The Aglets 2.0.2 User's Manual

Aglets Development Group

March 12, 2009

Contents

Introduction

This document provides a guideline for those, either developers or users, who want to deal with the Aglets mobile agent platform.

Originally developed at the IBM Tokyo Research Laboratory, Aglets was appreciated for its clear and easy to use API, good modularity and design. Since the initial effort of IBM, several versions of Aglets have been released, until the project become officially open source and hosted at SourceForge. This was the time when the release 2 become the official release (releases 1.x were on charge of IBM). In 2004, when the latest release was 2.0.2, the project administrator changed, and new commits were done on the CVS towards the 2.1 release, that is still not available as an installable package.

Since a while there was not official documentation about the Aglets Software Development Kit, and this is the main reason why this document has been created. Moreover, it must be took into account that due to the first development of the IBM laboratories, there is on the Internet a bit of confusion about the platform versions and documentation. This manual wants to substitute and concentrate most of the documentation, being "the entry point" for everyone who wants to start using Aglets. Of course, as the platform itself, this manual is always under construction, thus you should check periodically for newer versions.

Since describing Aglets is not a trivial task, due to its complexity, this document starts looking at what really lacks in the available documentation: how to install the server, how to write sample (and simple) aglets, how to manage server configuration etc. We hope this manual will, one day, substitute all other documents available. It has not yet reached that level, in fact you will not find here information about concepts like proxies, messages, etc. This means that, before you start developing agents, you have to consult the Aglets Working Draft (left unfinished) at

`http://www.research.ibm.com/trl/aglets/spec10.htm`

just to learn basic concepts about activation/deactivation, message passing, etc. It is expected that this manual will include all the relevant Aglets information/documentation, including and uptating the concepts explained in the Aglets Working Draft referred to above.

On-line Resources

There are not a lot of documentation and examples available on-line, and furthermore they are often quite old. It is for this reason that comes this manual. However, if you need further help, you should take a look at the Aglets web site: *http://aglets.sourceforge.net*. There you can find information and documentation about the Aglets project, and of course about the Aglets mailing lists. So far, Aglets provides four mailing lists:

- aglets-users:
  around usage of the ASDK (installing and running the server, developing aglets);
- aglets-developers:
  around development of the ASDK (proposing new features, coordinating the launch of new releases);
- aglets-commit:
  informs about the CVS commits and new available releases;
- agletsnet-users *deprecated*:
  a mailing list related to the *aglets-net* project, a collection of Aglets-based applications.

Actually, mailing lists are the fastest and easiest way to get help about Aglets.

About this document

The first version of this document has been written by Luca Ferrari using LaTeX. Raimondas Berniunas improved the first version correcting some mistakes. Thomas Herlea also helped with the manual and re-organized the CVS structure of both documentation and source code. Fernando G. Tinetti helps improving the manual since Nov 2008.

This document is under the Aglets CVS repository, and the latest version of the document can be obtained via anonymous checkout as follows:

```
export CVSROOT=:pserver:aglets.cvs.sourceforge.net:/cvsroot/aglets
cvs co docs
```

The document will be in the `docs` subdirectory and is named `manual.tex`; in order to build a `dvi` version you have to run LaTeXtwice, and then optionally to run a dvi-postscript converter as follows:

```
latex manual.tex
latex manual.tex
dvi2ps manual.dvi manual.ps
```

People who want to collaborate on this document (fixing mistakes, adding images or code samples, etc.) can contact *cat4hire@users.sourceforge.net* or can post a message to the *aglets-developers* mailing list.

Chapter 1

Mobile Agents: Introduction and Concepts

Agents are autonomous and social entities used to develop complex applications. The idea behind the concept of "agent" is the one of an active and autonomous module that can work cooperating and/or competing with other modules/agents and the sorrounding environment. In other words, an agent is a piece of executable code that, during its life, executes one or more task coordinating with other entities (e.g., other agents and the surrounding environment). Therefore, at first look, agents could be thought as a set of processes, each running alone; however, agents have some properties different from a "simple" process. In particular agents are *social*, that means they are prone to communicate with other agents (e.g., by mean of messages). Of course, even processes can cooperate each other, but the idea is that communication among agents is simpler and more natural.

Agents cannot live by theirselves, but must rely on a specific environment, called *Agent Platform* (AP) that is in charge of providing agents a set of resources and of controlling the agents life cycle. The Agent Platform acts therefore as a framework, where agents are the component that can be installed and run on top of it. Thanks to this architecture, agents result smaller and lighter if compared to normal processes. The Agent Platform controls the agent lifecycle, deciding when an agent must be started and stopped (these usually depends on a user request), must be destroyed, messages must be delivered to, and so on. Moreover an Agent Platform is in charge of keeping the state of each agent, as well as providing an unique way to identify each agent. Of course, the AP provides a set of tools to allow users and developers to deal with all the platform capabilities. Last but not least, the Agent Platform usually represents a "front-end" for agents that want to gain host resources (e.g., files); this means that generally speaking the Agent Platform controls the resources accessed by agents.

1.1 Mobile Agents

A special kind of agents are those called *mobile*, a type of agents that can move spountanously between two or more different Agent Platforms. Mobility is a very important feature, since enhance agents allowing them to migrate across the hosts of a network. Thanks to mobility, the agent (i.e., the running process) can move towards the data instead of migrating the data near to the process or requesting remotely the data. Since agents are often smaller than the data they are going to work on, mobility can lead to a bandwidth saving. There are other situations where mobility is fundamental: for instance agents can migrate from a corrupted or halting host to another one in order to continue their computations, thus surviving to host problems.

In order to support mobility, the Agent Platform must provide a set of dedicated services in order to support the migration in both directions (in and out the platform), as well as remote messaging between local agents and migrated (i.e., remote) ones. A platform that supports mobile agents is called *Mobile Agent Platform* (MAP).

Aglets is a **Mobile Agent Platform**, and an installation is usually composed of the following indipendent parts:

- *Aglet Mobile Agent Platform* (Aglets MAP): is the core platform, able to manage mobile agents.
- *Tahiti*: is the main server in charge of managing the mobility of agents. It comes with a Graphical User Interface (GUI) that helps administrators taking care of running agents.
- *Aglets Software Development Kit* (Aglets SDK - ASDK): is a library that provides developers all the facilities required to write mobile agents compliant to the Aglets MAP.

