A Java/XML guide for prolog programmers on Windows 9X

Tsutomu Fujinami
(http://www.jaist.ac.jp/~fuji)

18th July, 2000

Abstract

Once upon a time, there was a language called **prolog**, a programming language in logic. We Japanese spent a lot to build a computer who can speak, think, and even dance, using the computer language. She is now gone without notice and a cup of coffee was instead brought in to us, a taste of, **Java**. Only drinking coffee was not good enough for us to entertain ourselves, but happily, we can now take a bite of Excel..., no, XML (ex, em, el). Allrihgt, we are all diving into this feast as a party of retired prolog programmers, left behind the scene for some time. Are you ready? Okay, let us go. The spotlight is waiting for you!

Contents

1 Preface **1**

2 AI on Web? **3**

I Building your web server 4

3 Preparing your working environment **5**

3.1 Cygnus 5

3.1.1 Making a directory for downloading 5

3.1.2 Downloading and installation 6

3.1.3 Settings 7

3.2 Meadow 7

3.2.1 Making your HOME directory 7

3.2.2 Downloading 9

3.2.3 Installation 9

3.3 Settings 10

3.3.1 Editing AUTOEXEC.BAT 10

3.3.2 Editing .bashrc 11

3.3.3 Editing .emacs 11

4 Installing Java, prolog, and the interface **16**

4.1 Java Development Kit 16

4.1.1 Obtaining the package 17
4.1.2 Installation 17
4.1.3 Defining Java_Home 17
4.2 Installing SWI-prolog 17
4.2.1 Downloading 17
4.2.2 Installation 18
4.2.3 Testing SWI-prolog 19
4.3 Installing JPL 20
4.3.1 Downloading 21
4.3.2 Installation 21
4.3.3 Testing JPL 31

5 Building your Web server running Servlet 33
5.1 Web servers running Servlet 33
5.1.1 Downloading Tomcat 34
5.1.2 Installing Tomcat 34
5.1.3 Testing Tomcat 35
5.2 Creating Web applications 35
5.2.1 Making your working directory 35
5.2.2 Making sub-directories 38
5.2.3 Editing the source code 39
5.2.4 Editing build.bat and build.xml 39
5.2.5 Editing web.xml 41
5.2.6 Build your application 41
5.2.7 Editing server.xml 42
5.2.8 Running your application 42
5.3 Servlet basics 43
5.3.1 Sending parameter values from Browser to Servlet 45
5.3.2 Receiving parameter values in Servlet 45

5.3.3 Sending HTML documents to Browser from Servlet . . . 47
5.3.4 Building and Running the sample 47

II Extending your web server with prolog and RDB 50

6 Integrating prolog and Java on Servlet 51
6.1 JPL: a Java interface to prolog 51
6.2 Sample prolog program 52
6.3 Handing prolog variables and atoms 54
6.4 Getting back atoms from prolog 55
6.5 Calling prolog from Servlet 55
6.6 Running the sample 57

7 Database Connection using Java 60
7.1 Which database management system? 60
7.2 Download and Installation 61
7.2.1 Installing MySQL Server 61
7.2.2 Register yourself 61
7.2.3 Installing JDBC driver 63
7.3 MySQL Basics 63
7.3.1 Creating Databases 63
7.3.2 Creating Tables 63
7.4 Database connection 65
7.4.1 Prerequisite 65
7.4.2 Sample Code 65
7.4.3 Test-run 67
7.4.4 Explanation about the code 67

8 Connecting prolog to Relational Database on Servlet 69
8.1 Why prolog and RDB? 69

8.2 The demo story 70
8.3 Implementing the demo 71
8.3.1 Handling facts dynamically in prolog 71
8.3.2 Retrieving data from RDB 74
8.3.3 Test-run 80
8.4 Combining prolog and MySQL on Servlet 81
8.4.1 The source code 81
8.4.2 Building the web application 87
8.4.3 Runing the proServ2 Servlet 87

9 Limitations 89
9.1 Problems with multi-threads 89
9.2 Managing RDB Connections 91
9.3 Managing prolog loading 91

III Integrating applications into XML 92

10 Installing XML tools 93

11 Communication between client and server 94

12 Applications 95

13 Conclusion 96

A Program Codes 97
A.1 proServ 97
A.2 proServ2 103

B On Linux 112
B.1 Building your web server on Linux 112
B.1.1 Java Development Kit 112

B.1.2 Installing SWI-prolog 112
B.1.3 Installing JPL 113
B.1.4 Web servers running Servlet 113
B.1.5 Creating Web applications 114
B.2 Extending your web server 114

Chapter 1

Preface

The computer language, **prolog**, was once a celebrity, convincing us that we would be soon surrounded by lots of intelligent things like talking newspaper, thinking kettles, refrigerators recommending us what to eat tonight, etc. A great amount of money was poured into the research projects to materialize intelligent agents and quite a few of people worked hard for the goal. The party was over, however, about ten years ago when people realized that we had to continue spending on the project for at least another thirty years till we will actually see smart agents helping us daily. Well, we could at worst educate younger generations of researchers with the money from the goverment (Oops, No, from tax payers), was it not? Forget about the dead boy. We had altogether nice time, did we not? Bye, bye, artificial intelligence...

Time passed. Internet came in. The rise of network community, where we are all interconnected to each other on-line (or in-air), must take us forward towards the society supported by many intelligent agents, artificial or natural. This is a chance for us, retired prolog programmers, is it not? We knew that the shortage of data was one of the major obstacles in bringing artificial intelligence to life, but look, we have now plenty of resources on the net, in hand, off-the-shelf, all over the world. Why do we not dive into the world with our knowledge and skills to turn our Internet community into the knowledge society?

Okey, you are moved, aren't you? a little bit? Good, here is the map to sail into the world.

- Preparing your working environment
- Installing tools
- Samples and test-runs

The first part invites you to building a Unix-like environment on you Wintel machine. I assume that you are more or less forced to work on Windows 9X for education or business. We cannot of course fight Microsoft, but there is no reason to accept everything provided by them. Let us build a working environment with which we are familiar. The second part instructs you to install various tools including jdk. There are really lots of tools available to us to build Internet applications. Our principle is termed as follows:

- We use open and free software. We academics can only accept the products whose sources we can read through. Another point is that we normally do not have enough money to buy each student a commercial software.
- We prefer the tools workable on Windows9X to ones workable on Linux. I am not particularly fond of the products by MS, but Wintel machines are so prevalent and the mojority of students have no experience with Unix. Asking them to install Linux on their machines and to switch between Windows and Unix is simply too much for them, I am afraid.

The last part presents you some sample codes to give you an insight into the possibilities of AI-applications on Internet.

Finally, I must confess that I myself have not built anything concrete along the line I am proposing in this guide. It took me sometime to realize how we can import our knowledge and skills of AI into Internet under the restrictions mentioned above, i.e., open and free software, using Windows9X, etc. I would say that it was impossible to produce a document similar to this two years ago when very few texts were available on XML. We had very few tools for Java/XML, too, at that time. The situation has greatly improved recently and we can now even choose among products. We rather encounter another problem, that is, it is rather hard to follow the progress, to catch up the speed in which people produce and upgrade tools. There is, thus, I believe, some reason to propose a way to select and combine tools available in public for particular purposes.

I had better stop here to move to more useful stuff. I appreciate your comments, corrections, criticisms, whatever, on this guide. Thank you.

Tsutomu Fujinami, 15th June, 2000 at JAIST

Chapter 2

AI on Web?

[ToDo] We claim that AI will contribute WWW community. Some survey, too?

Part I

Building your web server

Chapter 3

Preparing your working environment

Overview: If you have worked longer with Unix than with Windows, you probably want to build some similar environment to Unix on your machine. This chapter explains how to build a Unix-like environment on you Windows9X. You will install the following products:

1. Cygwin tools and utilities to build a Unix-like environmenton on Windows9X
2. Meadow 1.1 (an Emacs-20 ported to Windows9X/NT)

3.1 Cygnus

Cygnus[1] project provides us with GNU development tools and utilities on Windows9X, allowing us to work on Windows9X as if we were on Unix. We will install the latest version, Cygwin-1.1.X. (as is on the 15rh June, 2000)

3.1.1 Making a directory for downloading

To download the package, you need an empty directry to which the setup program imports all the files (mostly, a gzipped tar file) necessary to install Cygnus. We assume that you make a directry, C:\tmp\cygnus, on your PC for installation using DOS-Window as shown below. (You can of course open 'my computer' window to create the new directories under C drive.)

[1] http://sourceware.cygnus.com/cygwin/

```
Microsoft(R) Windows 98
        (C)Copyright Microsoft Corp 1981-1999

C:\WINDOWS>cd ..

C:\>mkdir tmp

C:\>cd tmp

C:\tmp>mkdir cygnus
```

Figure 3.1: Making c:\tmp\cygnus directory under C drive

3.1.2 Downloading and installation

1. Visit Cygnus[2] page for a link[3] to the list of mirror sites.
2. Choose the nearest site to you. You are brought into the site, e.g., University of Aizu[4], upon clicking.
3. Open the folder called **latest**, e.g., latest at University of Aizu[5], to download **setup.exe** to the directory, C:\tmp\cygnus, on your PC.
4. Click or execute **setup.exe** you have downloaded. You are then prompted to type in your Root directory. The default is, C:\cygnus. Type Return key if you are fine with the default setting. (You have to type in the directory name by yourself if the directory proposed by the installer is different from C:\cygnus.)
5. You are subsequently asked if you download all the files through net or you have already those files on your PC. Choose to download them through net by typing [i] if you have not downloaded the files beforehands. You are prompted to choose a site for downloading. Type [d] if the installation is second or more time and you keep those files on your PC.
6. Cygnus is installed automatically after downloading those files. Check if the followings appear under C:\cygnus to confirm your installation:

[2] http://sourceware.cygnus.com/cygwin/
[3] http://sourceware.cygnus.com/cygwin/mirrors.html
[4] ftp://ftp.u-aizu.ac.jp/pub/gnu/gnu-win32
[5] ftp://ftp.u-aizu.ac.jp/pub/gnu/gnu-win32/latest/

```
This is the Cygwin setup utility (v1.45.2.3),
built on Jun  8 2000 12:08:45.

Use this program to install the latest version of the Cygwin Utilities
from the Internet.

Alternatively, if you already have already downloaded the appropriate files
to the current directory (and subdirectories below it), this program can use
those as the basis for your installation.

If you are installing from the Internet, please run this program in an empty
temporary directory.

Press <enter> to accept the default value.
Root directory? [C:\CYGNUS]
```

Figure 3.2: Setting the root

- the directories, bin, etc, lib, tmp, usr, and var
- the files, inetutils-1.3.2.README and login.README
- the executable, uninst.bat

3.1.3 Settings

You can proceed to setting values to variables. You might however want to use Emacs to edit files. If you prefer using Emacs to other editors available on Windows9X, install Meadow now (See below).

3.2 Meadow

Meadow is the emacs-20 ported to Windows9X/NT by Hisashi Miyashita.

3.2.1 Making your HOME directory

You have to set your home to some directory to install Meadow. We assume that you use C:\cygnus\home as your home directory. Create the directory as below if you use DOS-Window.

```
Microsoft(R) Windows 98
        (C)Copyright Microsoft Corp 1981-1999

C:\WINDOWS>cd ..

C:\>cd cygnus

C:\cygnus> mkdir home
```

Figure 3.3: Making c:\cygnus\home directory using DOS-Window

You can alternatively make C:\cygnus\home using BASH. Invoke BASH to type 'mkdir /home' if you choose this way. Beaware that you do not need to specify your home as 'C:/cygnus/home' because the ROOT directory is already set to 'C:\cygnus' within BASH when you downloaded those files from net. [6] That is, '/' in BASH corresponds to 'C:\cygnus' on Windows9X.

```
C:\cygnus\bin>
BASH.EXE-2.04$ mkdir /home
```

Figure 3.4: Making c:\cygnus\home directory using BASH

A shortcut to cygwin.bat

You will need to invoke BASH frequently. It is thus better to put a short cut to C:\cygnus\bin\cygwin.bat on your desktop. Click the icon of cygwin.bat with the right button to choose the item to make a shortcut to the batch file. Drug the shortcut to your desktop.

[6] We have been recommended to create our HOME as C:\home, but I believe it is easier to move to HOME if you set it to C:\cygnus\home because you can specify your home just as /home in BASH. You only need, for example, to type 'cd /home' to move to your HOME on BASH. It is also good for us to enclose our UNIX-like environment to C:\cygnus, separate it from other directories.

3.2.2 Downloading

A gzipped tar file, Meadow-1.10-i386.tar.gz, for Meadow Win32-Intel binary package is available from Vector [7]. The current version is 1.10. Click the link at right bottom to open a window for download. Then, click the button, [Ftp Download] to import the gzipped file. Be patient as the file size is about 17MB.

3.2.3 Installation

1. We need to use Cygwin tools to install Meadow on our PC. Double-click the shortcut to cygwin.bat if you have already created it on your desktop. Execute otherwise 'C:\cygnus\bin\cygwin.bat' to invoke BASH. You can either double-click the file in window or execute it directly from the [start] buttom by specifying the file name in the dialogue window. Upon execution, you will see a window named **BASH** appearing on the screen.

2. Try **pwd** in the window. The current directory must be shown as **/usr/bin**. You are actually looking at C:\cygnus\usr\bin.

3. Move to /usr/local/lib by typing 'cd /usr/local/lib' and the return key. We are going to install Meadow in this directory.[8]

4. Place Meadow-1.10-i386.tar.gz under C:\cygnus\usr\local\lib. You can use **cp** or **mv** command on BASH to place the file there. You can alternatively drag the icon to the folder on Windows9X.

5. Type 'tar xvfz Meadow-1.10-i386.tar.gz' under '/usr/local/lib'. The files are then extracted and installed under the directory.

6. Check if the directory, Meadow, is successfully created under /usr/local/lib. You can remove the archive file, Meadow-1.10-i386.tar.gz, if you like to clean up.

7. Move to /usr/local/lib/Meadow/1.10 to execute 'install.exe'. A window is opend upon clicking, asking you to specify your HOME directory. Type in C:\cygnus\home to tell the installer that your HOME directory is set to the directory. Note that you MUST specify your HOME. (Beaware also that 'cygnus' and 'home' are case-sensitive.) The installation will otherwise fail.

8. The binaries are installed and a window is opened titled C:\Windows\StartMenu\Program\Meadow. Click the icon, Meadow.lnk, to see if Meadow works.

---

[7] http://www.vector.co.jp/soft/win95/writing/se068653.html

[8] It is of course up to you in which directory you will install the package. Other directories such as '/usr/local/share' as is employed on Solaris or '/usr/share' as is employed on Linux, are as well acceptable. I like to install tools for my own into a directory that Cygnus does not keep any file, say, the directories under '/usr/local'.

9. Finally, we link the executable, '/usr/local/lib/Meadow/1.10/Meadow95.exe', to our bin directory, i.e., '/usr/local/bin', as 'emacs.exe'. Once we set our path to the directory, we can invoke emacs from anywhere by typing 'emacs'.

```
BASH.EXE-2.04$ cd /usr/local/bin
BASH.EXE-2.04$ ln /usr/local/lib/Meadow/1.10/Meadow95.exe emacs.exe
```

Figure 3.5: Link Meadow95.exe to /usr/local/bin/emacs.exe

3.3 Settings

To use Meadow, you have to set your HOME among others. We are going to edit AUTOEXEC.BAT to edit environment variables. Some parameters may be easily set within $HOME/.bashrc. We conclude this chapter with a note on $HOME/.emacs. I found most settings presented in this section at various sites. (Thank you guys, and I am sorry that I cannot list up all your names here. Too many to name, even to recall.).

3.3.1 Editing AUTOEXEC.BAT

You need to set your HOME to c:\cygnus\home. Time zone should also be set to yours. You can optionally set your your path to the bin directories of cygnus if you would like to invoke the shell within DOS-Windows or Emacs, too. (You do not need to do so if you only use the shell within Cygnus window because the path is specified in /bin/cygwin.bat.)

list 1: Variables to be added to AUTOEXEC.BAT

```
REM ---
REM --- Time Zone is set to that of Japan
REM ---
set TZ=JST-9

REM ---
REM --- your home directory
REM ---
set HOME=c:\cygnus\home
```

```
REM ---
REM --- Below is optional
REM ---
set PATH=.;c:\cygnus\bin;c:\cygnus\usr\local\bin;%PATH%
```

Memo: We had to set the variables, TMP, SHELL, and MAKE_MODE, by ourselves in beta version, but it seems that we no longer need to set those variables because the values are set by Cygnus.

3.3.2 Editing .bashrc

You can tune your BASH by editing .bashrc in your HOME directory. Following is my recommendtation. After editing AUTOEXEC.BAT and $HOME/.bash, you restart your PC so that the change takes effect.

list 2: A mininal definitions for .bashrc

```
# --- set prompt
PS1='\w> '

# --- set EDITOR
EDITOR="c:/cygnus/usr/local/lib/Meadow/1.10/bin/Meadow95.exe"
export EDITOR

# --- as you like
alias ls='ls -F'
alias cp='cp -i'
alias rm='rm -i'
alias mv='mv -i'
alias del='/bin/rm -f \!*'
alias h='history 20'
alias more=less
alias clear='COMMAND.COM /c cls'
```

3.3.3 Editing .emacs

We can control the behaviour of Meadow by editing $HOME/.emacs. We go through some useful definitions.

