

Pedro

version 1.3, 25 March 2009

Peter Robinson (pjr@itee.uq.edu.au)

This manual is for Pedro (version 1.3, 25 March 2009), a subscription/notification system.
Copyright © 2006, 2007, 2008 Peter Robinson

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Table of Contents

1	Introduction	1
2	Running Pedro	2
3	Syntax	3
3.1	Numbers	3
3.2	Atoms	3
3.3	Strings	3
3.4	Variables	4
3.5	Compound Terms	4
3.6	Lists	4
3.7	Operators	4
4	Notifications	6
5	Subscriptions	7
6	Removing Subscriptions	11
7	Registrations	12
8	Removing Registrations	13
9	Connecting to Pedro	14
10	Denial of Service	15
11	Library	16
12	APIs	17
12.1	Python API	17
12.2	Java API	18
Appendix A Copying This Manual		19
A.1	GNU Free Documentation License	19
Index		26

1 Introduction

Pedro is a subscription/notification system based on Prolog technology. The main component of the system is the Pedro server that is responsible for managing subscriptions from clients and for forwarding client notifications to clients with matching subscriptions.

From the clients point of view subscriptions and notifications are strings that represent Prolog terms. The server parses these strings into Prolog terms and treats subscriptions in a similar way to Prolog clauses and notifications in a similar way to Prolog queries to be matched against subscriptions.

Readers unfamiliar with Prolog are referred to the following books.

- *Prolog Programming for Artificial Intelligence*, Bratko
- *The Art of Prolog*, Sterling and Shapiro
- *Programming in Prolog*, Clocksin and Mellish
- *Techniques of Prolog Programming*, Van Lee

2 Running Pedro

Pedro is a daemon and is started using the following command.

```
pedro [OPTIONS..]
```

Using the help option `-?` produces the following output.

Usage:

```
pedro [OPTION...]
```

Help Options:

```
-?, --help          Show help options
```

Application Options:

```
--version          Show version  
-P, --Port=p       Use port p for connecting  
-S, --Size         The size for various areas  
-L, --Logfile=logfile Use logfile for logging  
-A, --Admin=admin_machine Use admin_machine for receiving all messages
```

If no log file is supplied then `/dev/null` is used. If `stdout` is used for the log file then Pedro will not be a daemon and the log information will be written to standard output.

If an admin machine is supplied then a process registering the name 'admin' on that machine will receive all message including peer-to-peer messages between other processes.

Otherwise the supplied log file must be writable by the caller.

3 Syntax

In this section we describe the syntax of the Prolog terms that are used for both subscriptions and notifications. The syntax we use is standard Prolog syntax with a cut-down set of the built-in prefix and infix operators.

3.1 Numbers

Pedro numbers are 32 bit integers in decimal notation and floating point numbers in decimal and scientific notation.

Examples:

```
Integers: 42, -100
Doubles: 3.14, -1.3e-5
```

3.2 Atoms

The syntax of Pedro atoms fall into four categories:

1. A lower case letter followed by any sequence consisting of “_” and alphanumeric characters.

Examples:

```
fred, a12, this_is_an_ATOM
```

2. Any combination of the following set of graphic characters.

```
-/+*<=>#@$%^&~‘:~?.
```

Examples:

```
==>, :?, -:-
```

3. Any sequence of characters enclosed by single quotes. The character “\” indicates an escape sequence.

Examples:

```
'Fred', 'it\'s', '\n'
```

4. The following.

```
;
[]
```

3.3 Strings

Pedro strings are sequences of characters enclosed by double quotes. As with quoted atoms, the character “\” indicates an escape sequence.

Examples:

```
"This is a string", "A string with a ', a \" and a newline \n"
```

3.4 Variables

The syntax of Pedro variables fall into two categories.

1. An upper case letter or “_” followed by any sequence consisting of “_” and alphanumeric characters.

Examples:

```
X, X_23, _fred_12
```

2. The “anonymous” variable “_”

Note that repeated occurrences of a variable token of the first category in a string representing a Pedro term represent the same variable. This is not the case for anonymous variables: each occurrence represents a different variable.

3.5 Compound Terms

The syntax of a compound Pedro term consists of an atom token (*the functor*) followed by an opening bracket followed by a comma separated list of syntax representing Pedro terms (*the arguments*) followed by a closing bracket.

