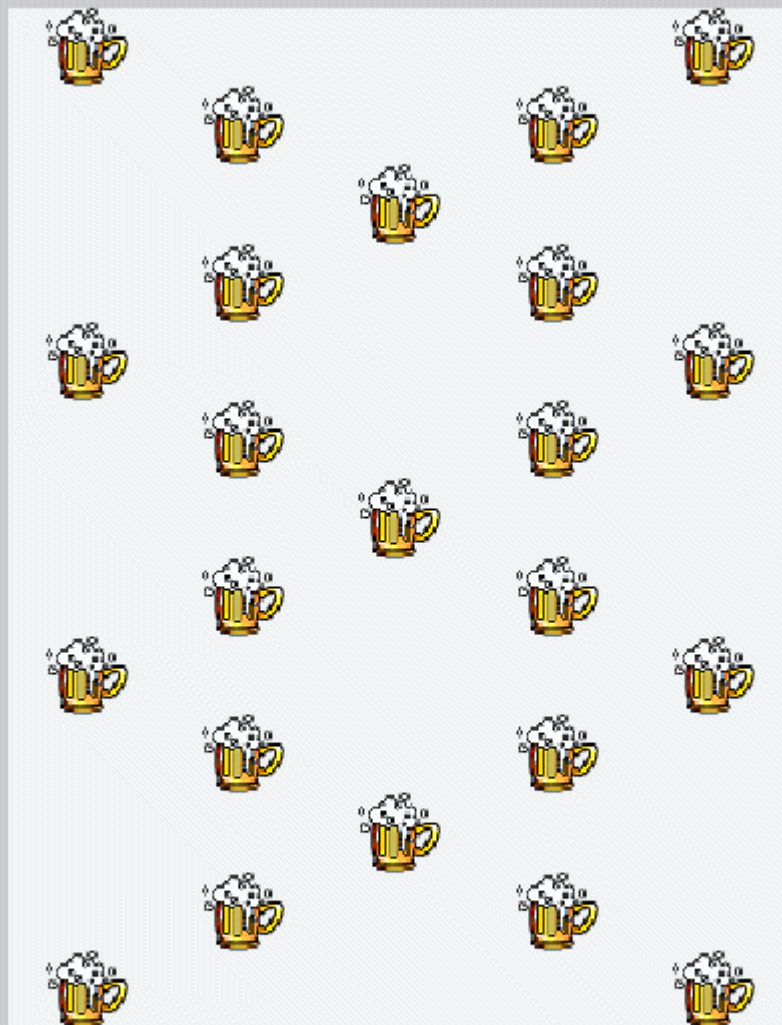




JIProlog 3.0.3 Reference Manual

v26 -13/06/2007

Copyright © 1999-2007 By Ugo Chirico. All rights reserved



Disclaimer of Liability

The content of this manual has been checked for agreement with the software described. Since deviations cannot be precluded entirely, full agreement is not guaranteed. However, the data in this manual are reviewed regularly and any necessary corrections will be included in subsequent versions. Suggestions for improvement are welcomed.

Java (TM) and all Java related trademarks and logos are trademarks of Sun Microsystems, Inc. Other company products and service names may be the trademark or service marks of others.

Thanks

Special thanks to Paulo Moura (the author of logtalk, <http://www.logtalk.otg>) for his kind help and for the hard and very precious work he made while testing JIProlog for ISO compatibility.

Many thanks to Ornella Simeoli for her good suggestions and her kind patience during the nights I spent to develop JIProlog.

Many thanks also to Dino Lupo for the time spent on testing XML/HTML extensions, Mathias Fonkam for his suggestions and corrections about this manual, Dave Hankinson for his test and suggestions about EPOC implementation, Wolfgang Chico-Töpfer for his kind help, suggestions and corrections, Yann-Gaël Guéhéneuc for his good works with JIProlog, Oluwole Okunseinde and Eugen Kupstov for their hard testing and their very good suggestions and ideas, Florian Fischer for his implementation on EPOC OS, Ulf Dittmer for his interesting feedbacks and suggestions.

Content:

<i>Abbreviations and Acronyms</i>	8
<i>About this manual</i>	8
<i>1 Introduction</i>	9
1.1 Compatibility	9
1.2 Support for PDAs.....	9
1.3 Support for Java enabled Mobile Phones	9
1.4 Interface Java \leftrightarrow Prolog	10
1.5 JIProlog Integrated Development Environment	10
1.6 JIProlog Console	10
1.7 JIProlog stand-alone interpreter	11
1.8 JIProlog Server.....	11
1.9 Support for XML - DOM	11
1.10 Support for HTML	11
1.11 Support for Java Reflection API.....	11
1.12 License.....	11
1.13 Mailing List.....	12
1.14 Contacts.....	12
<i>2.0 JIProlog Interpreter</i>	13
2.1 Features.....	13
2.1.1 Built-In Predicates.....	13
2.1.1.1 Using predicates in an extension package.....	14
2.1.1.2 Kernel Module	14
2.1.1.3 I/O Module	15
2.1.1.4 Windows Module	16
2.1.1.5 Database Module	17
2.1.1.6 ISO Exception Module	17
2.1.1.7 Sets Module	17
2.1.1.8 List Module	17
2.1.1.9 Term Module	17
2.1.1.10 Reflect Module.....	18
2.1.1.11 System Module.....	19
2.1.1.12 Operating System Module	19
2.1.1.13 XML Module	20
2.1.1.14 HTML Module	22
2.1.2 Operators	23
2.1.2 Modules.....	24
2.1.2.1 Name-based and Predicate-based Modules.....	24
2.1.2.2 Default Module Spaces	24
2.1.2.3 Declaring a Module	24
2.1.2.4 Importing a Module.....	24
2.1.2.5 Definition and Context Module	25
2.1.2.5 Calling a module predicate directly.....	25
2.2 DCG Grammar rules	25
2.3 Arithmetic Functions	25
<i>3.0 JIProlog Integrated Development Environment</i>	27
3.1 Installation	27
3.1.1 Installation for Windows OS.....	27

3.1.2 Installation for Unix/Linux.....	27
3.1.3 Installation for Mac OS	27
3.1.4 Installation on other platforms with JRE 1.2 or later	27
3.1.5 Java Enable Mobile Phones – MIDP	27
3.2 Usage	27
3.2.1 Running JIProlog IDE.....	27
3.2.1.1 Windows	27
3.2.1.2 Unix/Linux	27
3.2.1.3 Mac OS	27
3.2.1.4 Novell Netware.....	27
3.3.1 Using JIProlog IDE.....	28
3.3.1.1 Projects Frame.....	28
3.3.1.2 JIPConsole Panel.....	28
3.3.1.3 Predicates Panel.....	29
3.3.1.4 Projects Panel.....	30
3.3.1.5 JIProlog's Menu.....	30
3.3.1.6 JIProlog's Button Bar	31
4.0 JIProlog Console.....	32
4.1 Installation	32
4.1.1 Installation for Windows	32
4.1.2 Installation for Linux/Unix.....	32
4.1.3 Installation for Mac OS	32
4.1.4 Installation for Windows CE - Pocket PC2002/2003 Windows Mobile with Jeode/CreME	32
4.1.5 Installation for EPOC OS	32
4.1.6 Installation for other Java 1.1 platforms	32
4.1.7 Java Enable Mobile Phones – JIProlog2ME	33
4.2 Usage	33
4.2.1 Running JIPConsole.....	33
4.2.1.1 Windows	33
4.2.1.2 Unix/Linux	33
4.2.1.3 Mac OS	33
4.2.1.4 NetWare 5.....	33
4.2.1.5 Windows CE-PocketPC 2002/2003 (Jeode/crème)	33
4.2.1.6 EPOC OS	33
4.2.1.7 Other Java 1.1 platforms	33
4.2.2 Running JIPConsole applet	33
4.2.3 Editing and Consulting a Prolog file	33
4.2.4 Tools.....	33
5 JIProlog stand-alone interpreter	35
5.1 Running the stand-alone interpreter	35
5.1.1 Windows	35
5.1.2 Unix/Linux	35
5.1.3 Mac OS	35
5.1.4 Personal Java and Windows CE – Pocket PC2002/2003	35
5.1.5 Java Enable Mobile Phones – MIDP.....	35
5.1.6 Other Platforms	35
6 JIProlog Server	36
6.1 Running the JIProlog Server	36
6.1.1 Windows	36
6.1.2 Unix/Linux	36
6.1.3 MacOS	36
6.1.4 Personal Java and Windows CE – PocketPC 2002/2003	36
6.1.5 Java Enable Mobile Phones – MIDP.....	36
6.1.6 Other Platforms	36
7 JIProlog2ME for Java Enable Mobile Phones.....	37
7.1 Minimum Requirements.....	37

7.2 Limitations.....	37
7.3 Running the JIProlog2ME MIDlet on a J2ME Mobile Phone.....	38
7.4 Loading Extension and Custom Packages.....	39
8 JIProlog SDK.....	40
8.1 Interface between Java and Prolog.....	40
8.1.1 Java to Prolog.....	40
8.1.1.1 Parsing a Prolog term.....	40
8.1.1.2 Parsing a Prolog Stream.....	40
8.1.1.3 Synchronous calls.....	41
8.1.1.4 Asynchronous calls.....	41
8.2.2 Prolog to Java.....	41
8.2.2.1 Implementing custom built-in predicates.....	42
8.2.2.2 Managing Java Objects from Prolog.....	42
8.2.2.3 Prolog to RDBMS and JDBC.....	42
8.3 JIProlog2ME (J2ME).....	43
9 Main Built-in Predicates.....	44
9.1 Main Built-in Predicates supplied with JIProlog.....	44
10 JIPxxml Package.....	55
10.1 Predicates Exported by JIPxxml Package.....	55
xml_append_attribute/3.....	55
xml_append_child/3.....	55
xml_attribute/3.....	55
xml_cdata/2.....	55
xml_child/1.....	55
xml_comment/2.....	55
xml_document/5.....	55
xml_doctype/4.....	55
xml_doctype_id/2.....	55
xml_document_encoding/2.....	56
xml_document_root/2.....	56
xml_document_version/2.....	56
xml_element/4.....	56
xml_element_attributes/2.....	56
xml_element_attribute_by_name/2.....	56
xml_element_children/2.....	56
xml_element_child_by_name/2.....	56
xml_object/1.....	56
xml_object_name/2.....	56
xml_object_type/2.....	56
xml_object_value/2.....	57
xml_pi/3.....	57
xml_read_document/1.....	57
xml_read_document/2.....	57
xml_remove_attribute/3.....	57
xml_remove_child/3.....	57
xml_text/3.....	57
xml_write_document/1.....	57
xml_write_document/2.....	57
11 JIPxhtml Package.....	57
11.1 Predicates Exported by JIPxhtml Package.....	57
html_append_attribute/3.....	57
html_append_child/3.....	58
html_attribute/3.....	58
html_child/1.....	58
html_comment/2.....	58

html_document/3.....	58
html_doctype/3.....	58
html_doctype_specs/2.....	58
html_document_root/2.....	58
html_object/1.....	58
html_object_name/2.....	58
html_object_type/2.....	59
html_object_value/2.....	59
html_read_document/1.....	59
html_read_document/2.....	59
html_remove_attribute/3.....	59
html_remove_child/3.....	59
html_tag/4.....	59
html_tag_attributes/2.....	59
html_tag_attribute_by_name/2.....	59
html_tag_children/2.....	59
html_tag_child_by_name/2.....	59
html_text/3.....	60
html_write_document/1.....	60
html_write_document/2.....	60
12 JIPxreflect package.....	61
12.1 Predicates Exported by JIPxreflect Package.....	61
create_object/3.....	61
invoke/4.....	61
get/3.....	61
get_class/2.....	62
get_constructors/2.....	62
get_methods/2.....	62
get_fields/2.....	62
set/3.....	62
release_object/1.....	62
13 JIPxWin package.....	63
13.1 Predicates Exported by JIPxWin Package.....	63
winputbox/2.....	63
wmsgbox/1.....	63
wyesnobox/1.....	63
wtextbox/1.....	63
Appendix A: How To.....	64
How to call the interpreter synchronously.....	64
How to call the interpreter asynchronously.....	64
How to write a custom built-in predicate:	65
How to link a JDBC external database of clauses:.....	66
How to link a table	67
How to link a view specified by an SQL SELECT query	67
How to write a custom bridge to an external database of clauses:	67
How to consult a very large Prolog file:	68
How to compile and distribute a Prolog program:	68
How to implement a JIProlog client	68
Appendix C: Prolog miscellaneous	70
The standard ordering of terms.....	70
Prolog Resources	70

Internet:	70
Books:	70
<i>Appendix C: List of predicates supplied by JIProlog</i>	71

Abbreviations and Acronyms

API	Application Programming Interface
DCG	Definite Clause Grammars
DOM	Document Object Model
EJB	Enterprise Java Beans
IDE	Integrated Development Environment
J2ME	Java 2 Micro Edition
JNI	Java Native Interface
JSP	Java Server Pages
RDBMS	Relational DataBase Management System
WAM	Warren Abstract Machine
XML	eXtended Markup Language
MIDP	Mobile Information Device Profile
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
JVM	Java Virtual Machine

About this manual

JIProlog Reference Manual is designed for experienced Java and Prolog developers. Both Prolog and Java are not described in this manual. For such languages refer to related literature shown in the reference section of the manual.

Examples shown in this manual are distributed together with the JIProlog SDK available online at JIProlog web site:

`http://www.ugosweb.com/jiprolog`.

1 Introduction

JIProlog - Java Internet Prolog is a cross-platform pure Java 100% Prolog interpreter which integrates Prolog and Java languages in a very easy way. JIProlog allows to call Prolog predicates from Java without dealing with native code (JNI) and it allows to invoke Java methods from Prolog in the same way you call predicates. JIProlog is designed to work with any Java 1.1 platform or later, including Java enabled application servers, Java enabled browsers, PDAs compliant with Java 1.1 platform or Personal Java 1.1 or later and mobile devices compliant with MIDP 1.0/2.0. JIProlog is supplied with a powerful Integrated Development Environment which help to edit, consult, run and test Prolog and Java programs. Finally JIProlog provides a set of java classes, that forms the JIProlog API, which allows to:

- call Prolog Predicates from Java without dealing with native code (JNI);
- call Java methods and classes from Prolog without dealing with foreign language interfaces;
- write custom Prolog predicates in Java in order to extend the default set of built-in predicates;
- extend the graphical user interface of a Prolog program implementing windows and dialogs in Java as custom built-in predicates;
- integrate databases in Prolog through a JDBC \leftrightarrow Prolog bridge, treating tables and queries as Prolog predicates

1.1 Compatibility

JIProlog is compliant with the major prolog interpreters. It supports Edimburgh specifications, most of the ISO specification, and many of the most common and ISO built-in predicates (see set of predicates). However may be some little differences between JIProlog implementation and ISO other standard implementations Syntax and standard operator declarations confirm to the 'Edinburgh standard'.

Most built in predicates are compatible with those described in "*Programming in Prolog*" [CL87], [CL03].

ISO compliant predicates are based on "*Prolog: The Standard*", [DE96]

JIProlog is compliant with Java 1.1 Platform or later, J2ME (Personal Java 1.1 or later, MIDP 1.0, MIDP 2.0-CLDC1.0/1.1).

As of this writing, it has been successfully tested on the following platforms:

- Windows/Linux: JRE 1.1.x, 1.2.x, 1.3.x, 1.4.x, 1.5.x, 1.6.x;
- MIDP 2.0-CLDC1.0 Mobile Devices:
J2ME Wireless Toolkit 2.2, Nokia Series 40, 60 (2nd and 3rd) and 80, SonyEricsson P900, P910, Z1010, K800, Motorola Razr V3, Siemens CXT65, SK65.
- Windows CE, Pocket PC 2002/2003: Jeode JVM, CrEME JVM on HP iPAQ, Pocket Loox;
- Windows Mobile 2003/2005: Jeode JVM, CrEME JVM on HP iPAQ, HTC, QTEK, i-mate.
- EPOC OS on psion-m5;
- Apache Tomcat;
- Internet Explorer 4.x, 5.x, 6.0, 7.0;
- Netscape Navigator 4.5, 4.7, 6.0;
- Mozilla 1.x;
- Firefox 1.0, 2.0;

1.2 Support for PDAs

JIProlog is compliant with Personal Java 1.1 or later thus it works with any PDA that has a Personal Java 1.1 VM like Jeode, CrEME, chaiVM, JV-Lite2, etc. Some of PDAs supported are: HP iPAQ, HTC, QTEK, i-mate, Fujitsu-Siemens Pocket Loox and SimPad, HHP, NEC, Samsung, Sharp Eletronics, Toshiba, psion m5. Potentailly it works on any other device with Windows CE/Pocket PC 2002/2003 or Windows Mobile 2003/2005).

1.3 Support for Java enabled Mobile Phones

JIProlog is compliant with J2ME MIDP 2.0 - CLDC1.0 or later, thus it works with any Java Enabled Mobile Phone supporting at least MIDP 2.0 with at least 400 KB of heap memory available for execution.

It has been successfully tested on the following devices:

- J2ME Wireless Toolkit 2.2,
- Nokia Series 40, 60 (2nd and 3rd) and 80: 6020, 6021, 6600, 6800, E65, 6080.
- SonyEricsson P900, P910, Z1010, K800,
- Motorola Razr V3,
- Siemens CXT65, SK65.