Set the load-path to site-lisp and TMP in .emacs

Some emacs-lisp programs are under /usr/local/lib/Meadow/site-lisp.

list 3: Set load-path

```
;
; Set load-path to site-lisp and TMP
;
(setq load-path
      (cons (expand-file-name
            (concat exec-directory "/cygnus/usr/local/lib/Meadow/site-lisp"))
            load-path))

(setenv "TMP" "/tmp")
```

Set your shell to bash

To envoke BASH within Meadow by typing [M-x shell], you must include the following lines into your .emacs file.

list 4: Set your shell to bash in .emacs

```
;;;
(setq shell-command-option "-c")
;
;--- from FAQ u-tokyo
;
(setq explicit-shell-file-name "/cygnus/bin/bash.exe")
(setq shell-file-name "bash.exe")
```

How to close BASH window when you exit?

Would you like the BASH window to close when you exit? Click the shortcut icon on your desk top to cygwin.bat with Right button and select [property] from the popup menu. A window listing up the properties of shortcut is displayed. Select the tub, [program]. There is an option in the menu to close the window when exit. Check it and select [ok] to finish the setting. Open a BASH window and exit by typing c-d. The window is now closed when you exit.

Other useful definitions

list 5: To go to Home when you start Meadow

```
;
; To start mule from the home directory
;
(cd "~")
```

list 6: To change the color of cursor in IME-mode (from FAQ u-tokyo)

```
(add-hook 'mw32-ime-on-hook
                  (function (lambda () (set-cursor-color "red"))))
(add-hook 'mw32-ime-off-hook
                  (function (lambda () (set-cursor-color "black"))))
```

list 7: Key bindings

```
; my favourite
(global-set-key "\C-h" 'backward-delete-char)
(global-set-key "\M-?" 'help-for-help)
```

list 8: My favourite settings

```
; variables
(setq inhibit-startup-message t)

;
; some settings to my taste
;
(setq-default fill-column 68)
(global-set-key "\C-cg" 'goto-line)
(setq visible-bell t)
(setq display-time-day-and-date t)
(setq default-tab-width 4)
(setq search-highlight t)
(setq query-replace-highlight t)
```

list 9: To hilite characters

```
;;;
;;; hilit: change colours to improve visibility of characters
;;;
(cond (window-system
       (setq hilit-mode-enable-list  '(not text-mode)
             hilit-background-mode   'light
             hilit-inhibit-hooks     nil
             hilit-inhibit-rebinding nil)
       (require 'hilit19)))
;
; to hilite regions (from FAQ u-tokyo)
;
(transient-mark-mode 1)
```

list 10: To use ange-ftp

```
(setq ange-ftp-ftp-program-name "/cygnus/usr/local/lib/Meadow/1.10/bin/ftp.exe")
; (setq ange-ftp-generate-anonymous-password "yourID@Domain")
```

list 11: To fix the problem when you access to Solaris using ange-ftp

```
(setq dired-move-to-filename-regexp
      (let* ((l "[A-Za-z\xa0-\xff]")
             (k "[^\x00-\xff]")
             (s " ")
             (yyyy "[0-9][0-9][0-9][0-9]")
             (mm "[ 0-1][0-9]")
             (dd "[ 0-3][0-9]")
             (HH:MM "[ 0-2][0-9]:[0-5][0-9]")
             (western (concat l l l s dd s "\\(" HH:MM "\\|" s yyyy "\\)"))
             (japanese (concat mm k " " dd k s
                               "\\(" s HH:MM "\\|" yyyy k "\\)")))
        (concat s "\\(" western "\\|" japanese "\\)" s)))
(eval-after-load
 "ange-ftp"
 '(progn
    (setq ange-ftp-date-regexp
         (let* ((l "[A-Za-z\xa0-\xff]")
```

```
    (k "[^\x00-\xff]")
    (s " ")
    (mm "[ 0-1][0-9]")
    (dd "[ 0-3][0-9]")
    (western (concat l l l s dd))
    (japanese (concat mm k s dd k)))
(concat s "\\(" western "\\|" japanese "\\)" s)))))
```

Chapter 4

Installing Java, prolog, and the interface

SWI-prolog[1] is an efficient prolog compiler/interpreter developed by Jan Wielemaker and his collegues at University of Amsterdam. We employ SWI-prolog as our prolog compiler/interpreter because it is runnable on various platforms including Solaris, Linux, and Windows9X. We will install SWI-prolog and the interface to Java.

You may notice that there is a binary package for installing SWI-prolog on Windows. We need, however, to compile it from source because we need the runtime environment, 'libpl.a', which will be created under '/usr/local/lib/pl-3.3.8/runtime', to add SWI-prolog the interface to Java, **JPL**.

We install Java Development Kit before installing SWI-prolog. After installing both jdk and SWI-prolog, we install **JPL**, a Java Interface to Prolog.

4.1 Java Development Kit

There is no drama in installing JDK. We install JDK1.2.2 for Windows provided by Sun Microsystems. Some people claim that the runtime environment provided by IBM runs faster than the one by Sun, but we take the safer side.[2]

[1] http://www.swi.psy.uva.nl/projects/SWI-Prolog/

[2] I have recently installed JDK1.3 provided by Sun. It works fine and looks faster than JDK1.2.2.(27.7.2000)

4.1.1 Obtaining the package

Wherever you can find it.

4.1.2 Installation

We install jdk1.2.2 under 'C:\cygnus\usr\local\jdk1.2.2' and the runtime environment under 'C:\cygnus\usr\local\JavaSoft'.

4.1.3 Defining Java_Home

We do not test jdk. (It is very unlikely that you will encounter any problem with installation.) You should instead add the following line to 'c:\AUTOEXEC.BAT'.

list 12: Adding JAVA_HOME to AUTOEXEC.BAT

```
set JAVA_HOME=c:\cygnus\usr\local\jdk1.2.2
```

It is also convenient to set a path to the binaries of jdk under bin so that you can for example compile and execute java programs anywhere. Set the path as follows:

list 13: Setting a path to the binaries

```
set PATH=c:\cygnus\usr\local\jdk1.2.2\bin;%PATH%
```

Restart your PC when you have finished.

4.2 Installing SWI-prolog

4.2.1 Downloading

You can down load the source from SWI-prolog download page[3]. The latest is version 3.3.8 (1.12MB). Place the file, pl-3.3.8.tar.gz, under /tmp (i.e., c:\cygnus\tmp).

[3] http://www.swi.psy.uva.nl/projects/SWI-Prolog/download.html

4.2.2 Installation

1. Unpack pl-3.3.8.tar.gz by typing 'tar xvfz pl-3.3.8.tar.gz'
2. Change directory to pl-3.3.8/src to execute './configure'. You will probably encounter an error message as below, 'Invalid configuration...'. We will avoid the problem later by editing a source file, say, 'pl-os.c'.

```
/tmp/pl-3.3.8/src> ./configure
creating cache ./config.cache
checking for gcc... gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking for ranlib... ranlib
checking host system type...
        Invalid configuration `i386-unknown-cygwin32':
        system `cygwin32' not recognized
checking for make... make
(...to be continued)
```

Figure 4.1: An error while configuration

3. Check Makefile to edit if you like.
4. Run 'make'. You will probably encounter the compilation problem with pl-os.c as shown below:
5. If this is the case, open the file, src/pl-os.c. You skip to Line 642 and comment out the line and Line 656 so that the part between the two lines are compiled.

list 14: Lines to be amended in pl-os.c L624–656

```
// #if unix (to be commented out)
char *
PrologPath(const char *p, char *buf)
{ strcpy(buf, p);

  return buf;
```

```
gcc -c -I. -I. -I./rc -Wall -O2   pl-os.c -o pl-os.o
pl-os.c: In function `Which':
pl-os.c:2608: warning: implicit declaration of function `okToExec'
pl-os.c:2608: warning: assignment makes pointer from integer without a cast
pl-os.c:2630: `PATHSEP' undeclared (first use in this function)
pl-os.c:2630: (Each undeclared identifier is reported only once
pl-os.c:2630: for each function it appears in.)
pl-os.c:2631: warning: assignment makes pointer from integer without a cast
pl-os.c:2646: warning: assignment makes pointer from integer without a cast
make: *** [pl-os.o] Error 1
/tmp/pl-3.3.8/src>
```

Figure 4.2: A problem while compilation

```
}

char *
OsPath(const char *p, char *buf)
{ strcpy(buf, p);

  return buf;
}
// #endif /*unix*/ (to be commented out)
```

You also need to comment out the lines Line 2567 and 2570 so that the lines between them are forced to compile in:

list 15: The lines between 2567 and 2570 to edit

```
// #if defined(OS2) || defined(__DOS__) || defined(__WINDOWS__) || defined(__WIN32__)
#define EXEC_EXTENSIONS { ".exe", ".com", ".bat", ".cmd", NULL }
#define PATHSEP ';'
// #endif
```

6. Type 'make clean' and then 'make' again.
7. If nothing wrong happend, run 'make install'

4.2.3 Testing SWI-prolog

Once you have successfully compiled SWI-prolog, check if it works.

```
/tmp/pl-3.3.8/src> ls -l /usr/local/bin
total 1767
-rwxr-xr-x   1 fuji     unknown   3612344 Jun 15 21:48 emacs.exe*
lrw-r--r--   1 fuji     unknown        33 Jun 21 14:09 pl -> ../lib/pl-3.3.8/bin/pl*
lrw-r--r--   1 fuji     unknown        35 Jun 21 14:09 plld -> ../lib/pl-3.3.8/bin/plld*
lrw-r--r--   1 fuji     unknown        35 Jun 21 14:09 plrc -> ../lib/pl-3.3.8/bin/plrc*
/tmp/pl-3.3.8/src>
```

Figure 4.3: Checking the executables

1. Check if the executables are installed under /usr/local/bin as below:

2. Invoke SWI-prolog in bash to test.

```
c:/cygnus/home> pl
Welcome to SWI-Prolog (Version 3.3.8)
Copyright (c) 1990-2000 University of Amsterdam.
Copy policy: GPL-2 (see www.gnu.org)

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- write('Hello World!').
Hello World!

Yes
2 ?-
```

Figure 4.4: Test SWI-prolog in bash

3. You have now successfully installed SWI-prolog on your PC.
4. Try 'man pl' on console, say, in Bash window. You will see the beginning of the manual.

4.3 Installing JPL

JPL is a Java Interface to Prolog, with which you can call SWI-prolog from within Java. **JPL** is developed by Fred Dushin. The package is available from

Separately distributed packages for SWI-Prolog[4]. The package is well documented, thus, I only explain how to install it on Windows.

4.3.1 Downloading

The package is available from Separately distributed packages for SWI-Prolog[5]. Download the package from SWI-prolog site by clicking the sentence, "a local copy of the source (version 1.0.1, June 16, 1999) ", not 'JPL'. You will obtain a gzipped tar file called 'jpl-1.0.1.tar.gz'(as is on the 21st July, 2000).

4.3.2 Installation

Prerequisite

You have to install the following packages:

- Java Development Kit (jdk1.1.8 or later)
- Cygnus tools (version 1.0.1 or later)

Points to note

We had better keep the followings in mind:

- SWI-prolog is called from Java through Java Native Interface(JNI).
- SWI-prolog has to be loaded dynamically upon request from Java.
- You have to rely on the mechanism of Dynamically Loadable Library(DLL) on windows to allow Java Virtual Machine to load prolog programs on demand.

The difficult point lies in creating the DLL module of JPL because the package only provides us with a means to compile it for Unix platforms, where modules are loaded dynamically through the mechanism of shared library object. For our relief, Cygnus tools include the DLL tool, with which we can create DLL modules.

Tips and examples are found in GNU Win32 related projects[6] written by Mumit Khan. Download first the file entitled as 'README.jni.txt' to read a brief

[4] http://www.swi.psy.uva.nl/projects/SWI-Prolog/packages/

[5] http://www.swi.psy.uva.nl/projects/SWI-Prolog/packages/

[6] http://www.xraylith.wisc.edu/~khan/software/gnu-win32/

introduction to compiling DLLs using Cygnus tools, which you can find as the sixth item of the list. Download 'java-jni-examples.tar.gz' if you want to take a look of the samples. (The README file is included in the package, too.) We are going to create our own Makefile by slightly modifying 'java-jni/c/Makefile.cyg' in the sample for C.

Compiling Java sources

We are going to compile Java sources by hand, not using Makefiles provided in the package of jpl-1.0.1 because the package is primary designed for installing it on Unix platforms, not for Windows.

You can unpack the archive, jpl-1.0.1.tar, wherever. We unpack it, for example, under /cygdrive/d/tmp in what follows. We are going to create the files, jpl.jar (a Java archive) and jpl.dll (the DLL), and copy them later to appropriate directories. Unpack the archive as follows:

```
/cygdrive/d/tmp> tar xvfz jpl-1.0.1.tar.gz
```

Figure 4.5: Unpacking jpl-1.0.1.tar

You must have created the following directories and files under jpl-1.0.1 as shown in figure4.6:

```
/cygdrive/d/tmp> ls jpl-1.0.1
ChangeLog    INSTALL            Makefile
README       TODO               demo/
doc/         export.script*     jsrc/
rules.mk     src/               stamp.script*
test/        version
```

Figure 4.6: The directories and files of jpl

The steps are depicted below:

1. You first compile the sources under jsrc/jpl/fli, and
2. go up to jsrc/jpl to make the directories, jsrc/jpl/jpl and jsrc/jpl/jpl/fli.

3. Move the class files under jsrc/jpl/fli to jsrc/jpl/jpl/fli before
4. compiling the sources under jsrc/jpl.
5. Finally, move the class files under jsrc/jpl to jsrc/jpl/jpl.

```
/cygdrive/d/tmp/jpl-1.0.1/jsrc/jpl/fli> javac *.java
/cygdrive/d/tmp/jpl-1.0.1/jsrc/jpl/fli> cd ..
/cygdrive/d/tmp/jpl-1.0.1/jsrc/jpl> mkdir jpl
/cygdrive/d/tmp/jpl-1.0.1/jsrc/jpl> mkdir jpl/fli
/cygdrive/d/tmp/jpl-1.0.1/jsrc/jpl> mv fli/*.class jpl/fli
/cygdrive/d/tmp/jpl-1.0.1/jsrc/jpl> javac *.java
/cygdrive/d/tmp/jpl-1.0.1/jsrc/jpl> mv *.class jpl
```

Figure 4.7: Compiling sources

You must have created the class files under jsrc/jpl/jpl and jsrc/jpl/jpl/fli. The last thing to do is to make the archive of files under jsrc/jpl as shown in *unresolved reference*??:

```
/cygdrive/d/tmp/jpl-1.0.1/jsrc/jpl> jar cvf jpl.jar jpl
added manifest
adding: jpl/(in = 0) (out= 0)(stored 0%)
adding: jpl/fli/(in = 0) (out= 0)(stored 0%)
adding: jpl/fli/DoubleHolder.class(in = 226) (out= 182)(deflated 19%)
(... suppressed by Fujinami for readability)
adding: jpl/fli/IntHolder.class(in = 220) (out= 181)(deflated 17%)
adding: jpl/fli/LongHolder.class(in = 245) (out= 203)(deflated 17%)
adding: jpl/Version.class(in = 432) (out= 302)(deflated 30%)
/cygdrive/d/tmp/jpl-1.0.1/jsrc/jpl> ls -lt jpl.jar
-rw-r--r--   1 fuji     unknown     27613 Jun 22 13:45 jpl.jar
```

Figure 4.8: Archivining the files

Check the size of 'jpl.tar' to see if you have successfully made the archive.

Installing the Java archive

Once you have successfully created the archive, 'jpl.jar', you can place it under an appropriate directory and extend your classpath in AUTOEXEC.BAT. Below we make a directory, /usr/local/lib/jpl, and place the jar file there (figure4.9).

```
/cygdrive/d/tmp/jpl-1.0.1/jsrc/jpl> mkdir /usr/local/lib/jpl
/cygdrive/d/tmp/jpl-1.0.1/jsrc/jpl> cp jpl.jar /usr/local/lib/jpl
```

Figure 4.9: Placing jpl.jar

You should then include the following line in AUTOEXEC.BAT (list16).

list 16: Setting Classpath to jpl.jar

```
set CLASSPATH=c:\cygnus\usr\local\lib\jpl\jpl.jar
```

Restart your PC to take the change into effect.