If you want to host agents on your machine, you need to run the MAP, that acts as a daemon waiting for new agents to be created. Agents can be created locally or can be received from a remote MAP; in the latter case there is not a real "agent creation", since the agent already exists somewhere (in the remote MAP) and is just transferred to the local MAP.

If you are interested in the development of mobile agents, you need only the ASDK, even if it is strongly suggested you install also the MAP in order to test your agents.

1.1.1 Strong Mobility vs. Weak Mobility

There are two main kinds of mobility to take into account: *strong* and *weak*. The latter is the simple one, and requires that only executable code migrates, without any particular information about the execution of the code

itself. This means that it is like you send a program (e.g., an agent) to a remote host, so that such agent is *re-started* from the beginning. Strong mobility, instead, requires that also the information about the execution state is migrated along with the code. Having such information, it is possible to continue (not re-start) the process on the remote destination exactly from the same execution point.

To better understand the above concepts, consider the following simple meta-code example:

```
for( int i = 0; i < 10; i++){
    migrate( remoteHosts[i] );
}
```

While the above code is running, the program (or the agent) executes the `for` cycle, and at the very first step a migration is required. Once on the remote destination, if the mobility adopted is strong, the code will continue within the `for` loop, so executing the second step migration. In the case of weak mobility, instead, the program will start from an *entry-point* defined within the code, for instance a standard method, and will probably re-execute the same cycle over and over.

Aglets supports only the weak mobility, and this is not a proper limitation of the MAP implementation, rather a limitation of the Java architecture. In fact, almost all the Java MAP support only the weak mobility. The reason behind this is that there is no support in Java to get enough information about the execution flow (i.e., stack trace, program counter, registers, and so on), so there is no way to restore a computation from the same point after a serialization. As a proof of this, the `Thread` class itself is not serializable, and so cannot be transmitted over the network.

In order to partially overtake the above limitation, Aglets ensures that an agent will keep its Java state (i.e., the values of its inner variables) among migrations (at least if the Java state is serializable). Moreover, when the agent is going to be executed on the remote machine, the execution will start from a well defined entry point (a method), and no re-initialization will happen (in order to not override the current agent state).

1.2 Agents and Proxies

In the Aglets MAP an agent is simply an object, that is the instance of a class, on which a thread executes on. This approach is very similar to the one of the *applets* or *servlets*: an agent is simply a component on which a thread performs some methods in order to make it active.

The Aglets SDK provides the base class for making agents, that is the `Aglet` class, that must be extendend in order to provide a specific agent

behaviour. Overriding methods of the base class and defining different behaviours when events and messages come, the subclass defines the behaviour the agent will have. Please note that, as opposite to standard Java object management, an agent creation is managed by the platform, so there is not need explicitly call a constructor; similarly the `finalize` method call will be managed by the platform itself.

An agent should not make any assumption on the thread that is assigned to it, since the platform, for efficienty reasons, could pool threads and dynamically assign them to the execution of an agent's method. This also means that on the same agent could run, at different times, different threads. However, the platform ensures that only one thread at a specific moment will be active on an agent, so there is no need for synchronization.

Agents cannot directly reference other agents, so it is not possible for an agent to handle a "pointer" to another agent. This is done for security reasons: having a "pointer" to another agent allows a malicious agent to perform method calls on the referencee and, therefore, to induct specific behaviours that are not under the control of the platform. However, as stated before, agents are social entities, and thus they must be able to communicate each other. In order to allow agents communicating, keeping a good level of security, the Aglets platform exploits the *proxy* pattern. A proxy is an object that masquerades another object (its owner) and that forwarders method calls to its owner. So, in the Aglets platform, agents are hidden by proxies, and other agents can send messages or perform method calls against the proxy, having the proxy to forward such method calls on the owner agent. Advantages of this technique are that the agent are always protected, being invisible, and that the system can create and manage a lot of proxies for the same agent without breaking the agent protection. Moreover, the disposing of a proxy does not causes the disposing of the agent it is masquerading. In fact, the Aglets MAP is the only one component that keeps a reference to agents, avoiding thus that the Java garbage collector collects agents if other agents are not communicating (i.e., handling any kind of refence) to them. Summarizing, it is important to note that agents are created and managed by the MAP, while proxies represents "handles" to other agents, thanks to which agents can communicate each other.

Chapter 2

Installation

This chapter describes how to install and run for the first time the Aglets 2 platform. Please consider that Aglets 2 is shipped with both the ASDK (Aglets Software Development Kit) and the run-time environment. The former is the Aglets library, that allows developers to compile Aglets-based applications; the latter is a set of pre-built agents and programs used to implement a stand-alone platform, thanks to which you can execute and dispatch agents on your machine.

To run the Aglets platform you need at least a Java 2 Run-time Environment (JRE), even if it is recommended to install the full Java 2 Source Development Kit (J2SDK), which allows you to compile agents. This chapter does not cover how to get and install the JRE or the J2SDK; for information about Java see the SUN web site: *http://java.sun.com*. Aglets can be installed on a Unix/Linux system, Microsoft Windows and Mac OS X. More generally, each architecture able to run the Java 2 platform is a possible target on which install Aglets.

The following paragraphs show how to install Aglets from the three available forms: binaries, CVS, sources. In the following, the installation under Unix will be shown, even if the steps are the same for all other supported platforms. It is assumed that you have all required Java commands in your PATH, thus they can be executed starting from their short name. Please read the section on binaries whichever form you choose to install, since it is the more detailed one and describes the directory structure, execution of common commands and so on, which are also used in the other two sections.

2.1 Installing binaries

This is the recommended way, since compiled packages contain stable versions of the platform and of the library (ASDK). If you are not a developer, you should install Aglets starting from compiled packages. Both the library and the platform are shipped within a single file, a jar archive (Java

ARchive), with a name that reflects the version of Aglets it contains. In the following we will refer to the version 2.0.2 of Aglets (the latest stable at the moment of writing), whose archive file is:

```
aglets-2.0.2.jar
```

The following steps detail how to install Aglets starting from the above archive.