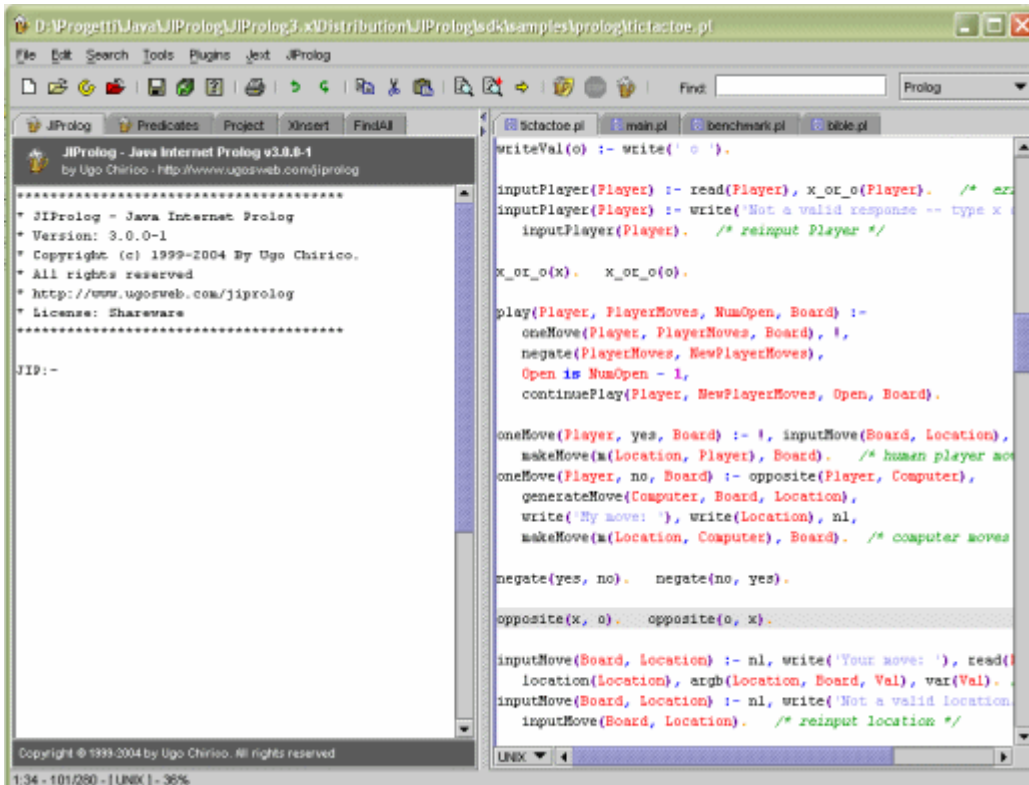
1.4 Interface Java \leftrightarrow Prolog

JIProlog comes with a complete API to link Prolog and Java languages from both sides. The API is composed of three parts: the first one regards the way to invoke Prolog from Java; the second one deals with the means to call Java methods from Prolog by implementing custom Prolog built-in predicates as Java classes or by calling directly Java methods through Java Reflection API; the third one treats the way to link Prolog predicates with tables in an external databases via JDBC \leftrightarrow Prolog bridge..

A complete description of the JIProlog API is available in the in SDK as javadoc formatted documentation.

1.5 JIProlog Integrated Development Environment

JIProlog is supplied with an Integrated Development Environment based on Jext editor (<http://www.jext.org>) Such an IDE supplies many interesting features to edit, compile, manage files and projects in different formats and languages (such as java, perl, C/C++, html). It gives also a syntax colorization for Prolog language as shown in the following snapshot:



JIProlog IDE requires at least JRE 1.5. For earlier JRE or for other poor platforms see JIPConsole.

1.6 JIProlog Console

JIProlog Console is a complete PureJava 100% Prolog console that allows editing, running and tracing Prolog programs. JIProlog Console can be run as a Java application on any Java 1.1 platform such as Windows, Unix/Linux, MacOS, Windows CE, Pocket PC 2002/2003, Windows Mobile 2003/2005, EPOC OS etc. or can be run in a html page as an applet. The following pictures shows a snapshot:



1.7 JIProlog stand-alone interpreter

JIProlog stand-alone interpreter is a java application that runs Prolog programs as stand-alone applications.

1.8 JIProlog Server

JIProlog Server is a daemon that brings JIProlog Interpreter's functionalities over the network via TCP/IP, supplying, in this way, any remote client with an interface to submit queries and to receives any related solutions.

Clients should be implemented using the a special class supplied with the API that manages the client-side protocol as described in the section "How to implement a JIProlog client" in the appendix A. The following picture is a screenshot of JIProlog Server:



1.9 Support for XML - DOM

The extension package `jipxxml` provides a set of Prolog predicates implementing a DOM-like API for parsing and creating XML documents. Each DOM object is mapped to a corresponding predicate and has a set of rules to manage it. The result is an API to manage DOM objects in Prolog. See the appendix B.

1.10 Support for HTML

The extension package `jipxhtml` provides a set of Prolog predicates for parsing and creating HTML 4.0 documents. Each HTML object is mapped to a corresponding predicate and has a set of rules to manage it. See the appendix C.

1.11 Support for Java Reflection API

The extension package `jipxreflect` provides a set of Prolog predicates that maps the functionalities of the Java Reflection API. The package allows creating Java objects, invoking object's and class's methods, retrieving values from object's fields and so on.

1.12 License

See JIProlog license agreement in the file `license.html` or visit the page <http://www.ugosweb.com/jiprolog/license.html>

More information about licensing and distribution are available at:

<http://www.ugosweb.com/jiprolog/licenseinfo.html>

1.13 Mailing List

To receive news and updates about JIProlog subscribe to JIProlog newsletter from JIProlog web site:

<http://www.ugosweb.com/jiprolog>

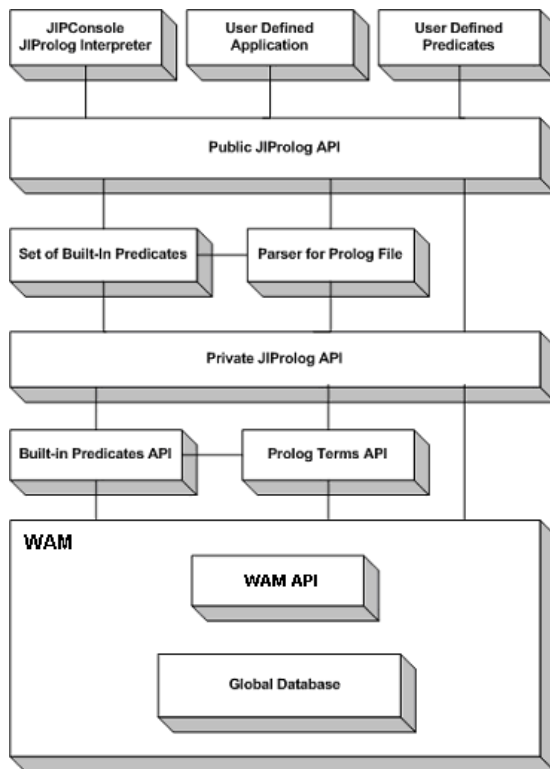
1.14 Contacts

Contact me at the address ugos@ugosweb.com to receive any kind of information about JIProlog.

If you discover a bug or you have some questions or want some explanations about JIProlog, please don't hesitate to write me.

2.0 JIProlog Interpreter

JIProlog Interpreter is based on a WAM (*Warren Abstract Machine*) implemented using a LISP-like algorithm to manage queries. It is composed of a set of Java classes implementing the WAM and the typical Prolog mechanisms (such as backtracking, cutting, etc.), a manager for built-in predicates, a parser for the Prolog language and an API by which other Prolog predicates can be implemented in Java. It also implements a mechanism to call Prolog predicates from any Java classes and vice versa, to invoke Java classes from Prolog code. The following picture shows briefly the JIProlog's architecture:



2.1 Features

JIProlog supplies most of the predicates defined in the ISO Prolog standard and in the Edinburgh specification. However may be some little differences between JIProlog implementation and the standard implementations.

2.1.1 Built-In Predicates

Most built-in predicates are compatible with those described in “*Programming in Prolog*” [CL87], [CL03].

ISO Prolog compliant predicates are based on “*Prolog: The Standard*”, [DE96].

Built-in predicates are divided in to different modules called *extension packages*.

Packages currently available are shown in the following table:

Package/Module	Description
jipxio.jar	I/O ISO predicates
jipxexception.jar	ISO Exception Handling predicates
jipxdb.jar	Database Connectivity (JDBC) predicates
jipxwin.jar	Windows predicates
jipxxml.jar	XML DOM-like predicates
jipxhtml.jar	HTML predicates
jipxreflect.jar	Predicates for Java Reflection API
jipxsystem.jar	Operating System related predicates
jipxsets.jar	Sets related predicates
jipxterm.jar	Terms management and conversion predicates
list.pl	List related predicates

sys.pl	Other system predicates
--------	-------------------------

2.1.1.1 Using predicates in an extension package

JIProlog IDE and JIProlog Console automatically load all extension packages.

If you don't use the former neither the latter, to use a predicate provided by an extension package it is necessary to load such a package by the predicate `load_library/1`. This predicate loads the Java classes related to the predicates exported by the library and consults the file `init.pl` (if the package contains any) to make the necessary initialization steps.

After the predicates are loaded they can be called as normal built-in predicates.

For example to call the predicate `see/2` in the package `jipxio.jar` type the following:

```
JIP:-load_library('jipxio.jar') % this is not needed with JIPrologIDE and JIPConsole
Yes
```

```
JIP:-see('MyFile', Handle)
Yes
Handle = #12345
```

Note: `load_library/1` needs the path to `.jar` file (absolute, relative or an URL)

2.1.1.2 Kernel Module

Kernel module is the default module always loaded by JIProlog interpreter at start-up.

The following is the list of predicates defined in the kernel:

```
expand_term/2
list/1
= /2
use_module/1
@=/2
\+/1
\=/2
@>=/2
[!/0
!]/0
is_list/1
string/1
@=</2
\==/2
number/1
append/3,nil/1
assert/1
asserta/1
assertz/1
@</2
op/3
retractall/1
;/2
simple/1
unload/1
->/2
false/0
not/1
callable/1,
phrase/3,
'C'/3
true/0
phrase/2
char/1
^/2
atomic/1
>=/2
compound/1
=</2
=\=/2
nonvar/1
@>/2
current_predicate/1
is/2
current_functor/2
nospy/1
clause/2
==/2
```

```

searchpath/1
!/0
current_atom/1
var/1
=../2
chars/1
fail/0
>/2
</2
module_transparent/1
retract/1
unconsult/1
float/1
ver/1
notrace/0
compile/1
consult/1
write/1
$error/1
spy/1
abort/0
integer/1
dynamic/1
load/1
garbage_collect/0
xcall/2
$current_query_handle/1
extern/3
nodebug/0
abolish/1
arg/3
load_library/1
compare/3
current_op/3
length/2
$free/1
ground/1
multifile/1
nl/0
:=/2
write_canonical/1
atom/1
debugging/0
ensure_loaded/1
functor/3
notify/2
trace/0
predicate_property/2.

```

See the appendix A for a comprehensive explanation of the supported built-in predicates.

2.1.1.3 I/O Module

I/O module is implemented in the package `jipxio.jar` and defines the ISO Prolog predicate for I/O. The following is the list of predicates defined in such a package:

```

see/1
see/2
seen/0
seen/1
seeing/1
seeing/2
read/1
read/2
read_term/2
read_term/3
read_clause/1
read_clause/2
get0/1
get0/2
get/1
get/2
get_byte/1
get_byte/2
get_code/1
get_code/2
get_char/1
get_char/2
peek_byte/1

```

```

peek_byte/2
peek_code/1
peek_code/2
peek_char/1
peek_char/2
tell/1
tell/2
told/0
told/1
telling/1
telling/2
write/2
writeq/1
writeq/2
writeln/1
writeln/2
write_canonical/2
write_term/2
write_term/3
put/1
put/2
put_byte/1
put_byte/2
put_code/1
put_code/2
put_char/1
skip/1
skip/2
put_char/2
nl/1
flush_output/0
flush_output/1
open/3
open/4
close/1
print/1
print/2
display/1
display/2
close/2
at_end_of_stream/0
at_end_of_stream/1
stream_property/2
access_file/2
exists_file/1
exists_directory/1
same_file/2
working_directory/2
chdir/1
absolute_file_name/2
is_absolute_file_name/1
file_attributes/7
file_name_extension/2
size_file/2
time_file/2
file_directory_name/2
file_base_name/2
delete_file/1
delete_directory/1
rename_file/2
dir/0
dir/1
make_directory/1
current_stream/3
cd/1
current_output/1
current_input/1
set_output/1
set_input/1

```

2.1.1.4 Windows Module

Windows module is implemented in the package `jipxwin.jar`. It provides the set of predicates for creating messageboxes. It defines the following predicates:

```

wmsgbox/1
winputbox/2
wyesnobox/1
wtextbox/1

```


2.1.1.5 Database Module

Database module is implemented in the package `jipxdb.jar`. It provides the Prolog \leftrightarrow JDBC bridge to import in Prolog any JDBC database and defines the following predicates:

```
declare_extern/3
```

2.1.1.6 ISO Exception Module

ISO Exception module is implemented in the package `jipxexception.jar`. It provides the ISO compliant exception handling by the predicates:

```
catch/3  
throw/1
```

2.1.1.7 Sets Module

Sets module is implemented in the package `jipxsets.jar`. It provides several predicates for managing sets of Prolog terms:

```
is_set/1  
list_to_set/2  
intersection/3  
union/3  
subset/2  
difference/3  
symdiff/3  
bsort/3  
qsort/3  
msort/3  
sort/2  
merge/2  
merge_set/2  
remove_duplicates/2  
findall/3  
setof/3  
bagof/3
```

2.1.1.8 List Module

List module is implemented in the package `list.pl` and provides several predicates for managing lists:

```
member/2  
member/3  
memberchk/2  
select/3  
nextto/3  
delete/3  
nth0/3  
nth1/3  
last/2  
reverse/2  
permutation/2  
flatten/2  
sumlist/2  
numlist/3
```

2.1.1.9 Term Module

Term module is implemented in the package `jipxterm.jar` and provides several predicates for managing and converting Prolog Terms:

```
numbervars/3  
free_variables/2  
copy_term/2  
name/2  
char_code/2  
atom_codes/2  
atom_chars/2  
number_codes/2  
number_chars/2  
atom_number/2  
atom_concat/3  
concat_atom/2  
concat_atom/3  
upcase/1  
downcase/1  
upcase_char/2
```

```

upcase_chars/2
downcase_char/1
downcase_chars/2
upcase_atom/2
downcase_atom/2
string_to_atom/2
string_to_list/2
string_length/2
string_concat/3

```

2.1.1.10 Reflect Module

Reflect module is implemented in the package `jipxreflect.jar` and supplies a set of custom built-in predicates that allows creating Java objects, invoking their methods, retrieving values from their fields and so on. Such predicates are implemented using the functionalities given by Java Reflection API:

```

create_object/3
release_object/1
invoke/4
get/3
set/3
get_class/2
get_constructors/2
get_methods/2
get_fields/2

```

The following table shows the map between operations and the supplied predicates.

Operation	Prolog predicate
Creating an object	<code>create_object(ConstructorProto, Params, ObjHandle)</code>
Invoking an object method	<code>invoke(ObjHandle, MethodProto, Params, RetVal)</code>
Getting the value of an object field	<code>get(ObjHandle, FieldName, Val)</code>
Setting the value of an object field	<code>set(ObjHandle, FieldName, Val)</code>
Invoking a class method (static)	<code>invoke(ClassName, MethodProto, Params, RetVal)</code>
Getting the value of a class field (static)	<code>get(ClassName, FieldName, Val)</code>
Setting the value of a class field (static)	<code>set(ClassName, FieldName, Val)</code>
Releasing an object	<code>release_object(ObjHandle)</code>
Getting the object class	<code>get_class(ObjHandle, ClassName)</code>
Getting a list of object constructors	<code>get_constructors(ObjHandle, MethodList)</code>
Getting a list of object methods	<code>get_methods(ObjHandle, MethodList)</code>
Getting a list of object field	<code>get_field(ObjHandle, MethodList)</code>
Getting a list of class methods	<code>get_methods(ClassName, MethodList)</code>
Getting a list of class field	<code>get_field(ClassName, MethodList)</code>

The following example show how to create an object `java.lang.String` containing the string *"Hello World"* and how to call its methods:

```

JIP:-load_library('jipxreflect.jar') % this is not needed with JIPrologIDE and JIPConsole
Yes

JIP:-create_object('java.lang.String'('java.lang.String'), ['Hello World'], Handle).
Yes
Handle = #12345

JIP:-invoke('#12345',substring(int, int), [0, 5], RetVal).
Yes
RetVal = Hello

JIP:-get_class('#12345', ClassName).
Yes
ClassName = java.lang.String

JIP:-release_object('#12345').
Yes

```

In the way from Prolog to Java, Prolog terms passed as parameters to the method call are translated in the corresponding Java objects. The same occurs with the values returned by the methods that are translated from Java objects in the corresponding Prolog terms. The following tables shows the translation rules:

Translation from Prolog to Java	
Prolog Term	Java Object
Integer	Integer if the value is < Integer.MAX_VALUE
	Long if the value is >= Integer.MAX_VALUE
Float	Double
Atom	String
String	String
true false	Boolean
List Functor Clause	Corresponding JIPTerm object

Translation from Java to Prolog	
Java Object	Prolog
Short Integer Long	Integer
Double Float	Double
String	Atom
Boolean	true false
Char	Char
Void Null	[] (NIL)
Other objects	Handle (#132423)

2.1.1.11 System Module

System module is implemented in the file `sys.pl` and supplies a set of predicates to be compliant with the major Prolog Interpreters:

```
meta_predicate/1
load_files/1
abolish_files/1
listing/1
listing/0
current_predicate/2
repeat/0
repeat/1
one/1
once/1
checklist/2
maplist/3
sublist/3
forall/2
apply/2
call/1
call/2
call/3
call/4
call/5
ignore/1
```

2.1.1.12 Operating System Module

System Extension module is implemented in the package `jipxsystem.jar` and supplies a set of custom built-in predicates to call OS specific functions:

```
sleep/1
shell/2
shell/1
statistics/0
get_time/1
time/1
time/4
time/5
time/7
time/8
```

```

convert_time/8
date/3
date/4
ms/2

```

2.1.1.13 XML Module

XML module is implemented in the package `jipxxml.jar` and provides a set of Prolog predicates implementing a DOM-like API for parsing and creating XML documents. Each DOM object is mapped to a corresponding predicate and has a set of rules to manage it. The result is an API to manage DOM objects in Prolog:

```

xml_write_document/1
xml_write_document/2
xml_read_document/1
xml_read_document/2
xml_object/1
xml_child/1
xml_object_name/2
xml_object_value/2
xml_object_type/2
xml_document/2
xml_document/5
xml_document_root/2
xml_document_version/2
xml_document_encoding/2,
xml_doctype/3,
xml_doctype/4
xml_doctype_id/2
xml_element/3
xml_element/4
xml_element_attributes/2
xml_element_attribute_by_name/3
xml_element_children/2
xml_element_child_by_name/3
xml_append_attribute/3
xml_remove_attribute/3
xml_append_child/3
xml_remove_child/3
xml_attribute/2
xml_attribute/3
xml_cdata/1
xml_cdata/2
xml_text/1
xml_text/2
xml_comment/1
xml_comment/2
xml_pi/1
xml_pi/2.