Compiling C sources

Before compiling the source, 'jpl_fli_Prolog.c', under (d:/tmp/)jpl-1.0.1/src, we have to do the followings:

- Copy 'libpl.a' under (/usr/local/lib/)pl-3.3.8/runtime to /usr/lib so that the archive can be linked to the compiled module, 'jpl.dll'.

```
~> cd /usr/local/lib/pl-3.3.8/runtime
/usr/local/lib/pl-3.3.8/runtime> ls
libpl.a
/usr/local/lib/pl-3.3.8/runtime> cp libpl.a /usr/lib
```

Figure 4.10: Copy libpl.a to /usr/lib

- Copy 'SWI-Prolog.h' and 'SWI-Stream.h' under (/usr/local/lib/)pl-3.3.8/include to (d:/tmp/)jpl-1.0.1/src so that they can be referred to while compiling the source, i.e., 'jpl_fli_Prolog.c'. You have to edit 'jpl_fli_Prolog.c' so that 'SWI-Prolog.h' can be read in the directory, i.e., #include <SWI-Prolog.h> has to be changed to #include "SWI-Prolog.h" in the header.

```
/usr/local/lib/pl-3.3.8/runtime> cd ../include
/usr/local/lib/pl-3.3.8/include> ls *.h
SWI-Prolog.h
SWI-Stream.h
/usr/local/lib/pl-3.3.8/include> cp *.h d:/tmp/jpl-1.0.1/src
```

Figure 4.11: Copying the header files to jpl-1.0.1/src

- Edit 'jni_mod.h' under JAVA_HOME/include/win32 as below so that GCC can compile the source with the type **long long** instead of **__int64**:

list 17: Editing jni_mod.h under JAVA_HOME/include/win32

```
/*
 * @(#)jni_md.h       1.9 98/09/21
 *
 * Copyright 1996-1998 by Sun Microsystems, Inc.,
 * 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information
 * of Sun Microsystems, Inc. ("Confidential Information").  You
 * shall not disclose such Confidential Information and shall use
 * it only in accordance with the terms of the license agreement
 * you entered into with Sun.
 */

#ifndef _JAVASOFT_JNI_MD_H_
#define _JAVASOFT_JNI_MD_H_

#define JNIEXPORT __declspec(dllexport)
#define JNIIMPORT __declspec(dllimport)
#define JNICALL __stdcall

typedef long jint;
#ifdef __GNUC__
```

```
typedef long long jlong;
#else
typedef __int64 jlong;
#endif
typedef signed char jbyte;

#endif /* !_JAVASOFT_JNI_MD_H_ */
```

- To create a DLL file, you copy Makefile.cyg below (*unresolved reference*??) to the src directory, (d:/tmp/)jpl-1.0.1/src, and edit the variable, JDK_ROOT, to fit it to your environment.

list 18: Editing JDK_ROOT

```
# JDK_ROOT = c:/jdk1.1.7A
JDK_ROOT = c:/cygnus/usr/local/jdk1.2.2
```

The crucial part is to set 'LDLIBS' as 'LDLIBS = -lpl -ltermcap -lm' so that 'libpl.a' under /usr/lib can be linked while building the load module.

list 19: Editing LDLIBS

```
# any extra libraries that your DLL may depend on.
# DLL_LDLIBS = c:\\usr\\local\\lib\\pl-3.6.6\\runtime\\libpl.a
# DLL_LDLIBDIR = c:/usr/local/lib/pl-3.6.6/runtime
# LDLIBS = -L/usr/local/lib/pl-3.3.6/runtime -lpl -ltermcap -lm
LDLIBS = -lpl -ltermcap -lm
```

You should also replace the following lines:

list 20: The lines to be commented out in Makefile.cyg

```
#$(DLL_NAME): $(DLL_OBJS)
#       $(DLLWRAP) $(DLLWRAP_FLAGS) -o $(DLL_NAME) \
#           $(DLL_OBJS) $(DLL_LDFLAGS) $(DLL_LDLIBS)
```

with the lines below:

list 21: The lines to be inserted

```
$(DLL_NAME): $(DLL_OBJS)
	$(DLLWRAP) $(DLLWRAP_FLAGS) -o $(DLL_NAME) \
	    $(DLL_OBJS) $(DLL_LDFLAGS) $(LDLIBS)
```

The Makefile is otherwise identical with the one provided by Mumit Khan.

list 22: Makefile.cyg

```
#
# Sample makefile to create Java JNI with Cygwin b20.1 tools. This *will not*
# work with Cygwin versions earlier than b20.1.
#
# The only difference from creating a regular DLL is to supply a different
# entry point, __cygwin_noncygwin_dll_entry@12, since Java is an MSVC app.
#
# See Makefile.nocyg if you want to use -mno-cygwin and build a Mingw JNI.
#

CC = gcc
CXX = c++


DEBUG = -g -Wall -O2
CXXFLAGS = $(DEBUG)
CFLAGS = $(DEBUG)
CPPFLAGS = -I. -I$(JDK_ROOT)/include -I$(JDK_ROOT)/include/win32

# JDK_ROOT = c:/jdk1.1.7A
JDK_ROOT = c:/cygnus/usr/local/jdk1.2.2

AS = as
DLLTOOL = dlltool
DLLWRAP = dllwrap

#
# Various targets to build.
#
DLL_NAME = jpl.dll
DLL_EXP_DEF = jpl.def

all: $(DLL_NAME)

#
# DLL related variables. These are used when building the DLL. See later.
#

# Some tools require special CPP macros when building a DLL (eg., _DLL etc).
# Here we don't need anything.
```

```
DLL_CFLAGS = -DBUILDING_DLL=1 -D_DLL=1

# The default entry point defined by dllwrap is __cygwin_dll_entry@12
# defined in libcygwin.a, but that's only appropriate for Cygwin apps,
# but since Java is a MSVC app, we need to provide a different entry
# point. Note the leading underscore and the trailing @12.
# The -s flag strips the DLL to shrink the size.

DLL_LDFLAGS = -Wl,-e,__cygwin_noncygwin_dll_entry@12 -s

# any extra libraries that your DLL may depend on.
# DLL_LDLIBS = c:\\usr\\local\\lib\\pl-3.6.6\\runtime\\libpl.a
# DLL_LDLIBDIR = c:/usr/local/lib/pl-3.6.6/runtime
# LDLIBS = -L/usr/local/lib/pl-3.3.6/runtime -lpl -ltermcap -lm
LDLIBS = -lpl -ltermcap -lm


DLL_SRCS  = jpl_fli_Prolog.c
DLL_OBJS  = $(DLL_SRCS:.cc=.o)
DLL_OBJS := $(DLL_OBJS:.c=.o)

###
#
# Making DLL
#
###
DLLWRAP_FLAGS = --output-def $(DLL_EXP_DEF) \
        --add-stdcall-alias \
        --driver-name $(CC) \
        $(IMAGE_BASE)

#$(DLL_NAME): $(DLL_OBJS)
#        $(DLLWRAP) $(DLLWRAP_FLAGS) -o $(DLL_NAME) \
#            $(DLL_OBJS) $(DLL_LDFLAGS) $(DLL_LDLIBS)

$(DLL_NAME): $(DLL_OBJS)
        $(DLLWRAP) $(DLLWRAP_FLAGS) -o $(DLL_NAME) \
            $(DLL_OBJS) $(DLL_LDFLAGS) $(LDLIBS)

#
# dependencies.
#

#
# default rules for building DLL objects. Note that client programs (ie.,
# the ones that *use* the DLL) have to be compiled without the DLL_CFLAGS
# flags.
#
.cc.o:
```

```
        $(CXX) -c $(DLL_CFLAGS) $(CPPFLAGS) $(CXXFLAGS) -o $@ $<
.c.o:
        $(CC) -c $(DLL_CFLAGS) $(CPPFLAGS) $(CFLAGS) -o $@ $<

# Note that we omit the $(DLL_CFLAGS) for client programs.
usedll.o: %o: %c
        $(CC) -c $(CPPFLAGS) $(CFLAGS) -o $@ $<

clean:
        -rm -f $(OBJS) $(DLL_OBJS) $(DLL_NAME) $(DLL_EXP_LIB) $(DLL_EXP_DEF) $(TESTPROGS)
```

- Finally, run 'make -f Makefile.cyg' as follows:[7]

```
/cygdrive/d/tmp/jpl-1.0.1/src> make -f Makefile.cyg
dllwrap --output-def jpl.def --add-stdcall-alias --driver-name gcc   -o jpl.dll \
    jpl_fli_Prolog.o -Wl,-e,__cygwin_noncygwin_dll_entry@12 -s -lpl -ltermcap -lm
Warning: no export definition file provided
dllwrap will create one, but may not be what you want
/cygdrive/d/tmp/jpl-1.0.1/src>
```

Figure 4.12: Running make

Installing jpl.dll

You can copy 'jpl.dll' to a directory that can be found by JVM, e.g., /usr/local/lib/jpl as follows:

You have to of course include the directory into your PATH by editing your AUTOEXEC.BAT as follows:

list 23: The line to be included for extending your PATH

[7] You may encounter a problem if you run 'make -f Makefile.cyg' within Meadow using BASH due to the different ways to referr to directories between Cygnus tools and Meadow. That is, /home for Cygnus corresponds to C:\cygnus\home for Meadow. If you have a trouble at this last stage, use Bash window instead.

```
/cygdrive/d/tmp/jpl-1.0.1/src> ls -l
total 250
-rw-r--r--   1 fuji     unknown      1658 May  5  1999 Makefile
-rw-r--r--   1 fuji     unknown      2556 Jun 22 17:46 Makefile.cyg
-rw-r--r--   1 fuji     unknown      2542 Jun 13 10:30 Makefile.cyg~
-rw-r--r--   1 fuji     unknown     25205 Jun 22 17:43 SWI-Prolog.h
-rw-r--r--   1 fuji     unknown      8719 Jun 22 17:43 SWI-Stream.h
-rw-r--r--   1 fuji     unknown      8788 Jun 22 17:51 jpl.def
-rw-r--r--   1 fuji     unknown    345600 Jun 22 17:51 jpl.dll
-rw-r--r--   1 fuji     unknown     39909 May  5  1999 jpl_fli_Prolog.c
-rw-r--r--   1 fuji     unknown     66834 Jun 22 17:47 jpl_fli_Prolog.o
```

Figure 4.13: The result after making

```
/cygdrive/d/tmp/jpl-1.0.1/src> cp jpl.dll /usr/local/lib/jpl
/cygdrive/d/tmp/jpl-1.0.1/src> ls /usr/local/lib/jpl
jpl.dll
jpl.jar
/cygdrive/d/tmp/jpl-1.0.1/src>
```

Figure 4.14: Copy jpl.dll to an appropriate directory

```
set PATH=c:\cygnus\usr\local\lib\jpl;%PATH%
```

Restart your PC to take the change into effect.

4.3.3 Testing JPL

JPL package includes test programs so that we can check if we have successfully installed JPL. For testing, change directory to jpl-1.0.1/test to compile and run Test.java program as follows:

```
~> cd d:/tmp/jpl-1.0.1/test
/cygdrive/d/tmp/jpl-1.0.1/test> javac Test.java
/cygdrive/d/tmp/jpl-1.0.1/test> java Test
test 0...Warning: (/cygdrive/d/tmp/jpl-1.0.1/test/test.pl:4):
        Singleton variables: [X]
Warning: (/cygdrive/d/tmp/jpl-1.0.1/test/test.pl:10):
        Singleton variables: [X, Y]
Warning: (/cygdrive/d/tmp/jpl-1.0.1/test/test.pl:13):
        Singleton variables: [Y]
Warning: (/cygdrive/d/tmp/jpl-1.0.1/test/test.pl:16):
        Singleton variables: [Y]
% test.pl compiled 0.02 sec, 2,416 bytes
passed.
test 1...passed.
test 2...passed.
test 3...passed.
test 4...passed.
test 5...passed.
test 6...passed.
test 7...passed.
test 8...passed.
test 9...passed.
test 10...passed.
test 11...passed.
test 101...0123456789012345678901234567890
1234567890123456789012345678901234567890l2
3456789012345678901234678901234567890l2345
6789012345678901234567890l234567890l234567
8901234567890123456789012345678901234567 89
5passed.
```

Figure 4.15: Running Test.java

No output visible?

Beaware that the output by Test.class may not be visible on Bash window. You should run the shell in Meadow if it is the case.

Chapter 5

Building your Web server running Servlet

5.1 Web servers running Servlet

We need first to build a Web server on which we can run Servlet programs. We have three options in building our server:

1. To buy a commercial server that can run Servlet programs
2. To use JServ as an add-on to Apache Web Server
3. To use Tomcat, a reference implementation of Servlet API

Among them, the first option is not available to us because we do not have money. The second option looks attractive because Apache is known to be very stable, but the version of Servlet API is still 2.1. With Tomcat we can use Servlet API version 2.2, the latest, but it may not be so stable because it is not intended for realistic applications. Having considered all these three options, we opt for the last, i.e., using Tomcat, because we are primary interested in what and how we can do using Servlet. We will take a look of the second option, using JServ, too, in Appendix, because the combination of JServ and Apache is attractive to test applications in more realistic environment.

5.1.1 Downloading Tomcat

Tomcat is developed in the Jakarta Project[1] as part of the activities within The Apache Software Foundation[2]. You can visit their download page[3] to download binaries. Choose 'Release Builds' to download. The latest version is Tomcat3.1 (on 23rd June, 2000). We are going to install Tomcat under /usr/local. Download the file, 'jakarta-tomcat.tar.gz', which builds up the web server.

5.1.2 Installing Tomcat

Copy or move 'jakarta-tomcat.tar.gz' to c:\cygnus\usr\local and unpack the archive as below:

```
~> ls d:/tmp/jakarta-tomcat.tar.gz
d:/tmp/jakarta-tomcat.tar.gz
~> cp d:/tmp/jakarta-tomcat.tar.gz c:/cygnus/usr/local/
~> cd c:/cygnus/usr/local/
/usr/local> tar xvfz jakarta-tomcat.tar.gz
jakarta-tomcat/
(... Suppressed ...)
/usr/local> ls jakarta-tomcat
LICENSE      bin/       conf/       doc/
lib/         src/       webapps/
/usr/local>
```

Figure 5.1: Copying 'jakarta-tomcat.tar.gz' to c:\cygnus\usr\local

Define TOMCAT_HOME and extend your classpath as shown in figure5.2 by adding the two lines to your AUTOEXEC.BAT:

```
set TOMCAT_HOME=c:\cygnus\usr\local\jakarta-tomcat
set CLASSPATH=c:\cygnus\usr\local\jdk1.2.2\lib\tools.jar;%CLASSPATH%
```

Figure 5.2: Editing AUTOEXEC.BAT for TOMCAT

Restart your PC to take the change into effect.

[1] http://jakarta.apache.org/
[2] http://www.apache.org/
[3] http://jakarta.apache.org/downloads/binindex.html

5.1.3 Testing Tomcat

To test Tomcat, change directory to jakarta-tomcat/bin and type 'startup.bat' as shown in figure5.3:

Another window subsequently pops up, showing the message as shown in figure5.4

Tomat is now running. To check if you have successfully invoked Tomcat, evoke a browser to enter the URL, http://127.0.0.1:8080. You will see the Tomcat Opening Window as shown in figure5.5 if everything went well:

To shutdown Tomcat, simply type 'shutdown.bat' under jakarta-tomcat/bin in the window where you envoked Tomcat. The message shown in figure5.6 appears and the window figure5.4 will close.

5.2 Creating Web applications

5.2.1 Making your working directory

You first make a directory where you produce all the files necessary to build your application. Let HOME/hello be a working directory (because we will create a very simple application that only displays 'Hello World' when invoked.

Starting Tomcat without defining TOMCAT_HOME

You can run Tomcat even before defining TOMCAT_HOME as long as you run jakarta-tomcat/bin/startup.bat under jakarta-tomcat/bin. The procedure to test Tomcat explained below was in fact tested before TOMCAT_HOME is defined in AUTOEXEC.BAT. (Note that the paths to jar files are shown as relative path. If you define TOMCAT_HOME, the paths are shown as absolute.)