1. Decompress the archive:
   Since Aglets comes as compressed archive, you need first to extract the files from it. The files are not grouped into a single folder in the archive, thus it is better to create a container directory for your aglet installation (for example */java/aglets*). Copy the archive file into it, change directory to it and execute the *jar* command like this:

   ```
   jar xvf aglets-2.0.2.jar
   ```

   During the decompression you will see a few of lines scrolling on the screen, indicating what is being extracted:

   ```
   luca@linux:/java/aglets> jar xvf aglets-2.0.2.jar
     created: META-INF/
   extracted: META-INF/MANIFEST.MF
     created: bin/
   extracted: bin/agletsd.bat.in
   extracted: bin/agletsd.in
   extracted: bin/ant
   extracted: bin/ant.bat
   extracted: bin/build.xml
   extracted: bin/daemoncontrol.bat.in
   extracted: bin/daemoncontrol.in
   extracted: bin/lcp.bat
     created: cnf/
   extracted: cnf/aglets.props
   extracted: cnf/agletslog.xml
   extracted: INSTALL.html
     created: lib/
   extracted: lib/jaxp.jar
   extracted: lib/tahiti.properties
   extracted: lib/log4j.jar
   extracted: lib/parser.jar
   extracted: lib/aglets-2.0.2.jar
   extracted: lib/ant.jar
   extracted: lib/crimson.jar
   ...
   ```

Once you have extracted the archive, you should see a set of subdirectories as follows:

- *bin* contains executable programs for the Aglets 2 platform, such as the daemon in charge of receiving incoming agents. Furthermore it contains files required by further installation steps;
- *cnf* contains configuration files for the Aglets platform;
- *public* contains a few examples of agents, and should be your root directory as base of your own agents;
- *lib* contains the Aglets 2 library (as a jar archive) and other libraries required by the Aglets technology.

2. Install the platform:
To install the platform you need to run Apache Ant, a tool expressely made to compile and install Java applications. Aglets 2 is shipped with a version of Ant that is suitable to install the platform, nevertheless it is possible to use another version of Ant (a version greater than 1.5 is recommended). Check the Ant project web site at the Apache Foundation site *http://www.apache.org* to get more information about Ant.

   To install Aglets with the shipped Ant, you need first to enter the *bin* directory, where the Ant buildfile `build.xml` is present, and then run *ant* like this:

```
luca@linux:/java/aglets> cd bin/
luca@linux:/java/aglets/bin> chmod 755 ant
luca@linux:/java/aglets/bin> ./ant
Buildfile: build.xml
...
BUILD SUCCESSFUL

Total time: 8 seconds
```

   During the library build/installation you will see messages coming from the Aglets maintainer about the current version; please read them since they might contain information not yet reported in this manual.

3. Set up policy:
Like other Java applications, the Aglets server requires entries in the Java policy file (usually `~/.java.policy`) to open sockets, execute agents, access local files and so on. You can copy entries directly from the file `bin/.java.policy` (of the Aglets installation) in your `~/.java.policy` or you can ask Ant to do it for you. This is the recommended way, since it can change depending on administrators wills

and since is a more transparent and standard way. Furthermore, Ant will install a base keystore for you. Aglets requires a keystore in order to contain keys for secure agent migrations; usually the keystore is contained in the `~/.keystore` file.

To install both the policy entries and the keystore in your home directory, launch `ant` specifying the `install-home` option:

```
luca@linux:/java/aglets/bin> ant install-home
Buildfile: build.xml

install-home:
     [echo] Copying .java.policy file...
     [copy] Copying 1 file to /home/luca
     [echo] Copying .keystore file...
     [copy] Copying 1 file to /home/luca

BUILD SUCCESSFUL
Total time: 1 second
```

Security note:
Please consider that both the policy entries and the keystore file are meant to allow Aglet users to quickly and easily start using the platform; you should strengthen the security before running the Aglets platform in a production environment.

4. Set up environment variables:
In order to get the Aglets platform running, you should set the following environment variables to the installation directory of Aglets: `AGLETS_HOME` and `AGLETS_PATH`. Furthermore, to run the Aglets platform in a more comfortable way, add the `bin` directory of the Aglets installation to your `PATH`. If you are running a Unix-Linux system with Bash, you can do the following:

```
export AGLETS_HOME=/java/aglets
export AGLETS_PATH=$AGLETS_HOME
export PATH=$PATH:$AGLETS_HOME/bin
```

while in a Microsoft Windows system you can do:

```
set AGLETS_HOME=c:\java\aglets
set AGLETS_PATH=%AGLETS_HOME%
set PATH=%PATH%;%AGLETS_HOME%\bin
```

or you can configure environment variables from the control panel.

5. Run the Aglets server:
   Once you have installed the Aglets platform and the keystore, you can run the default Aglets server, which is called Tahiti. Tahiti can be executed through the command `agletsd`, that starts the Aglets server. If authentication does not matter, it's worth launching the server with the properties file supplied with the binaries:

   ```
   luca@linux:/java/aglets/bin> agletsd -f ../cnf/aglets.props
   ```

   Without the properties file Tahiti will ask the user to authenticate at every startup and server reboot (see figure 2.1). If the user has installed the default keystore the username can be either *anonymous*

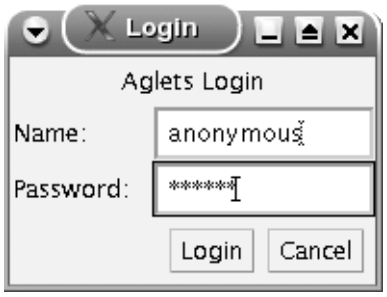

Figure 2.1: Tahiti login window.

   or *aglet_key* and the password is *aglets*. If a keystore with custom keys is used, the user name is the alias of a key and the password is the password of the keystore. Once the user has logged in, the Tahiti main window is displayed (see figure 2.2). Within this window the user can