```

The following table shows the mapping between DOM objects and Prolog predicates:

DOM Object	Prolog structure	Related Prolog predicates
Document	<code>xml_document(Prolog, Root)</code> ¹	<code>xml_document/5</code> <code>xml_document_root/2</code> <code>xml_document_version/2</code> <code>xml_document_encoding/2</code> <code>xml_object/1</code> <code>xml_object_type/2</code>
DocumentType	<code>xml_doctype(Name, Id, Content)</code> ²	<code>xml_doctype/4</code> <code>xml_doctype_id/2</code> <code>xml_object/1</code> <code>xml_object_type/2</code> <code>xml_object_name/2</code>
Element	<code>xml_element(Name, Attributes, Children)</code>	<code>xml_element/4</code> <code>xml_element_attributes/2</code> <code>xml_element_children/2</code> <code>xml_element_attribute_by_name/3</code> <code>xml_element_child_by_name/3</code> <code>xml_object/1</code> <code>xml_object_type/2</code> <code>xml_object_name/2</code>

¹ Prolog has the following form: `[[version = V, encoding = E], DocType]` where encoding is optional and DocType can be `[]`.

² Id specify both PUBLIC or SYSTEM ids. Content specifies the rest of DTD definitions in the form of a Prolog atom

Attribute	xml_attribute(Name, Value)	xml_attribute/3 xml_object/1 xml_object_type/2 xml_object_name/2 xml_object_value/2
CDATA	xml_cdata(Content)	xml_cdata/2 xml_object/1 xml_object_type/2 xml_object_value/2
Text	xml_text(Content)	xml_text/2 xml_object/1 xml_object_type/2 xml_object_value/2
ProcessingInstruction	xml_pi(Name, Content)	xml_pi/3 xml_object/1 xml_object_type/2 xml_object_name/2 xml_object_value/2
Comment	xml_comment(Content)	xml_comment/2 xml_object/1 xml_object_type/2 xml_object_value/2

The objects Entity, EntityReference, Notation are not supported. The predicate xml_doctype/3 embed in the term Content their values as atom. For example a DTD like this:

```
<!DOCTYPE personnel [
<!ELEMENT personnel (person)+>
<!ELEMENT person (name,email*,url*,link?)>
<!ELEMENT name ((family,given)|(given,family))>
...
<!NOTATION gif PUBLIC "-//APP/Photoshop/4.0" 'photoshop.exe'>
]>
```

is transformed to:

```
xml_doctype(personnel,[], `<!DOCTYPE personnel [<!ELEMENT personnel (person)+> <!ELEMENT
person (name,email*,url*,link?)> <!ELEMENT name ((family,given)|(given,family))> ...
<!NOTATION gif PUBLIC "-//APP/Photoshop/4.0" 'photoshop.exe'>]>`).
```

To read an XML document use the predicates xml_read_document/1 or xml_read_document/2.

They parse an XML document that should be *well-formed* and *valid*³. xml_read_document/1 parses the current input stream (set by see/1) and unifies its Prolog representation with the argument. xml_read_document/2 parses the input stream pointed to by the handle in the first argument (set by see/2 in jipxio package) and unifies its Prolog representation with the second argument.

Ex.

```
JIP:-load_library('jipxxml.jar') % this is not needed with JIPrologIDE and JIPConsole
Yes
```

```
JIP:-see('myfile.xml').
Yes
```

```
JIP:-xml_read_document(X).
Yes
X = xml_document([[version = 1.0, encoding = UTF-8], [], xml_element(personnel
...

```

```
JIP:-seen.
Yes
```

```
JIP:-load_library('jipxio.jar') % this is not needed with JIPrologIDE and JIPConsole
Yes
```

```
JIP:-see('myfile.xml', Handle), xml_read_document(Handle, X), seen(Handle).
Yes
Handle = 12345
X = xml_document([[version = 1.0, encoding = UTF-8], [], xml_element(personnel
...

```

See also the predicate browse_xml_file/1 and the predicate create_xml_file/1. the file xml_sample.pl

³ The parser checks the validity of the XML document through DTD, thus the xml document should be necessarily well formed and valid.

2.1.1.14 HTML Module

HTML module is implemented in the package `jipxsystem.jar` and provides a set of Prolog predicates for parsing and creating HTML 4.0 documents. Each HTML object is mapped to a corresponding predicate and has a set of rules to manage it:

```
html_write_document/1
html_write_document/2
html_read_document/1
html_read_document/2
html_object/1
html_child/1
html_object_name/2
html_object_value/2
html_object_type/2
html_document/2
html_document/3
html_document_root/2
html_doctype/2
html_doctype/3
html_doctype_specs/2
html_tag/3
html_tag/4
html_tag_attributes/2
html_tag_attribute_by_name/3
html_tag_children/2
html_tag_child_by_name/3
html_append_attribute/3
html_remove_attribute/3
html_append_child/3
html_remove_child/3
html_attribute/2
html_attribute/3
html_text/1
html_text/2
html_comment/1
html_comment/2
```

The following table shows the mapping between HTML objects and Prolog predicates:

HTML Object	Prolog structure	Related Prolog predicates
HTMLDocument	<code>html_document(Prolog, Root)</code> ⁴	<code>html_document/3</code> <code>html_document_root/2</code> <code>html_object/1</code> <code>html_object_type/2</code>
HTMLDocumentType	<code>html_doctype(Name, IdSpecs)</code>	<code>html_doctype/3</code> <code>html_doctype_specs/2</code> <code>html_object/1</code> <code>html_object_type/2</code> <code>html_object_name/2</code>
HTMLTag	<code>html_tag(Name, Attributes, Children)</code>	<code>html_tag/4</code> <code>html_tag_attributes/2</code> <code>html_tag_children/2</code> <code>html_tag_attribute_by_name/3</code> <code>html_tag_child_by_name/3</code> <code>html_object/1</code> <code>html_object_type/2</code> <code>html_object_name/2</code>
HTMLTagAttribute	<code>html_attribute(Name, Value)</code>	<code>html_attribute/3</code> <code>html_object/1</code> <code>html_object_type/2</code> <code>html_object_name/2</code> <code>html_object_value/2</code>
HTMLText	<code>html_text(Content)</code>	<code>html_text/2</code> <code>html_object/1</code> <code>html_object_type/2</code> <code>html_object_value/2</code>
HTMLComment	<code>html_comment(Content)</code>	<code>html_comment/2</code> <code>html_object/1</code> <code>html_object_type/2</code> <code>html_object_value/2</code>

To read an HTML document use the predicates `html_read_document/1` or `html_read_document/2`.

They parse an HTML document. `html_read_document/1` parses the current input stream (set by `see/1`) and unifies its Prolog representation with the first argument. `html_read_document/2` parses the input stream

⁴ Prolog has the following form: `[DocType]` where `DocType` can be `[]` if HTML document has no prolog.

pointed to by the handle in the first argument (set by `see/2` in `jipxio` package) and unifies its Prolog representation with the second argument.

Ex.

1)

```
JIP:-load_library('jipxhtml.jar') % this is not needed with JIPrologIDE and JIPConsole
Yes
```

```
JIP:-see('myfile.html').
Yes
```

```
JIP:-html_read_document(X).
Yes
X = html_document([], html_tag(html, [], [html_tag(head, [], [], ...
```

```
JIP:-seen.
Yes
```

2)

```
JIP:-load_library('jipxio.jar') % this is not needed with JIPrologIDE and JIPConsole
Yes
```

```
JIP:-see('myfile.html', Handle), html_read_document(Handle, X)
seen(Handle).
Yes
Handle = 12345
X = html_document([], html_tag(html, [], [html_tag(head, [], [] ...
```

See also the predicate `browse_html_file/1` and `create_html_file/1` in the file `html_sample.pl`

2.1.2 Operators

Syntax and standard operator declarations agree with the ISO standard.

The following table shows the operator declarations:

Name	Associativity	Precedence
dynamic	fx	1150
rem	xfx	300
multifile	fx	1150
\	yfx	500
>>	yfx	400
@>=	xfx	700
>=	xfx	700
\+	fy	900
spy	fy	900
>	xfx	700
=	xfx	700
\==	xfx	700
<	xfx	700
i	xfy	1100
mod	xfx	300
:	xfy	600
:=	xfx	700
/\	yfx	500
@>	xfx	700
@=	xfx	700
@<	xfx	700
is	xfx	700
module_transparent	fx	1150
-->	xfx	1200
/	yfx	400
->	xfy	1000
-	yfx	500
-	fx	500
,	xfy	1000
+	yfx	500
+	fx	500
==	xfx	700
@=<	xfx	700
*	yfx	400
}	xf	900
=<	xfx	700
{	fy	901

not	fy	900
=\=	xfx	700
nospy	fy	900
xor	yfx	400
:-	fx	1200
:-	xfx	1200
**	xfy	200
cd	fx	900
^	xfy	200
<<	yfx	400
//	yfx	400
\	fx	500
\=	xfx	700
?-	fx	1200
meta_predicate	fx	1150
=..	xfx	700

2.1.2 Modules

In many Prolog systems the predicate space was flat. This means that all predicates are defined in the same database and, thus, it will be visible from all source modules that compose an application.

Developing large applications it is desirable instead that each source module has it's own predicate space and only some predicates it defines will be visible (public) to all source modules, while the other defined predicates will be private. This approach allows the programmer to use local names for private predicates without worrying about name conflicts with other modules.

2.1.2.1 Name-based and Predicate-based Modules

There are two approaches commonly used in Prolog to deal with modules.

The first one is the name-based modules where each private atom or functor is prefixed with the module name while the public atoms are not. The second one is the predicate-based modules where each module actually implements its own predicate space.

Each of the mentioned approach has pros and cons (see) however the first approach can be ever used by the programmer whatever is the Prolog environment while the second requires a Prolog environment that support predicate-based modules.

JIProlog v3.0 uses the predicate-based approach while the version 2.0 used named-based approach.

2.1.2.2 Default Module Spaces

JIProlog v3.0 defines three default module spaces:

- 1) `kernel` space contains all internal predicates used by JIProlog Interpreter;
- 2) `system` space contains all system predicates;
- 3) `user` space will contain all predicates defined without any module specification and all predicates defined in modules as public.

2.1.2.2 Declaring a Module

A module consists of an header that declares the module's name and what are the public predicates by using `module/2`, followed by the predicate definitions:

```
:- module(myModule, [myPublicPredicate/1]).
```

```
myPublicPredicate(X):-  
    myPredicate(X).
```

```
myPredicate(1).
```

`myPublicPredicate` is declared as public predicate and will be added to the user module in order to be visible to other source modules. `myPredicate` instead will be private.

2.1.2.3 Importing a Module

Importing a module means compile a module importing all public predicates.

To import a module use `use_module/1`:

```
:- use_module(myModule).
```


2.1.2.4 Definition and Context Module

Each predicate of the program is assigned a module, called it's definition module, that is the module in which the predicate was originally defined.

Each active goal has a context module assigned to it, that is module space where to search for predicates needed to prove the goal.

By default, the context module is the definition module of the predicate unless it is a meta-predicate declared as module transparent using `module_transparent/1`. For a meta-predicate the context module is the context module of the goal that invoked it.

2.1.2.5 Calling a module predicate directly

There are cases in which may be useful to overrule this schema of managing modules and explicitly call a predicate in some module or assert explicitly in some module. The former is useful to call a predicate in some module even if it is declared as private. The latter is useful to create and modify dynamic modules.

For this purpose, there is the predicate `:/2`. Each call to a predicate check whether the call is of the form `module:goal`. If so, `goal` is searched for in `module` instead of the goal's context module:

```
JIP:- assert(myModule:myPredicate). % asserts myPredicate/0 into module myModule
Yes
```

```
JIP:- myModule:assert(myPredicate). % the same
Yes
```

```
JIP:- myPredicate
no
```

```
JIP:- myModule:myPredicate. % calls myPredicate/0 in module myModule
Yes
```

2.2 DCG Grammar rules

Definite Clause Grammar (DCG) may be used to define the syntax of a language and to define a parser for that language. A Prolog program may contain one or more grammar rules.

A grammar rule takes the form:

```
head --> body.
```

Where `head` is a non-terminal symbol optionally followed by a terminal symbol. The body of the grammar rule is a sequence of terminals, non-terminals or grammar conditions, each separated by commas or semi-colons. A grammar condition is a sequence of Prolog call terms surrounded by curly brackets ('{' and '}').

The following is an example of grammar rules:

```
sentence --> noun_ph.
verb_ph.
verb_ph --> verb.
noun_ph.
verb --> [likes] ; [hates].
noun_ph --> determiner.
noun.
determiner --> [the].
noun --> [boy] ; [dog].
```

The body of a grammar rule can contain three types of terms. A compound term interpreted as a reference to a grammar-rule. Code between `{...}` is interpreted as a reference to ordinary Prolog code and finally, a list is interpreted as a sequence of literals.

Grammar rule-sets are called using the builtin predicates `phrase/2` and `phrase/3`.

2.3 Arithmetic Functions

JIProlog implements the following arithmetic functions:

Function	Definition
<code>+/2</code>	sum
<code>-/2</code>	subtraction
<code>//2</code>	quotient
<code>*/2</code>	multiplication
<code>-/1</code>	minus
<code>sin/1</code>	sin

cos/1	cosin
asin	arcsin
acos	arccosin
tan	tangent
atan	arctangent
log	natural logarithm \log_e
exp	exponential e^x
abs	absolute value
sqrt	square root
int	integer value nearer to
integer	integer value nearer to
pow	power x^y
mod	modulus (rest)
min	minimum value
max	maximum value
rand	random number
e	value of e
pi	value of π
cputime	current cpu time
ceil	smallest integer larger than arg
ceiling	smallest integer larger than arg
floor	largest integer below argument
float	convert to float
float_fractional_part	fractional part of a float
float_integer_part	integer part of a float
sign	sign of value
\	bitwise negation
random	generate random number
//	integer division
rem	remainder of division
/\	bitwise and
\/	bitwise or
<<	bitwise left shift
>>	bitwise right shift
round	round to nearest integer
truncate	truncate float to integer
xor	bitwise exclusive or

3.0 JIProlog Integrated Development Environment

JIProlog is supplied with an Integrated Development Environment for Prolog based upon [Jext 1.5](http://www.jext.org) editor (<http://www.jext.org>).

Jext is a powerful, pure Java text editor primarily targeted for use by programmers. For this reason, Jext provides many useful programming functions: syntax colorization, auto indentation, a source code browser, a class browser, and an integrated console. Jext is fully customizable with the help of plugins and XML configuration files.

Because of graphical advanced features given by Jext JIProlog IDE runs on any Java Virtual Machine that supports Sun's JVM 1.2 specification or later. Thus, as of this writing, it is available only for Windows and Linux Platforms.

3.1 Installation

Download the latest version of the installation package for your operating system from the following web page:

<http://www.ugosweb.com/jiprolog/Download.html>

3.1.1 Installation for Windows OS

Run the file `JIPSetupIDE.exe` and follow the instructions given by the installation program.

3.1.2 Installation for Unix/Linux

Unzipping `JIPPrologIDE.zip` in a directory such as `usr/JIPProlog`

3.1.3 Installation for Mac OS

Unzipping `JIPPrologIDE.zip` in a directory such as `JIPProlog`

3.1.4 Installation on other platforms with JRE 1.2 or later

Unzipping `JIPPrologIDE.zip` in a directory such as `usr/JIPProlog`

3.1.5 Java Enable Mobile Phones – MIDP

Currently JIProlog IDE is not supported for this platform.