Iincluding tools.jar to your CLASSPATH

You need to include 'tools.jar' in your classpath when you build a Web application using Ant tools, though the jar file is not requied for running Servlet programs. You had better therefore include 'tools.jar' at this stage.

```
~> cd /usr/local/jakarta-tomcat/bin/
/usr/local/jakarta-tomcat/bin> ls
ant*            jspc.bat*       shutdown.sh*    tomcat.bat*
ant.bat*        jspc.sh*        startup.bat*    tomcat.sh*
antRun*         shutdown.bat*   startup.sh*     tomcatEnv.bat*
/usr/local/jakarta-tomcat/bin> startup.bat
Starting tomcat in new window
Using classpath:
..\classes;..\lib\webserver.jar;..\lib\jasper.jar;..\lib\xml.jar;
..\lib\servlet.jar;c:\cygnus\usr\local\jdk1.2.2\lib\tools.jar;
c:\cygnus\usr\local\lib\jpl\jpl.jar;
/usr/local/jakarta-tomcat/bin>
```

Figure 5.3: Starting Tomcat

```
C:\cygnus\usr\local\jakarta-tomcat\bin>
Context log: path="/examples" Adding context path="/examples"
docBase="webapps/examples"
Context log: path="" Adding context path=""  docBase="webapps/ROOT"
Context log: path="/test" Adding context path="/test"  docBase="webapps/test"
Starting tomcat. Check logs/tomcat.log for error messages
Starting tomcat install=".." home="C:\cygnus\usr\local\jakarta-tomcat"
classPath="..\classes;..\lib\webserver.jar;..\lib\jasper.jar;..\lib\xml.jar;
..\lib\servlet.jar;c:\cygnus\usr\local\jdk1.2.2\lib\tools.jar;
c:\cygnus\usr\local\lib\jpl\jpl.jar;"
Context log: path="/admin" Automatic context load
docBase="C:\cygnus\usr\local\jakarta-tomcat\webapps\admin"
```

Figure 5.4: Running Tomcat

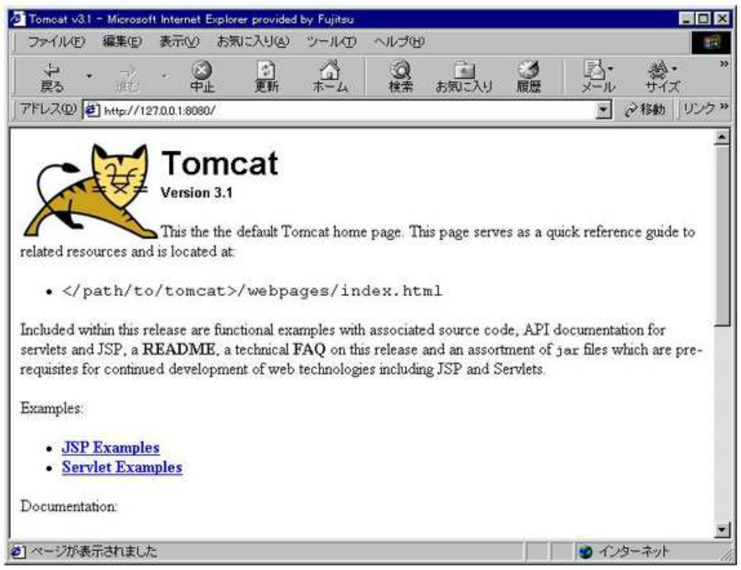

Figure 5.5: Opening Tomcat

```
/usr/local/jakarta-tomcat/bin> shutdown.bat
Using classpath:
..\classes;..\lib\webserver.jar;..\lib\jasper.jar;
..\lib\xml.jar;..\lib\servlet.jar;
c:\cygnus\usr\local\jdk1.2.2\lib\tools.jar;
c:\cygnus\usr\local\lib\jpl\jpl.jar;
Stop tomcat
/usr/local/jakarta-tomcat/bin>
```

Figure 5.6: Shutting down Tomcat

Tomcat User's Guide

Consult "Tomcat - A Minimalistic User's Guide"(jakarta-tomcat/doc/uguide/tomcat_ug.html) for more detail.

```
~> mkdir hello
```

Figure 5.7: Making a working directory

5.2.2 Making sub-directories

Create **etc**, **lib**, and **src** directories under HOME/hello. We will put 'web.xml' under 'etc' and 'Hello.java' under 'src'. The 'lib' directory does not store any file, but it must exist while compilation. You need **web** directory, too. You can copy the directory, 'jakarta-tomcat/doc/appdev/sample/web', to your HOME/hello with all the files under it.

```
~> cd hello
~/hello> mkdir etc lib src
~/hello> ls
etc/  lib/  src/
~/hello> cp -R /usr/local/jakarta-tomcat/doc/appdev/sample/web .
~/hello> ls
etc/  lib/  src/  web/
~/hello> ls web
hello.jsp   images/     index.html
```

Figure 5.8: Making sub-directories

The 'web' derectory includes 'index.html' which you will open to invoke a Servlet program. The crucial part of the HTML file is the line 24, where a link to 'hello' is set to invoke our Servlet.

list 24: The part to invoke a Servlet in web/index.html

```
21:<p>To prove that they work, you can execute either of the following links:
22:<ul>
23:<li>To a <a href="hello.jsp">JSP page</a>.
24:<li>To a <a href="hello">servlet</a>.
25:</ul>
```

5.2.3 Editing the source code

The class called 'Hello.java' is as shown in list25 and put it under 'src'. You do not need to compile the Java program at this moment; It will be compiled automatically with other files comprising the sample web application when you run the script.

list 25: Hello.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Hello extends HttpServlet {
        public void doGet (
                HttpServletRequest req,
                HttpServletResponse res
                ) throws ServletException, IOException
        {
                res.setContentType( "text/html" );
                PrintWriter out = res.getWriter();
                out.println( "<html><head><title>Hello World Servlet</title></head>" );
                out.println( "<body>Hello, World!</body></html>" );
                out.close();
        }
}
```

5.2.4 Editing build.bat and build.xml

The batch file, 'build.bat', contains a script to evoke Ant tool that creates the application. You can copy any file bearing the same file name in the distribution to your HOME/hello. Copy, for example, 'jakarta-tomcat/doc/appdev/sample/build.bat' to HOME/hello.

Below list26 shows 'build.xml' file for our example. The definition is almost the same as given in samples except the first two lines which specify the project name, 'Hello World', and application name, 'mytest', respectively. When you build your application, the files comprising your application are all put under '/usr/local/jakarta-tomcat/webapps/mytest'.

list 26: build.xml

```
<!-- The project name is 'Hello World' -->
```

```
<project name="Hello World" default="compile" basedir=".">

<!-- The application name is 'mytest' -->
  <property name="app.name"       value="mytest"/>
  <property name="deploy.home"    value="${tomcat.home}/webapps/${app.name}"/>
  <property name="dist.home"      value="${deploy.home}"/>
  <property name="dist.src"       value="${app.name}.jar"/>
  <property name="dist.war"       value="${app.name}.war"/>
  <property name="javadoc.home"   value="${deploy.home}/javadoc"/>

  <target name="prepare">
    <mkdir dir="${deploy.home}"/>
    <copydir src="web" dest="${deploy.home}"/>
    <mkdir dir="${deploy.home}/WEB-INF"/>
    <copyfile src="etc/web.xml" dest="${deploy.home}/WEB-INF/web.xml"/>
    <mkdir dir="${deploy.home}/WEB-INF/classes"/>
    <mkdir dir="${deploy.home}/WEB-INF/lib"/>
    <copydir src="lib" dest="${deploy.home}/lib"/>
    <mkdir dir="${javadoc.home}"/>
  </target>

  <target name="clean">
    <deltree dir="${deploy.home}"/>
  </target>

  <target name="compile" depends="prepare">
    <javac srcdir="src" destdir="${deploy.home}/WEB-INF/classes"
           classpath="${deploy.home}/WEB-INF/classes"
           debug="on" optimize="off" deprecation="off"/>
  </target>

  <target name="javadoc" depends="prepare">
    <!-- TODO -->
  </target>

  <target name="all" depends="clean,prepare,compile,javadoc"/>

  <target name="dist" depends="prepare,compile">
    <jar jarfile="${dist.home}/${dist.src}"
         basedir="."/>
    <jar jarfile="${dist.home}/${dist.war}"
         basedir="${deploy.home}"/>
  </target>
</project>
```

5.2.5 Editing web.xml

The 'web.xml' as shown in list27 specifies how the Servlet is invoked. 'servlet-name' stores the name of the Servlet, i.e., 'HelloServlet', whose (actual) class file is specified by 'servlet-class', that is, Hello(.class). The definition under 'servlet-mapping' specifies the way the Hello Servlet is called as URL. In the definition, a servlet named 'HelloServlet' is invoked as a URL, (http://127.0.0.1:8080/mytest)/hello, where 'http://127.0.0.1:8080/mytest' is its application domain and suppressed in the definition. The class file, 'Hello.class', is thus, invoked as a URL, 'hello', via 'servlet-name'.

list 27: etc/web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">

<web-app>
    <servlet>
        <servlet-name>
            HelloServlet
        </servlet-name>
        <servlet-class>
            Hello
        </servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>
            HelloServlet
        </servlet-name>
        <url-pattern>
            /hello
        </url-pattern>
    </servlet-mapping>
</web-app>
```

5.2.6 Build your application

Thanks to Ant tool, it is easy to build an application. Change your directory to HOME/hello. Then, run 'build.bat' as shown in figure5.9. You can check if the directory, /usr/local/jakarta-tomcat/webapps/mytest, is created with the directory structure as presented.

```
~/hello> build.bat
Buildfile: build.xml
Project base dir set to: C:\cygnus\home\hello
Executing Target: prepare
Created dir: C:\cygnus\usr\local\jakarta-tomcat\webapps\mytest
Copying 2 files to C:\cygnus\usr\local\jakarta-tomcat\webapps\mytest
Created dir: C:\cygnus\usr\local\jakarta-tomcat\webapps\mytest\WEB-INF
Created dir: C:\cygnus\usr\local\jakarta-tomcat\webapps\mytest\WEB-INF\classes
Created dir: C:\cygnus\usr\local\jakarta-tomcat\webapps\mytest\WEB-INF\lib
Created dir: C:\cygnus\usr\local\jakarta-tomcat\webapps\mytest\javadoc
Executing Target: compile
Compiling 1 source files to
C:\cygnus\usr\local\jakarta-tomcat\webapps\mytest\WEB-INF\classes
Completed in 6 seconds
~/hello> ls /usr/local/jakarta-tomcat/webapps/mytest/
WEB-INF/    images/     index.html  javadoc/
~/hello>
```

Figure 5.9: Building your application

5.2.7 Editing server.xml

The last stub is to edit '/usr/local/jakarta-tomcat/conf/server.xml' so that Tomcat can recognize your application. Open the file to edit its last part as shown in list28. You need to add to it a definition with a tag, 'Context', which enables Tomcat to load your application kept under jakarta-tomcat/webapps/mytest upon request to 'mytest'.

list 28: Editing jakarta-tomcat/conf/server.xml

```
        <Context path="/mytest" docBase="webapps/mytest" debug="0" reloadable="true">
        </Context>

    </ContextManager>
</Server>
```

5.2.8 Running your application

Stop Tomcat if it is running. Start Tomcat and enter the URL, 'http://127.0.0.1:8080/mytest/index.html' or 'http://127.0.0.1:8080/mytest/'. You should see the window opening as de-

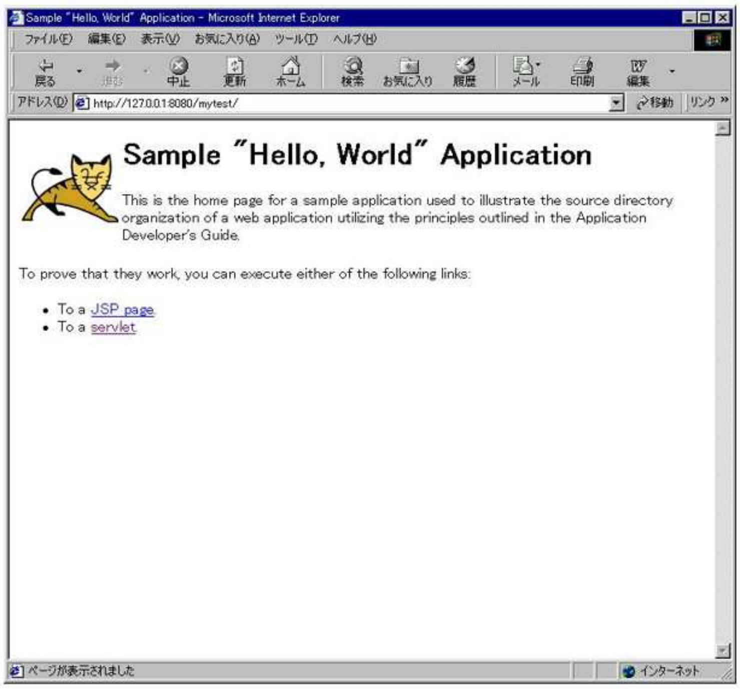

Figure 5.10: Hello World Page

picted in figure5.10.

Upon cliking 'servlet' in the page, you will be carried into another page as depicted in figure5.11. Note that you can evoke the same page directly by entering 'http://127.0.0.1:8080/mytest/servlet/Hello', too.

5.3 Servlet basics

We cover very basics of Servlet. Consult Servlet API Specification [4] at SUN site and other publically available texts on Servlet for more detai.

[4] http://java.sun.com/products/servlet/download.html

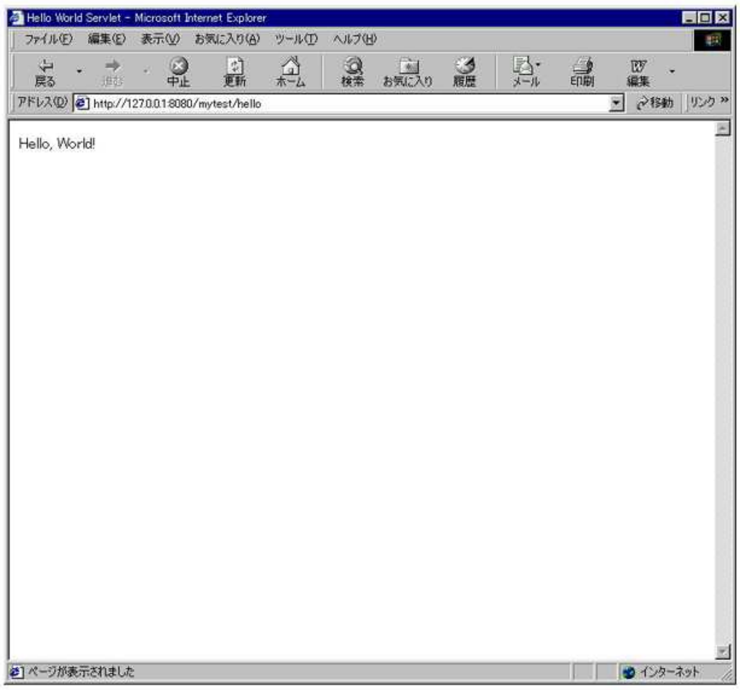

Figure 5.11: Hello World Servlet

A note on developing web applications

Consult **Developing Applications With Tomcat**, which you can find under 'jakarta-tomcat/doc/appdev' for more detail on developing web applications with Tomcat. You should especially take a look of Chapter 5, 'Dvelopment Process'.

5.3.1 Sending parameter values from Browser to Servlet

The easiest way to send a pair of parameter and value from Browser is to use 'POST' or 'GET' method in 'form' tag. Below example list29 illustrates how a Servlet can be invoked from a Browser. In the description of action="/parTest/paramTest", '/parTest' denotes the application name, which we decided to call 'parTest'. The other string, '/paramTest', is a link name (URL) of the Servlet to be invoked.

list 29: paramTest.html

```
<html>
<head>
<meta HTTP-EQUIV="Content-Type" CONTENT="text/html">
<title>Testing parameter-value pairs</title>
</head>
<body>
<form action="/parTest/paramTest" method="POST">
<input type="text" name="name">
<input type=submit value="SUBMIT">
</form>
</body>
</html>
```

Let 'HOME/paramTest' be your working directory for this application and put 'paramTest.html' under 'HOME/paramTest/web'. When you have built your application, you will see the window as shown in figure5.12 appearing when you enter 'http://127.0.0.1:8080/parTest/ParamTest.html'. You can write any string, e.g., "fujinami" for test, and click the button, 'SUBMIT' to perform the action.

5.3.2 Receiving parameter values in Servlet

An instance, 'req', of HttpServletRequest class serves to receive data from Browser. To receive the list of parameter names, the method, **getParameterNames**(), is employed. To retrieve a value of each parameter, another method, **getParameter**() is employed as shown in list30

list 30: paramServ.java

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

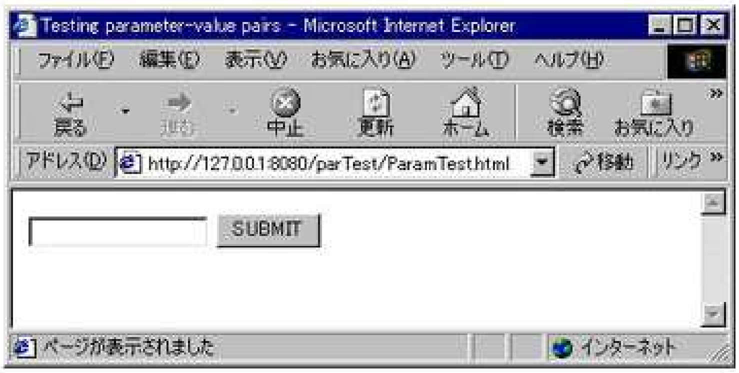

Figure 5.12: A sample form

```
public class paramServ extends HttpServlet {
        public void doGet (
                HttpServletRequest req,
                HttpServletResponse res
                ) throws ServletException, IOException
        {
                res.setContentType( "text/html" );
                PrintWriter out = res.getWriter();
                out.println( "<html><head><title>Receiving parameter-value pairs</title></head>" );
                out.println( "<body>Using getParameter() method<br>" );
                Enumeration enum = req.getParameterNames();
                while( enum.hasMoreElements() ) {
                        String name = (String)enum.nextElement();
                        String value = req.getParameter( name );
                        out.println( name + "=" + value + "<br>" );
                }
                out.println( "</body></html>" );
                out.close();
        }

        public void doPost (
                HttpServletRequest req,
                HttpServletResponse res
                ) throws ServletException, IOException
        {
                doGet( req, res );
        }
}
```

5.3.3 Sending HTML documents to Browser from Servlet

To send back a HTML file from Servlet, an instance, 'res', of HttpServletResponse class is employed. The method, getwriter(), creates an instance, 'out', of PrintWriter class as shown in list30. You can write strings by using println() method of 'out'.