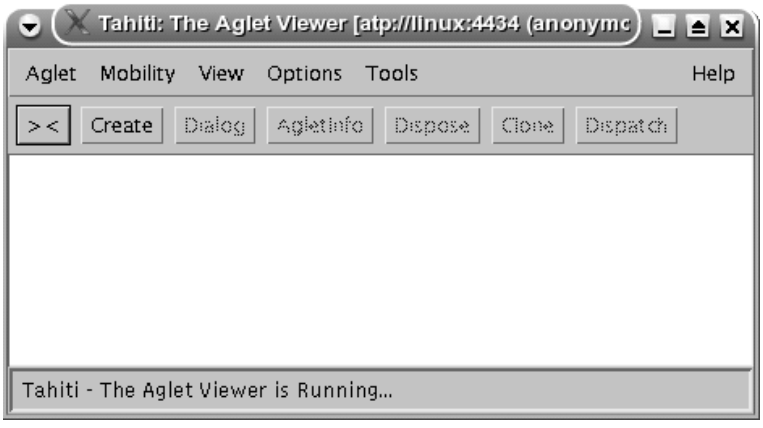

Figure 2.2: Tahiti main window.

   manage the server, create, and dispose agents, get server information and so on.

Do not worry too much if, at start time, the `agletsd` command shows a few warnings like the following:

```
AgletRuntime is requested to get unknown user's certificate
Signature of shared secret is incorrect.
secret is null.
[Warning: The hostname seems not having domain name.
 Please try -resolve option to resolve the
 fully qualified hostname
 or use -domain option to
 manually specify the domain name.]
```

These are warnings related to your network connection, and you will see how to fix them in further chapters.

Please note that the Tahiti main window gives information about the server; it suffices to look at the main window title to know which port the server is listening on (by default 4434) and within who is running (the username).

To stop the server, simply click on either the close button in the window title bar (usually an 'x') or select *Exit* from the *Aglet* menu. In both cases, Tahiti will ask you for a confirmation (see figure 2.3); clicking on 'OK' will shutdown the Aglets server (killing all running agents and

Figure 2.3: Confirmation required for server shutdown

freeing resources), 'Cancel' will leave Tahiti running and 'Reboot' will force a server restart.

2.2 Installing from CVS

You can install the Aglets platform from the CVS repository. The following are the required steps:

(a) Create the directory for the repository:
You need to create a directory playing as a container for the CVS repository. In this directory you will download a copy of all sources currently in the CVS repository.

(b) Log in to the CVS server:
To log in to the CVS server do the following:

```
luca@linux:/java/aglets/bin> cvs
    -d:pserver:anonymous@cvs.sf.net:/cvsroot/aglets
    login
```

The server will respond with

```
Logging in to
  :pserver:anonymous@cvs.sf.net:2401/cvsroot/aglets
CVS password:
```

No password is required for anonymous access, so simply leave it blank. After the login, the command prompt of your shell will be shown again. Now you are logged in the CVS server, and you can download the source tree.

(c) Download the source tree:
You need to download from the *aglets* module, thus do:

```
luca@linux:/java/aglets/bin> cvs
   -d:pserver:anonymous@cvs.sf.net:/cvsroot/aglets
   checkout aglets
```

The system will download (or update if you have already a version of the CVS repository) each source file in the on-line repository, placing files into a subdirectory with the same name of the module (in this case *aglets*). After that you can logout doing:

```
luca@linux:/java/aglets/bin> cvs
    -d:pserver:anonymous@cvs.sf.net:/cvsroot/aglets
    logout
```

(d) Compile the downloaded source tree:
The source tree you have downloaded must be compiled in order to build the Aglets library and platform. Enter in the *src* subdirectory and run Ant there, you will see the compilation of all sources. At the end of the compilation, the library and the platform will be installed in the module directory (i.e., the parent directory of the *src* one).

2.3 Compile from the source

The compilation of the source tree can be done easily through Ant, as already described in the previous sections. Once you have downloaded the source tree (either from HTTP or CVS), compile the whole tree entering in the tree directory (the one that contains

a file called *build.xml*) and running *ant*, as already described in this chapter.

Chapter 3

Using the Tahiti server

This chapter describes how to use the Tahiti server, that is the default server for the Aglets platform, in order to manage agents on your system.

3.1 Tahiti GUI

This section covers the use of the Tahiti GUI (Graphical User Interface), that is used as default user interface to the user when you launch the *agletsd* command (see figure 2.2). Tahiti presents a main window, with a menu bar, a list of running agents, and toolbar. The main area of the window is covered by the running agent list ("agent list" henceforth), which gives information about agents. Most operations are aglet-dependent, that means act on a specific agent. To specify to the server which aglet you are referring to, you have to select the agent from the agent list clicking on its row with the mouse; the row will become highlighted to notify that you are working on that agent.

Following sections cover how to use Tahiti in both GUI and command line mode.

3.1.1 The *Aglet* menu

Entries of the Aglet menu are displayed also in the toolbar as buttons. This menu allows administrator to handle the agent life cycle: creating/disposing, dispatching/retracting, etc. (see figure 3.1). Please consider that a lot of entries of this menu act on specific agent instances, so you need to select an agent in the Tahiti agent list before you can work on it. Each entry of the menu is detailed in the following.

- **Create**
  Allows administrators to create new agent instances. Once selected, a dialog window will appear, requesting to insert the agent class name

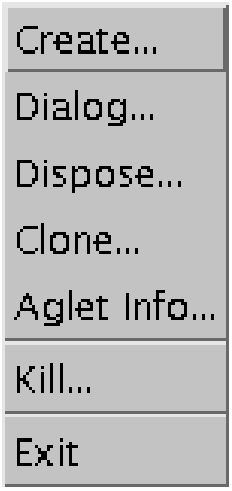

Figure 3.1: The Aglet menu.

(fully qualified, with the name of the package), the URL and other options (see figure 3.2).

In the creation dialog window you have to specify the class name of the aglet you want to create. This can be specified either manually writing the class name (with its package) in the *Aglet name* field or selecting an existing class from the list of known agents. Once you have inserted the aglet name, you can click on the *Create* button to create the new agent (a new row will appear in the Tahiti main window, specifying the agent name and other information about it). The *Add to list* and *Remove from list* buttons allow users to insert and remove new agent names in the known agent list. The *Reload class and create* button forces an instantation of the agent class without using the class loaders cache. This can be useful if you have modified the agent class and have already loaded it.