A compound term must have at least one argument. The *arity* of a compound term is the number of arguments.

Examples:

```
f(a), g(Var, "string", 42, h(a, g(b)))
```

3.6 Lists

The atom [] represents the empty list.

The syntax of a non-empty Pedro list consists of an opening square bracket followed by a comma separated list of syntax representing Pedro terms (optionally followed by a “|” followed by syntax representing Pedro term) followed by a closing square bracket.

Examples:

```
[], [V, a, f(2), [a,b,c], "string'], [H1, H2|T]
```

The list syntax involving “|” is typically used to describe list patterns. For example, the third example above might be used to match against lists with at least two elements (H1 and H2) and whose tail is T.

3.7 Operators

Below is the table of operators used in Pedro. The table lists the declarations of the operators as used in Prolog. In Prolog, the declaration `op(Prec, Assoc, Ops)` declares the list of operators `Ops` to have precedence `Prec` and associativity `Assoc`. The smaller the precedence, the more tightly the operator binds. For the associativity argument `xfy` describes a right-associative infix operator, `xfx` describes a non-associative infix operator, `yfx` describes a left-associative infix operator, and `fy` describes an associative prefix operator.

```
op(1100, xfy, [ ; ]).
op(1050, xfy, [ -> ]).
op(1000, xfy, [ ', ' ]).
op(700, xfx, [ =, is, <, =<, >, >= ]).
```

```

op(500, yfx, [ + , - , /\ , \/ ]).
op(400, yfx, [ * , / , // , rem , mod , << , >> ]).
op(200, xfx, [ ** ]).
op(200, fy, [ + , - ]).
op(100, xfx, [ @ ]).
op(50, xfx, [ : ]).

```

Examples:

The string "X is A + - B + C*D" parses to the term `is(X, +(+(A, -(B)), *(C, D)))`■

The string "G1 -> G2 ; G3" parses to the term `;->(G1, G2), G3`

Note that comma is used both as an argument separator and as an infix operator. Each operator used at the top-level of an argument of a list or compound term has to have precedence less than 1000. If an argument has an operator of higher precedence then the argument needs to be enclosed in brackets.

Examples:

The string "f((G1, G2))" parses to the compound term `f(','(G1, G2))`

The string "[G, (G1;G2)]" parses to the list `[G, ;(G1, G2)]`

4 Notifications

Notifications are newline terminated strings sent from clients to the Pedro server. A valid notification is a string that represents an atom, a list or a compound Pedro term (with a following newline). The Pedro server reads characters from a client until it gets a newline. The server will then attempt to parse the characters up to (but not including) the newline as a compound Pedro term. If this succeeds then the server processes the notification term; otherwise the string is ignored.

Examples (trailing newline removed):

Valid:

```
info(fred, 42, "some string")
```

Invalid (not compound, atom or list):

```
"bad"
```

```
X
```

Invalid (does not parse):

```
f(a;b)
```

```
f([a,b))
```

The server will acknowledge the client with a 1 if the notification is valid and a 0 otherwise.

5 Subscriptions

Like notifications, subscriptions are newline terminated strings that parse as compound Pedro terms. The functor of each subscription term is always `subscribe` and has arity 3. In other words, each subscription term is of the form `subscribe(Head, Body, Rock)`. Following the similarity between subscriptions and Prolog clauses we refer to the first argument as the *head* of the subscription and the second argument as the *body* of the subscription.

The third argument is commonly referred to as a *rock*. This is an integer and its meaning is determined by each client. When the Pedro server matches a notification against the subscription, the server sends the notification string together with the rock to the subscribing client. A given client can use the rock to, for example, refer to a message queue or a thread and thereby determine how to process the notification.

The Pedro server will match a notification against a subscription if the notification term unifies with the head of the subscription and, with this unifier, the goal in the body of the subscription is satisfied. Readers are referred to Prolog references for descriptions of unification, variable binding, dereferencing, occurs check, backtracking and goal evaluation.

When the server matches a notification against a subscription it will send the string consisting of the subscribers rock followed by a space followed by the notification string (including the newline).