3.2 Usage

Because JIProlog IDE is based upon Jext editor, here we describe only the main features related to JIProlog Interpreter. For a comprehensive explanation about how to use Jext and all its interesting features refer to the documentation supplied in the directory: `<InstallationDirectory>/docs`

Here we assume that the reader has a minimal knowledge about Prolog language and knows what's mean running a query. For such a stuff refer to the book "*Programming in Prolog*" [CL03] or visit the site:

<http://pauillac.inria.fr/~deransar/prolog/docs.html>

3.2.1 Running JIProlog IDE

3.2.1.1 Windows

Click on Start→Programs→JIProlog IDE

3.2.1.2 Unix/Linux

Run `<InstallationDirectory>/bin/jext`

3.2.1.3 Mac OS

Run `<InstallationDirectory>/bin/jext`

3.2.1.4 Novell Netware

Run `<InstallationDirectory>/bin/jext.ncf`

3.3.1 Using JIProlog IDE

JIProlog IDE is composed by two frames: the Projects Frame to manage project files and to call the interpreter, the Edit Frame to edit Prolog and Java (or other languages) files

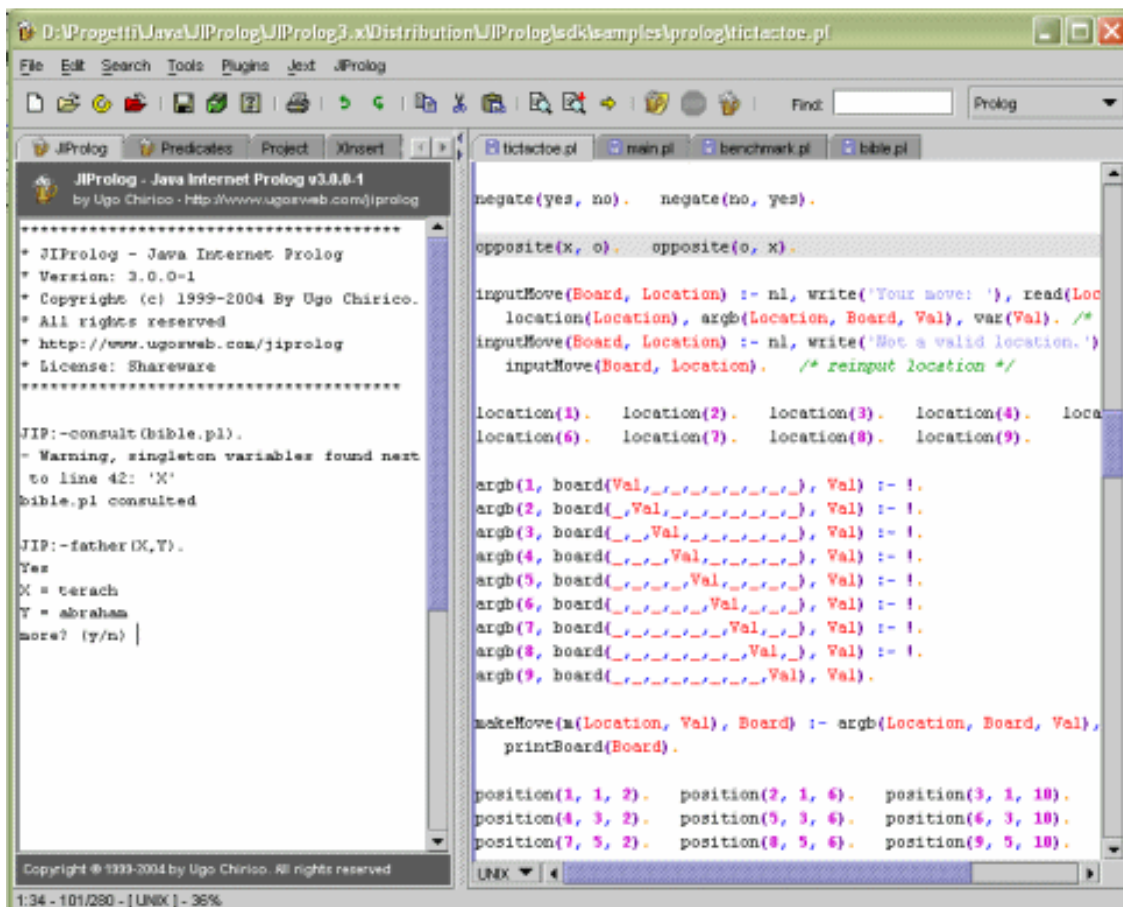
3.3.1.1 Projects Frame

The Projects Frame allows to manage projects, project files in Prolog, Java, Text, (or other languages) and supply a console to call the JIProlog interpreter. It is composed by the following panels:

- JIConsole Panel to call the JIProlog Interpreter;
- Predicates Panel that shows all predicates defined in the opened files;
- Project Panel to manage projects and project files.

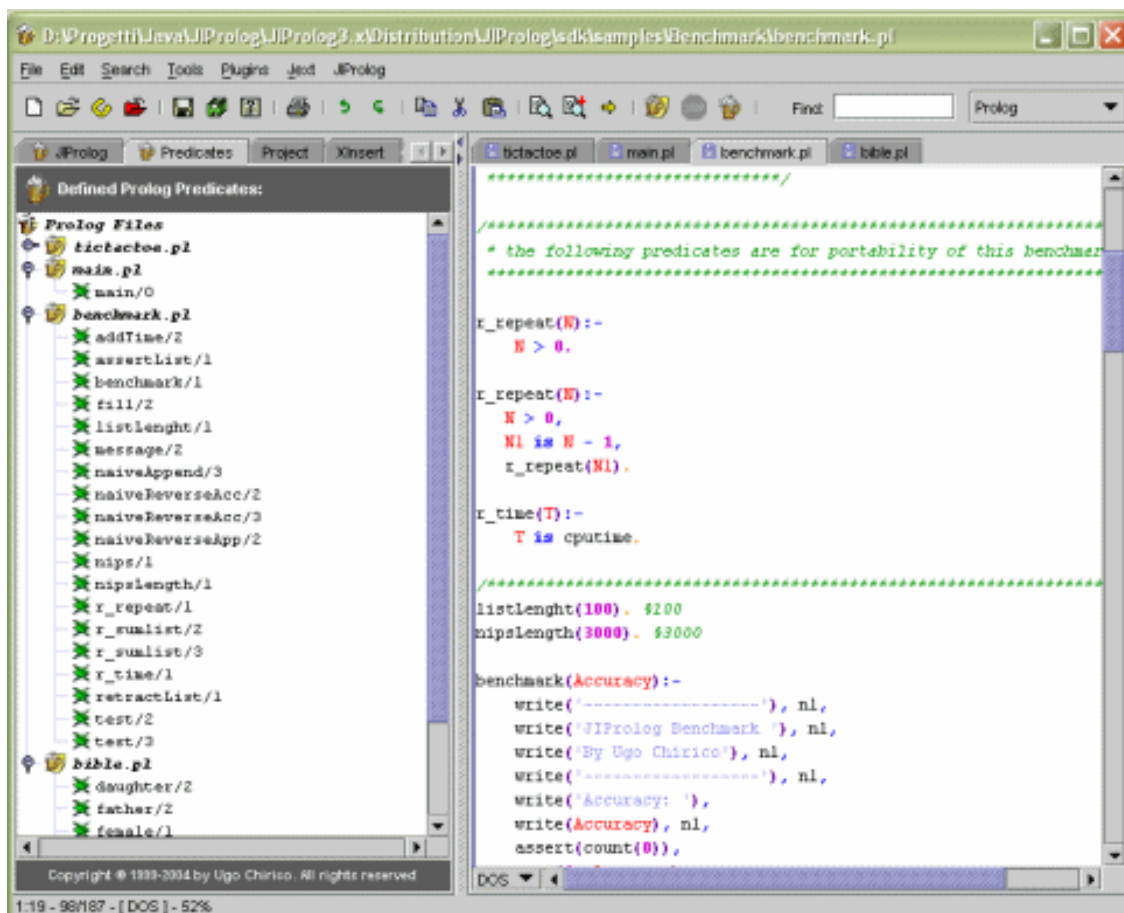
3.3.1.2 JIConsole Panel

JIConsole Panel allows to manage the JIProlog Interpreter running queries, and so on.
At left In the following figure is shown the Project Frame with the JIConsole panel.








3.3.1.3 Predicates Panel

Predicates Panel shows all predicates defined in the files open in the Edit Frame (at right):



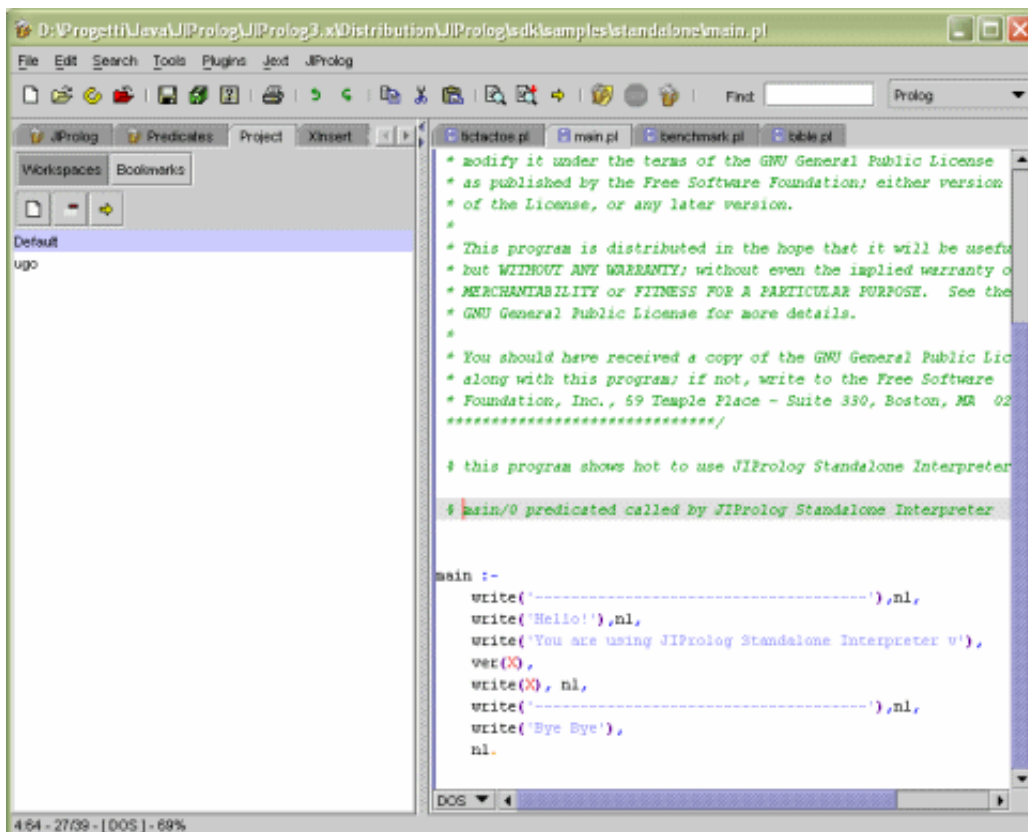
Clicking on a predicate in the Predicates Panel the cursors will be positioned to the corresponding predicate in the Edit Frame.

Prolog Items in the predicate panel is represented by special icons in the following way:

Prolog files is represented by	
Directive call is represented by	
Module definitions is represented by	
Public predicates is represented by	
Private predicates is represented by	

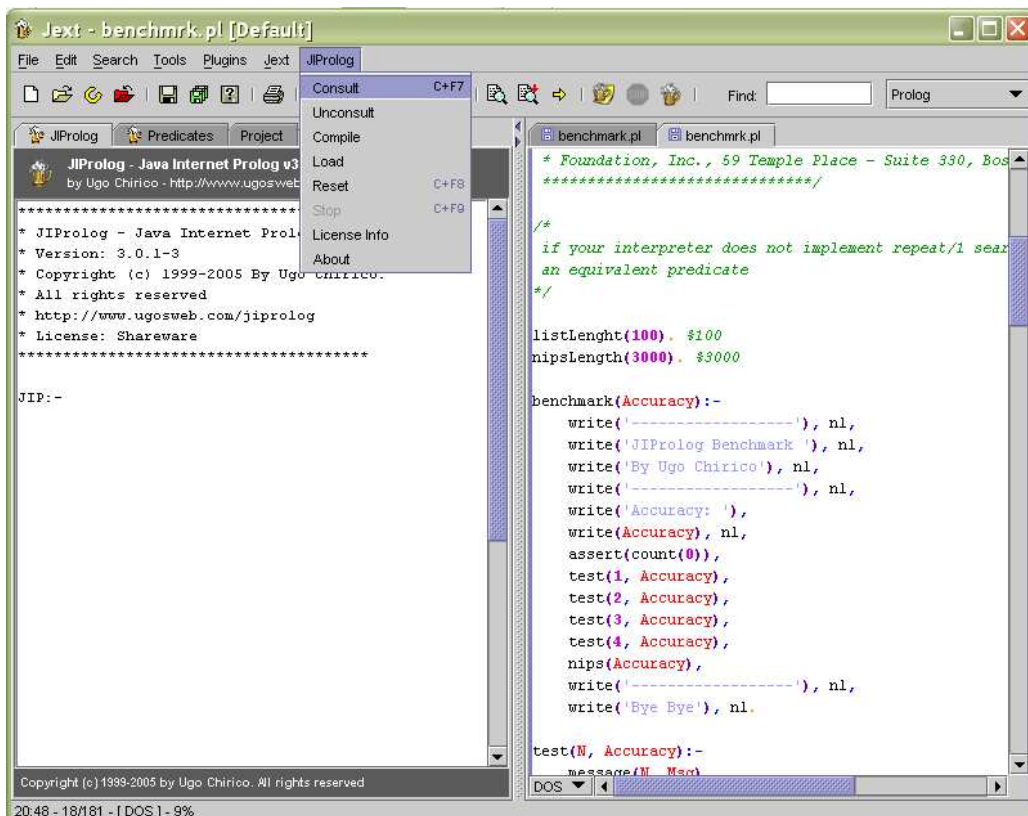
3.3.1.4 Projects Panel

Projects Panel allows to manage projects in Prolog, Java, or other languages:



3.3.1.5 JIProlog's Menu

JIProlog menu supplies all actions that can be done on a Prolog file:






Menu Item	Description
Consult	Consults the selected prolog file
Unconsult	Unconsults the selected prolog file
Compile	Compiles the selected prolog file and produce a compiled prolog file with the extension .jip
Load	Loads a compiled prolog file with the extension .jip
Reset	Resets interpreter database and clears all consulted clauses
Stop	Aborts the execution of the current query

3.3.1.6 JIProlog's Button Bar

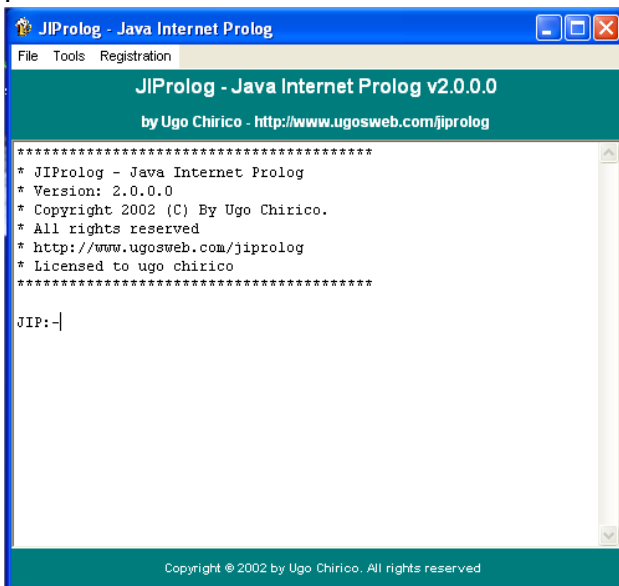
Some of the action supplied by JIProlog Menu can be accomplished by using the JIProlog's Button Bar in the next's button bar.



	Consults the selected prolog file
	Aborts the execution of the current query
	Shows JIProlog's About Box

4.0 JIProlog Console

JIProlog Console is a complete PureJava 100% Prolog console that allows editing, running and tracing Prolog programs. JIProlog Console can be run as a Java application on any platform such as Windows, Unix/Linux, MacOS, Windows CE, Pocket PC 2002/2003, Windows Mobile 2003/2005, EPOC OS etc. or can be run in a html page as an applet:



4.1 Installation

Download the latest version of the package JIPConsole.zip from the following web page:

<http://www.ugosweb.com/jiprolog/Download.html>

4.1.1 Installation for Windows

Download JIPConsole.zip for windows from the following web page:

<http://www.ugosweb.com/jiprolog/Download.html>

and unzip it in a directory such as c:\JIProlog

4.1.2 Installation for Linux/Unix

Download JIPConsole.zip for Linux/Unix from the following web page:

<http://www.ugosweb.com/jiprolog/Download.html>

and unzip it in a directory such as usr/JIProlog

4.1.3 Installation for Mac OS

Download JIPConsole.zip for Linux/Unix from the following web page:

<http://www.ugosweb.com/jiprolog/Download.html>

and unzip it in a directory such as JIProlog

4.1.4 Installation for Windows CE - Pocket PC2002/2003 Windows Mobile with Jeode/CreME

Download jiprologjeode.cab or jiprologcreme.cab depending on your platform (Jeode or CrEME) from the following web page:

<http://www.ugosweb.com/jiprolog/Download.html>

copy the cab file on your pda via active sync, Bluetooth or irda, and tap on it to install.

4.1.5 Installation for EPOC OS

Download the latest version of JIPrologSIS.zip installation package from

<http://www.ugosweb.com/jiprolog/Download.html>

Copy it in your PDA and install it by clicking on .sis file or by ControlPanel>add/remove applet.

4.1.6 Installation for other Java 1.1 platforms

Download JIPConsole.zip for Linux/Unix from the following web page:

<http://www.ugosweb.com/jiprolog/Download.html>

and unzip it in a directory such as usr/JIProlog

4.1.7 Java Enable Mobile Phones – JIProlog2ME

Download `jip2me.jad` (or `jip2me.jar` if your phone supports installation from jar) on your mobile phone from the following urls:

<http://www.ugosweb.com/Download/jip2me.jar>
or
<http://www.ugosweb.com/Download/jip2me.jad>

4.2 Usage

Here we assume that the reader has a minimal knowledge about Prolog language and knows how to run a query. For such a stuff refer to the book “*Programming in Prolog*” [CL03].

4.2.1 Running JIPConsole

4.2.1.1 Windows

Double-click on `JIPConsole.js` or `JIPConsole.bat`

4.2.1.2 Unix/Linux

Run `jipconsole.sh`

4.2.1.3 Mac OS

Run `jipconsole`

4.2.1.4 NetWare 5

Run `jipconsole.ncf`

4.2.1.5 Windows CE-PocketPC 2002/2003 (Jeode/crème)

From start menu select `Programs->JIPConsole->JIPConsole` and wait patiently while JIPConsole starts.

4.2.1.6 EPOC OS

Run JIProlog Console selecting the “extras” icon in the programs bar and clicking the JIProlog icon.