The *Source URL* field can be useful to load agent which classes are not in the aglet root (usually `public`). You can specify the location starting from which the class name should be found, thus the agent name results fully qualified by the URL and the class name.

- **Dialog**
  Sends a message of the kind *dialog* to the selected agent. This can be useful to display user windows on request. For example, the aglet `HelloAglet` shows a dialog window only if the *Dialog* option (i.e., a "dialog" message) is activated, as shown in the following code:

```
public void dialog(Message msg) {
        // check and create a dialog box
```

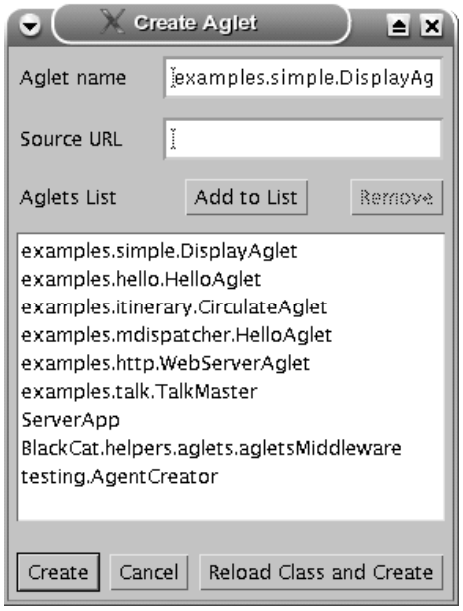

Figure 3.2: Agent creation dialog window.

```
        if (my_dialog == null) {
                my_dialog = new MyDialog(this);
                my_dialog.pack();
                my_dialog.setSize(my_dialog.getPreferredSize());
        }

        // show the dialog box
        my_dialog.setVisible(true);
}

public boolean handleMessage(Message msg) {
        if (msg.sameKind("atHome")) {
                atHome(msg);
        } else if (msg.sameKind("startTrip")) {
                startTrip(msg);
        } else if (msg.sameKind("sayHello")) {
                sayHello(msg);
        } else if (msg.sameKind("dialog")) {
                dialog(msg);
        } else {
                return false;
        }
        return true;
}
```

- **Dispose**
  The *Dispose* entry allow administrators to kill a running agent. Once you have selected an agent in the Tahiti agent list, and have clicked

the *Dispose* button (or have selected the entry from the menu), Tahiti will ask you a confirmation before it proceeds (see figure 3.3). If you

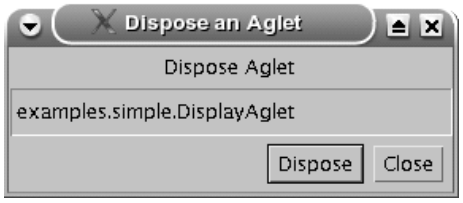

Figure 3.3: Confirmation required to dispose an agent.

are sure that you want to kill that aglet, click on the *Dispose* button in the dialog window, otherwise click on *Close*.

- **Clone**
  The *Clone* entry allows administrators to create an identical copy of a running agent. Tahiti will show you a confirmation dialog (see figure 3.4) where you can click on the *Clone* button in order to proceed. If

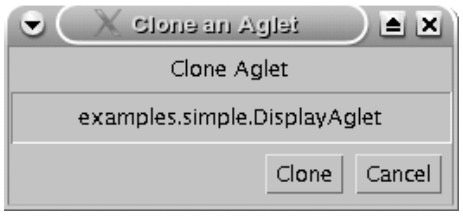

Figure 3.4: Confirmation required to clone an agent.

you do not want to clone the agent, click on *Close*. After cloning the selected agent, Tahiti will show a new row in the agent list, since a new agent has been created.

- **Aglet Info**
  This entry opens a dialog window with different data related to the agent, such as the key, the owner ID, the creation date, class name, etc. All the information about security comes from the certificate stored in the keystore database (see figure B), while the agent class information is that specified at creation time. The dialog window (see figure 3.5) does not allow users to modify the agent information. To close the window click on the *Close* button.

- **Kill**
  This option is present in the menu only, and is similar to the *Dispose* one, except it forces an agent to shutdown without waiting its disposing operations. In other words, disposing an agent awaits the exit of the

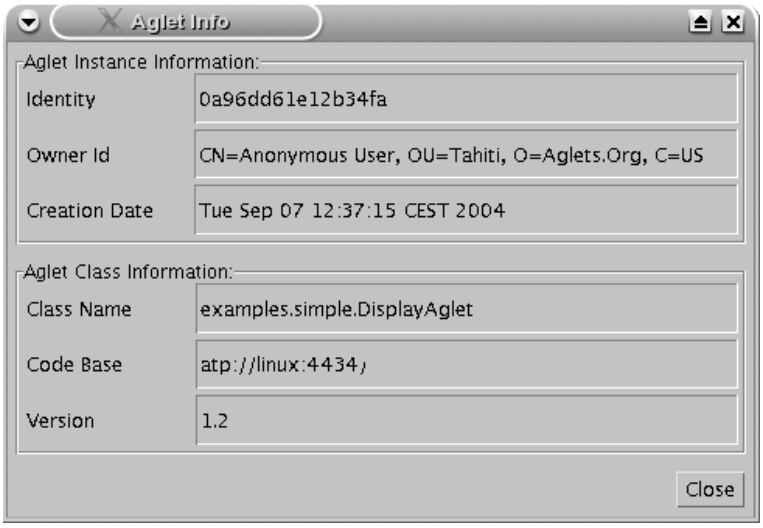

Figure 3.5: Main information about an agent.

agent's cleanup method, while killing an agent proceeds directly to the elimination of the agent from the server's data structures. The *kill* option is useful for stopping unresponsive agents, such as those stuck in infinite loops, or to quickly terminate malicious agents, but disposing is preferred over killing for normal agent shutdown.

- **Exit**
  The *Exit* option causes Tahiti to shutdown, disposing each running agent. Tahiti will ask the user whether a server shutdown or a server restart is desired and will offer the option to abandon the exit.

3.1.2 The *Mobility* menu

The *Mobility* menu allows control over the migration of agents and their activation/deactivation. Similarly to the *Aglet* menu, since each option works on a specific agent instance, you need first to select an agent in the Tahiti agent list. Each entry of the menu is detailed in the following, the menu is shown in figure 3.6.

