Prolobjectlink Projects

prolobjectlink-jpi-ip

v. 1.2-SNAPSHOT

User Guide

Prolobjectlink Project **2023-12-10**

Table of Contents

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 betters features from each solution with the propose to achieve a common integration interface. The last feature allows switch the under laying 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:

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

Thanks to Prolobjectlink Development Team, Contributors and Sponsors.

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 **prolobjectlink-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 pllink –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
docs	Documentation
prt	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

3.1 Prolog Programming

3.1.1 Introduction

Prolog is a programming language that originated in the early 1970s and was initially developed for natural language processing. It became popular with the introduction of interpreters for various computer systems. Prolog evolved from being interpreted to a semi-interpreted language, thanks to the creation of a compiler. Its adoption for the fifth-generation computer project and the establishment of an ISO standard (ISO/IEC 13211-1) further contributed to its widespread use.

Prolog belongs to the paradigm of logic and declarative languages, which sets it apart significantly from more popular languages such as FORTRAN, Pascal, C, or Java. In the aforementioned programming languages, instructions are typically executed sequentially, one after another, in the same order as they are written. The order only changes when a control instruction is reached (such as a loop, conditional statement, or transfer).

Prolog programs consist of Horn clauses that represent "modus ponens" rules, meaning "If the antecedent is true, then the consequent is true." However, the way Horn clauses are written is the opposite of the usual convention. First, the consequent is written, followed by the antecedent. The antecedent can be a conjunction of conditions referred to as a sequence of goals. Each goal is separated by a comma and can be seen as similar to an instruction or procedure call in imperative languages. In Prolog, there are no control instructions. Execution is based on two concepts: unification and backtracking.

Thanks to unification, each goal determines a subset of clauses that can be executed. Each of these subsets is called a choice point. Prolog selects the first choice point and continues executing the program until determining whether the goal is true or false. If the goal is false, backtracking comes into play. Backtracking involves undoing everything that has been executed, placing the program in the same state it was in just before reaching the choice point. Then, the next pending choice point is taken, and the process is repeated. All goals conclude their execution either successfully ("true") or unsuccessfully ("false").

3.1.2 Data types

Prolog has a single data type called "term." Terms can be atoms, numbers, variables, or compound terms. Atoms are general-purpose names with no built-in meaning. Examples of atoms are x, red, 'Taco', and 'some atom'. Numbers can be floats or integers. Prolog systems compatible with the ISO standard can check the "bounded" flag. Most major Prolog systems support integers of arbitrary length. Variables are represented by strings consisting of letters, numbers, and underscores. They start with an uppercase letter or underscore. Variables closely resemble logic variables as they act as placeholders for any term.

A compound term consists of an atom called a "functor" and a number of "arguments," which are themselves terms. Compound terms are typically written as a functor followed by a list of comma-separated argument terms enclosed in parentheses. The number of arguments is referred to as the term's "arity." An atom can be seen as a compound term with arity zero. For example, person_friends(zelda, [tom, jim]) is a compound term.

Special cases of compound terms: - Lists: An ordered collection of terms denoted by square brackets. The terms are separated by commas. An empty list is represented by []. For instance, [1, 2, 3] or [red, green, blue].

- Strings: A sequence of characters surrounded by quotes. Depending on the value of the Prolog flag "double_quotes," a string can be treated as a list of character codes, a list of single-character atoms, or simply as an atom. For example, "to be, or not to be".

ISO Prolog provides predicates like atom/1, number/1, integer/1, and float/1 for type-checking.

3.1.3 Rules and Facts

Prolog programs define relationships using clauses. Pure Prolog is limited to Horn clauses. There are two types of clauses: facts and rules. A rule has the form:

Head :- Body.

This signifies that "Head is true if Body is true." The body of a rule consists of calls to predicates, which are the goals of the rule. The built-in logical operator ,/2 (denoting a binary operator named ",") represents the conjunction of goals, while ;/2 represents disjunction. Conjunctions and disjunctions can only appear in the body of a rule, not in the head.

Clauses with empty bodies are referred to as facts. An example of a fact is:

cat(tom).

This is equivalent to the rule:

cat(tom) :- true.

The built-in predicate true/0 is always true. Based on the given fact, we can ask: Is tom a cat?

?- cat(tom).

The answer is "Yes." We can also inquire about the things that are cats: What things are cats?

?- cat(X).