Note that on some platforms JIProlog Console starts showing `stdout` window. Move on JIProlog Console window clicking on the “System” icon and selecting JIProlog Console program.

4.2.1.7 Other Java 1.1 platforms

The JIProlog Console can be run on any Java 1.1 platform. Depending on the specific platform the right way to run JIProlog Console is to follow the usual way used by that platform to run a Java application.

Note that the packages needed to run JIProlog Console are: `jiprolog.jar` and `jipgui.jar`

The JIPConsole application is implemented in the class `com.ugos.JIProlog.gui.JIPConsole`

For example to run JIProlog on a Windows platform the command is:

```
java -classpath JIProlog.jar;JIPgui.jar com.ugos.JIProlog.gui.JIPConsole
```

4.2.2 Running JIPConsole applet

To run JIPConsole as an applet in your Java 1.1 enabled web browser open your browser and point it to the file `JIPConsole.html` contained in the distribution package.

4.2.3 Editing and Consulting a Prolog file

To edit a new file select the menu: `File/New` or type `CTRL+N`

To edit an existing file select the menu: `File/Open` or type `CTRL+O`

An edit window appears empty or containing the file selected.

To consult your Prolog program in the edit window select the menu: `File/Consult` or type `CTRL+T`

4.2.4 Tools

The Tools menu allows do the following things:

- `Reset database` resets interpreter database and clears all consulted clauses
- `Stop execution` forces the execution of a query to stop

`Colors` allows to change background color of the console.

5 JIProlog stand-alone interpreter

JIProlog stand-alone interpreter is a Java application that runs Prolog programs as stand-alone applications. The only requirement to run a Prolog program as stand-alone application is that it should define the predicate `main/0` that is the entry point called by the interpreter when it starts.

5.1 Running the stand-alone interpreter

JIProlog stand-alone interpreter is an application implemented in the class `com.ugos.JIProlog.JIProlog` supplied with the package `jiprolog.jar`. It should be run as any Java application but requires a Prolog file that defines the predicate `main/0`. This predicate is the entry point and is called by the interpreter when it starts. In the following, let `<mainfile.pl>` be a Prolog file containing the predicate `main/0`.

5.1.1 Windows

Run one of the following:

```
JIProlog.js <mainfile.pl>  
JIProlog.bat <mainfile.pl>
```

5.1.2 Unix/Linux

Run the following:

```
jiprolog.sh <mainfile.pl>
```

5.1.3 Mac OS

Run the following:

```
jiprolog <mainfile.pl>
```

5.1.4 Personal Java and Windows CE – Pocket PC2002/2003

Depending on the specific platform the right way to run JIProlog Console is to follow the usual way used by that platform to run a Java application.

Note that the packages needed to run JIProlog stand-alone Interpreter on Personal Java is: `jiprolog.jar`.

The JIProlog stand-alone Interpreter is implemented in the class:

```
com.ugos.JIProlog.JIProlog
```

For example to run on a standard Personal Java platform the command is:

```
pjava.exe -classpath jiprolog.jar com.ugos.JIProlog.JIProlog <mainfile.pl>
```

5.1.5 Java Enable Mobile Phones – MIDP

Currently JIProlog stand-alone Interpreter is not supported for this platform

5.1.6 Other Platforms

The JIProlog stand-alone Interpreter can be run by any Java 1.1 platform. Depending on your specific platform the right way to run it is to follow the usual way used to run any Java application.

For example to run a JIProlog stand-alone Interpreter on a Windows platform the command is:

```
java -classpath jiprolog.jar com.ugos.JIProlog.JIProlog <mainfile.pl>
```

6 JIProlog Server

JIProlog Server is a daemon that brings JIProlog Interpreter's functionalities over the network via TCP/IP channel, supplying, in this way, any remote client with an interface to submit queries and to receive the related solutions.

JIProlog Server can serve multiple simultaneous connections from multiple clients. Each connection is managed in a separate thread while each query is referenced by a unique handle.

Clients should be implemented using the class `com.ugos.JIProlog.client.JIPClient` that implements the client-side protocol. Refer to the API's javadoc documentation in the directory `doc` and see the section "*How to implement a JIProlog client*" in the appendix A.

The following picture gives a screenshot of JIProlog Server:



6.1 Running the JIProlog Server

JIProlog server is supplied in the JIProlog SDK. Download it from:

- <http://www.ugosweb.com/jiprolog/Download.html>

JIProlog Server is an application implemented in the class `com.ugos.JIProlog.server.JIPServer` supplied with the package `jipserver.jar` in the directory `<SDKDirectory>/clientserver`.

6.1.1 Windows

Go in the directory `<SDKDirectory>/clientserver` and run one of the following:

```
JIPServer.js  
JIPServer.bat
```

6.1.2 Unix/Linux

Go in the directory `<SDKDirectory>/clientserver` and run the following:

```
jipserver.sh
```

6.1.3 MacOS

Go in the directory `<SDKDirectory>/clientserver` and run the following:

```
jipserver
```

6.1.4 Personal Java and Windows CE – PocketPC 2002/2003

Currently JIProlog Server is not supported for this platform

6.1.5 Java Enable Mobile Phones – MIDP

Currently JIProlog Server is not supported for this platform

6.1.6 Other Platforms

The JIProlog Server can be run by any Java 1.1 platform. Depending on your specific platform the right way to run it is to follow the usual way used to run any Java application.

For example to run a JIProlog Server on a Windows platform the command is:

```
java -classpath jiprolog.jar;jipserver.jar com.ugos.JIProlog.server.JIPServer
```

7 JIProlog2ME for Java Enable Mobile Phones

JIProlog has been optimized for J2ME MIDP 2.0-CLDC 1.0 or later platform.

7.1 Minimum Requirements

JIProlog2ME has the following minimum requirements:

- MIDP 2.0:
- CLDC 1.0
- 400 KB RAM (heap)

7.2 Limitations

Due to limitations given by MIDP specification JIProlog2ME doesn't support the following:

- access to files using `file://` or `jar://` protocols;
- predicates:
 - o `compile/1`
 - o `load/1`
 - o `load_library/1`

The following is the set of built-in predicates available with JIProlog2ME basic runtime.

```
expand_term/2,  
=</2  
phrase/3  
phrase/2  
^/2  
current_predicate/1  
'C'/3  
number/1  
=\\= /2  
@</2  
string/1  
false/0  
op/3  
->/2  
@>= /2  
char/1  
\\== /2  
true/0  
>= /2  
= /2  
nonvar/1  
@= /2  
[! /0  
\\+ /1  
not/1  
append/3  
retractall/1  
unload/1  
assertz/1  
simple/1  
nil/1  
list/1  
atomic/1  
callable/1  
\\= /2  
; /2  
@> /2  
use_module/1  
is_list/1  
compound/1  
@=< /2  
is/2  
current_functor/2  
clause/2  
== /2
```

```

searchpath/1
!/0
current_atom/1
var/1
=../2
chars/1
fail/0
>/2
</2
module_transparent/1
retract/1
unconsult/1
float/1
ver/1
consult/1
write/1
$error/1
abort/0
integer/1
dynamic/1
asserta/1
!]/0
garbage_collect/0
xcall/2
$current_query_handle/1
extern/3
abolish/1
arg/3
compare/3
assert/1
current_op/3
length/2
$free/1
ground/1
multifile/1
nl/0
:=/2
write_canonical/1
atom/1
ensure_loaded/1
functor/3
notify/2
predicate_property/2

```

The other ISO predicates are implemented in extension packages (see above) and can be recompiled as needed for the specific platform. The SDK contains the source code for such extension packages (see the chapter 8 JIProlog SDK):

7.3 Running the JIProlog2ME MIDlet on a J2ME Mobile Phone

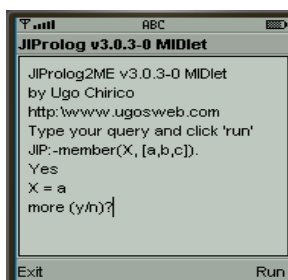
Download `jip2me.jad` (or `jip2me.jar` if your phone supports installation from jar) on your mobile phone from the following urls:

<http://www.ugosweb.com/Download/jip2me.jar>

OR

<http://www.ugosweb.com/Download/jip2me.jad>

This is a screenshot of JIProlog2ME console:



7.4 Loading Extension and Custom Packages

External extension predicates, as well as custom predicates, must be recompiled for the specific platform and the related packages must be inserted in the final .jar package (together with .jad file).

This is due to the lack of the predicate `load_library/1` that, in J2ME version is not available.

For the source code of the extension packages supplied with JIProlog see the directory `samples` supplied with JIProlog SDK.

8 JIProlog SDK

JIProlog SDK supplies the JIProlog runtime for all supported platforms, the extension packages, the documentation needed to use JIProlog API, several useful samples in Prolog and Java and the source code of the extension packages.

8.1 Interface between Java and Prolog

JIProlog comes with a complete API to link Prolog and Java worlds from both sides. The API can be described as composed of three parts: the first one regards the way to invoke the Prolog Interpreter from Java; the second one deals with the way to call Java objects from Prolog programs, that is, by implementing custom Prolog built-in predicates as Java classes or by calling Java methods through Java Reflection API; the third one treats the way to link Prolog predicates with tables in an external database, via Prolog \leftrightarrow JDBC bridge. A complete description of the JIProlog API is in the javadoc formatted documentation supplied in the directory `<SDKInstallationDirectory>/jiprolog/docs`

8.1.1 Java to Prolog

JIProlog provides several Java classes to manage the Prolog interpreter in the same way you call generic Java methods, without resorting to the JNI link to native code (and, as result, without the need for a signed applet).

`JIPEngine` is the main class. It supplies the methods to initialize the Prolog interpreter, to submit and manage queries, to consult/unconsult files, and to set several interpreter properties. In addition there are several classes that wrap Prolog terms such as atoms, lists, clauses, etc. Some of them are: `JIPTerm`, `JIPAtom`, `JIPList`, etc. (all those classes are described in depth in the javadoc documentation in the JIProlog distribution)

Queries can be submitted synchronously or asynchronously. In the former case, the method that submits the query exits when the goal is satisfied or when it fails. In other words, the method waits until the interpreter has finished its work. In the latter case, the method that submits the query exits immediately after submission and the result of the query is notified asynchronously to a special listeners registered before hand.

A synchronous call is simpler and fits needs in most cases. An asynchronous call, even though it is more complex, allows a more powerful and accurate management of the interpreter.

The following paragraphs describe in depth how to call Prolog from Java.

8.1.1.1 Parsing a Prolog term

Parsing a Prolog term means creating in Java a Prolog term by a String object. The method

```
JIPTerm JIPTermParser.parseTerm(String strTerm)
```

creates a Prolog term from a string if it represents a valid Prolog term, otherwise it raises a `JIPSyntaxErrorException`.

The following are valid parsing procedures:

```
JIPEngine engine = new JIPEngine();
JIPTermParser termParser = engine.getTermParser();
Atom atom = (JIPAtom)termParser.parseTerm("hello");
JIPAtom atom = (JIPAtom)termParser.parseTerm("hello");
JIPAtom quotedAtom = (JIPAtom)termParser.parseTerm("'Hello World'");
JIPFunctor functor = (JIPFunctor)termParser.parseTerm("write('Hello World')");
JIPList list = (JIPList)termParser.parseTerm("[ 'Hello' 'World' ]");
```

8.1.1.2 Parsing a Prolog Stream

Parsing a Prolog stream means translate a prolog file in an enumeration of Prolog terms.

The method

```
Enumeration JIPTermParser.parseStream(InputStream ins
String strStreamName)
```


creates an enumeration of Prolog terms from the given Prolog stream (if in the stream there is a syntax error it raises a `JIPSyntaxErrorException`).

The following is a valid parsing procedure:

```
JIPEngine engine = new JIPEngine();
JIPTermParser termParser = engine.getTermParser();
FileInputStream fins = new FileInputStream("myprologfile.pl");
Enumeration enum = termParser.parseStream(fins
"myprologfile.pl");
```

8.1.1.3 Synchronous calls

A synchronous call can be made by the method:

```
JIPQuery JIPEngine.openSynchronousQuery(JIPTerm query)
```

It returns a `JIPQuery` object that gives the methods to invoke the interpreter synchronously:

```
JIPTerm JIPQuery.nextSolution()
```

waits for the interpreter and returns the next solution or null if the goal fails;

```
boolean JIPQuery.hasMoreChoicePoints()
```

checks if the goal has another choice-point i.e. there are is another solution on backtracking.

See also the section "How to call the interpreter synchronously" in the appendix A.

8.1.1.4 Asynchronous calls

An asynchronous call can be made by the method:

```
void JIPEngine.openQuery(JIPTerm query)
```

It starts the interpreter in a separate thread passing it the query and returns immediately. The results of the query are notified to the registered `JIPEventListener` in the methods:

```
void JIPEventListener.solutionNotified(JIPEvent e)
void JIPEventListener.failNotified(JIPEvent e)
void JIPEventListener.endNotified(JIPEvent e)
void JIPEventListener.openNotified(JIPEvent e)
void JIPEventListener.closeNotified(JIPEvent e)
void JIPEventListener.moreNotified(JIPEvent e)
void JIPEventListener.termNotified(JIPEvent e)
```

If the query raises an error it is notified in the method:

```
void JIPEventListener.errorNotified(JIPEvent e)
```

Note that listeners should be registered before hand by the methods:

```
void JIPEngine.addEventListener(JIPEventListener el)
```

See also the section "How to call the interpreter asynchronously" in the appendix A.

8.2.2 Prolog to Java

JIProlog supplies two ways to call Java from Prolog. The first one allows extending the set of built-in predicates implementing them in Java classes. As result it is feasible to add to Prolog custom predicates, custom dialogs and windows or custom algorithms implemented in Java. To do that JIProlog supplies an abstract class and an interface to extend Prolog with custom predicates, dialogs and windows.

The second one allows to manage directly Java objects from prolog through the Java Reflection API mapped on a set of Prolog predicates implemented in the extension package `jipxreflect.jar`.

Both methods are described below.

8.2.2.1 Implementing custom built-in predicates

To implement a custom built-in predicate create a new class derived from the abstract class `JIPXCall`:

```
public class MyCustomBuiltIn extends JIPXCall
```

and implement three abstract methods:

```
boolean JIPXCall.unify(JIPCons params  
    Hashtable varsTbl)  
  
boolean JIPXCall.hasMoreChoicePoints()
```

The first one is called when the interpreter unifies the custom predicate giving the parameter passed in `params`. The second one is called to check if the predicate has another choice-point on backtracking. For example consider the predicate `sumall/2` that sums all numbers in the list passed as first parameter and unifies the result with the second one. The method `unify` returns true if the first parameter is a list of numbers and if the second parameter unifies with the sum. Finally, because the sum can be made in only one way, thus it gives only one result, the method `hasMoreChoicePoints` must return false.

See the sections "How to write a custom predicates" in the appendix A for more information.

8.2.2.2 Managing Java Objects from Prolog

The extension package `jipxreflect.jar` supplies a set of custom built-in predicates that allows creating Java objects, invoking their methods, retrieving values from their fields and so on. Such predicates are implemented using the functionalities given by Java Reflection API. See the §2.1.1.10

8.2.2.3 Prolog to RDBMS and JDBC

JIProlog supports external tables of facts that are sets of facts not stored in memory but in external systems such as RDBMSs. Sometimes when you need to manage in Prolog a lot of facts that change dynamically and not deterministically it is preferable to have dynamic links to them in order to avoid repeated imports in Prolog memory every time they change.

JIProlog provides a way to dynamically link a Prolog predicate to a table maintained in an external database. After the link has been established the linked predicate can be called in Prolog as all other predicates and each change in the table is viewed immediately in Prolog.

Currently JIProlog provides in an extension package, the bridges to link JDBC databases, to bind tables stored in text files and tables stored in Prolog format.

As an example, to link the predicate `employee/5` to the table `Employee` with 5 columns in `samples/ExternalDB` directory connected via `JIPTextClausesDatabase` type the following:

```
JIP:-load_library('jipxdb.jar') % this is not needed with JIPrologIDE and JIPConsole  
Yes
```

```
JIP:-extern(employee/5, 'com.ugos.JIProlog.extensions.database.TextClausesDatabase',  
'c:\JIProlog\samples\ExternalDB\employee.txt').  
Yes
```

Or, in a simpler way, using the helper supplied by the extension package:

```
JIP:-declare_extern(text, employee/5,  
'com.ugos.JIProlog.extensions.database.TextClausesDatabase', 'c:\JIProlog\samples\ExternalD  
B\employee.txt').  
Yes
```

From now on the predicate `employee/5` is linked to the table `Employee`:

```
JIP:-employee(A,B,C,D,E).  
Yes  
A = Albert  
B = Einstein  
C = Physicist  
D = 13579  
E = 3000
```

JDBC connections are discussed in depth in the section "How to link an external database of clauses via JDBC" in appendix A.

JIProlog allows also to implement one's own custom bridge to an external database. It supplies two abstract classes by which to implement custom bridges. See the sections "*How to write a custom bridge to an external database of clauses*" in the appendix A.

8.3 JIProlog2ME (J2ME)

In the directory `<SKDDirectory>J2ME` there are the package for J2ME and the related documentation. Refer to the file `JIPrologMIDlet.java` in `samples/midlet` as skeleton to write a MIDlet using JIProlog2ME.

9 Main Built-in Predicates

JIProlog supplies almost all ISO Prolog built-in predicates.

This section describes only a subset of the built-in predicates supplied with JIProlog.

The complete list of built-in predicates is available in the appendix D

For a complete reference manual of ISO Prolog predicates visit the following url:

<http://paullac.inria.fr/~deransar/prolog/docs.html>

9.1 Main Built-in Predicates supplied with JIProlog

`=/2`

`term1 = term2`

Succeeds if `term1` and `term2` unify.