- **Dispatch**
  This entry orders an agent to migrate to another Aglets platform. You need first to select the agent instance to migrate, and then select the *Dispatch* option. A dialog window will pop up, asking for the destination URL (see figure 3.7). The URL should generally use "atp" as protocol, from ATP - Agent Transfer Protocol, thus for example a valid URL could be `atp://somehost`. If you are running a couple of Tahiti instances on ports 4434 (default) and 5000, you can move the

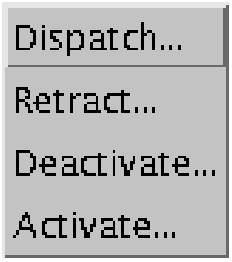

Figure 3.6: The *Mobility* menu.

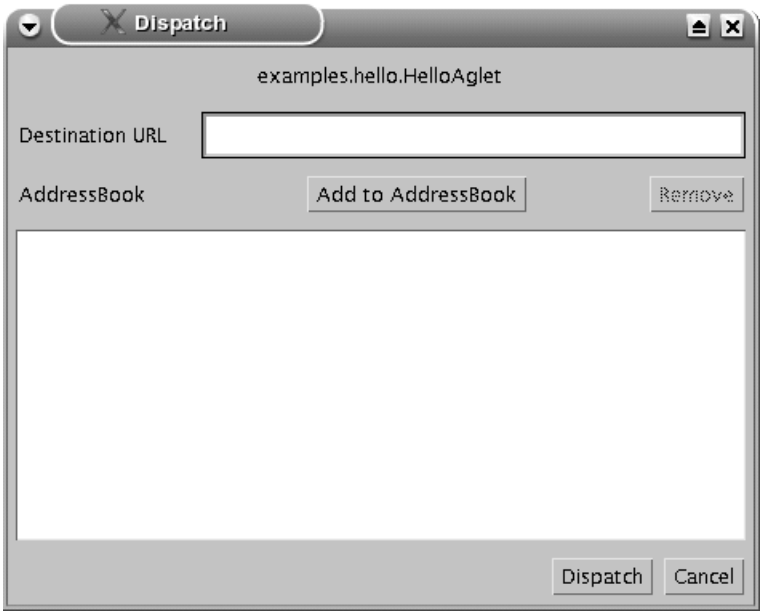

Figure 3.7: The dispatch dialog window.

example agent `DisplayAgent` to the latter platform using a URL as `atp://localhost:5000`. When you click on the *Dispatch* button, your aglet will be sent to the destination platform. If, for some reason (the aglet cannot migrate, is not serializable, etc.), the migration cannot be successfully done, your agent will stay on the current platform and a dialog window will notify to you the exception (see figure 3.8).

The dispatch dialog window offers to you the capability to store the URL to which you are sending an agent to a list of known URLs, called *Address Book*. The buttons *Add to AddressBook* and *Remove* gives to you the capability to add and remove entries (as URLs) from the above list. **Please note that, until you add an URL to the address book, you will not be able to retract agents sent to that URL.**

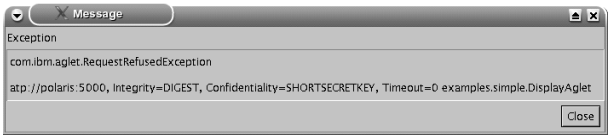

Figure 3.8: An exception during the migration occurred.

- **Retract**
  The *Retract* menu entry does the same thing as *Dispatch*, except from the other end of the trip. While *Dispatch* triggers an agent migration from the starting point, *Retract* triggers it from the destination. Once selected, the *Retract* option will show to you a dialog window as that in 3.9. Often, retracting brings "back home" a previously dispatched

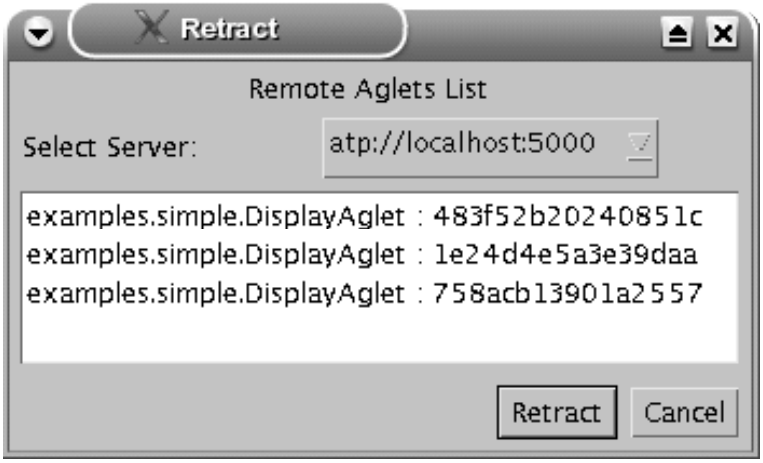

Figure 3.9: The retract dialog window.

agent, that is why the destination of agents that will later be called back should be saved to the address book and that is why only servers from the Address Book are offered in the list from the dialog window. After selecting the server, Tahiti queries it about running agents and presents them to the user in a list. After you have selected the agent to bring over, you can click on the *Retract* button. Now the server will be asked to send the chosen agent to this server and if the migration is successful, you should see it running again on your platform.

Not only agents previously dispatched to another server can be retracted. It is possible to start the steps for dispatching an agent, but to click on *Cancel* after the server has been added to the Address Book. The Address Book is persistent across server shutdowns, so from the moment a server has been added and until it is removed, agents can be retracted from it. Any agent can be selected from a remote server's

agent list, even agents that did not originate on the local server and never visited it. Whether the remote server carries out the retraction request or not depends on the protections present on the remote server.

- **Deactivate**
  This option forces Tahiti to stop the execution of the selected agent, serializing it locally, and deserializing when the agent is reactivated. Tahiti will pop up a dialog window (see figure 3.10) where you can insert a sleeping time (in seconds) for the agent. Clicking on the *De-*

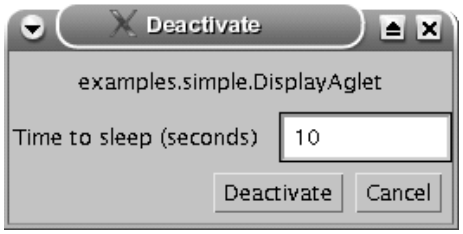

Figure 3.10: The deactivate dialog window.

activate button causes the agent to be deactivated.

