symbol: ©, ©, ©.

5 Development Tools

Paragraph 1, line 1. Paragraph 1, line 2.

Paragraph 2, line 1. Paragraph 2, line 2.

5.1 Section title

5.1.1 Sub-section title

5.1.1.1 Sub-sub-section title

5.Sub-sub-sub-section title

5.Sub-sub-sub-sub-section title

- List item 1.
- List item 2.
Paragraph contained in list item 2.
 - Sub-list item 1.
 - Sub-list item 2.
- List item 3. Force end of list:

Verbatim text not contained in list item 3

1. Numbered item 1.

A.Numbered item A.

B.Numbered item B.

2. Numbered item 2.

List numbering schemes: [[1]], [[a]], [[A]], [[i]], [[I]].

Defined term 1

of definition list.

Defined term 2

of definition list.

Verbatim text
in a box

--- instead of +- suppresses the box around verbatim text.

Figure caption

Centered cell 1,1	Left-aligned cell 1,2	Right-aligned cell 1,3
cell 2,1	cell 2,2	cell 2,3

Table caption

No grid, no caption:

cell	cell
cell	cell

Horizontal line:

5.2 ^L New page.

Italic font. **Bold** font. Monospaced font.

Anchor. Link to [anchor](#). Link to <http://www.pixware.fr>. Link to [showing alternate text](#). Link to [Pixware home page](#).

Force line
break.

Non breaking space.

Escaped special characters: ~, =, -, +, *, [,], <, >, {, }, \.

Copyright symbol: ©, ©, ©.

6 Contribution

6.1 Contribution

6.1.1 Issues

See the issue tracker at <https://github.com/proobjectlink/proobjectlink-jpx-jpl-swi> to create a new issue or take an existing one.

6.1.2 Changes and Build

Fork the repository in GitHub.

Clone your forked repository in your preferred IDE

Proobjectlink development requires.

- Java 1.8 - Maven 3.1.0 or above

Make changes in your cloned repository

Run all test to see if the system still consistent after your changes

Create unit-tests and make sure that the include changes are covered to 100%

Run the benchmark to see if the system performance still consistent after your changes

Add a description of your changes in CHANGELOG.txt and src/changes/changes.xml

Commit the changes.

Run an integration test on Travis-CI

Submit a pull request.

6.1.3 New Implementations

The project start with some adapters implementations over most used open source prolog engines.

We accept any new adapter implementation of another prolog engine not covered at this moment.

For this propose create a new GitHub source code repository naming this follow the project convesion:

proobjectlink-jpi- new engine implementation name

Create an new maven project in your preferred IDE named like repository.

Copy the src/assembly/dist.xml descriptor

Copy the src/build/filters folder and change by your console main entry point

Copy and clean src/changes/changes.xml to go reporting every change

Copy src/site folder to generate a similar project site.

Copy the pom.xml properties, build, report, etc... from another implementation

Change the project information.

Add your dependencies including Java Prolog Interface API

```

<dependencies>
    ...
    <dependency>
        <groupId>org.proobjectlink</groupId>
        <artifactId>proobjectlink-jpi</artifactId>
        <version>[1.0.0, )</version>
    </dependency>
    ...
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>[4.10, )</version>
        <scope>test</scope>
    </dependency>
    ...
</dependencies>

```

In test package copy the unit-tests cases from another implementation to develop in test driven mode.

We suggest like adapter implementation order begin with data types, parsers, engine and finally query.

Run all test to see if the system to see if your implementation pass all.

Create unit-tests and make sure that the include changes are covered to 100%

Create the benchmark to see if the system performance.

Add a description of your changes in CHANGELOG.txt and src/changes/changes.xml

Commit the changes.

Run an integration test on Travis-CI or another CI system

6.1.4 Version Numbering

Proobjectlink version signature is Major.Minor.Micro.

Major version is change when the API compatibility is broken. Minor version is change when a new feature is include in the release. Micro version is change when some bug is fixed or some maintenance take place

Proobjectlink suggest work over the started 1.Y.Z version to preserve compatibility all the time. You are free of make any change adding new features, fixing bugs or code maintenance.

6.1.5 Contact us

Please contact us at our project mailing list <https://groups.google.com/group/proobjectlink> to debat over project evolution

Thanks for contributing to Proobjectlink!

7 Related Works

Paragraph 1, line 1. Paragraph 1, line 2.

Paragraph 2, line 1. Paragraph 2, line 2.

7.1 Section title

7.1.1 Sub-section title

7.1.1.1 Sub-sub-section title

7.Sub-sub-sub-section title

7.Sub-sub-sub-sub-section title

- List item 1.
- List item 2.
Paragraph contained in list item 2.
 - Sub-list item 1.
 - Sub-list item 2.
- List item 3. Force end of list:

Verbatim text not contained in list item 3

1. Numbered item 1.

A.Numbered item A.

B.Numbered item B.

2. Numbered item 2.

List numbering schemes: [[1]], [[a]], [[A]], [[i]], [[I]].

Defined term 1

of definition list.

Defined term 2

of definition list.

Verbatim text
in a box

--- instead of +- suppresses the box around verbatim text.

Figure caption

Centered cell 1,1	Left-aligned cell 1,2	Right-aligned cell 1,3
cell 2,1	cell 2,2	cell 2,3

Table caption

No grid, no caption:

cell	cell
cell	cell

Horizontal line:

7.2 ^L New page.