5.3.4 Building and Running the sample

You can run the sample as follows:

1. Create a working directory, e.g., HOME/paramTest
2. Make sub-directories, etc, lib, src, and web under HOME/paramTest
3. Copy build.bat (as above) to HOME/paramTest
4. Copy 'ParamTest.html' above to HOME/paramTest/web
5. Copy 'paramServ.java' to HOME/paramTest/src
6. Create the 'web.xml' as below under HOME/paramTest/etc. The definition ensures that the URL pattern, /paramTest, is related to ParamServ Servlet.

   list 31: web.xml for parTest application

```
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">

<web-app>
    <servlet>
        <servlet-name>
            paramServlet
        </servlet-name>
        <servlet-class>
            paramServ
        </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>
            paramServlet
```

```
        </servlet-name>
        <url-pattern>
            /paramTest
        </url-pattern>
    </servlet-mapping>
</web-app>
```

7. Create HOME/paramTest/build.xml. You only need to edit build.xml above in two points. For project name, you can for example define it as follows:

```
<project name="Testing Parameter-Value pairs" default="compile" basedir=".">
```

   For the base directory under which all the files are stored, we name it 'pTest'.

```
  <property name="app.name"       value="pTest"/>
```

8. Execute 'build.bat' under HOME/paramTest. You have now created the directory structure under (/usr/local/)jakarta-tomcat/webapps/pTest.

9. The reminding task is to edit (/usr/local/)jakarta-tomcat/conf/server.xml. Open the file and move towards the end of the file to add the two lines below. The definition ensures that the application named 'parTest' stores its files under (/usr/local/)jakarta-tomcat/webapps/pTest directory.

```
<Context path="/parTest" docBase="webapps/pTest" debug="0" reloadable="true" >
</Context>
```

Start Tomcat and enter the ULR address, http://127.0.0.1:8080/parTest/ParamTest.html as explained above. You can fill the form with any string, say, "fujinami". Click 'SUBMIT' button. You are shortly seeing the window as shown in figure5.13 appearing on display.

Figure 5.13: The result upon clicking SUBMIT

Part II

Extending your web server with prolog and RDB

Chapter 6

Integrating prolog and Java on Servlet

6.1 JPL: a Java interface to prolog

We learn in this chapter how to call prolog programs from Servlet and return the result on Servlet. The mechanism allows you to build in your Web page various AI technologies implemented in prolog.

We employ JPL, the Java interface to prolog, to call prolog programs from Java. JPL is developed by Fred Dushin and detailed explanation is given in the distribution, under doc directory. We will not therefore repeat the explanation of the interface below. We illustrate how the interface is used on Servlet. The sample programs are taken from the distribution, under 'demo/getting-started' directory. It is good idea to test-run Test.java under the directory before proceeding to check if JPL was installed correctly as shown in 5347. You may have to run it within Meadow if the result is not visible in Bash Window.

A note on Windows95: I have tested the samples below on Windows95 and Windows98 and found that calling prolog program from Servlet does not work on Windows95. I am not sure if it is generally so. Please tell me what happened if you run the samples on Windows95.

```
/tmp/jpl/demo/getting-started> java Test
% test.pl compiled 0.02 sec, 1,084 bytes
child_of(joe, ralf) = true
descendent_of(steve, ralf) = true
querying descendent_of( X, ralf )
X = joe
querying descendent_of( X, ralf )
X = joe
X = mary
X = steve
/tmp/jpl/demo/getting-started>
```

Figure 6.1: Testing JPL

6.2 Sample prolog program

We assume that you have an elementary level of knowledge about prolog programming. Below (list32) is a sample prolog program, 'test.pl', taken from 'demo/getting-started'. The program says that 'joe' is a child of 'ralf', and 'mary' and 'steve' are children of 'joe'. It defines descendent_of relation, too, such that X is descendent of Y if X is a child of Y or if there is a person Z who is child of Y and a parent of X.

list 32: test.pl

```
child_of( joe, ralf ).
child_of( mary, joe ).
child_of( steve, joe ).

descendent_of( X, Y ) :-
        child_of( X, Y ).
descendent_of( X, Y ) :-
        child_of( Z, Y ),
        descendent_of( X, Z ).
```

You can load the program on SWI-prolog to test it as shown in (figure6.2.

```
/tmp/jpl/demo/getting-started> pl
Welcome to SWI-Prolog (Version 3.3.6)
Copyright (c) 1990-2000 University of Amsterdam.
Copy policy: GPL-2 (see www.gnu.org)

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- [test].
% test compiled 0.02 sec, 1,156 bytes
Yes

2 ?- child_of(joe, ralf).
Yes

3 ?- descendent_of( X, ralf ).
X = joe ;
X = mary ;
X = steve ;

No
4 ?-
```

Figure 6.2: Test run

6.3 Handing prolog variables and atoms

The program, demo/getting-started/Test.java, and the documentation illustrate well how to hand SWI-prolog variables and atoms. Let us examine the method, test_3, in Test.java (list33)

list 33: Test.java

```
static void
test_3()
{
        Variable X = new Variable();
        Term args[] = {
                X,
                new Atom( "ralf" )
        };
        Query query =
                new Query(
                        "descendent_of",
                        args );

        System.out.println( "querying descendent_of( X, ralf )" );
        java.util.Hashtable solution =
                query.oneSolution();
        System.out.println( "X = " + solution.get( X ) );
}
```

The method prepares an inquiry, descendent_of(X, ralf), to emit it to SWI-prolog and receives back an atom instantiating X. We modify the method slightly so that it returns an Atom bound to the variable, X. Let us call the method 'test3'. The method is defined as below (list34). The type of the method is changed from 'void' to 'Atom' and the line, 'return (Atom)solution.get(X);' replaces for 'System.out.println("X = " + solution.get(X));'

list 34: Method test3

```
//
//  This is test3 method.
//  The type of the return value is Atom.
//
static Atom
test3( String person )
{
        //
```

```
        // We construct the query, descendent_of( X, Person )
        // where Person is instantiated already when called
        Variable X = new Variable();
        Term args[] = {
                X,
                new Atom( person )
        };
        Query query =
                new Query(
                        "descendent_of",
                        args );

        //
        // Obtain the first solution to return
        //
        java.util.Hashtable solution =        query.oneSolution();
        return (Atom)solution.get( X );
}
```

6.4 Getting back atoms from prolog

Executing 'query.oneSolution()' calls SWI-prolog, emitting the variable and name. By executing 'solution.get(X)', the Java method receives back some atom bound to X. We have to cast the return value to Atom type because the returned value is of 'Object' type.

6.5 Calling prolog from Servlet

We show the part of the program up to test3 method (list35). The sample is almost identical with the one shown to illustrate basic usage of Servlet. Points to note are:

1. The location of 'test.pl' must be specified in full, as 'new Atom("/usr/local/jakarta-tomcat/webapps/proServ/WEB-INF/classes/test.pl")' because Tomcat does not know how to relate 'test.pl' to the file itself. (test.pl is not a class file.)
2. We use the method, name(), to retrieve the name of Atom, i.e., 'String descendent = test3(value).name();' Consult the chapter, 'High-Level Interface' (doc/high-level_interface.html), for more detai.

list 35: prologServ.java

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
//
//   JPL package
//
import jpl.Atom;
import jpl.Variable;
import jpl.Term;
import jpl.Query;
import jpl.JPL;

public class prologServ extends HttpServlet {
        public void doGet (
                HttpServletRequest req,
                HttpServletResponse res
                ) throws ServletException, IOException
        {
                res.setContentType( "text/html" );
                PrintWriter out = res.getWriter();
                out.println( "<html><head><title>Finding decendents of the person</title></head>" );
                out.println( "<body>Using getParameter() method<br>" );

                JPL.init(); // We initialize JPL
                Term consult_arg[] = {
                        //
                        // We have to specify the location of prolog program in full
                        //
                        new Atom( "/usr/local/jakarta-tomcat/webapps/proServ/WEB-INF/classes/test.pl"
                };
                //
                // We consult 'test.pl'
                //
                Query consult_query =
                        new Query(
                                "consult",
                                consult_arg );

                boolean consulted = consult_query.query();

                if ( !consulted ){
                        out.println( "Consult failed" );
                        System.exit( 1 );
                }

                Enumeration enum = req.getParameterNames();
                while( enum.hasMoreElements() ) {
                        String name = (String)enum.nextElement();
                        String value = req.getParameter( name );
```

```
                    out.println( name + "=" + value + "<br>" );
                    //
                    //   We pass the task to test3 below
                    //
                    String descendent = test3( value ).name();
                    out.println( "descendent = " +  descendent + "<br>" );
            }
            out.println( "</body></html>" );
            out.close();
    }

    static Atom
    test3( String person )
    {
        ( ... Suppressed ... )
    }

    public void doPost
        ( ... Suppressed ... )
}
```

6.6 Running the sample

We create the application named 'proServ' under some working directory and build it as explained above. Upon entering the address, 'http://127.0.0.1:8080/proServ/index.html', you are prompted to enter a name of person, whose descendent we like to know of (figure6.3). We type in 'ralf' for this example.

Upon cliking 'SUBMIT', you are shown the result that 'joe' is a descendent of 'ralf', which was inferred by prolog (figure6.4).

You can check that the prolog program is loaded on the starting window of Tomcat (Monitoring invocation of prolog.tex)

Figure 6.3: Query window

Figure 6.4: Result

```
( ... Suppressed ... )
Context log: path="/admin" Automatic context load
             docBase="C:\cygnus\usr\local\jakarta-tomcat\webapps\admin"
Context log: path="/admin" Adding context path="/admin"
             docBase="C:\cygnus\usr\local\jakarta-tomcat\webapps\admin"
% /usr/local/jakarta-tomcat/webapps/proServ/WEB-INF/classes/test.pl
compiled 0.09 sec, 1,344 bytes
```

Figure 6.5: Monitoring invocation of prolog

Source code

All the files comprising proServ application are found in Appendix.

Calling prolog program from Servlet did not work on Windows95 for my case. Please share your experience if you tried to do the same on Windows95.

Chapter 7

Database Connection using Java

7.1 Which database management system?

The goal of this part is to show you how best we can combine Database and prolog. We have shown you how to call prolog on Servlet. We will show you in next section how to connect prolog to Relational Database on Servlet/Java, but before that we have to learn how to use Relational Database on Servlet/Java.

The first question should be, then, "Which database shall we employ?" Sincerely speaking, we do not have a best solution because there is no free, open-source relational database runnable on Windows 9X. We may try one of the followings:

- MySQL on Linux: We install MySQL on Linux, calling it from our PC.
- PostgreSQL on Linux: We install PostgreSQL on Linux, calling it from our PC.
- MySQL on PC: We install MySQL for Windows on our PC.
- Access on PC: We install Access on our PC.

There is of course a trade-off between these options. If you choose the first or second option, you can use the database for free, but need another PC running Linux. If you opt for the third or fourth option, you do not need to buy another PC, but have to pay for the package. The best solution seems to me that you install Linux on your PC and will do everything on Linux, throwing away Windows9X. It is, however, our intention to work on Windows9X, thus we cannot put away Windows9X.

My first choice is the first option, that is, we install MySQL on a Linux machine and call the database from our PC running on Windows9X. Why? well, I suppose that you work on Windows9X only for developing software, not for running your server to respond to the requests from public. In the end you may want to install your web applications on more reliable machines, i.e, a PC running on Linux. It makes sense, therefore, to install the database management system on the PC running on Linux, to which you will import your web applications for public access. Duplicating a database on your PC for development and the other running as webserver, is not good idea at all.

It is more sensitive question, why we opt for MySQL, not for PostgreSQL. Generally speaking, MySQL is smaller and efficient than PostgreSQL, but some of the functionalities available with PostgreSQL are missing. I believe that MySQL is sufficient to building web applications for experimentation. We may have to employ PostgreSQL to build bigger systems in the future, but for the moment we only build small systems to study how we can import ideas from AI to Web. For rapid prototyping, we prefer MySQL for its efficiency to PostgreSQL.

There are a number of good texts on JDBC. I do therefore not go into detail in the following and only explain the points which you might miss.

7.2 Download and Installation

7.2.1 Installing MySQL Server

You can download the package from MySQL Page[1]. You can find in the package a detailed instruction for installing mySQL on Linux.

7.2.2 Register yourself

After installing MySQL server, you have to add yourself to the user list. The crucial point in registering yourself is to grant you to get an access to the server from a remote PC other than localhost. If you allow yourself only to login the server on localhost, you will never establish a connection from your PC to the server through LAN.

Consult the section 6.13 titled "Adding new user privileges to MySQL" of **MySQL Rererence Manual** (mine is Version 3.23.7-alpha). To repeat the explanation, you can grant yourself the previlege to login the server with the right to do anything you like from any machine on the net as follows:

[1] http://www.mysql.com

```
shell> mysql -u root -p
Enter password: (Here you enter the password for root)
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 38 to server version: 3.22.27

Type 'help' for help.

mysql> use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> GRANT ALL PRIVILEGES ON *.* TO fuji@"%"
       IDENTIFIED BY 'YourPassword' WITH GRANT OPTION;
```

Figure 7.1: Adding yourself to the user list

Note that you should type in an appropriate password for 'YourPassword'. The symbol, '%', in the above works as wild card in SQL, thus 'fuji@"%"' means that the user 'fuji' is allowed to enter from any computer. Beaware that you must always specify the host name after then to operate on data on the server even when you login it locally, that is, you have to login the server as below:

```
shell> mysql -h ks15e0f00 -u fuji -p
```

Figure 7.2: Specifying your host to login MySQL Server

(Type your host name for 'ks15e0f00'.) Your access to particular databasese will be denied, otherwise.

Check finally if you have registered yourself as you intended. That is, you login the server as explained above, choosing the database, 'mysql'. Execute the command, 'select * from user;', then. You will see your host privilege displayed as '%' in the first column if everything went well.

7.2.3 Installing JDBC driver

My preference goes to MM MySQL JDBC Driver developed by Mark Matthews. Download the package from his web page[2]. The latest is the version 2.0.2, whose file you can find as mm.mysql-2.0.2-bin.jar (10.7.2000). To install the driver, simply place the jar file in your classpath.

7.3 MySQL Basics

We go through very basics of MySQL. The material given here is apparently insufficient even to build toy systems. The reader not familiar with relational databases and SQL, is strongly rocommended to consult other texts on these topics.

7.3.1 Creating Databases

We build a sample database which will store the pairs of people who stand in 'child_of' relation. Let 'mysqltest' be the database. To create the database, you type in as follows:

```
mysql> CREATE DATABASE mysqltest;
Query OK, 1 row affected (0.00 sec)
```

Figure 7.3: Creating mysqltest database

Try 'show databases' command to list up the databases on your server.

7.3.2 Creating Tables

Before creating a table in a database, you have to first choose the database as your working database. To choose 'mysqltest' database, type in the command as below:

We create a table to keep records on the people who stand in child_of relation. An image of the table is depicted as follows:

[2] http://www.worldserver.com/mm.mysql/

```
mysql> use mysqltest;
Database changed
```

Figure 7.4: Choosing mysqltest database

Table 7.1: child_of relation

child	parent
joe	ralf
mary	joe
steve	joe

We assume the lengh of names is at most 10. The table, child_of, is thus specified as composed of two colums, each stores the name of child and that of parent.