- **Activate**
  This option makes the opposite of the *Deactivate* one: activate a sleeping agent. The agent will be deserialized and its execution will start again. Please note that this command runs silently, and the only thing you will see in the Tahiti window is a message in the status bar, that notify the activation of the agent.

3.1.3 The *View* menu

This menu (see figure 3.11) offers a few tools to take care of what is going on:

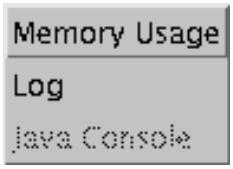

Figure 3.11: The *View* menu.

memory usage and Tahiti logs. In the following, each menu entry is detailed.

- **Memory Usage**
  This options opens a dialog window with a progress bar that shows the memory usage respect the Java run-time system (see figure 3.12). The red part of the bar represents the memory used by the Aglets platform,

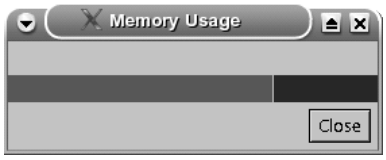

Figure 3.12: The dialog window that shows the memory usage.

while the blue bar represents the memory still available from the Java run time environment. The dialog window is managed by a separated thread, thus the progression bar updates itself every second.

- **Log**
  This options opens the dialog window shown in figure 3.13, that reports a brief log of operations done by the Aglets platform (agent creation,

Figure 3.13: The log dialog window.

dispatching, etc.). The *Clear Log* button causes the flush of the log content and its reset, thus a new clean log is used.

- **Java Console**
  This option is not available.

3.1.4 The *Options* menu

This menu, shown in figure 3.14, allows administrators to change settings

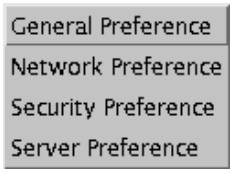

Figure 3.14: The *Options* menu.

about the whole aglet server engine, to set up protections and policies, and so on. In the following, a detailed explanation of each entry is given.

- **General Preference**
  This entry opens a dialog window that allows the user to set up global preferences, related to the start up of Tahiti and to its look and feel (see figure 3.15). The *Font* section allows the user to select which font

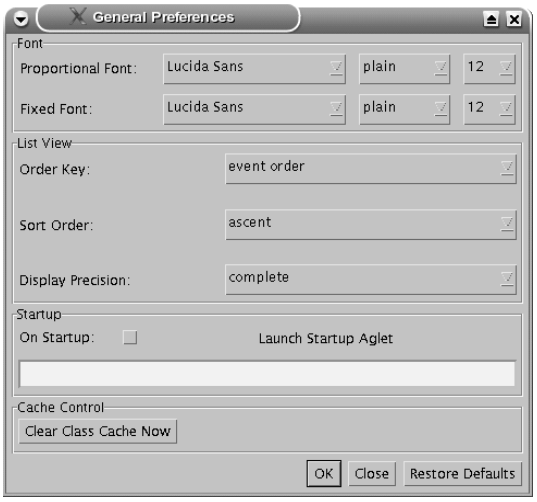

Figure 3.15: The general options dialog window.

Tahiti should use to display information, with its style (e.g., bold) and its size (in points). The effective use of the selected font depends on which fonts are available to the Java system. The *List view* section allows to set up how Tahiti have to show agents in the agent list. The order can be ascent/descent, and can be done by the agent class name, the creation time, the event order (i.e., what happened to the agent), etc.

The *Startup* section allows you to select a specific agent to be loaded at the Tahiti start time. You have to click on the *On Startup* check button and then to enter the fully qualified agent class name in the following text field.

The *Clear Class Cache Now* button of the *Class Cache* section, allows the administrator to reset the class loader's cache, thus new instances of already loaded agents will be created after a reload of their class. This can be useful if you are testing an agent during development, when its class(es) are changing frequently.

To apply all modifications you have done through the above dialog, you have to click on the *OK* button, while the *Close* one will abandon the modifications. The *Restore Defaults* button reset any changes to the Tahiti default.

- **Network Preferences**
  This entry opens a dialog window that allows users to manage network settings, like the use of proxies, HTTP tunneling, authentication requests, and so on. The dialog is shown in figure 3.16.

Figure 3.16: The network preferences dialog window.

The *Http Tunneling* section allows you to specify if Aglets should accept HTTP requests, if it must send agents through the http protocol (useful if you are running Tahiti behind a firewall), and which proxy should be use. You can specify either a DNS host name or an IP address, along with the port the proxy is accepting connections on. You can also specify a domain to which to dispatch agents without passing through the proxy, that means with a direct connection.

The *Authentication* section contains several buttons to configure security on incoming connections. The *Do Authentication on ATP Requests* checkbox forces, if checked, authentication on each incoming connection over ATP, that means on each incoming agent.

The *Create a new shared secret* button allows users to create new secrets for a specific domain. The button will open a dialog like the one in figure 3.17. The user has to enter a domain and a couple username (called *alias*) and password that must match a couple in the keystore database.

The *Remove a shared secret* button allows you to remove a secret by selecting it from the list of registered secrets, as shown in figure 3.18. Please note that you have to provide the password that holds the alias (i.e., the username) the secret has been created with.

Figure 3.17: Adding a shared secret.

Figure 3.18: Removing a shared secret.

The *Export a shared secret* button opens a dialog as the one shown in figure 3.19, that allows users to select the domain the secret is asso-

Figure 3.19: Exporting a shared secret.

ciated to, and to store it in a file which name is written in the *File name* text field. Once you have saved the secret in the file, Tahiti will show you a dialog window with the absolute path of the secret file, for making it easy to find (see figure 3.20).

The *Import a shared secret* button opens the dialog window shown in figure 3.21, which asks the user for the file name of the secret to import.

- **Security Preferences**
  This entry opens a dialog like the one in figure 3.22, that allows administrators to set up Java permission for agents and other Java classes.