The server will acknowledge an attempt by a client to subscribe with a string consisting of an integer (an ID) followed by a newline. The ID will be 0 if the subscription attempt fails (the string is too long, it doesn't parse, or does not represent a valid subscription term). If the subscription attempt succeeds then the ID will be a unique (for that client) positive integer. This ID is used when the client chooses to unsubscribe.

The following table lists the “basic” valid subscription goals and their semantics that can be used in the body of subscriptions. As with Prolog, whenever a unification is carried out, the variable bindings implied by the unifier are applied.

Note that, unlike most Prologs, unification in Pedro uses the occurs check.

<code>true</code>	Always succeeds
<code>fail</code>	Always fails
<code>T1 = T2</code>	Succeeds if and only if the terms <code>T1</code> and <code>T2</code> unify.
<code>T1 is T2</code>	Succeeds if and only if <code>T1</code> unifies with the evaluation of the the arithmetic expression <code>T2</code> . The goal produces a type error if <code>T2</code> does not represent an arithmetic expression that can be fully evaluated (to a number). The valid arithmetic expressions are described later.
<code>T1 < T2</code>	Succeeds if and only if the arithmetic expression <code>T1</code> evaluates to a number that is less than the evaluation of the arithmetic expression <code>T2</code> . The goal produces a type error if either term does not represent an arithmetic expression that can be fully evaluated (to a number).
<code>T1 =< T2</code>	The same as above, except that a less-or-equal test is applied.
<code>T1 > T2</code>	The same as above, except that a greater-than test is applied.
<code>T1 >= T2</code>	The same as above, except that a greater-or-equal test is applied.

<code>member(X, L)</code>	Succeeds if and only if <code>L</code> is a “cons” pair and <code>X</code> unifies with the head element of <code>L</code> or <code>X</code> is a member of the tail of <code>L</code> .
<code>split(L1, L2, L3)</code>	Succeeds if and only if the concatenation of the lists <code>L2</code> and <code>L3</code> unifies with <code>L1</code> . The list <code>L1</code> must be supplied.
<code>splitstring(S1, S2, S3)</code>	Succeeds if and only if the concatenation of the strings <code>S2</code> and <code>S3</code> unifies with <code>S1</code> . The string <code>S1</code> must be supplied.
<code>number(T)</code>	Succeeds if and only if <code>T</code> is a number.
<code>atom(T)</code>	Succeeds if and only if <code>T</code> is an atom.
<code>string(T)</code>	Succeeds if and only if <code>T</code> is a string.
<code>list(T)</code>	Succeeds if and only if <code>T</code> is either the empty list or a “cons” pair.

The following table lists the valid meta-level subscription goals – i.e. goals that take valid goals as arguments.

<code>G1 , G2</code>	Conjunction: succeeds if and only if first the goal <code>G1</code> succeeds and then the goal <code>G2</code> succeeds.
<code>G1 ; G2</code>	Disjunction: succeeds if and only if either the goal <code>G1</code> succeed or the goal <code>G2</code> succeeds.
<code>G1 -> G2 ; G3</code>	If-then-else: If <code>G1</code> succeeds then alternative solutions for <code>G1</code> are removed and the goal succeeds if and only if <code>G2</code> succeeds. Otherwise, the goal succeeds if and only if <code>G3</code> succeeds.
<code>not(G)</code>	Negation: succeeds if and only if <code>G</code> fails. In Prolog this form of negations is called “unsafe negation”.
<code>once(G)</code>	Succeeds if and only if <code>G</code> succeeds. Alternative solutions are removed.

The following are valid arithmetic expressions. Numbers are valid arithmetic expressions and in the table below, `E1` and `E2` are valid arithmetic expressions.

<code>pi</code>	Pi
<code>e</code>	E
<code>- E1</code>	Negation
<code>E1 + E2</code>	Addition
<code>E1 - E2</code>	Subtraction
<code>E1 * E2</code>	Multiplication
<code>E1 / E2</code>	Division
<code>E1 // E2</code>	Integer division