```
mysql> CREATE TABLE child_of
    -> (child CHAR(10),
    -> parent CHAR(10));
Query OK, 0 rows affected (0.09 sec)

mysql> show tables;
+---------------------+
| Tables in mysqltest |
+---------------------+
| child_of            |
+---------------------+
1 row in set (0.00 sec)
```

You add to the table the pair of 'joe' and 'ralf' as follows:

list 36: Inserting the pair of 'joe' and 'ralf'

```
mysql> INSERT into child_of (child, parent) VALUES('joe', 'ralf');
Query OK, 1 row affected (0.00 sec)

mysql> select * from child_of;
+-------+--------+
| child | parent |
+-------+--------+
| joe   | ralf   |
+-------+--------+
1 row in set (0.00 sec)
```

Other pairs can be inserted to the table as well:

list 37: Inserting other pairs

```
mysql> INSERT into child_of (child, parent) VALUES('mary', 'joe');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT into child_of (child, parent) VALUES('steve', 'joe');
Query OK, 1 row affected (0.00 sec)

mysql> select * from child_of;
+-------+--------+
| child | parent |
+-------+--------+
| joe   | ralf   |
| mary  | joe    |
| steve | joe    |
+-------+--------+
3 rows in set (0.00 sec)
```

7.4 Database connection

7.4.1 Prerequisite

We use the JDBC driver to connect to our mySQL server from Java programs. We assume that you have already installed mm.mysql, a JDBC driver for MySQL, from Mark Matthews' web page[3]. Installation is simple; you only need to place mm.mysql-2.0.2-bin.jar in your classpath.

7.4.2 Sample Code

We show you the simplest program which logs in our MySQL server to read and display all the records in child_of table of mysqltest database. Below is the sample code:

list 38: JDBCSample.java

[3] http://www.worldserver.com/mm.mysql/

```
/*
  A sample code to connect to mySQL server using mm.mysql(2.0.2)
*/

import java.io.*;
import java.sql.*;

public class JDBCSample
{
    public static void main( String argv[] )
    {
        try {
            Class.forName( "org.gjt.mm.mysql.Driver" );
            String url = "jdbc:mysql://ks15e0f00/mysqltest?user=YourID&password=YourPassword";
            Connection con = DriverManager.getConnection( url );

            Statement select = con.createStatement();
            String sql = "select * from child_of";
            sql = new String(sql.getBytes("SJIS"), "8859_1" );
            ResultSet res = select.executeQuery( sql );

            int count = 0;
            while( res.next() ) {
                count++;
                String result = res.getString( "child" ) + "\t" + res.getString( "parent" );
                result = new String( result.getBytes("8859_1"), "SJIS" );
                System.out.println( result );
            }
            System.out.println( "Record count=" + count );
            select.close();
            con.close();
        }
        catch ( Exception e ) {
            e.printStackTrace();
        }
    }
}
```

You must replace your machine name for 'ks15e0f00', your userID for 'yourID', and your password 'YourPassword'. (Your userID and password are for loging in MySQL server, not to enter the Linux machine.) If you are familiar with the programing using JDBC, there must be no mystery in the code. We explain some details of the code below.

7.4.3 Test-run

Before going into detail, we had better show you how it works. You have to compile JDBCSample.java by typing 'javac JDBCSample.java'. You can then execute the program as follows:

```
~/docs/fp/programs/jdbc> java JDBCSample
joe     ralf
mary    joe
steve   joe
Record count=3
```

7.4.4 Explanation about the code

Packages

You have to use the packages under java.sql.

list 39: Importing the SQL packages

```
import java.io.*;
import java.sql.*;
```

Establishing a connection

The first line loads the driver for connecting the MySQL server. The second line specifies the database to connect with its machine name and the name of the database (i.e., 'mysqltest'), followed by your userID and password. The third line establishes the connedtion.

```
Class.forName( "org.gjt.mm.mysql.Driver" );
String url = "jdbc:mysql://ks15e0f00/mysqltest?user=YourID&password=YourPassword";
Connection con = DriverManager.getConnection( url );
```

Executing the query

The first line prepares for the statement to be sent to MySQL server. The second line construct the query sentence. The third line will convert a query coded in SJIS (Japanese) to binary. (We store data coded in Japanese as binary on our MySQL server. You do not need to include this line if you will not use Japanese characters.) The last line execute the query to receive the result to 'res'.

```
Statement select = con.createStatement();
String sql = "select * from child_of";
sql = new String(sql.getBytes("SJIS"), "8859_1" );
ResultSet res = select.executeQuery( sql );
```

Reading out the result

We go read out the result line by line. The first line sets the counter to zero, which will be referred to for printing out the number of lines retrieved at end. In the while loop, we read out the data in the columns of 'child' and 'parent' and concatenate them to store the result in 'result'. The fourth line is for converting the data format to SJIS from binary. The result is printed out to the last line. At end, the number of lines is displayed.

```
int count = 0;
while( res.next() ) {
    count++;
    String result = res.getString( "child" ) + "\t" + res.getString( "parent" );
    result = new String( result.getBytes("8859_1"), "SJIS" );
    System.out.println( result );
}
System.out.println( "Record count=" + count );
```

Closing up

We finally finish the query and close the connection.

```
select.close();
con.close();
```

Chapter 8

Connecting prolog to Relational Database on Servlet

8.1 Why prolog and RDB?

We will conclude this part by showing how to combine prolog and relational databases in Java. One may ask why we should use both prolog and relational databases. We would reply to the question, for example, as follows:

- For prolog programmers, it is nice to be able to read in data from RDB. The relational databases may provide prolog with huge amount of data. Prolog programmers can concentrate on implementing **logic** of data processing.
- For database developers, it is nice to be able to use the strong inference engine provided by prolog. Constructing queries may be made easier by employing prolog to translate requests from user (or some other agent) to SQL sentences. (Ultimately, users may ask their questions in natural language.)

The combination of prolog, RDB, and Java, may develop into an intelligent system with huge supply of data both from RDB and through Internet. We do not know yet how we can exploit the possibility, but the goal sounds attractive, especially to the ear of retired prolog programmers.

8.2 The demo story

We demonstrate how prolog works with RDB with an example. Recall the prolog program we examined to test the connection between prolog and Servlet. The program is reproduced below for readers' convenience:

list 40: test.pl

```
child_of( joe, ralf ).
child_of( mary, joe ).
child_of( steve, joe ).

descendent_of( X, Y ) :-
        child_of( X, Y ).
descendent_of( X, Y ) :-
        child_of( Z, Y ),
        descendent_of( X, Z ).
```

Of the definitions, the first three lines defining the child_of relation will vary, depending on the domain to which you apply the 'descendent_of' program. The relation is therefore a good candidate to encode in RDB. We remove the lines to slim down 'test.pl' program as follows:

list 41: test2.pl

```
descendent_of( X, Y ) :-
        child_of( X, Y ).
descendent_of( X, Y ) :-
        child_of( Z, Y ),
        descendent_of( X, Z ).
```

The items of information we have removed from the prolog program, test.pl, is obtainable from the database, mysqltest, we built in the previous section to teach you the very basics of MySQL:

list 42: The items in child_of table

```
mysql> use mysqltest;
Database changed

mysql> select * from child_of;
+-------+--------+
```

```
| child | parent |
+-------+--------+
| joe   | ralf   |
| mary  | joe    |
| steve | joe    |
+-------+--------+
3 rows in set (0.00 sec)
```

We will modify the demo system we presented in the beginning of this part, the system that calls prolog to return the name of parent given name of a person. We replace a database connection module for the prolog predicate specifying child_of relation.

8.3 Implementing the demo

8.3.1 Handling facts dynamically in prolog

To allow prolog to handle facts dynamically, we have to declare the predicate, child_of, to be dynamic. The program, test2.pl, must be therefore inserted the first line as below:

list 43: test2.pl with dynamic declaration

```
:- dynamic child_of/2.

descendent_of( X, Y ) :-
        child_of( X, Y ).
descendent_of( X, Y ) :-
        child_of( Z, Y ),
        descendent_of( X, Z ).
```

When you load the program, the definition of child_of is of course missing:

With the declaration, you can insert the set of facts about child_of relation as follows:

With the facts of child_of, one can correctly calculate who is descendent of Ralf:

```
programs/proServ2/src> pl
Welcome to SWI-Prolog (Version 3.3.6)
Copyright (c) 1990-2000 University of Amsterdam.
Copy policy: GPL-2 (see www.gnu.org)

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- [test2].
% test2 compiled 0.02 sec, 860 bytes

Yes
2 ?- listing.


descendent_of(A, B) :-
        child_of(A, B).
descendent_of(A, B) :-
        child_of(C, B),
        descendent_of(A, C).

Yes
```

Figure 8.1: Loading test.pl

```
3 ?- assert( child_of( joe, ralf ) ).

Yes
4 ?- assert( child_of( mary, joe ) ).

Yes
5 ?- assert( child_of( steve, joe ) ).

Yes
6 ?- listing.

child_of(joe, ralf).
child_of(mary, joe).
child_of(steve, joe).

descendent_of(A, B) :-
        child_of(A, B).
descendent_of(A, B) :-
        child_of(C, B),
        descendent_of(A, C).

Yes
```

Figure 8.2: Inserting the facts of child_of

```
7 ?- descendent_of( X, ralf ).

X = joe ;

X = mary ;

X = steve ;

No
```

Figure 8.3: Inferring of the descendent of Ralf

```
8 ?- abolish( child_of/2 ).

Yes
9 ?- listing.

descendent_of(A, B) :-
        child_of(A, B).
descendent_of(A, B) :-
        child_of(C, B),
        descendent_of(A, C).

Yes
```

Figure 8.4: Cleaning up the facts

After getting the result, you should not forget to clean up the data by executing abolish:

8.3.2 Retrieving data from RDB

What we need to do to realize the demo story, is to do the same operation as presented in the previous section using Java. Basic idea is to combine the Java program to call prolog, e.g., Test.java presented in the first chapter of this part, and the other to get access to RDB, JDBCSample.java shown above. The whole program, which we call Test2.java, is shown below:

list 44: Test2.java

```
//****************************************************************************/
//
// This program is drived from Test.java included in the JPL package under
// 'demo/getting_started'. The original was implemented by Fred Dushin,
// who developed JPL package.
//
//****************************************************************************/

import java.util.Hashtable;
import jpl.Atom;
import jpl.Variable;
import jpl.Term;
import jpl.Query;
```

```
import jpl.JPL;
import jpl.Compound;
import jpl.Util;
import jpl.Integer;
import java.io.*;
import java.sql.*;

public class Test2
{
    public static void
    main( String argv[] )
    {
        load_prog();
        assert_facts();
        test_4();
        abolish_facts();
    }

    static void abolish_facts() {
        Term pred = new Atom( "child_of" );
        Term num = new Integer( 2 );
        Term rem = new Compound( "/", Util.toTermArray( pred, num ) );

        Query query =
                new Query(
                         "abolish",
                             rem );
        query.oneSolution();
    }

    static void
    assert_facts() {
        try {
            Class.forName( "org.gjt.mm.mysql.Driver" );
            String url = "jdbc:mysql://ks15e0f00/mysqltest?user=YourID&password=YourPassword";
            Connection con = DriverManager.getConnection( url );

            Statement select = con.createStatement();
            String sql = "select * from child_of";
            ResultSet res = select.executeQuery( sql );

            while( res.next() ) {
                Term child = new Atom( res.getString( "child" ) );
                Term parent = new Atom( res.getString( "parent" ) );
                Term pair = new Compound( "child_of", Util.toTermArray( child, parent ) );

                Query assert_query =
                    new Query(
                             "assert",
                             pair );
```

```
            assert_query.oneSolution();
        }
        select.close();
        con.close();
    }
    catch ( Exception e ) {
        e.printStackTrace();
    }
}

static void
load_prog()
{
    JPL.init();

    Term consult_arg[] = {
        new Atom( "test2.pl" )
    };
    Query consult_query =
        new Query(
            "consult",
            consult_arg );

    boolean consulted = consult_query.query();

    if ( !consulted ){
        System.err.println( "Consult failed" );
        System.exit( 1 );
    }
}

static void
test_4()
{
    Variable X = new Variable();
    Term args[] = {
        X,
        new Atom( "ralf" )
    };
    Query query =
        new Query(
            "descendent_of",
            args );

    System.out.println( "querying descendent_of( X, ralf )" );

    while ( query.hasMoreSolutions() ){
        java.util.Hashtable solution =
            query.nextSolution();
        System.out.println( "X = " + solution.get( X ) );
```

```
        }
    }
}
```

The explanation is in order.

Imported packages

We import 'java.util.Hashtable' to recieve the result from prolog. The packages under 'jpl' are part of JPL, the java interface to prolog. The last, 'java.sql.*', is to get access to the RDB.

list 45: Imported packages

```
import java.util.Hashtable;
import jpl.Atom;
import jpl.Variable;
import jpl.Term;
import jpl.Query;
import jpl.JPL;
import jpl.Compound;
import jpl.Util;
import jpl.Integer;
import java.io.*;
import java.sql.*;
```

Flow of control

The top level defines the control flow. The method, 'load_prog()', first loads the prolog program, 'test2.pl'. Another method, 'assert_facts()', establishes the connection to RDB and retrieves data. The method asserts each datum into prolog every time it retrieves a line of data. The method, 'test_4()', prints out the result in prolog. This method is as implemented by Fred Dushin, the author of JPL. We finally clean up the facts of prolog interpreter.

list 46: Control flow

```
load_prog();
assert_facts();
test_4();
abolish_facts();
```

Loading prolog program

We first laod the prolog program, 'test2.pl'. Nothing new to the method as we have already explained about the procedure.

list 47: Loading test2.pl

```
static void
load_prog()
{
    JPL.init();

    Term consult_arg[] = {
        new Atom( "test2.pl" )
    };
    Query consult_query =
        new Query(
            "consult",
            consult_arg );

    boolean consulted = consult_query.query();

    if ( !consulted ){
        System.err.println( "Consult failed" );
        System.exit( 1 );
    }
}
```

Porting data from RDB to prolog

The next thing we have to do is to establish the connection to MySQL. We have already gave an explanation on the procedure. There is nothing new in retrieving data, too. In 'while' loop, however, we insert the data into prolog through JPL. The actoin occurs each time the method reads out a line from RDB.

list 48: porting data from RDB into prolog

```
static void
assert_facts() {
    try {
        Class.forName( "org.gjt.mm.mysql.Driver" );
        String url = "jdbc:mysql://ks15e0f00/mysqltest?user='YourID'&password='YourPassword";
```

```
            Connection con = DriverManager.getConnection( url );

            Statement select = con.createStatement();
            String sql = "select * from child_of";
            ResultSet res = select.executeQuery( sql );

            while( res.next() ) {
                Term child = new Atom( res.getString( "child" ) );
                Term parent = new Atom( res.getString( "parent" ) );
                Term pair = new Compound( "child_of", Util.toTermArray( child, parent ) );

                Query assert_query =
                    new Query(
                              "assert",
                              pair );
                assert_query.oneSolution();
            }
            select.close();
            con.close();
        }
        catch ( Exception e ) {
            e.printStackTrace();
        }
    }
```

Executing the query to display the result

The method, 'test_4()', executes the query, descendent_of(X, ralf), in prolog and receives the result through JPL.

list 49: test_4

```
    static void
    test_4()
    {
        Variable X = new Variable();
        Term args[] = {
            X,
            new Atom( "ralf" )
        };
        Query query =
            new Query(
                "descendent_of",
                args );

        System.out.println( "querying descendent_of( X, ralf )" );
```

```
        while ( query.hasMoreSolutions() ){
            java.util.Hashtable solution =
                query.nextSolution();
            System.out.println( "X = " + solution.get( X ) );
        }
    }
```

Cleaning up

The last thing to do is to clean up the prolog facts, i.e., child_of/2, we inserted for the query. The point to note is that the expression, child_of/2, is in fact a compound term such as /(child_of, 2). No mystery, otherwise. You can skip the execution of this method because the inserted facts disappear anyway when Tomcat terminates.[1]

list 50: abolish_facts

```
    static void abolish_facts() {
        Term pred = new Atom( "child_of" );
        Term num = new Integer( 2 );
        Term rem = new Compound( "/", Util.toTermArray( pred, num ) );

        Query query =
                new Query(
                          "abolish",
                              rem );
        query.oneSolution();
    }
```

8.3.3 Test-run

The program, Test2.class, works as follows when compiled. Do not forget to replace your userID and password for 'YourID' and 'YourPassword' in Test2.java.

[1]We come back to this point later. The trouble is that the method, JPL.halt(), causes Tomcat crash, thus we cannot terminate prolog within a Servlet. Fred Dushin, the developer of JPL, claims that this is due to the implementation of SWI-prolog.