The following example tests that two given lists can be unified, and binds the variables as required:

```
JIP:- [1,X,Y] = [Z,2,3].
yes
X = 2
Y = 3
Z = 1
```

`\=/2`

`term1 \= term2`

Succeeds if `term1` does not unify with `term2`, and never binds any variables.

The following call succeeds, because the two specified terms do not unify:

```
JIP:- foo(bar) \= foo(sux).
yes
```

The following call fails, because although the given terms are different, they can unify:

```
JIP:- foo(bar) \= foo(X).
no
```

The `\=/2` predicate is the exact opposite of `=/2`, and is effectively defined as:

```
X \= Y :-
    \+ X = Y.
```

`==/2`

`term1 == term2`

Succeeds if `term1` is identical to `term2`. No variables in `term1` and `term2` are bound as a result of the testing.

The following call succeeds because both given terms are identical:

```
JIP:- foo(A,B) == foo(A,B).
A = _? ,
B = _?
```

The following call fails, because the terms are not identical:

```
JIP:- foo(A,a) == foo(A,B).
no
```

`\==/2`

`term1 \== term2`

Succeeds if `term1` is not identical to `term2`

The following call succeeds, because the two specified terms are different from each other:

```
JIP:- foo(bar) \== foo(sux).
yes
```

The following call also succeeds, because even though the two terms could unify, nonetheless they are different:

```
JIP:- foo(bar) \== foo(X).  
X = _?
```

The `\==/2` predicate is the exact opposite of `==/2`, and is effectively defined as:

```
X \== Y :-  
    \+ X == Y.
```

>/2, </2, >=/2, <=/2, ==/2, \=/2

`exp1 > exp2`

Succeeds if the relation between arithmetic expression `exp1` `exp2` is the same as indicated by the operator. An arithmetic expression can be a number or an arithmetic function.

@>/2, @</2, @>=/2, @<=/2

`term1 @>= term2`

Succeeds if the relation between `term1` and `term2` is the same as indicated by the operator according to the standard ordering of terms (see appendix C).

The difference between the `@`-prefixed operators and the others shown above is that the first refers to Prolog terms (atoms, lists, functors, etc.) while the second refers only to numbers. For example:

```
JIP:- 3 > 2.  
Yes
```

```
JIP:- b > a.  
Error 2...
```

```
JIP:- b @> a.  
Yes
```

```
JIP:- 3 @> 2.  
Yes
```

==./2

`Term ==. List`

List is a list whose first element is the functor of the Term, and whose tail is the list of arguments of Term.

If Term is an uninstantiated variable, then List must be instantiated to a list of determinate length. A compound term will be constructed from the list. The functor of the term will be the head of the list, while the arguments of the term will be the tail of the list. If Term is not a variable, then the corresponding list is constructed and unified with List.

The following call creates a compound term from the given list:

```
JIP:- X ==. [likes,brian,Prolog]. <enter>  
X = likes(brian, Prolog)
```

When a single-element list is provided, an atomic term is returned:

```
JIP:- X ==. [123]. <enter>  
X = 123
```

You can also convert a compound term into a list:

```
JIP:- foo(bar) ==. L. <enter>  
L = [foo, bar]
```

The conversion can work in both directions at once, filling in any variables automatically:

```
JIP:- foo(123, A, 789, bar) ==. [X, 123, 456 | Y]. <enter>  
A = 456 ,  
X = foo ,  
Y = [789, bar]
```

The `==./2` predicate is usually called "univ", and is widely used in meta-programming, where terms representing calls are pulled apart, modified as lists, and rebuilt into terms which are then executed.

!/0

!

The predicate `!/0` always succeeds, but it has the important side effect of removing pending choice points within the current clause and relation. Its purpose is to enable programs to commit to solutions, and to retrieve space that would otherwise be tied up in unnecessary choice points.

[!/0, !/0

[!, foo, bar, !]

Snip.

Snip is a set predicates, delimited by an open snip [! and a close snip !]. It is similar to a cut in that, when it is encountered during backtracking, it affects program execution.

When a snip is encountered during backtracking the interpreter skips over the predicates delimited by snip and program execution continue from the predicate preceding the snip.

Example:

```
a:- b, [!, c, d, !], e.
```

if e fails, backtracking skips over goals c and d and retry goal b

```
a:- b, c, d, !, e.
```

if e fails, backtracking encounters the cut and the entire predicate, a fails.

^/2

X ^ Goal.

Existential quantification. ^/2 means "there exists an X such that Goal is true". This predicate is not really programmed into ^/2 at all: normally this predicate simply calls the given Goal, completely ignoring the value of X.

->/2

goal -> then

Succeeds if goal succeeds and the then goal also succeeds. It performs a "local cut" after executing its first goal, preventing backtracking into it.

;/2 - Disjunction

Either ; Or.

Disjunction. This predicate succeeds if either or both of its goals, Either and Or, are succeed. Either call may itself be a conjunction or disjunction, or a simple goal.

;/2 - if-then-else

If -> Then ; Else.

IF – THEN – ELSE construct. This predicate succeeds if IF succeeds and THEN Succeeds or if IF fails and ELSE succeeds. It performs a "local cut" after executing its first goal, preventing backtracking into it.

\+/1

\+ Goal.

Logical Not. Fails if goal succeeds.

abolish/1

abolish(Pred).

Delete all occurrences of the predicate Pred. Pred must be in the format name/arity.

```
JIP:-abolish(foo/1)
```

abolish_files/1

abolish_files(Files).

Delete all predicates defined in the files contained in the list Files

```
JIP:-abolish_files(['foo1.pl', 'foo2.pl'])
```

abort/0

abort.

Abort the program that is currently being executed.

Because the abort/0 predicate directly resets all internal stacks and forces JIP to stop the execution , it is normally only used to abandon execution of a query when an error has occurred for which there is no other method of recovery.

append/3

append(First, Second, Whole)

if First and Second are instantiated, the Second list is appended to the end of the First list to give the Whole list

```
JIP:-append([a,b,c,d], [1,2,3,4], A ).  
A = [a,b,c,d,1,2,3,4].
```

If Whole is instantiated, the Whole list is split at an arbitrary place to form two new sub-lists First and Second. Alternative solutions will be provided on backtracking.

```
JIP:-append(First, Second, [a,b,c,d,1,2,3,4] ).  
First = []  
Second = [a,b,c,d,1,2,3,4]  
Yes. more (y/n): y  
First = [a]  
Second = [b,c,d,1,2,3,4].
```

If First and Whole are instantiated and the Whole list starts with the First list, then the remainder of the Whole list is returned as Second.

```
JIP:-append([a,b,c,d], Second, [a,b,c,d,1,2,3,4] ).  
Second = [1,2,3,4].
```

If Second and Whole are instantiated and the Whole list ends with the Second list then the beginning of the Whole list is returned as First.

```
JIP:-append(First, [1,2,3,4], [a,b,c,d,1,2,3,4] ).  
First = [a,b,c,d].
```

If First, Second and Whole are instantiated, then test that the Whole list begins with the First list and that the remainder of the Whole list is the Second list.

```
JIP:-append ([a,b,c,d], [1,2,3,4], [a,b,c,d,1,2,3,4] ).  
yes.
```

arg/3

arg(N, Term, Arg)

Unifies Arg with the Nth argument of Term. N must be a positive integer, and Term must be a compound term (or a list). The arguments are numbered 1 upwards.

assert/1, assertz/1

assert(clause)

add clause to the database at the end of the sequence of clauses defining its predicate name.

clause maybe a fact:

atom

atom(arg1, arg2, ...)

or a rule (in that case it must be enclosed in parentheses):

(atom:-term1,term2,...)

(atom(arg1, arg2, ...):-term1, term2, ...)

Note: a clause defined as built-in predicate cannot be asserted

asserta/1

asserta(clause)

add clause to the database at the beginning of the sequence of clauses defining its predicate name.

clause maybe a fact:

atom

atom(arg1, arg2, ...)

or a rule (in that case it must be enclosed in parentheses):

(atom:-term1,term2, ...)

(atom(arg1, arg2, ...):-term1, term2, ...)

Note: a clause defined as built-in predicate cannot be asserted

atom/1

atom(Atom)

Succeeds if Atom is currently instantiated to an atom.

atomic/1

atomic(Term)

Succeeds if Term is currently instantiated to an integer, float, atom or string. It will fail if term is an unbounded variable or a compound term.

call/1

call(Call)

Calls the goal Call. Succeeds if Call succeeds, and fails otherwise.

callable/1

callable(Term)

Succeeds if Term is an atom or a functor.

char/1

char(Term)

Succeeds if Term is a character i.e. is an integer in the range [0, 255]

chars/1

char(Term)

Succeeds if Term is a list of character i.e. is a list composed by integers in the range [0, 255].

Ex. [65,66,67,68] that is equivalent to "ABCD".

clause/2

clause(Head, Body)

Succeeds if there is a clause in the database whose head matches Head and whose body matches Body. It searches through the database until it finds the first clause whose head matches Head. The body of this clause will be unified with Body. If the matching clause is a fact, Body will be unified with the single goal true. The clause/2 predicate is non-deterministic. That is, it can be used to backtrack through all the clauses in the database that match a given Head and Body. It will fail when there are no (more) matching clauses.

compare/3

compare(Rel, Term1, Term2)

Succeeds if Rel is the relationship between Term1 and Term2.

The order is found according to the standard ordering of terms.

If Rel is an unbound variable, it will be bound to =, <, or > according to the relationship between Term1 and Term2 otherwise Rel must be an atom in the set {=, <, >}. In such a case compare/3 will succeed if Rel describes the relationship between Term1 and Term2.

compile/1

compile(filename)

Compiles the source file specified by filename. The result is a file with the extension .jip containing the compiled version of the program specified by filename.

Compiled program can be read and load in JIProlog by load/1 predicate.

compile/1 and load/1 are very useful to distribute your Prolog program in an unreadable format (see the section "How to compile and distribute a Prolog program" in the appendix A)

compound/1

compound(Term)

Succeeds if Term is currently instantiated to a compound term. It will fail if Term is an atom, integer, float, string or an unbounded variable.

consult/1, reconsult/1

consult(filename), consult([file1, file2,...])

consult/1 loads the source file specified by filename.

```
JIP:- consult('foo.pl').  
Yes
```

```
JIP:- consult(['foo.pl', 'bar.pl']).  
Yes
```

Filename can be a full path, a relative path or an URL of the type:

- <http://www.ugosweb.com/jiprolog/packages/hanoi.txt>
- file:/user/Prolog/foo.pl
- jar://user/Prolog/prologarchive.jar#foo.pl
- loc:/resource/file.pl' for reading file inside jiprolog.jar.

Filename can be also a list of paths.

If Filename doesn't specify any extension, .pl is assumed.

reconsult/1 is identical to consult. It is defined only for compatibility with other Prolog interpreters

Note: JIProlog allows consulting files containing up to 5000 clauses. It is due to the optimization of the memory and the computational speed. If you need to consult a larger file you should declare it as an external database of clauses (see the section "How to consult a very large Prolog file" in the appendix A).

cons/1

cons(Term)

Succeeds if Term is currently instantiated to a list.

current_output/1

current_output(Stream)

See telling/1

current_input/1

current_input(Stream)

See seeing/1

current_op/3

current_op(Prec, Type, Name)

Succeeds when the atom Name is an operator of type Type and precedence Prec. Any of the arguments may be uninstantiated variables.

declare_extern/3

declare_extern(Bridge, Predicate, Params) (package JIPxdb.jar)

Declare the predicate specified in Predicates as external predicate linked to an external table through the bridge specified in Bridge. Params should be an atom or a string containing additional parameters to pass to the bridge (ex. name of the table).

Currently the supported bridge are:

- jdbc
- text
- prolog

Ex.

```
JIP:-declare_extern(jdbc, employee/5,  
"sun.jdbc.odbc.JdbcOdbcDriver,jdbc:odbc:EmpoloyeeDSN,table:Employee,user,password").  
Yes
```

ensure_loaded/1

ensure_loaded(FileName).

Loads or Consults the file specified in FileName. If FileName refers to a Prolog file it is consulted, if FileName refers to a compiled Prolog if is loaded.

If filename can be a full path, a relative path or an URL of the type:

- http://www.Prolog.com/foo.pl
- file:/user/Prolog/foo.jip
- jar:/user/Prolog/prologarchive.jar#foo.pl
- loc:/resource/file.pl' %for reading file inside jiprolog.jar.

expand_term/2

expand_term(Term1, Term2)

Succeeds if Term1 is a grammar rule and unifies with Term2 that is the corresponding Prolog representation.

extern/3

extern(Pred, DBClassName, DBAttributes).

Declares Pred as an external database of clauses (i.e. a database of clauses not stored in the Prolog database but in an external device such as JDBC database, text database and so on).

Pred must be in the format name/arity, DBClassName must be derived from JIPClausesDatabase and is the class that implements the functionality to manage the external database of clauses (see the section "How to write an external database of clauses" in the appendix A), DBAttributes specifies some attributes needed to the database (such as db filename, userid and password, etc.).

```
JIP:-extern(foo/3, "MyPackage.JDBCdb", "file=foo.db+userid=me+pass=hello").
```

halt/0, halt/1

halt.

halt(Code).

Exits from the execution of the interpreter with the code Code.

fail/0, false/0

Always fails. Can be used to force backtracking in a query.

float/1

float(Term)

Succeeds if Term is currently instantiated to a float number

functor/1

functor(Term)

Succeeds if Term is currently instantiated to a functor

functor/3

functor(Term, Name, Arity)

Succeeds if Term is a functor with the specified Name and Arity.

If Term is an uninstantiated variable, then Name must be an atom and Arity must be an integer greater than 0.

If Arity is greater than 0, then Term will be bound to a compound term whose functor is the term Name, and with arity Arity. Each argument of this compound term will be a distinct unbounded variable.

garbage_collect/0

garbage_collect.

Invokes the immediate garbage collection of the memory area.

Note: when a program is running automatic garbage collection will always be done.

get/1

get(Char)

Read a char from the current input stream and unify it to Char.

ground/1

ground(Term)

Succeeds if Term has no uninstantiated variables.

integer/1

integer(Term)

Succeeds if Term is currently instantiated to an integer number.

is/2

exp1 is exp2

Evaluates the expression exp2 and unifies the result with the value of the expression exp1.

An arithmetic expression can be a number or an arithmetic function.

Note: if exp2 contains some uninstantiated variable is/2 raise an exception.

length/2

length(List, Length)

gets the length of a Prolog term.

The length/2 predicate can also be used to create a list of either a specified or infinite length.

If Term is unbound and Length is an integer, Term will be unified with a list of variables of the specified Length.

```
JIP:- length( Term, 3 ).
Term = [_32356,_32358,_32360]
```

If both Term and Length are unbound, Term will be unified, through backtracking, with a list of variables of infinite Length:

```
JIP:- length( Term, Length ).
Term = [ ] ,
Length = 0
Yes. more (y/n)? : y

Term = [_32580] ,
```

```
Length = 1 ;
Yes. more (y/n)? : y

Term = [_32580,_34760] ,
Length = 2
```

load/1

load(FileName)

Loads a compiled Prolog file (.jip).

compile/1 and load/1 are very useful to distribute your Prolog program in an unreadable format (see the section "How to compile and distribute a Prolog program" in the appendix A)

load_files/1

load(FileList)

Loads all files in FileList

load_library/1

load_library(FileName)

Load an extensions library. The library should be a valid .jar file containing a set of extension classes.

If in the .jar there is a file named init.pl it is consulted as any other Prolog file.

lst/1

lst(Term)

Succeeds if Term is currently instantiated to a list.

member/2

member(element, list)

Succeeds if element is a member of the list list.

If element is instantiated, member/2 will check that it is a member of the list (if list is a variable, it will be bound to a list that contains element).

If element is a variable, it will be bound to the first element of list. On backtracking, it will be bound to successive elements of list.

module/2

:-module(moduleName, ExportList).

Defines the name of the current module and declares the list of exported predicates.

module_transparent/1

:-module_transparent ExportList.

Defines the predicates specified in ExportList as module_transparent.

multifile/1

multifile(Pred).

Defines the predicate specified in Pred as a predicate that can be declared in more files.

```
:-multifile(foo/2).
```

nil/1

nil(Term)

Succeeds if Term is instantiated to [].

nl/0

nl

Writes a carriage return followed by line feed to the current stream.

nl/1

nl(Handle) (package JIPxio.jar)

Writes a line-feed to the output stream pointed by Handle.

nonvar/1

nonvar(Term)

Succeeds if Term is instantiated to a non variable term.

not/1, \+/1

not Term

Logical not. Succeeds if predicate fails and vice versa. It is defined as:

```
not(Call):-call(Call), !, fail.  
not(Call).
```

notify/2

notify(Exp, Term)

Notify the JIPEventListeners that an Event occurred with ID in the expression Exp and term in Term

number/1

number(Term)

Succeeds if Term is instantiated to a float or an integer number.

op/3

op(Prec, Type, Name)

Declares an operator with a given type and precedence. The Name argument must be an atom that is the name of the operator. The Prec must be an integer between 0 and 1199, and whose type is Type.

open/3

open(File, Mode, Handle) (package JIPxio.jar)

Opens a file as defined in Mode and unify Handle with an handle to that file.

Mode can assume the following values:

read to open an input stream

write to open an output stream

phrase/2

phrase(Phrase, List)

Invokes the currently defined grammar rules in order to parse a sequence of symbols.

It succeeds if List is a phrase of type Phrase (according to the current grammar rules).

phrase/3

phrase(Phrase, List, Rest)

Invokes the currently defined grammar rules in order to parse a sequence of symbols.

It succeeds when the list List starts with a phrase of type Phrase, according to the currently defined grammar rules. Rest is that part of List that is left over after you have found the phrase.

put/1

put(Char).