<code>E1 ** E2</code>	Exponentiation
<code>E1 rem E2</code>	Remainder (<code>E1</code> and <code>E2</code> are integer expressions)
<code>E1 mod E2</code>	Modulo (<code>E1</code> and <code>E2</code> are integer expressions)
<code>E1 /\ E2</code>	Bitwise And (<code>E1</code> and <code>E2</code> are integer expressions)
<code>E1 \/ E2</code>	Bitwise Or (<code>E1</code> and <code>E2</code> are integer expressions)
<code>\E1</code>	Bitwise Negation (<code>E1</code> is an integer expression)
<code>E1 << E2</code>	Shift Left (<code>E1</code> and <code>E2</code> are integer expressions)
<code>E1 >> E2</code>	Shift Right (<code>E1</code> and <code>E2</code> are integer expressions)
<code>abs(E1)</code>	Absolute Value
<code>round(E1)</code>	Round
<code>floor(E1)</code>	Floor
<code>ceiling(E1)</code>	Ceiling
<code>sqrt(E1)</code>	Square Root
<code>sin(E1)</code>	Sin
<code>cos(E1)</code>	Cos
<code>tan(E1)</code>	Tan
<code>asin(E1)</code>	Arcsin
<code>acos(E1)</code>	Arccos
<code>atan(E1)</code>	Arctan
<code>log(E1)</code>	Log (base e)

The following are examples of valid subscriptions. In all these examples, the rock is zero but can be any integer (e.g. thread ID).

- `subscribe(info(fred, X), true, 0)` : in this example, the goal is `true` and will always succeed. Hence, this subscription will match against any notification that unifies with the head term – i.e. any compound term whose functor is `info`, has arity 2 and has first argument `fred`
- `subscribe(data(L), (member(height = H, L), H > 1000), 0)` : the head of this subscription matches against any notification with functor `data` and arity 1. The subscription will match a notification if the notification's argument is a list that contains a term of the form `height = X` and `X` is a number greater than 1000. Note that, as with Prolog, operators can be used to build terms (even when the operator semantics is not being used). In this case the operator `=` is just a convenient infix operator used to construct an arity 2 compound term.

- `subscribe(foo(X, X), (X < 10; X > 20), 0)` : the head matches against any notification with functor `foo`, has arity 2, and whose arguments are unifiable. The body succeeds if `X` is either less than 10 or greater than 20.
- `subscribe(str(S), (splitstring(S, _, S2), splitstring(S2, "hello", _)), 0)`: this subscription matches any notification with functor `str` and arity 1 and whose argument is a string containing "hello" as a substring.
- `subscribe(foo(X, Y), (atom(X) -> number(Y), Y > 0 ; atom(Y)), 0)`: this subscription matches any notification with functor `foo`, has arity 2, and if its first argument (`X`) is an atom then `Y` is a number greater than 0, else `Y` is an atom.

The following example illustrates what happens when type errors occur in an attempted match of a subscription and a notification.

Consider the subscription

```
subscribe(foo(X, Y), (X < 0 -> Y > 10 ; Y < 10), 0)
```

and the notification

```
foo(bar, 0)
```

In this case the head of the subscription matches the notification but `X` is not a number and so the test `X < 0` produces a type error which causes the attempted match to fail.

If the intention of the subscription was to test if `X` is a number less than 0 then it should be written as follows.

```
subscribe(foo(X, Y), (number(X), X < 0 -> Y > 10 ; Y < 10), 0)
```

In this case the notification above will match.

6 Removing Subscriptions

A client can remove a subscription by sending the newline terminated string `unsubscribe(ID)` to the server, where `ID` is the ID of the subscription to be removed. Recall the when a client makes a subscription, the server returns a client-unique ID for the subscription. It is this ID that is used to remove the subscription.

It is the clients responsibility to keep track of the mapping between subscriptions and IDs.

The server will acknowledge the client with a `1` if the unsubscription succeeds and a `0` otherwise.

7 Registrations

A client can register a name with the Pedro server. This name can be used by other registered clients to provide peer-to-peer communication. This is done by sending the following string (with a newline termination) to the Pedro server (where `name` is the name being registered).

```
register(name)
```

The registered name must be an atom not containing the characters `'`, `,`, `:` and `@`.

The server will acknowledge the client with a 1 if the registration succeeds and a 0 otherwise.

Semantically, a registration is the same as a restricted form of subscription. Specifically, a registration can be thought of as a subscription of the following form.

```
subscribe(p2pmsg(_:name@machine, _, _), true)
```

where `name` is the name being registered (the name of this process) and `machine` is the name of the machine on which the process is running. The restriction is that at most one process on any given machine can register a given name.