The answer is X = tom. Clauses with bodies are known as rules. An example of a rule is:

animal(X) :- cat(X).

If we include this rule and ask what things are animals:

?- animal(X).

The answer is X = tom.

Due to the relational nature of many built-in predicates, they can be used in multiple ways. For instance, length/2 can be used to find the length of a list (length(List, L)) given a list List, generate a list skeleton of a specific length (length(X, 5)), or generate both list skeletons and their lengths together (length(X, L)). Similarly, append/3 can be employed to append two lists (append(ListA, ListB, X)) given lists ListA and ListB, or split a given list into parts (append(X, Y, List)) given a list List. Consequently, a relatively small set of library predicates is sufficient for many Prolog programs.

As a general-purpose language, Prolog also offers various built-in predicates for common tasks such as input/output, graphics usage, and interaction with the operating system. These predicates do not have relational meanings and are only useful for their system-related effects. For example, the predicate write/1 displays a term on the screen.

3.1.4 Execution

To run a Prolog program, you start by entering a single goal called the query. The Prolog engine then attempts to find a resolution refutation of the negated query. Prolog uses a method called SLD resolution. If the negated query can be proven false, it means that the original query, with the appropriate variable assignments, is a logical consequence of the program. In this case, all the variable assignments are displayed, and the query is considered successful.

Operatively, Prolog's execution strategy can be seen as an extension of function calls in other programming languages. One difference is that multiple clause heads can match a particular call. When this happens, the system creates a choice point, where it matches the goal with the clause head of the first alternative and proceeds with that alternative's goals. If any goal fails during program execution, all variable assignments made since the most recent choice point was created are undone, and execution continues with the next alternative of that choice point. This strategy is known as chronological backtracking.

For example:

```
mother_child(trude, sally).
father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).
sibling(X, Y)      :- parent_child(Z, X), parent_child(Z, Y).
parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

Executing the following query will yield a true result:

```
?- sibling(sally, erica).
Yes
```

Here's how the result is obtained: Initially, the only clause head that matches the query `sibling(sally, erica)` is the first one. Therefore, proving the query is equivalent to proving the body of that clause with the appropriate variable assignments, which in this case is the conjunction `(parent_child(Z,sally), parent_child(Z,erica))`. The next goal to prove is the leftmost part of this conjunction: `parent_child(Z, sally)`. There are two clause heads that match this goal. The system creates a choice point and attempts the first alternative, which has the body `father_child(Z, sally)`. This goal can be proven with the fact `father_child(tom, sally)`, leading to the assignment `Z = tom`. The next goal to prove is the second part of the conjunction: `parent_child(tom, erica)`. This is also proven by the corresponding fact. Since all the goals have been proven, the query is considered successful. As the query doesn't contain any variables, no assignments are displayed to the user.

A query that includes variables, such as `?- father_child(Father, Child).`, will list all valid answers through backtracking. Note that with the given code, the query `?- sibling(sally, sally).` also succeeds. If there are specific restrictions, additional goals should be added to the code.

ISO Prolog is a technical standard developed by the International Organization for Standardization (ISO). It consists of two main parts. The first part, ISO/IEC 13211-1, was published in 1995 with the goal of standardizing the core elements of Prolog. This standard aims to bring clarity and remove ambiguities in the language, making it easier to write portable programs. Additionally, there have been three corrigenda issued: Cor.1:2007, Cor.2:2012, and Cor.3:2017.

The second part, ISO/IEC 13211-2, was published in 2000 and provides support for modules within the standard. The maintenance of this standard is overseen by the ISO/IEC JTC1/SC22/WG17 working group. In the United States, the US Technical Advisory Group for the standard is ANSI X3J17.

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/prolobjectlink/prolobjectlink-jpi-ip 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

Prolobjectlink 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:

prolobjectlink-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

```
<repositories>
        <repository>
                <id>ossrh</id>
                <name>Sonatype Nexus Snapshots</name>
                <url>https://oss.sonatype.org/content/repositories/snapshots</url>
                <releases>
                        <enabled>false</enabled>
                </releases>
                <snapshots>
                        <enabled>true</enabled>
                </snapshots>
        </repository>
</repositories>
...
<dependencies>
        ...
        <dependency>
                <groupId>org.prolobjectlink</groupId>
                <artifactId>prolobjectlink-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

Prolobjectlink 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

Prolobjectlink 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/prolobjectlink to debat over project evolution

Thanks for contributing to Prolobjectlink!

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]