Writes to the current output stream the character whose ASCII code is Char.

read/1

read(Term)

Reads the next term from the current input stream and unifies with Term. In the input stream, the term must be followed by a dot ('.') and at least one white space character. The dot and white space character are read in but are not considered part of the term.

repeat/0

repeat

Succeeds when called and on backtracking. Any calls which textually precede the repeat in the body of a clause will never be reached on backtracking.

It is defined as:

```
repeat.  
repeat :- repeat.
```

repeat/1

repeat(Count)

Succeeds when initially called, and succeeds for the given "Count" of times on backtracking.

retract/1

retract(clause)

Searches for the first clause in the database that matches clause. If such a clause is found, it is deleted. Any variables in clause are bound as a result of the unification. This predicate is non-deterministic. On

backtracking there is an attempt to find and delete another matching clause. This search always starts at the beginning of the list of clauses for the relation name of clause. So all clauses asserted between the retract and the redo of the call (even if added using asserta/1) are candidates for deletion on the redo. The call to retract/1 fails when there are no (more) clauses that match clause.
A clause defined as built-in predicate cannot be retracted.

retractall/1

retractall(head)

Deletes every clause in the database whose head matches head. Variables in head are left uninstantiated by the call. On backtracking there is no attempt to redo the call, even though matching clauses may have been asserted.

A clause defined as built-in predicate cannot be retracted.

retractone/1

retractone(clause)

Searches for the first clause in the database that matches clause. If such a clause is found, it is deleted. Any variables in clause are bound as a result of the unification. Differently from retract This predicate is deterministic. On backtracking there is no attempt to find another matching clause.

reverse/2

reverse(List, RevList)

Succeeds if RevList unify with is the reverse of List.

searchpath/1

searchpath (Path)

Set or get current search path. The search path is the directory where consult/1, load/1, tell/1/2 and see/1/2 search for files. If Path is an unbounded variable Path is bound to the current search path. If Path is an atom representing a valid path the search path is set to its value.

Path can be a relative path or an absolute path of the form

- d:\Prolog\
- http://www.prolog.com
- file:/user/prolog
- jar://user/prolog/prologarchive.jar

```
JIP:-searchpath(X).
```

```
Yes
```

```
X = file:/c:/jiprolog
```

```
JIP:-searchpath('c:/code/myprolog/').
```

```
Yes
```

simple/1

simple(Term)

Succeeds if Term is an atom, a number or a variable.

Note that Month is 1 based different from java.util.GregorianCalendar that is 0 based

true/0

true

Always succeeds.

unconsult/1

unconsult(filename)

Abolishes all clauses previously consulted or loaded in the file specified by FileName.

unify/2

unify(term1, term2)

Succeeds if Term1 unify with Term2

It is defined as =/2

unload/1

unload(FileName)

Abolishes all clauses previously consulted or loaded in the file specified by FileName.

var/1

var(Term).

Succeeds if Term is instantiated to a variable.

ver/1

ver(X)

Unifies X with JIProlog version number.

write/1

write(Term)

Writes the term Term to the current output stream. If Term will be read by read/1 or read/2, the call write('.') is needed.

write_canonical/1

write_canonical(Term)

Writes the Term in canonical way, i.e. without considering any operator definitions.

xcall/2

xcall(ClassName, Params).

Create an instance of the extension class (a class derived from JIPXCall class) specified in ClassName and calls the method JIPXCall.unify passing the Params argument. If unify succeeds the method JIPXCall.unify bind is called to bind Params to some term.

The developer can create your own extension class simply deriving his class from the JIPXCall class

See the section "How to write a custom predicates"

```
xcall('my.package.MyRandomNumGen', [1, 6, Rand]).
```

Creates an instance of the class "my.package.MyRandomNumGen" passing it the list [1, 6, Rand]

10 JIPxxml Package

10.1 Predicates Exported by JIPxxml Package

xml_append_attribute/3

xml_append_attribute(Attribute, Element, NewElement)

Appends Attribute to the attribute list of Element and unifies the result with NewElement.

xml_append_child/3

xml_append_child(Child, Element, NewElement)

Appends Child to the children of Element and unifies the result with NewElement.

xml_attribute/3

xml_attribute(Name, Value, AttributeTerm)

Succeeds if AttributeTerm unifies with the term xml_attribute(Name, Value).

It can be used to create an *attribute* object

xml_cdata/2

xml_cdata(Content, CDataTerm)

Succeeds if CDataTerm unifies with the term xml_cdata(Content).

It can be used to create a *CDATA* object

xml_child/1

xml_child(Element)

Succeeds if Element can be a child of some element, that is: *CDATA*, *Comment*, *Text*, *Element*, *Processing Instruction*.

xml_comment/2

xml_comment(Content, CommentTerm)

Succeeds if CommentTerm unifies with the term xml_comment(Content).

It can be used to create a *comment* object

xml_document/5

xml_document(Version, Encoding, DocType, Root, DocumentTerm)

Succeeds if DocumentTerm unifies with the term:

xml_document([version = Version, encoding = Encoding], DocType, Root).

It can be used to create a *document* object

xml_doctype/4

xml_doctype(Name, Id, Content, DocTypeTerm)

Succeeds if DocTypeTerm unifies with the term:

xml_doctype(Name, Id, Content).

It can be used to create a *document type* object

xml_doctype_id/2

xml_doctype_id(DocTypeTerm, Id)

Succeeds if id unifies with the id of DocTypeTerm.

xml_document_encoding/2

xml_document_encoding(DocumentTerm, Encoding)
Succeeds if Encoding is the encoding of DocumentTerm

xml_document_root/2

xml_document_root(DocumentTerm, Root)
Succeeds if Root is the root of DocumentTerm

xml_document_version/2

xml_document_version(DocumentTerm, Version)
Succeeds if Version is the root of DocumentTerm

xml_element/4

xml_element(Name, Attributes, Children, ElementTerm)
Succeeds if ElementTerm unifies with the term xml_element(Name, Attributes, Children)
It can be used to create a *element* object

xml_element_attributes/2

xml_element_attributes(Element, Attributes)
Succeeds if Attributes is the attribute list of Element

xml_element_attribute_by_name/2

xml_element_attribute_by_name(Name, Element, Attribute)
Succeeds if Element has an attribute with Name and unifies Attribute with it

xml_element_children/2

xml_element_children(Element, Children)
Succeeds if Children is the children list of Element

xml_element_child_by_name/2

xml_element_child_by_name(Name, Element, Attribute)
Succeeds if Element has an attribute with Name and unifies Attribute with it

xml_object/1

xml_object(Element)
Succeeds if Element is an xml object, that is: CDATA, Comment, Text, Element, Processing Instruction, Document, DocumentType, Attribute

xml_object_name/2

xml_object(Element, Name)
Succeeds if Name unifies with Element 's Name.
Note: only Element, Attribute, Processing Instruction and DocumentType objects has a Name

xml_object_type/2

xml_object_type(Element, Type)
Succeeds if Type unifies with Element 's type

xml_object_value/2

xml_object_value(Element, Value)

Succeeds if Value unifies with Element's Value

Note: only Attribute CDATA, Text, Comment, and Processing Instruction has a Value

xml_pi/3

xml_pi(Name, Content, PITerm)

Succeeds if PITerm unifies with the term xml_pi(Name, Content)

It can be used to create a *processing instruction* object

xml_read_document/1

xml_read_document(Term)

Reads an XML document from the current input stream.

The document is validated if it specifies a DTD.

Note: input stream cannot be the user input stream

xml_read_document/2

xml_read_document(Handle, Term)

Reads an XML document from the input stream pointed by Handle

The document is validated if it specifies a DTD.

xml_remove_attribute/3

xml_remove_attribute(Attribute, Element, NewElement)

Removes Attribute to the attribute list of Element and unifies the result with NewElement.

xml_remove_child/3

xml_remove_child(Child, Element, NewElement)

Removes Child from the children of Element and unifies the result with NewElement.

xml_text/3

xml_text(Content, TextTerm)

Succeeds if TextTerm unifies with the term xml_text(Content)

It can be used to create a *text* object

xml_write_document/1

xml_write_document(XMLDoc)

Writes the XML document in XMLDoc to the current output stream.

xml_write_document/2

xml_write_document(Handle, Term)

Writes the XML document in XMLDoc to the output stream pointed by Handle

11 JIPxhtml Package

11.1 Predicates Exported by JIPxhtml Package

html_append_attribute/3

html_append_attribute(Attribute, Tag, NewTag)

Appends Attribute to the attribute list of Tag and unifies the result with NewTag.

html_append_child/3

html_append_child(Child, Tag, NewTag)

Appends Child to the children of Tag and unifies the result with NewTag.

html_attribute/3

html_attribute(Name, Value, AttributeTerm)

Succeeds if AttributeTerm unifies with the term html_attribute(Name, Value).

It can be used to create an *attribute* object

html_child/1

html_child(Element)

Succeeds if Element can be a child of some element, that is: Comment, Text, Tag.

html_comment/2

html_comment(Content, CommentTerm)

Succeeds if CommentTerm unifies with the term html_comment(Content).

It can be used to create a *comment* object

html_document/3

html_document(DocType, Root, DocumentTerm)

Succeeds if DocumentTerm unifies with the term: html_document(DocType, Root).

It can be used to create a *html document* object

html_doctype/3

html_doctype(Name, Specs, DocTypeTerm)

Succeeds if DocTypeTerm unifies with the term: html_doctype(Name, Specs).

It can be used to create a *document type* object

html_doctype_specs/2

html_doctype_specs(DocTypeTerm, Specs)

Succeeds if id unifies with the id specification of DocTypeTerm.

html_document_root/2

html_document_root(DocumentTerm, Root)

Succeeds if Root is the root of DocumentTerm

html_object/1

html_object(Element)

Succeeds if Element is an html object, that is: Comment, Text, Tag, Document, DocumentType, Attribute

html_object_name/2

html_object(Element, Name)

Succeeds if Name unifies with Element 's Name.

Note: only Tag, Attribute, and DocumentType objects has a Name

html_object_type/2

html_object_type(Element, Type)
Succeeds if Type unifies with Element 's type

html_object_value/2

html_object_value(Element, Value)
Succeeds if Value unifies with Element 's Value
Note: only Attribute, Text and Comment has a Value

html_read_document/1

html_read_document(Term)
Reads an HTML document from the current input stream.
Note: input stream cannot be the user input stream

html_read_document/2

html_read_document(Handle, Term)
Reads an HTML document from the input stream pointed by Handle

html_remove_attribute/3

html_remove_attribute(Attribute, Tag, NewTag)
Removes Attribute to the attribute list of Tag and unifies the result with NewTag.

html_remove_child/3

html_remove_child(Child, Tag, NewTag)
Removes Child from the children of Tag and unifies the result with NewTag.

html_tag/4

html_tag(Name, Attributes, Children, TagTerm)
Succeeds if TagTerm unifies with the term html_tag(Name, Attributes, Children)
It can be used to create a tag object

html_tag_attributes/2

html_tag_attributes(Tag, Attributes)
Succeeds if Attributes is the attribute list of Tag

html_tag_attribute_by_name/2

html_tag_attribute_by_name(Name, Tag, Attribute)
Succeeds if Tag has an attribute with Name and unifies Attribute with it

html_tag_children/2

html_tag_children(Tag, Children)
Succeeds if Children is the children list of Tag

html_tag_child_by_name/2

html_tag_child_by_name(Name, Tag, Attribute)
Succeeds if Tag has an attribute with Name and unifies Attribute with it

html_text/3

html_text(Content, TextTerm)

Succeeds if TextTerm unifies with the term html_text(Content)

It can be used to create a *text* object

html_write_document/1

html_write_document(HTMLDoc)

Writes the HTML document in HTMLDoc to the current output stream.

html_write_document/2

html_write_document(Handle, Term)

Writes the HTML document in HTMLDoc to the output stream pointed by Handle

12 JIPxreflect package

12.1 Predicates Exported by JIPxreflect Package

create_object/3

`create_object(ConstructorProto, Params, Handle)`

Creates the object specified in ConstructorProto calling the constructor whose prototype matches with the given one with the parameters specified in Params. If creation succeeds Handle is unified with a reference to the created object.

ConstructorProto can be a functor or an Atom. In the former case the functor name specifies the class name while the parameters specify the class of the constructor parameters. In the latter case default constructor is used and Params must be [].

Es.:

```
JIP:-load_library('jipxreflect.jar') % this is not needed with JIPrologIDE and JIPConsole
Yes
```

```
JIP:-create_object('java.lang.String'('java.lang.String'), ['Hello World'], Handle).
Yes
Handle = #12345
```

```
JIP:-create_object('java.lang.StringBuffer', [], Handle).
Yes
Handle = #67890
```

invoke/4

`invoke(Handle, MethodProto, Params, RetVal)`

Invokes the method specified in MethodProto on the object or class referenced by Handle passing the parameters specified in Params. If the invocation succeeds RetVal is unified with the value returned by the method.

Handle can be a reference to an object obtained by `create_object/3` or can be a class name. The former case causes a call to an object method. The latter case causes a call to class method (static).

MethodProto can be a functor or an Atom. In the former case the functor name specifies the method name while the parameters specify the class of the method parameters. In the latter case method is assumed having no parameters and Params must be [].

Es.

```
JIP:-invoke('#12345', substring(int, int), [0, 5], RetVal).
Yes
RetVal = Hello
```

```
JIP:-invoke('java.lang.String', valueOf(int), [5], RetVal).
Yes
RetVal = 5
```

get/3

`get(Handle, FieldName, Val)`

Gets the value of a field related to the object or class referenced by Handle.

Handle can be a reference to an object obtained by `create_object/3` or can be a class name (static)

Es.:

```
JIP:-get('#12345', myName, Val).
Yes
Val = Ugo
```

```
JIP:-get('java.io.File', separator, Val).
Yes
Val = \
```

get_class/2

get_class(Handle, ClassName)

Retrieves the class of the object specified in Handle

Es.:

```
JIP:-get_class('#12345', ClassName).  
Yes  
ClassName = java.lang.String
```

get_constructors/2

get_constructors(Handle, MethodList)

Retrieves the constructors of an object or class.

Handle can be a reference to an object obtained by create_object/3 or can be a class name.

Es.:

```
JIP:-get_constructors('java.lang.Integer', X).  
Yes  
X = [java.lang.Integer(java.lang.String), java.lang.Integer(int)]
```

get_methods/2

get_methods(Handle, MethodList)

Retrieves the methods of an object or class.

Handle can be a reference to an object obtained by create_object/3 or can be a class name.

Es.:

```
JIP:-get_methods('java.lang.Object', MethodList).  
Yes  
MethodList = [hashCode, getClass, wait(long, int), wait, wait(long),  
equals(java.lang.Object), toString, notify, notifyAll]
```

get_fields/2

get_fields(Handle, FieldList)

Retrieves the fields of an object or class.

Handle can be a reference to an object obtained by create_object/3 or can be a class name.

Es.:

```
JIP:-get_fields('java.lang.String', X).  
Yes  
X = [CASE_INSENSITIVE_ORDER]
```

set/3

set(Handle, FieldName, Val)

Sets the value of a field related to the object or class referenced by Handle.

Handle can be a reference to an object obtained by create_object/3 or can be a class name (static)

Es.:

```
JIP:-set('#12345', myName, ugo).  
Yes
```

release_object/1

release_object(Handle)

Releases the object referenced by Handle. Java Garbage Collector will free the related memory.

```
JIP:-release_object('#12345').  
Yes
```

13 JIPxWin package

13.1 *Predicates Exported by JIPxWin Package*

winputbox/2

winputbox(Message, Output).

Shows a dialog with the message contained in Message, an input field and an OK and Cancel buttons. If the user click on OK, when winputbox returns Output is instantiated to the string obtained from the text typed in the input field.

wmsgbox/1

wmsgbox(Message)

Shows a dialog with the message contained in Message.

wyesnobox/1

wyesnobox(Message)

Shows a dialog with Yes/No button showing the message contained in Message.

wtextbox/1

wtextbox(Message)

Shows a dialog with the message contained in Message.

Useful for long text

Appendix A: How To...

How to call the interpreter synchronously

To call the interpreter synchronously refer to the following skeleton:

```
// New instance of Prolog engine
JIPEngine jip = new JIPEngine();

JIPTerm query = null;

try
{
    // parse query
    JIPTermParser parser = jip.getTermParser();
    query = parser.parseTerm("write('hello world'), nl.");
}
catch(JIPSyntaxErrorException ex)
{
    // there is a syntax error in the query
    ex.printStackTrace();
    System.exit(0);
}

// open a synchronous query
JIPQuery jipQuery = jip.openSynchronousQuery(query);
JIPTerm solution;

// Loop while there is another solution
while (!jipQuery.hasMoreChoicePoints())
{
    solution = jipQuery.nextSolution();
    System.out.println(solution);
}
```

How to call the interpreter asynchronously

To call the interpreter asynchronously refer to the following skeleton:

```
// New instance of Prolog engine
JIPEngine jip = new JIPEngine();

JIPTerm query = null;

try
{
    // parse query
    JIPTermParser parser = jip.getTermParser();
    query = parser.parseTerm("?- write('hello world'), nl.");
}
catch(JIPSyntaxErrorException ex)
{
    // there is a syntax error in the query
    ex.printStackTrace();
    System.exit(0);
}

// add listeners previously instantiated
jip.addEventListener(eventListener);

// opens query
starts the interpreter and exits immediately
int m_nQueryHandle = jip.openQuery(query);
```

The result of the query is notified to registered event listeners as showed below

```
// open event occurred
public void openNotified(JIPEvent e)
{
    System.out.println("Query n." + e.getQueryHandle());
    System.out.println("open");
}
```



```

// more event occurred
public void moreNotified(JIPEvent e)
{
    System.out.println("Query n." + e.getQueryHandle());
    System.out.println("more");
}

// A solution event occurred
public void solutionNotified(JIPEvent e)
{
    System.out.println("Query n." + e.getQueryHandle());
    System.out.println("solution:");
    System.out.println(e.getTerm());
}

// A Term has been notified with notify/2
public void termNotified(JIPEvent e)
{
    System.out.println("Query n." + e.getQueryHandle());
    System.out.println("notified");
}

// The end has been reached because there wasn't more solutions
public void endNotified(JIPEvent e)
{
    System.out.println("Query n." + e.getQueryHandle());
    System.out.println("end");

    // get the source of the query
    JIPEngine jip = e.getSource();

    // close query
    jip.closeQuery(m_nQueryHandle);
}

public void closeNotified(JIPEvent e)
{
    System.out.println("Query n." + e.getQueryHandle());
    System.out.println("close");
}

// An error (exception) has been raised up by Prolog engine
public void errorNotified(JIPEngineEvent e)
{
    System.out.println("Query n." + e.getQueryHandle());
    System.out.println("Error:");
    System.out.println(e.getError());

    // get the source of the query
    JIPEngine jip = e.getSource();

    // close query
    jip.closeQuery(m_nQueryHandle);
}

```

How to write a custom built-in predicate:

To write a custom predicate you need to write a class derived from the abstract class `JIPXCall` implementing the following abstract methods:

```

public boolean unify(JIPCons params, Hashtable varsTbl)

public boolean hasMoreChoicePoints()

```

The method `unify` must return true if the corresponding predicate succeeds (i.e. unifies) given the arguments in `params`. The parameter `varsTbl` on exit will contain all variables instantiated during unification. The method `hasMoreChoicePoints` must return true if the corresponding predicate has more choice-points on backtracking.