The idea is that a notification of the form

```
p2pmsg(ToAddr, MyAddr, Msg)
```

where `ToAddr` and `MyAddr` are addresses of the form `ID:Name@Machine` will match a registration subscription if the `ToAddr` of the notification unifies with the first argument of the subscription.

The `ID` part of an address is optional and is ignored by the Pedro server. It is up to the clients to determine how this argument is used. This might be, for example, used as the name of a thread or message queue. The rock used when sending such notifications to clients is always set to 0.

The `ID` part of an address can be elided from the notification if it is not required. It is up to the clients to determine what to do if no `ID` is supplied.

The `Name` and `Machine` parts of `ToAddr` are either atoms or variables. The characters `'`, `,`, `:` and `@` are not allowed to appear in `Name` or `Machine`. It is also possible for `ToAddr` to be a variable in which case the semantics is the same as having an address with both `Name` and `Machine` variables.

If `Name` is a variable then the notification will be sent to all registered processes on that machine. If `Machine` is a variable then the notification will be sent to all processes with that registered name. If both are variables then the notification will be sent to all registered processes on all machines that have registered processes.

Note that `MyAddr` can be any valid address, but in practice the client should make sure that this is its address. It will typically be used by the receiving client when responding to a message.

Example:

The notification `p2pmsg(foo@localhost, bar@localhost, info(fred, [1,2,3]))` is intended for the process with registered name `foo` on this machine. This client is registered with the name `bar`.

8 Removing Registrations

Registrations are removed by sending a newline terminated string of the form

```
deregister(name)
```

where `name` is the registered name of the process.

The server will acknowledge the client with a 1 if the deregistration succeeds and a 0 otherwise.

9 Connecting to Pedro

The communication between each client and the Pedro server is via a pair of sockets. One is used for two-way data transfer (e.g. notifications, subscriptions), the other is used for the server to acknowledge messages from the client. This will be a 0 for invalid data from the client and non-zero for valid data.

The creation of the two sockets is done as follows. Examples are given in the code for the Python and Java APIs.

1. Create a socket in the client (this will be used for acknowledgements)
2. Use `connect` to connect the socket to the Pedro server for acknowledgements. Pedro will be listening for the connection on a given port. The default port is 4550.
3. Read the client ID on this socket (sent by the server as a newline terminated string).
4. Create another socket in the client (this will be used for data)
5. Use `connect` to connect the socket to the Pedro server for data. Pedro will be listening for the connection on the next port after the previous port. The default port is 4551.
6. Send the client the client ID on the data socket (the same string the server sent above).
7. Read the status on the data socket. If the connection succeeds the status will be the string `"ok\n"`.

10 Denial of Service

Pedro's was designed as a message transport for communicating agents, typically in a local area network. It was not designed to deal with security issues relating to open networks. Even so, Pedro needs to take into account various DOS issues because client programs can have bugs and may block in ways that might cause problems for the Pedro server.

Consequently, the Pedro server is designed to overcome various problems caused by badly behaving clients.

Firstly, the length of any newline terminated string sent to the server has to be less than a bound that can be set using the `-S` switch at startup. The default is 1024. If the server does not find a newline (from a given client) before it reaches the bound then that string will be ignored.

Secondly, in Prolog, there are many ways to create infinite computation, either by constructing infinite (cyclic) terms and then processing them or by calling other non-terminating goals. Because Pedro uses occurs checking within unification then infinite terms cannot be constructed. Also, the valid goals used in subscriptions cannot produce infinite computation.

Thirdly, when a client disconnects, all its subscriptions are automatically removed.

Finally, there is the issue of blocking in the Pedro server. Because Pedro was designed to support communicating agents then we need to guarantee that messages from a given client will be processed in the order sent and that no messages will be dropped.

Consider a client that has subscribed but does not read notifications sent to it. Eventually, the Pedro server will not be able to send any more notifications to this client and so will have to block on any notifications that this client should receive. This may then cause notifying clients to block. To avoid this problem the Pedro server attaches a timeout to clients that causes the Pedro server to block. When the timeout expires (without the client consuming more notifications), the client is disconnected and all its subscriptions removed. The timeout is currently set to 5 seconds.