```
programs/proServ2/src> java Test2
% test2.pl compiled 0.02 sec, 784 bytes
querying descendent_of( X, ralf )
X = joe
X = mary
X = steve
```

Figure 8.5: Runing Test2.class

8.4 Combining prolog and MySQL on Servlet

We have now everything at our hand to combine prolog and MySQL on Servlet. We will extend the sample Servlet, proServ, which we developed in the first chapter of this part. The Servlet was to return the name of a parent given a person name by calling prolog program. The modified version of the Servlet will return all the names of parents of the person.

8.4.1 The source code

We show you above all the source code. The reader who have read through the text in every corer should understand it without difficulties. The explanation comes in order.

list 51: prologServ2.java

```
//
//   for Servlet
//
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
//
//   JPL package
//
import jpl.Atom;
import jpl.Variable;
import jpl.Term;
import jpl.Query;
import jpl.JPL;
import java.util.Hashtable;
import jpl.Compound;
import jpl.Util;
```

```
import jpl.Integer;
//
//   for SQL
//
import java.sql.*;

public class prologServ2 extends HttpServlet {
    public void doGet (
        HttpServletRequest req,
        HttpServletResponse res
        ) throws ServletException, IOException
    {
        res.setContentType( "text/html" );
        PrintWriter out = res.getWriter();
        out.println( "<html><head><title>Finding decendents of the person</title></head>" );
        out.println( "<body>Using SWI-prolog and MySQL<br>" );

        load_prog();    // Loading test2.pl
        assert_facts(); // Asserting facts into prolog

        Enumeration enum = req.getParameterNames();
        while( enum.hasMoreElements() ) {
            String name = (String)enum.nextElement();
            String value = req.getParameter( name );
            out.println( name + "=" + value + "<br>" );
            test4( value, out );
        }
        // JPL.halt(); this causes Tomcat shutdown
        abolish_facts();
        out.println( "</body></html>" );
        out.close();
    }

    public void doPost (
        HttpServletRequest req,
        HttpServletResponse res
        ) throws ServletException, IOException
    {
        doGet( req, res );
    }

    static void
    load_prog()
    {
        JPL.init();

        Term consult_arg[] = {
            new Atom( "/WinApps/jakarta-tomcat/webapps/proServ2/WEB-INF/classes/test2.pl" )
        };
        Query consult_query =
```

```
        new Query(
            "consult",
            consult_arg );

    boolean consulted = consult_query.query();

    if ( !consulted ){
        System.err.println( "Consult failed" );
        System.exit( 1 );
    }
}

static void
assert_facts() {
    try {
        Class.forName( "org.gjt.mm.mysql.Driver" );
        String url = "jdbc:mysql://ks15e0f00/mysqltest?user=YourID&password=YourPassword";
        Connection con = DriverManager.getConnection( url );

        Statement select = con.createStatement();
        String sql = "select * from child_of";
        ResultSet res = select.executeQuery( sql );

        while( res.next() ) {
            Term child = new Atom( res.getString( "child" ) );
            Term parent = new Atom( res.getString( "parent" ) );
            Term pair = new Compound( "child_of", Util.toTermArray( child, parent ) );

            Query assert_query =
                new Query(
                         "assert",
                         pair );
            assert_query.oneSolution();
        }
        select.close();
        con.close();
    }
    catch ( Exception e ) {
        e.printStackTrace();
    }
}

static void
test4( String person, PrintWriter out )
{
    Variable X = new Variable();
    Term args[] = {
        X,
        new Atom( person )
    };
```

```
        Query query =
            new Query(
                "descendent_of",
                args );

        out.println( "querying descendent_of( X, " + person + " ) <br>" );

        while ( query.hasMoreSolutions() ){
            java.util.Hashtable solution =
                query.nextSolution();
            out.println( "X = " + solution.get( X ) + "<br>" );
        }
    }

    static void abolish_facts() {
        Term pred = new Atom( "child_of" );
        Term num = new Integer( 2 );
        Term rem = new Compound( "/", Util.toTermArray( pred, num ) );

        Query query =
                new Query(
                          "abolish",
                              rem );
        query.oneSolution();
    }
}
```

Imported packages

We import the packages for Servlet, JPL, and SQL.

list 52: Imported packages

```
//
//   for Servlet
//
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
//
//   JPL package
//
import jpl.Atom;
import jpl.Variable;
```

```
import jpl.Term;
import jpl.Query;
import jpl.JPL;
import java.util.Hashtable;
import jpl.Compound;
import jpl.Util;
import jpl.Integer;
//
//   for SQL
//
import java.sql.*;
```

Sending header

We firstly send back the browser the header and the line to be printed out in the top.

list 53: Sending header, etc.

```
public class prologServ2 extends HttpServlet {
    public void doGet (
        HttpServletRequest req,
        HttpServletResponse res
        ) throws ServletException, IOException
    {
        res.setContentType( "text/html" );
        PrintWriter out = res.getWriter();
        out.println( "<html><head><title>Finding decendents of the person</title></head>" );
        out.println( "<body>Using SWI-prolog and MySQL<br>" );
```

Loading test2.pl and asserting the facts

The program, test2.pl, is then loaded, and the facts of child_of relation is imported from RDB to prolog. The methods, load_prog() and assert_facts(), are identical with the ones presented above.

list 54: Loading test2.pl and asserting the facts

```
    load_prog();    // Loading test2.pl
    assert_facts(); // Asserting facts into prolog
```

Displaying the names of parents

In the next step, the program receives the person's name to hand it to the method, test4. The method, test4, will evoke prolog to infer who are parents of the person and display the result.

list 55: Listing up the parents

```
    Enumeration enum = req.getParameterNames();
    while( enum.hasMoreElements() ) {
        String name = (String)enum.nextElement();
        String value = req.getParameter( name );
        out.println( name + "=" + value + "<br>" );
        test4( value, out );
    }
```

Closing up

When finished with displaying the names of parents, the program terminates by abolishing the inserted facts and closing the connection to the browser.

list 56: Closing up

```
    abolish_facts();
    out.println( "</body></html>" );
    out.close();
}
```

Evoking prolog and showing the result

The first half is to construct the query, descendent_of(X, 'ralf'). The last part is to print out each name of parent to the browser by executing the query in prolog.

list 57: test4

```
    static void
    test4( String person, PrintWriter out )
    {
```

```
        Variable X = new Variable();
        Term args[] = {
            X,
            new Atom( person )
        };
        Query query =
            new Query(
                "descendent_of",
                args );

        out.println( "querying descendent_of( X, " + person + " ) <br>" );

        while ( query.hasMoreSolutions() ){
            java.util.Hashtable solution =
                query.nextSolution();
            out.println( "X = " + solution.get( X ) + "<br>" );
        }
    }
```

8.4.2 Building the web application

We build the web application as 'proServ2'. The files are appended in appendix. Do not forget to add the following lines to 'conf/server.xml', before restarting Tomcat.

list 58: In conf/server.xml

```
<Context path="/proServ2" docBase="webapps/proServ2" debug="0" reloadable="true" >
</Context>
```

8.4.3 Runing the proServ2 Servlet

You start Tomcat by executing 'startup.bat' under jakarta-tomcat/bin. Open the page, 'http://127.0.0.1:8080/proServ2/index.html' and enter 'ralf' as before in the place of Name. (figure8.6) Then click 'SUBMIT' button.

You will then see the result displayed on the window as (figure8.7)

Figure 8.6: Opening the input window

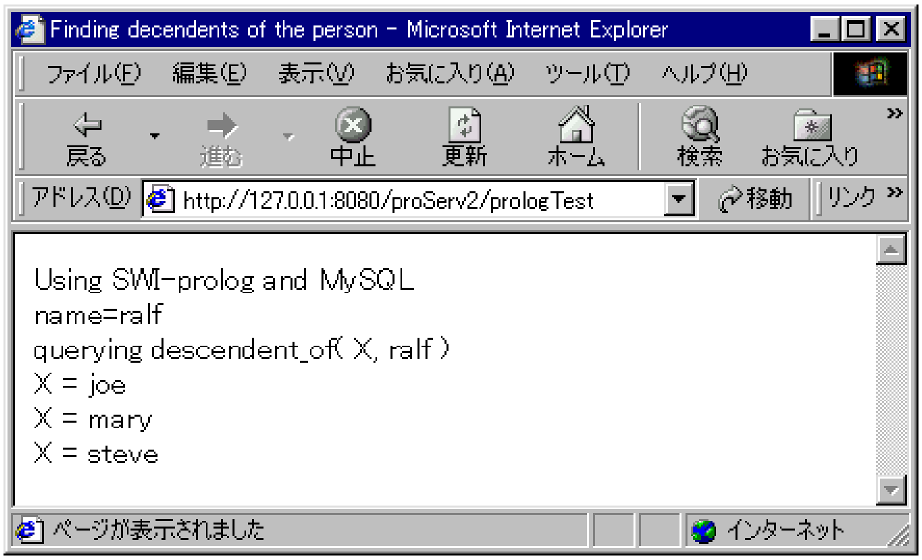

Figure 8.7: Getting the result

Chapter 9

Limitations

9.1 Problems with multi-threads

The Servlet we have implemented for illustrating the use of prolog, MySQL, and Java, can only be safely used by a single user. Let us see first a problem occured by commenting out the line, 'abolish_facts();' in prologServ2.java. The modification hinders prolog to clear up the facts imported from RDB. When you submit your request, you get the same result as above, but if you re-submit the same request to the Servlet again, then you will be embarassed by the odd results (figure9.1)

This abnormality is due to the fact that the Servlet imports the set of facts into prolog **again** when it was evoked second time. The prolog process was running (or available) while Tomcat is alive, serveing you. Each time you send a requet to the Servlet, it imports the facts of child_of relation from RDB to prolog. You must, therefore, clear up the set of facts each time you call the Servlet.

The implication of this malfunction is that allowing another user to call prolog may lead to similar abnormalities. If the second user lets his Servlet to assert a set of facts of child_of relation and he does not clear them up immediately, The same set of facts are called by the Servlet developped by the first user. The point here is that prolog runs only in one process while Tomcat responds to the requests from several users. Sharing the same prolog process among different users may lead to inconsistency to some user.

The best solution would be to spawn out a new process for running prolog each time a user calles the Servlet. The solution is, however, unavailable to us because a servlet can only run multi-threads, on a single process. We have to, therefore, carefully control multi-threads on prolog so that they do not interfere each other by inserting or removing facts.

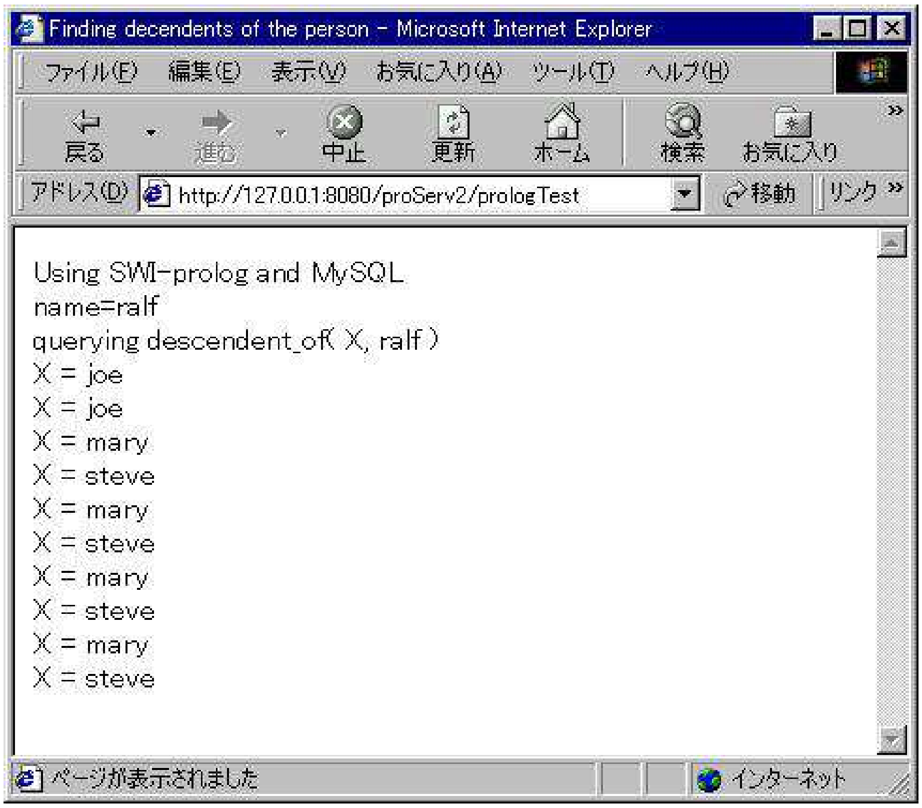

Figure 9.1: An odd result

We can control threads in two levels, one within SWI-prolog and the other within JPL. SWI-prolog enables us to spawn out multi-threads and put control over them. The function is, however, in still pre-alpha state and can only be tested on Unix platforms. Moreover, JPL did not work well with the multi-thread-enabled SWI-prolog when I tested it on Linux.

The only possible way to control multi-threads in prolog is, therefore, to control them in JPL. The methods such as **query()**, **oneSolution()**, and **allSolutions()**, are in fact thread-safe. There is also a method lock(). (See "Multi-threaded Queries" in "High-Level Interface" for more detail.) We conclude that we should do our best by controling multi-threads at the level of JPL. (Needless to say, the inconsistency is unlikely to occur if executing prolog program does not involve updating facts.)

9.2 Managing RDB Connections

It is known that establishing connection with RDB using JDBC takes time. We had better prepare for some connections in the beginning and reuse them for letting users to get access to RDB. Some author claims that sharing connections reduces sometimes the accessing time to one tenth. I believe that we should defenitely employ such a mechanism for real web applications, but we do not need to be so sensitive to the loss as long as only the single developer runs the Servlet on Windows9X.

9.3 Managing prolog loading

In analogy with the case of RDB connections, one may want to manage the loading SWI-prolog. That is, one may want to terminate the prolog process when no thread uses it. We cannot, however, terminate a prolog process as the attempt leads to shutting down Tomcat. We can therefore only hold the process to the end of a Tomcat session once it was evoked by some servlet.

Part III

Integrating applications into XML

Chapter 10

Installing XML tools

[ToDo] We are going to install various XML tools to use the language as the intermediate language between prolog and SQL.

Chapter 11

Communication between client and server

[ToDO] We will discuss how better the user can interact with our server. Topics include serialization, thread, etc.

Chapter 12

Applications

[ToDO] We will cover applications that can be implemented in our environment, which includes a natural language interface to database, cuncurrent processing of speech recognition and generation, etc. (hm.... really?)

Chapter 13

Conclusion

[ToDO] We will discuss how prolog/AI community can contribute to World Wide Web. Mobility?

Appendix A

Program Codes

A.1 proServ

list 59: build.xml

```
<!-- A "project" describes a set of targets that may be requested
     when Ant is executed.  The "default" attribute defines the
     target which is executed if no specific target is requested,
     and the "basedir" attribute defines the current working directory
     from which Ant executes the requested task.  This is normally
     set to the current working directory.
-->


<project name="Calling prolog from Servlet" default="compile" basedir=".">


<!-- Property Definitions

     Each of the following properties are used by convention in this
     build file.  The values specified can be overridden at run time by
     adding a "-Dname=value" argument to the command line that invokes Ant.
     This technique is normally used to copy the values of the ANT_HOME
     and TOMCAT_HOME environment variables into the "ant.home" and
     "tomcat.home" properties, which are normally not defined explicitly.

     app.name          Base name of this application, used to
                       construct filenames and directories.

     deploy.home       The name of the directory into which the
```

```
                         deployment hierarchy will be created.
                         Normally, this will be the name of a
                         subdirectory under $TOMCAT_HOME/webapps.

    dist.home                  The name of the base directory in which
                         distribution files are created.

    dist.src             The name of the distribution JAR file
                         containing the application source code,
                         to be stored in the "dist.home" directory.
                         This filename should end with ".jar".

    dist.war             The name of the Web ARchive (WAR) file
                         containing our deployable application.
                         This filename should end with ".war".

    javadoc.home         The name of the base directory in which
                         the JavaDoc documentation for this application
                         is generated.

    tomcat.home          The name of the base directory in which
                         Tomcat has been installed.  This value is
                         normally set automatically from the value
                         of the TOMCAT_HOME environment variable.

    In the example below, the application being developed will be deployed
    to a subdirectory named "myapp", and will therefore be accessible at:

      http://localhost:8080/myapp
-->

  <property name="app.name"      value="proServ"/>
  <property name="deploy.home"   value="${tomcat.home}/webapps/${app.name}"/>
  <property name="dist.home"     value="${deploy.home}"/>
  <property name="dist.src"      value="${app.name}.jar"/>
  <property name="dist.war"      value="${app.name}.war"/>
  <property name="javadoc.home"  value="${deploy.home}/javadoc"/>


<!-- The "prepare" target is used to construct the deployment home
     directory structure (if necessary), and to copy in static files
     as required.  In the example below, Ant is instructed to create
     the deployment directory, copy the contents of the "web/" source
     hierarchy, and set up the WEB-INF subdirectory appropriately.
-->

  <target name="prepare">
    <mkdir dir="${deploy.home}"/>
    <copydir src="web" dest="${deploy.home}"/>
    <mkdir dir="${deploy.home}/WEB-INF"/>
```

```
    <copyfile src="etc/web.xml" dest="${deploy.home}/WEB-INF/web.xml"/>
    <mkdir dir="${deploy.home}/WEB-INF/classes"/>
    <mkdir dir="${deploy.home}/WEB-INF/lib"/>
    <copydir src="lib" dest="${deploy.home}/lib"/>
    <mkdir dir="${javadoc.home}"/>
  </target>


<!-- The "clean" target removes the deployment home directory structure,
     so that the next time the "compile" target is requested, it will need
     to compile everything from scratch.
-->

  <target name="clean">
    <deltree dir="${deploy.home}"/>
  </target>


<!-- The "compile" target is used to compile (or recompile) the Java classes
     that make up this web application.  The recommended source code directory
     structure makes this very easy because the <javac> task automatically
     works its way down a source code hierarchy and compiles any class that
     has not yet been compiled, or where the source file is newer than the
     class file.  After compilation is complete, any non-Java files (such as
     properties files containing resource bundles) found in the source code
     hierarchy are copied to a corresponding position in the destination
     directory hierarchy.

     Feel free to adjust the compilation option parameters (debug,
     optimize, and deprecation) to suit your requirements.  It is also
     possible to base them on properties, so that you can adjust this
     behavior at runtime.