The following is an implementation of a custom predicate that generates random numbers in the range [min, max]:

```

import com.ugos.JIProlog.engine.*;
import java.util.*;

public class RandomNumberGen3 extends JIPXCall

```

```

{
    Random m_random;

    int m_nMin = -1
    m_nMax = -1;

    public boolean unify(JIPCons input
    Hashtable varsTbl)
    {
        // input is a List like [Min
    Max
    Rand]

        // set min max parameters only the first time
        if(m_random == null)
        {
            // initialize the parameters
            m_random = new Random()

            JIPTerm term = input.getNth(1);
            if(term instanceof JIPVariable)
                term = ((JIPVariable)term).getValue();

            JIPNumber min = (JIPNumber)term;

            term = input.getNth(2);
            if(term instanceof JIPVariable)
                term = ((JIPVariable)term).getValue();

            JIPNumber max = (JIPNumber)term;

            m_nMin = (int)min.getValue();
            m_nMax = (int)max.getValue();
        }

        int n = (int)(m_random.nextDouble() * (m_nMax - m_nMin)) + m_nMin;
        JIPNumber rand = JIPNumber.create(n);
        return input.getNth(3).unify(rand
varsTbl);
    }

    public boolean hasMoreChoicePoints()
    {
        return true;
    }
}

```

Compile your class and put the .class file in myjar.jar archive, then load it by load_library/1

```

JIP:-load_library('myjar.jar')
yes

```

Now call your custom predicate by typing:

```

JIP:-xcall("RandomNumberGen3", [1, 10, X]).
yes
X = 8
more (y/n):

```

For a smart usage you can attach your custom predicate to a Prolog predicate by defining the following clause:

```

(1) randomNumberGen(Min, Max, Rand):-
    xcall("RandomNumberGen3", [Min, Max, Rand]).

```

Note that load_library/1 loads the library specified and consults the file init.pl if in the .jar file there is one. Thus, if the definition (1) is added into your init.pl, when the library is loaded the new class is automatically linked to the related Prolog predicate.

See the samples RandomNumberGen in samples supplied with the SDK

How to link a JDBC external database of clauses:

JIPxdb package supply two classes: JDBCCLausesDatabase and JDBCCLausesEnumeration. Such classes allow linking a predicate to a table or a view (SQL query) in a database by JDBC.

The syntax to link a predicate to a specific table or view in a database by JDBC is:

```
JIP:-extern(myPred/n, 'com.ugos.JIProlog.extensions.database.JDBCClausesDatabase',
'driverName,view').
```

where

- driverName is the name used in JDBC to load the driver related to the RDBMS;
- url specifies the url used to open a connection to the database via the method DriverManager.getConnection (see JDBC specification);
- view identifies a table or a valid SQL SELECT query as specified below that has *n* columns (where *n* is the arity of *myPred*).

If the access to the database requires username and password they can be added in the url field or also in the following way:

```
JIP:-extern(myPred/n, 'com.ugos.JIProlog.extensions.database.JDBCClausesDatabase',
'driverName,url,view@username,password').
```

How to link a table

Suppose you have a table "myTable" with 3 columns in a ODBC database with DSN="myDSN".

To link such table to the predicate "myPred/3" type:

```
JIP:-load_library('jipxdb.jar'). % this is not needed with JIPrologIDE and JIPConsole
yes
```

```
JIP:-extern(myPred/3, "com.ugos.JIProlog.extensions.database.JDBCClausesDatabase",
"sun.jdbc.odbc.JdbcOdbcDriver,jdbc:odbc:myDSN,table:myTable").
```

How to link a view specified by an SQL SELECT query

Suppose you have a SQL select query with 2 columns in an ODBC database with DSN="myDSN".

To link such view to the predicate "myPred/2" type:

```
JIP:-load_library('jipxdb.jar'). % this is not needed with JIPrologIDE and JIPConsole
yes
```

```
JIP:-extern(myPred/2, "com.ugos.JIProlog.extensions.database.JDBCClausesDatabase",
"sun.jdbc.odbc.JdbcOdbcDriver,jdbc:odbc:myDSN,sql:SELECT FIELD1, FIELD2 FROM myTable").
```

General notes:

1. JDBCClausesDatabase allows asserting and retracting clauses only if the predicate is linked to a table. It fails if the predicate is linked to a view.
 2. to retract a clause JDBCClausesDatabase uses the query "DELETE ONCE FROM".
- Because not all JDBC drivers support the statement "ONCE" some drivers may have different behavior
-

How to write a custom bridge to an external database of clauses:

An external database of clauses is a database not stored in the Prolog memory but in an external system such as an RDBMS.

To implement a custom bridge to an external database of clauses you must derive your class from the abstract class JIPClausesDatabase and implement a JIPClausesEnumeration to enumerate all facts in the table.

In the class derived from JIPClausesDatabase you must implement:

```
/** Set attributes to pass to the database (i.e. login info, filename, etc.)
public abstract void setAttributes(String strAttribs);
/** Add a clause to the database at the position specified
public abstract boolean addClauseAt(int nPos, JIPClause clause);
/** Append a clause to the database
public abstract boolean addClause(JIPClause clause);
/** Remove a clause to the database
public abstract boolean removeClause(JIPClause clause);
/** Return an enumeration of clauses contained in the database
public abstract Enumeration clauses();
```

In the class derived from JIPClausesEnumeration must implement the method:

```
/** Get next clause in the enumeration of clauses
public abstract JIPClause nextClause();
```

Refer to the sample "jipxdb" in samples directory.

The files TextClausesDatabase.java and TextClausesEnumeration.java link a predicate to a table stored in a text file.

The files JDBCCLausesDatabase.java and JDBCCLausesEnumeration.java implement a database of clauses stored in JDBC connected database.

The files PrologClausesDatabase.java and PrologClausesEnumeration.java link a predicate to a table stored in stored in Prolog file.

How to consult a very large Prolog file:

If you need to use larger files you can declare them as an external Prolog database of clauses.

Load the package jipxdb.jar by typing:

```
JIP:-load_library('jipxdb.jar'). % this is not needed with JIPrologIDE and JIPConsole
yes
```

```
JIP:- extern(yourpredicate/n, 'com.ugos.JIProlog.extensions.database.PrologClausesDatabase'
"yourprologfile.pl").
yes
```

In this way yourpredicate/n is linked to the facts in yourprologfile.pl and can be used as a standard predicate.

Note that in this version Prolog file should contain only facts without comments or rules.

See the example "database" in the "samples" directory.

How to compile and distribute a Prolog program:

JIProlog allows compiling source files transforming Prolog code in compiled code in an unreadable format.

To compile a file you should use compile/1 predicate:

```
JIP:-compile('myprologfile.pl').
```

The result is a compiled file called "myprologfile.jip"

To load a compiled file you should use load/1 predicate

```
JIP:-load('myprologfile.jip').
```

How to implement a JIProlog client

JIProlog supplies a daemon that allows a remote client to submit queries over the network via TCP/IP protocol.

Implementing a client results very easy using the class com.ugos.JIProlog.client.JIPClient

The code below shows an implementation of a simple client that submit a query:

```
import com.ugos.JIProlog.client.*
public static void main(String[] args)
{
    try
    {
        // Create new client
        JIPClient jipClient = new JIPClient("localhost");

        // open a query
        jipClient.openQuery("?-member(X, [a,b,c,d,e,f,g])");

        // get all solutions
        JIPSolution solution;
        while((solution = jipClient.nextSolution()) != null)
        {
            System.out.println(solution);
        }

        // close query
```

```
        jipClient.closeQuery();  
        // leave the connection  
        jipClient.leave();  
    }  
    catch(IOException ex)  
    {  
        ex.printStackTrace();  
    }  
    catch(JIPServerException ex)  
    {  
        ex.printStackTrace();  
    }  
}
```

See also the sample JIPClientSample in the samples directory

Appendix C: Prolog miscellaneous

The standard ordering of terms

variables are less than:

integers and floats which are less than:

atoms which are less than:

strings which are less than:

lists which are less than:

compound terms which are less than:

true conjunctions which are less than:

true disjunctions

Prolog Resources

Internet:

- ISO Prolog: J.P.E. Hodgson, <http://pauillac.inria.fr/~deransar/prolog/docs.html>
- Nilsson, U., Maluszynski, J., *Logic, Programming and Prolog (2ed)*, <http://www.ida.liu.se/~ulfni/lpp/>
- A comprehensive Prolog course: <http://www.coli.uni-sb.de/~kris/prolog-course/>
- Interactive Prolog Guide: <http://ktlinux.ms.mff.cuni.cz/~bartak/prolog/>
- Advanced Example: <http://kti.ms.mff.cuni.cz/~bartak/prolog.old/examples.html>
- Logic Programming: <http://www.afm.sbu.ac.uk/logic-prog/>
- FAQ: <http://www.cs.cmu.edu/Groups/AI/html/faqs/lang/prolog/prg/top.html>

Books:

- I. Bratko, *Prolog Programming for Artificial Intelligence*, 1990.
- Y. Shoham, *Artificial Intelligence Techniques in Prolog*, 1994.
- Leon Sterling, Ehud Shapiro: *Prolog*, Addison-Wesley, 1988
- W.F. Clocksin, C.S.Mellish, *Programming in Prolog*, Springer-Verlag

Appendix C: List of predicates supplied by JIProlog

! / 0	merge2 / 4
!] / 0	merge_set / 3
\$current_query_handle / 1	meta_predicate / 1
\$error / 1	module_transparent / 1
\$free / 1	mother / 2
-> / 2	ms / 2
/ / 2	msort / 3
< / 2	msort1 / 5
= / 2	multifile / 1
=.. / 2	name / 2
:= / 2	nextto / 3
=< / 2	nil / 1
== / 2	nl / 0
=\= / 2	nl / 1
> / 2	no_end_tag / 1
>= / 2	nodebug / 0
@< / 2	nonvar / 1
@= / 2	nospy / 1
@=< / 2	not / 1
@> / 2	notify / 2
@>= / 2	notrace / 0
C / 3	nth0 / 3
[! / 0	nth1 / 3
\+ / 1	number / 1
\= / 2	number_chars / 2
\== / 2	number_codes / 2
^ / 2	numbervars / 3
abolish / 1	numlist / 3
abolish_files / 1	numlist_ / 3
abort / 0	once / 1
absolute_file_name / 2	one / 1
absolute_file_name / 3	op / 3
access_file / 2	open / 3
append / 1	open / 4
append / 2	order / 3
append / 3	peek_byte / 1
apply / 2	peek_byte / 2
arg / 3	peek_chars / 1
assert / 1	peek_chars / 2
asserta / 1	peek_code / 1
assertz / 1	peek_code / 2
at_end_of_stream / 0	permutation / 2
at_end_of_stream / 1	permutation / 3
atom / 1	phrase / 2
atom_chars / 2	phrase / 3
atom_codes / 2	print / 1
atom_concat / 3	print / 2
atom_length / 2	put / 1
atom_number / 2	put / 2
atom_prefix / 2	put_byte / 1
atomic / 1	put_byte / 2
bagof / 3	put_char / 1
bsort / 3	put_char / 2
call / 1	put_code / 1
call / 2	put_code / 2
call / 3	qsort / 3
call / 4	read / 1
call / 5	read / 2
callable / 1	read_clause / 2
catch / 3	read_clause / 3
cd / 1	read_term / 2
char / 1	read_term / 3
char_atom / 2	release_object / 1
char_code / 2	remove_duplicates / 2
chars / 1	rename_file / 1
chdir / 1	repeat / 0
check_eof / 2	repeat / 1

check_handle / 2	retract / 1
checklist / 2	retractall / 1
clause / 2	reverse / 2
close / 1	reverse / 4
close / 2	same_file / 2
compare / 3	searchpath / 1
compile / 1	see / 1
compound / 1	see / 2
concat_atom / 2	seeing / 1
concat_atom / 3	seeing / 2
consult / 1	seek / 4
convert_chars / 2	seen / 0
convert_time / 8	seen / 1
copy_term / 2	select / 3
create_object / 3	set / 3
current_atom / 1	set_eof / 2
current_functor / 2	set_input / 1
current_input / 1	set_output / 1
current_op / 3	set_properties / 2
current_output / 1	set_stream / 2
current_predicate / 1	set_stream_position / 2
current_predicate / 2	setof / 3
current_stream / 3	shell / 1
date / 3	shell / 2
date / 4	simple / 1
daughter / 2	size_file / 2
dcg_and / 3	skip / 1
dcg_or / 4	skip / 2
dcg_rhs / 4	sleep / 1
debugging / 0	son / 2
declare_extern / 3	sort / 2
delete / 3	split / 5
delete_directory / 1	spy / 1
delete_file / 1	statistics / 0
difference / 3	stream_property / 2
dir / 0	string / 1
dir / 1	string_concat / 3
display / 1	string_length / 2
display / 2	string_to_atom / 2
downcase / 1	string_to_list / 2
downcase_atom / 2	sublist / 3
downcase_char / 2	subset / 2
downcase_chars / 2	sumlist / 2
dynamic / 1	sumlist / 3
ensure_loaded / 1	syndiff / 3
exists_directory / 1	tab / 1
exists_file / 1	tab / 2
expand_term / 2	tag / 4
extern / 3	tell / 1
fail / 0	tell / 2
false / 0	telling / 1
father / 2	telling / 2
female / 1	throw / 1
file_attributes / 7	time / 1
file_base_name / 2	time / 4
file_directory_name / 2	time / 5
file_name_extension / 3	time / 7
findall / 3	time / 8
flatten / 2	time_file / 2
flatten1 / 3	told / 0
flatten2 / 2	told / 1
float / 1	trace / 0
flush_output / 0	translate / 2
flush_output / 1	true / 0
forall / 2	unconsult / 1
free_variables / 2	union / 3
functor / 3	unload / 1
garbage_collect / 0	upcase / 1
get / 1	upcase_atom / 2
get / 2	upcase_char / 2
get / 3	upcase_chars / 2

get0 / 1	use_module / 1
get0 / 2	var / 1
get_byte / 1	ver / 1
get_byte / 2	ver / 2
get_char / 1	wait / 1
get_char / 2	winputbox / 2
get_class / 2	wmsgbox / 1
get_code / 1	working_directory / 2
get_code / 2	write / 1
get_constructors / 2	write / 2
get_fields / 2	write_attributes / 1
get_methods / 2	write_attributes / 1
get_time / 1	write_attributes / 2
granfather / 2	write_attributes / 2
ground / 1	write_canonical / 1
html_append_attribute / 3	write_canonical / 2
html_append_child / 3	write_dir / 1
html_attribute / 3	write_doctype / 1
html_child / 1	write_doctype / 2
html_comment / 2	write_elements / 1
html_doctype / 3	write_elements / 1
html_doctype_specs / 2	write_elements / 2
html_document / 3	write_elements / 2
html_document_root / 2	write_id / 1
html_object / 1	write_id / 2
html_object_name / 2	write_prolog / 1
html_object_type / 2	write_prolog / 1
html_object_value / 2	write_prolog / 2
html_read_document / 1	write_prolog / 2
html_read_document / 2	write_term / 2
html_remove_attribute / 3	write_term / 3
html_remove_child / 3	writeln / 1
html_tag / 4	writeln / 2
html_tag_attribute_by_name / 3	writeln / 2
html_tag_attributes / 2	writeln / 2
html_tag_by_name / 3	writeln / 2
html_tag_child_by_name / 3	writeln / 2
html_tag_children / 2	writeln / 2
html_text / 2	writeln / 2
html_write_document / 1	writeln / 2
html_write_document / 2	writeln / 2
ignore / 1	writeln / 2
integer / 1	writeln / 2
intersection / 3	writeln / 2
invoke / 4	writeln / 2
is / 2	writeln / 2
is_absolute_file_name / 1	writeln / 2
is_list / 1	writeln / 2
is_set / 1	writeln / 2
last / 2	writeln / 2
last_ / 3	writeln / 2
length / 2	writeln / 2
likes / 2	writeln / 2
list / 1	writeln / 2
list_to_set / 2	writeln / 2
list_to_set_ / 2	writeln / 2
listing / 0	writeln / 2
listing / 1	writeln / 2
load / 1	writeln / 2
load_files / 1	writeln / 2
load_library / 1	writeln / 2
make_connects / 4	writeln / 2
make_directory / 1	writeln / 2
male / 1	writeln / 2
maplist / 3	writeln / 2
member / 2	writeln / 2
member / 3	writeln / 2
memberchk / 2	writeln / 2
merge / 3	writeln / 2