Consequently, when writing a client program, it would be best to consume notifications as soon as they arrive.

This timeout is also used when a client is trying to connect. If the handshake is not completed within this time, the client will be disconnected.

Note that, although dealing with open networks was not a design concern for Pedro, we believe the above defenses against DOS would allow Pedro to work in such environments.

11 Library

The Pedro C-library provides support for writing Pedro clients. It is built on top of the glib library.

`Client` is an opaque datatype used to store information about a Pedro client. The intention is to use `g_main_loop_run` with the callback declared in the creation of the client called each time data is sent to the client from the Pedro server. The program `consumer.c` in the examples directory is a simple example using this approach.

The following functions define the client interface.

```
Client* client_new(void (*cb) (char*, gpointer, gpointer), gpointer
user_data);
```

This function returns a pointer to a new client object connected to the Pedro server on this machine using the default port (4550). The callback `cb` (user defined) takes the data from the server, the rock specified in the matching subscription and a pointer to user data and is called each time data arrives from the server. The last argument, `user_data`, is a pointer to the user data used as the third argument to the callback.

```
Client* client_new_full(int port, char* hostname, void (*cb) (char*, gpointer,
gpointer), gpointer user_data);
```

This function is the same as the previous one except the port and machine name of the Pedro server are also supplied.

```
void client_destroy(Client* client);
```

Shut down the connection with the Pedro server and free the client object data.

```
int subscribe(Client* client, char* term, char* goal, gpointer rock);
```

Make a subscription for the given client and return the ID of the subscription (0 if the subscription is unsuccessful).

```
gboolean unsubscribe(Client* client, guint id);
```

Remove the subscription with the supplied ID. It returns true iff the operation was successful.

```
gboolean notify(Client* client, char* term);
```

Send a notification to the server. It returns true iff the operation was successful.

```
gboolean register_name(Client* client, char* name);
```

Register the supplied name with the server. It returns true iff the operation was successful.

```
gboolean deregister_name(Client* client);
```

Deregister the current name with the server. It returns true iff the operation was successful.

```
gboolean p2p(Client* client, char* toaddr, char* msg);
```

Send a peer-to-peer message to the supplied address. It returns true iff the operation was successful.

12 APIs

The directories `src/python_api` and `src/java_api` contain APIs for Python and Java.

12.1 Python API

The directory `src/python_api` contains a definition for a Pedro client class in the file `pedroclient.py`. This directory also contains a program `pedro_gui.py` that uses this API and provides an interface to subscriptions and notifications.

Below is a simple example using this class withing the python interpreter.

```
>>> from pedroclient import *
>>> me = PedroClient()
>>> me.subscribe('f(X)')
1
>>> me.notify('f(a)')
1
>>> me.get_notification()
('f(a)', 0)
>>> me.notify('f(g(12, "hi"))')
1
>>> print me.get_term()[0]
f(g(12, "hi"))
>>>
```

`PedroClient(machine='localhost', port=4550, async = True)`

Create a Pedro client object connected to the Pedro server on `hostname` using the port `port`. If `async` is true a thread is created to process Pedro messages. If not, a call to `notification_ready` will process any Pedro messages. If the main program has its own event loop (e.g. in `pygame`) then the program runs faster setting `async` to `False` and adding `notification_ready` to the loop.

`disconnect()`

A method that disconnects the client.

`connect()`

A method that (re)connects the client.

`notify(term)`

A method that sends `term` as a notification to the Pedro server.

`subscribe(term, goal = 'true', rock = 0)`

A method that subscribes with the supplied term, goal and rock. This method, if successful, returns the ID of the subscription.

`unsubscribe(ID)`

A method that unsubscribes to the subscription with the given ID.

`register(name)`

A method that registers the supplied name with the Pedro server.

`deregister()`

A method that deregisters its name.

`p2p(toaddr, term)`

A method that sends `term` to the address `toaddr` using the peer-to-peer support. The machine part of the address can be elided if it is to a process on the same machine. Do not use `localhost` as the name of the machine.

`get_notification()`

A method that returns the next notification received by the client as a pair consisting of the notification string together with the rock supplied in the corresponding subscription. This method blocks until a message is available if `async == True` and returns `None` if `async == false` .