     The "compile" task depends on the "prepare" task, so the deployment
     home directory structure will be created if needed the first time.
-->

  <target name="compile" depends="prepare">
    <javac srcdir="src" destdir="${deploy.home}/WEB-INF/classes"
           classpath="${deploy.home}/WEB-INF/classes"
           debug="on" optimize="off" deprecation="off"/>
  </target>



<!-- The "javadoc" target is used to create the Javadoc API documentation
     for the Java classes in this web application.  It is assumed that
     this documentation is included in the deployed application, so the
     example below generates the Javadoc HTML files in a subdirectory under
     the deployment home directory.
```

```
-->

  <target name="javadoc" depends="prepare">
    <!-- TODO -->
  </target>


<!-- The "all" target rebuilds everything by executing the "clean"
     target first, which forces the "compile" target to compile all
     source code instead of just the files that have been changed.
-->

  <target name="all" depends="clean,prepare,compile,javadoc"/>


<!-- The "dist" target builds the distribution Web ARchive (WAR) file
     for this application, suitable for distribution to sites that wish
     to install your application.  It also creates a JAR file containing
     the source code for this application, if you wish to distribute
     that separately.
-->

  <target name="dist" depends="prepare,compile">
    <jar jarfile="${dist.home}/${dist.src}"
         basedir="."/>
    <jar jarfile="${dist.home}/${dist.war}"
         basedir="${deploy.home}"/>
  </target>


</project>
```

list 60: etc/web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">

<web-app>
    <servlet>
        <servlet-name>
            prologServlet
        </servlet-name>
        <servlet-class>
            prologServ
```

```
        </servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>
            prologServlet
        </servlet-name>
        <url-pattern>
            /prologTest
        </url-pattern>
    </servlet-mapping>

</web-app>
```

list 61: src/prologServ.java

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
//
//   JPL package
//
import jpl.Atom;
import jpl.Variable;
import jpl.Term;
import jpl.Query;
import jpl.JPL;

public class prologServ extends HttpServlet {
        public void doGet (
                HttpServletRequest req,
                HttpServletResponse res
                ) throws ServletException, IOException
        {
                res.setContentType( "text/html" );
                PrintWriter out = res.getWriter();
                out.println( "<html><head><title>Finding decendents of the person</title></head>" );
                out.println( "<body>Using getParameter() method<br>" );

                JPL.init(); // We initialize JPL
                Term consult_arg[] = {
                        //
                        // We have to specify the location of prolog program in full
                        //
                        new Atom( "/WinApps/jakarta-tomcat/webapps/proServ/WEB-INF/classes/test.pl" )
                };
```

```
        //
        // We consult 'test.pl'
        //
        Query consult_query =
                new Query(
                        "consult",
                        consult_arg );

        boolean consulted = consult_query.query();

        if ( !consulted ){
                out.println( "Consult failed" );
                System.exit( 1 );
        }

        Enumeration enum = req.getParameterNames();
        while( enum.hasMoreElements() ) {
                String name = (String)enum.nextElement();
                String value = req.getParameter( name );
                out.println( name + "=" + value + "<br>" );
                //
                //   We pass the task to test3 below
                //
                String descendent = test3( value ).name();
                out.println( "descendent = " +  descendent + "<br>" );
        }
        out.println( "</body></html>" );
        out.close();
}

//
//  This is test3 method.
//  The type of the return value is Atom.
//
static Atom
test3( String person )
{
        //
        // We construct the query, descendent_of( X, Person )
        // where Person is instantiated already when called
        Variable X = new Variable();
        Term args[] = {
                X,
                new Atom( person )
        };
        Query query =
                new Query(
                        "descendent_of",
                        args );
```

```
                //
                // Obtain the first solution to return
                //
                java.util.Hashtable solution =        query.oneSolution();
                return (Atom)solution.get( X );
        }

        public void doPost (
                HttpServletRequest req,
                HttpServletResponse res
                ) throws ServletException, IOException
        {
                doGet( req, res );
        }
}
```

list 62: web/index.html

```
<html>
<head>
<meta HTTP-EQUIV="Content-Type" CONTENT="text/html"">
<title>Calling prolog from Java</title>
</head>
<body>
<form action="/proServ/prologTest" method="POST">
Name: <input type="text" name="name">
<input type=submit value="SUBMIT">
</form>
</body>
</html>
```

list 63: part of server.xml

```
<Context path="/proServ" docBase="webapps/proServ" debug="0" reloadable="true" >
</Context>
```

A.2 proServ2

list 64: build.xml

```
<!-- A "project" describes a set of targets that may be requested
     when Ant is executed.  The "default" attribute defines the
     target which is executed if no specific target is requested,
     and the "basedir" attribute defines the current working directory
     from which Ant executes the requested task.  This is normally
     set to the current working directory.
-->


<project name="Calling prolog and RDB from Servlet" default="compile" basedir=".">


<!-- Property Definitions

     Each of the following properties are used by convention in this
     build file.  The values specified can be overridden at run time by
     adding a "-Dname=value" argument to the command line that invokes Ant.
     This technique is normally used to copy the values of the ANT_HOME
     and TOMCAT_HOME environment variables into the "ant.home" and
     "tomcat.home" properties, which are normally not defined explicitly.

     app.name          Base name of this application, used to
                       construct filenames and directories.

     deploy.home       The name of the directory into which the
                       deployment hierarchy will be created.
                       Normally, this will be the name of a
                       subdirectory under $TOMCAT_HOME/webapps.

     dist.home               The name of the base directory in which
                       distribution files are created.

     dist.src          The name of the distribution JAR file
                       containing the application source code,
                       to be stored in the "dist.home" directory.
                       This filename should end with ".jar".

     dist.war          The name of the Web ARchive (WAR) file
                       containing our deployable application.
                       This filename should end with ".war".

     javadoc.home      The name of the base directory in which
                       the JavaDoc documentation for this application
                       is generated.

     tomcat.home       The name of the base directory in which
                       Tomcat has been installed.  This value is
                       normally set automatically from the value
                       of the TOMCAT_HOME environment variable.
```

```
     In the example below, the application being developed will be deployed
     to a subdirectory named "myapp", and will therefore be accessible at:

       http://localhost:8080/myapp
-->

  <property name="app.name"       value="proServ2"/>
  <property name="deploy.home"    value="${tomcat.home}/webapps/${app.name}"/>
  <property name="dist.home"      value="${deploy.home}"/>
  <property name="dist.src"       value="${app.name}.jar"/>
  <property name="dist.war"       value="${app.name}.war"/>
  <property name="javadoc.home"   value="${deploy.home}/javadoc"/>


<!-- The "prepare" target is used to construct the deployment home
     directory structure (if necessary), and to copy in static files
     as required.  In the example below, Ant is instructed to create
     the deployment directory, copy the contents of the "web/" source
     hierarchy, and set up the WEB-INF subdirectory appropriately.
-->

  <target name="prepare">
    <mkdir dir="${deploy.home}"/>
    <copydir src="web" dest="${deploy.home}"/>
    <mkdir dir="${deploy.home}/WEB-INF"/>
    <copyfile src="etc/web.xml" dest="${deploy.home}/WEB-INF/web.xml"/>
    <mkdir dir="${deploy.home}/WEB-INF/classes"/>
    <mkdir dir="${deploy.home}/WEB-INF/lib"/>
    <copydir src="lib" dest="${deploy.home}/lib"/>
    <mkdir dir="${javadoc.home}"/>
  </target>


<!-- The "clean" target removes the deployment home directory structure,
     so that the next time the "compile" target is requested, it will need
     to compile everything from scratch.
-->

  <target name="clean">
    <deltree dir="${deploy.home}"/>
  </target>


<!-- The "compile" target is used to compile (or recompile) the Java classes
     that make up this web application.  The recommended source code directory
     structure makes this very easy because the <javac> task automatically
     works its way down a source code hierarchy and compiles any class that
     has not yet been compiled, or where the source file is newer than the
     class file.  After compilation is complete, any non-Java files (such as
     properties files containing resource bundles) found in the source code
```

```
     hierarchy are copied to a corresponding position in the destination
     directory hierarchy.

     Feel free to adjust the compilation option parameters (debug,
     optimize, and deprecation) to suit your requirements.  It is also
     possible to base them on properties, so that you can adjust this
     behavior at runtime.

     The "compile" task depends on the "prepare" task, so the deployment
     home directory structure will be created if needed the first time.
-->

  <target name="compile" depends="prepare">
    <javac srcdir="src" destdir="${deploy.home}/WEB-INF/classes"
           classpath="${deploy.home}/WEB-INF/classes"
           debug="on" optimize="off" deprecation="off"/>
  </target>



<!-- The "javadoc" target is used to create the Javadoc API documentation
     for the Java classes in this web application.  It is assumed that
     this documentation is included in the deployed application, so the
     example below generates the Javadoc HTML files in a subdirectory under
     the deployment home directory.
-->

  <target name="javadoc" depends="prepare">
    <!-- TODO -->
  </target>


<!-- The "all" target rebuilds everything by executing the "clean"
     target first, which forces the "compile" target to compile all
     source code instead of just the files that have been changed.
-->

  <target name="all" depends="clean,prepare,compile,javadoc"/>


<!-- The "dist" target builds the distribution Web ARchive (WAR) file
     for this application, suitable for distribution to sites that wish
     to install your application.  It also creates a JAR file containing
     the source code for this application, if you wish to distribute
     that separately.
-->

  <target name="dist" depends="prepare,compile">
    <jar jarfile="${dist.home}/${dist.src}"
         basedir="."/>
```

```
    <jar jarfile="${dist.home}/${dist.war}"
         basedir="${deploy.home}"/>
  </target>


</project>
```

list 65: etc/web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">

<web-app>
    <servlet>
        <servlet-name>
            prologServlet
        </servlet-name>
        <servlet-class>
            prologServ2
        </servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>
            prologServlet
        </servlet-name>
        <url-pattern>
            /prologTest
        </url-pattern>
    </servlet-mapping>

</web-app>
```

list 66: src/prologServ2.java

```
//
//   for Servlet
//
import java.io.*;
import java.util.*;
```

```
import javax.servlet.*;
import javax.servlet.http.*;
//
//   JPL package
//
import jpl.Atom;
import jpl.Variable;
import jpl.Term;
import jpl.Query;
import jpl.JPL;
import java.util.Hashtable;
import jpl.Compound;
import jpl.Util;
import jpl.Integer;
//
//   for SQL
//
import java.sql.*;

public class prologServ2 extends HttpServlet {
    public void doGet (
        HttpServletRequest req,
        HttpServletResponse res
        ) throws ServletException, IOException
    {
        res.setContentType( "text/html" );
        PrintWriter out = res.getWriter();
        out.println( "<html><head><title>Finding decendents of the person</title></head>" );
        out.println( "<body>Using SWI-prolog and MySQL<br>" );

        load_prog();    // Loading test2.pl
        assert_facts(); // Asserting facts into prolog

        Enumeration enum = req.getParameterNames();
        while( enum.hasMoreElements() ) {
            String name = (String)enum.nextElement();
            String value = req.getParameter( name );
            out.println( name + "=" + value + "<br>" );
            test4( value, out );
        }
        // JPL.halt(); this causes Tomcat shutdown
        abolish_facts();
        out.println( "</body></html>" );
        out.close();
    }

    public void doPost (
        HttpServletRequest req,
        HttpServletResponse res
        ) throws ServletException, IOException
```

```
{
    doGet( req, res );
}

static void
load_prog()
{
    JPL.init();

    Term consult_arg[] = {
        new Atom( "/WinApps/jakarta-tomcat/webapps/proServ2/WEB-INF/classes/test2.pl" )
    };
    Query consult_query =
        new Query(
            "consult",
            consult_arg );

    boolean consulted = consult_query.query();

    if ( !consulted ){
        System.err.println( "Consult failed" );
        System.exit( 1 );
    }
}

static void
assert_facts() {
    try {
        Class.forName( "org.gjt.mm.mysql.Driver" );
        String url = "jdbc:mysql://ks15e0f00/mysqltest?user=YourID&password=YourPassword";
        Connection con = DriverManager.getConnection( url );

        Statement select = con.createStatement();
        String sql = "select * from child_of";
        ResultSet res = select.executeQuery( sql );

        while( res.next() ) {
            Term child = new Atom( res.getString( "child" ) );
            Term parent = new Atom( res.getString( "parent" ) );
            Term pair = new Compound( "child_of", Util.toTermArray( child, parent ) );

            Query assert_query =
                new Query(
                          "assert",
                          pair );
            assert_query.oneSolution();
        }
        select.close();
        con.close();
    }
```

```
        catch ( Exception e ) {
            e.printStackTrace();
        }
    }

    static void
    test4( String person, PrintWriter out )
    {
        Variable X = new Variable();
        Term args[] = {
            X,
            new Atom( person )
        };
        Query query =
            new Query(
                "descendent_of",
                args );

        out.println( "querying descendent_of( X, " + person + " ) <br>" );

        while ( query.hasMoreSolutions() ){
            java.util.Hashtable solution =
                query.nextSolution();
            out.println( "X = " + solution.get( X ) + "<br>" );
        }
    }

    static void abolish_facts() {
        Term pred = new Atom( "child_of" );
        Term num = new Integer( 2 );
        Term rem = new Compound( "/", Util.toTermArray( pred, num ) );

        Query query =
                new Query(
                          "abolish",
                              rem );
        query.oneSolution();
    }
}
```

list 67: web/index.html

```
<html>
<head>
<meta HTTP-EQUIV="Content-Type" CONTENT="text/html"">
<title>Calling prolog from Java</title>
</head>
```

```
<body>
<form action="/proServ2/prologTest" method="POST">
Name: <input type="text" name="name">
<input type=submit value="SUBMIT">
</form>
</body>
</html>
```

Appendix B

On Linux

[ToDO] We show how more stable system can be built on Linux. The content present in this guide mostly concerns the programming environment where programmers can rapidly develop prototypes. This appendix should show how to develop prototypes into more stable, reliable systems, using only open source software.

B.1 Building your web server on Linux

First of all, happily, you do not need to build a Unix-like environment as you have to do on Windows9X because Linux is already a UNIX. Installation of Java, prolog, and JPL, is not as complicated as you have to manage on Windows.

B.1.1 Java Development Kit

Sun provides us with JDK-1.2.2 for Solaris, which works fine. You can alternatively try JDK-1.3 distributed by IBM.[1] I found no problem in installing and using JDK-1.2.2 provided by Sun. You should, however, not forget to include lib/tools.jar in your classpath.

B.1.2 Installing SWI-prolog

There is not any problem in installing SWI-prolog on Linux. Do not forget to copy (or move) runtime/libpl.a to /usr/lib.

[1] I have installed it on 17th July, 2000, and found it work okay. It looks faster, too, but I had to recompile the samples presented in this guide.

You can create a SWI_prolog capable of handling multi-threads. The version does, however, not work with JPL, a Java Prolog interface. So do not be bothered with it.

B.1.3 Installing JPL

You follow the same instruction as given in the content of this guide. You should, however, use jpl/src/Makefile to compile libjpl.so. Copy libjpl.so which you have created under jpl/lib to some directory, e.g., /usr/lib, and add it to the variable, LD_LOAD_PATH, so that the shared library object can be evoked on the spot.

B.1.4 Web servers running Servlet

You only need to install Tomcat to run servlets, but you may want to add it to your Apache Web Server. The instruction is given in the reference manual of Tomcat under jakarta-tomcat/doc/uguide/tomcat_ug.html. In short, you first run Tomcat and terminate it, then. There must be a file, jakarta-tomcat/conf/tomcat-apache.conf. You should then edit httpd.conf of Apache to include the file by adding a few lines.

Some other minor points to note are:

1. Some other program may occupy the port 8080. If it is the case, you have to either remove the service occupying 8080 or force Tomcat to use some other port. Edit /etc/services if you choose the former.
2. You have to run tomcat as root (if you have installed it on somewhere like /usr/local.
3. Some Linux distributions come with an Apache server pre-installed. Replace the latest apache for the pre-installed one. Do not forget in that case to force Linux to read the httpd.conf you have installed on it. The setting can be changed using linuxconf. (Again you have to be root.)
4. If you choose to install Tomcat as an add-on to apache, you have to enable your apache server to dynamically load objects. (The apache server must be DSO capable). Follow the instruction given in Apache install guide to add DSO module to your apache server. Do not foget to compile mod_so by yourself. You go down to src, edit Configuration to uncomment 'AddModule mod_so', do 'make', and go up to apache HOME to 'make install'.
5. You need mod_jserv.so to enable your apache server to invoke Tomcat on demand. You can find a pre-compiled shared libary object in a page to

download Tomcata binaries. Follow the link, linux. Download and copy it under apache/libexec.

6. If you run Tomcat from Apache, you have to first run Tomcat, and then Apache because Apache reads in jakarta-tomcat/conf/tomcat-apache.conf when it starts up.

7. The content of web.xml is not read by Apache when it deligates a job to Tomcat. Your servlets can, therefore, not be evoked using aliases as specified in web.xml. You may evoke Servlets from Apache if you include the items of information into httpd.conf, but it may not be worth trying. We had better wait for future releases of Tomcat as it is likely to be integrated into Apache in the long run and we expect that servlets will be easier to use, then.

B.1.5 Creating Web applications

The shell script did not work on my environment. If you encounter the same problem, try the following:

```
java -classpath $CLASSPATH org.apache.tools.ant.Main -Dtomcat.home=$TOMCAT_HOME
```

You have to include ant.jar, xml.jar, jasper.jar, servlet.jar, and webserver.jar to your classpath, though. (They are all found under $TOMCAT_HOME/lib)

B.2 Extending your web server

There is nothing special to Linux. You can just do the same thing as you are on Windows.