Italic font. **Bold** font. Monospaced font.

Anchor. Link to [anchor](#). Link to <http://www.pixware.fr>. Link to [showing alternate text](#). Link to [Pixware home page](#).

Force line
break.

Non breaking space.

Escaped special characters: ~, =, -, +, *, [,], <, >, {, }, \.

Copyright symbol: ©, ©, ©.

8 FAQ

8.1 Frequently Asked Questions

General

1. [What is the difference between `mvn site` and `mvn site:site`?](#)
2. [How do I Integrate static \(X\)HTML pages into my Maven site?](#)
3. [How to include a custom Doxia module, like Twiki?](#)
4. [How can I validate my xdoc/fml source files?](#)
5. [How does the Site Plugin use the `<url>` element in the POM?](#)

Specific issues

1. [Why do my absolute links get translated into relative links?](#)
2. [Why don't the links between parent and child modules work when I run "`mvn site`"?](#)
3. [Can I use entities in xdoc/fml source files?](#)

8.2 General

What is the difference between `mvn site` and `mvn site:site`?

`mvn site`

Calls the *site* **phase** of the site **lifecycle**. Full site lifecycle consists in the following life cycle phases: `pre-site`, `site`, `post-site` and `site-deploy`. See [Lifecycle Reference](#). Then it calls plugin goals associated to `pre-site` and `site` phases.

`mvn site:site`

Calls the *site* **goal** of the site **plugin**. See [site:site](#).

[\[top\]](#)

How do I Integrate static (X)HTML pages into my Maven site?

You can integrate your static pages by following these steps:

- Put your static pages in the resources directory, `${basedir}/src/site/resources`
- Create your `site.xml` and put it in `${basedir}/src/site`
- Link to the static pages by modifying the menu section, create items and map them to the filenames of the static pages

[\[top\]](#)

How to include a custom Doxia module, like Twiki?

The site plugin handles out-of-box `apt`, `xdoc` and `fml` formats. If you want to use a custom format like Twiki, Simple DocBook, or XHTML (or any other document format for which a doxia parser exists, see the list of [Doxia Markup Languages](#)), you need to specify the corresponding Doxia module dependency, e.g. for Twiki:

```

<project>
  ...
  <build>
    <plugins>
      ...
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-site-plugin</artifactId>
        <dependencies>
          <dependency>
            <groupId>org.apache.maven.doxia</groupId>
            <artifactId>doxia-module-twiki</artifactId>
            <version><!-- doxia version appropriate to the site plugin version -->
          </dependency>
        </dependencies>
      </plugin>
    </plugins>
  </build>
  ...
</project>

```

Note that the doxia version has to be adjusted to the site-plugin version you are using, see the [Migration Guide](#). In particular, for site plugin versions `>=2.1` you need to use doxia `>=1.1`.

[\[top\]](#)

How can I validate my xdoc/fml source files?

Since version 2.1.1 of the Site Plugin, there is a `validate` configuration parameter that switches on xml validation (default is off). Note that in the current implementation of the parser used by Doxia, validation requires an independent parsing run, so that every source file is actually parsed twice when validation is switched on.

If validation is switched on, **all** xml source files need a correct schema and/or DTD definition. See the Doxia documentation on [validating xdocs](#), and the schema definitions for [xdoc](#) and [fml](#).

[\[top\]](#)

How does the Site Plugin use the <url> element in the POM?

The Site Plugin does not use the `<url>` element in the POM. The project URL is just a piece of information to let your users know where the project lives. Some other plugins (e.g. the `project-info-report-plugin`) may be used to present this information. If your project has a URL where the generated site is deployed, then put that URL into the `<url>` element. If the project's site is not deployed anywhere, then remove the `<url>` element from the POM.

On the other hand, the `<distributionManagement.url>` is used in a multi-module build to construct relative links between the generated sub-module sites. In a multi module build it is important for the parent and child modules to have **different** URLs. If they have the same URL, then links within the combined site will not work. Note that a proper URL **should** also be terminated by a slash (`"/`).

[\[top\]](#)

8.3 Specific issues

Why do my absolute links get translated into relative links?

This happens because the Site Plugin tries to make all URLs relative, when possible. If you have something like this defined in your `pom.xml`:

```
<url>http://www.your.site.com/</url>
```

and create links in your `site.xml` (just an example) like this:

```
<links>
  <item name="Your Site" href="http://www.your.site.com/" />
  <item name="Maven 2" href="http://maven.apache.org/maven2/" />
</links>
```

You will see that the link to "Your site" will be a relative one, but that the link to "Maven 2" will be an absolute link.

There is an [issue for this in JIRA](#), where you can read more about this.

[\[top\]](#)

Why don't the links between parent and child modules work when I run "mvn site"?

What "mvn site" will do for you, in a multi-project build, is to run "mvn site" for the parent and all its modules **individually**. The links between parent and child will **not** work here. They **will** however work when you deploy the site.

If you want to test this, prior to deployment, you can run the `site:stage` goal as described in the [usage documentation](#) instead.

[\[top\]](#)

Can I use entities in xdoc/fml source files?

Yes. Entity resolution has been added in Doxia version 1.1, available in Site Plugin 2.1 and later.

There is a catch however. In the current implementation (as of maven-site-plugin-2.1.1), entities are only resolved by an independent [validation](#) run. Therefore, if you want to use entities, you **have** to switch on validation for your xml source files. See [MSITE-483](#).

[\[top\]](#)