`get_term()`

This has the same behaviour as `get_notification` except that the notification is parsed into a Prolog term.

`notification_ready()`

A method that tests if a notification is ready to get.

12.2 Java API

The directory `src/java_api` contains a definition for a Pedro client class and support for Prolog terms (including a Prolog parser).

The file `Client.java` contains a very simple program that connects to the Pedro server, makes a subscription based on the runtime arguments, and enters a loop that parses the received notification into Prolog terms then pretty prints them.

The Pedro client object creation and methods are the same as in the Python case from the previous section except there is no `get_term` method. This is replaced by the use of a parser that converts strings (from a call to `get_notification()`) to a Prolog term.

The files `PedroEvent.java` and `PedroListener.java` provide support for data events from the Pedro server. Clients can add listeners to data events using the `addPedroListener` method. The file `PedroGUI.java` is similar to `pedro_gui.py` and contains an example of using this method.

Appendix A Copying This Manual

A.1 GNU Free Documentation License

Version 1.2, November 2002

Copyright © 2000,2001,2002 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

- be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
 - C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
 - D. Preserve all the copyright notices of the Document.
 - E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
 - F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
 - G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
 - H. Include an unaltered copy of this License.
 - I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
 - J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
 - K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
 - L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
 - M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
 - N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
 - O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts. A copy of the license is included in the section entitled ‘‘GNU
Free Documentation License’’.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

-
- E1 8
- \
- \E1 9
- A**
- abs(E1) 9
- acos(E1) 9
- asin(E1) 9
- atan(E1) 9
- Atom syntax 3
- atom(T) 8
- C**
- ceiling(E1) 9
- Client* client_new(void (*cb) (char*,
gpointer, gpointer), gpointer user_data);
..... 16
- Client* client_new_full(int port, char*
hostname, void (*cb) (char*, gpointer,
gpointer), gpointer user_data); 16
- Compound term syntax 4
- connect() 17
- cos(E1) 9
- D**
- deregister() 17
- disconnect() 17
- E**
- e 8
- E1 * E2 8
- E1 ** E2 9
- E1 + E2 8
- E1 - E2 8
- E1 / E2 8
- E1 // E2 8
- E1 /\ E2 9
- E1 << E2 9
- E1 >> E2 9
- E1 \\/ E2 9
- E1 mod E2 9
- E1 rem E2 9
- F**
- fail 7
- FDL, GNU Free Documentation License 19
- floor(E1) 9
- G**
- G1 , G2 8
- G1 -> G2 ; G3 8
- G1 ; G2 8
- gboolean deregister_name(Client* client);
..... 16
- gboolean notify(Client* client, char* term);
..... 16
- gboolean p2p(Client* client, char* toaddr,
char* msg); 16
- gboolean register_name(Client* client, char*
name); 16
- gboolean unsubscribe(Client* client, guint
id); 16
- get_notification() 18
- get_term() 18
- I**
- int subscribe(Client* client, char* term,
char* goal, gpointer rock); 16
- L**
- List syntax 4
- list(T) 8
- log(E1) 9
- M**
- member(X, L) 8
- N**
- not(G) 8
- notification_ready() 18
- notify(term) 17
- Number syntax 3
- number(T) 8
- O**
- once(G) 8
- Operator syntax 4
- P**
- p2p(toaddr, term) 18
- PedroClient(machine='localhost', port=4550,
async = True) 17

pi	8	Syntax, variables	4
R			
register(name)	17		
round(E1)	9		
S			
sin(E1)	9		
split(L1, L2, L3)	8		
splitstring(S1, S2, S3)	8		
sqrt(E1)	9		
String syntax	3		
string(T)	8		
subscribe(term, goal = 'true', rock = 0) ...	17		
Syntax, atoms	3		
Syntax, compound terms	4		
Syntax, lists	4		
Syntax, numbers	3		
Syntax, operators	4		
Syntax, strings	3		
T			
T1 < T2	7		
T1 = T2	7		
T1 =< T2	7		
T1 > T2	7		
T1 >= T2	7		
T1 is T2	7		
tan(E1)	9		
true	7		
U			
unsubscribe(ID)	17		
V			
Variable syntax	4		
void client_destroy(Client* client);	16		