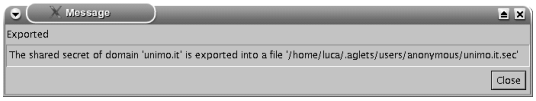

Figure 3.20: Tahiti gives you information about the full path of the secret file.

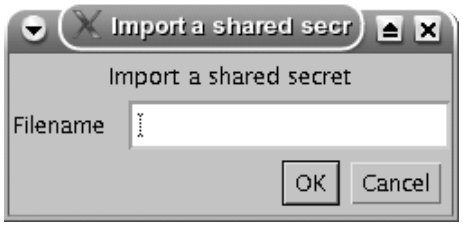

Figure 3.21: Importing a shared secret.

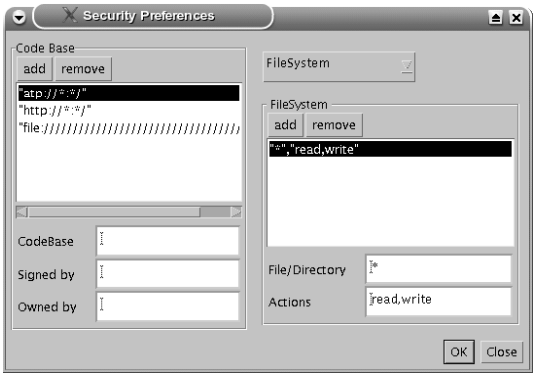

Figure 3.22: The security options dialog window.

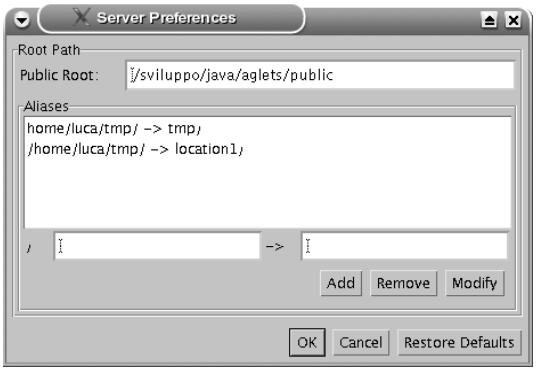

Figure 3.23: The server preferences dialog window.

The window is splitted in two main parts: on the left the user can select the codebase of the Java classes (either an agent or a normal class) for which to use the permissions and protections defined on the right. The use of this window is very similar to the use of the Java *policytool* program. Furthermore, since it works like the Java security mechanism, all permissions will not be explained here; you can find more details on the Java 2 documentation. Modifications will be applied to the `~/.aglets/security/aglets.policy` file. **Please take care when using this option, since it does not always work as expected**; thus you should check that the policy file has changed.

- **Server Preferences**
  This entry opens a dialog window (see figure 3.23) that allows users to set a few parameters like the server public root, that is the directory where Tahiti searches for agents. Unfortunately, this option seems to have a few bugs and does not work very well.

3.1.5 The *Tools* menu

This menu gives users access to a few tools more related to the Java virtual machine than to the Aglets platform itself. Figure 3.24 shows the menu appearance, while in the following you can find a detailed description of each entry.

- **Invoke GC**
  The selection of this entry will force a call to the Java garbage collector, in order to force a memory check and to free no more used objects/agents. You can use this menu entry if you believe your system memory has not been freed, or after killing a large agent.

- **Threads**
  This option causes Tahiti to dump a brief information about all ex-

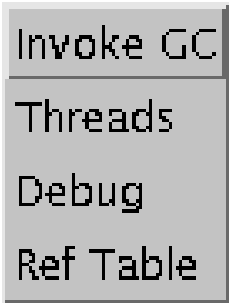

Figure 3.24: The *Tools* menu.

isting threads in the JVM. The dump is displayed in the Java console (terminal), and is similar to the following one:

```
{java.lang.ThreadGroup[name=system,maxpri=10]}
 + Threads
  - Thread[Reference Handler,10,system] alive
  - Thread[Finalizer,8,system] alive
  - Thread[Signal Dispatcher,10,system] alive
  - Thread[CompilerThread0,10,system] alive
```

- **Debug**
  The only visible thing is the showing of the string "Debug off" in the Java console. Probably this option was used to enable debug prints for Tahiti components.

- **Ref Table**
  Does not show anything. Probably it was a dump mechanism for the Tahiti and Aglets internal reference table.

3.1.6 The *Help* menu

This menu does not provide a real help, rather credit information. Most of the entries are not working in the current release of Tahiti due to the absence of an external program, called `openurl`, used to point the web browser to a web page. For this reason, do not worry too much if you see an exception in the Java console when you select this menu entry.

This menu will probably be fixed in a future release of the Aglets platform.

3.2 The Tahiti command line

You can run the Aglets server also from the command line. To enter in the command line, specify the `-nogui` option to the `agletsd` command. (it does not matter where the option is placed respect the other parameters). Tahiti will start in command line mode, asking for the username and the password as for the GUI mode.

Starting Tahiti with the `-noconsole` or `-commandline` parameter seems no different that starting it with no parameter. The `-daemon` parameter probably starts the aglet server listening on a port to which the control client `daemoncontrol` can connect.

Once the user is logged, Tahiti presents a command prompt that allows administrators to manage agents. The command line prompt is not so powerful as the Tahiti GUI, but can be faster and can be used in extreme situations (e.g., when the X server crashes). You can ask for help writing "help", and you will see a list of available commands:

```
> help
help                    Display this message.
shutdown                Shutdown the server.
reboot                  Reboot the server.
list                    List all aglets in the server.
prompt                  Display or changes the prompt.
msg on|off              Message printing on/off.
create [codeBase] name  Create new aglet.
<aglet> dispatch URL    Dispatch the aglet to the URL.
<aglet> clone           Clone the aglet.
<aglet> dispose         Dispose the aglet.
<aglet> dialog          Request a dialog to interact with.
<aglet> property        Display properties of the aglet.
Note: <aglet> is a left most string listed
in the result of list command.
```

As an example, if you want to create a new agent, you have to use the `create` command:

```
> create examples.hello.HelloAglet
```

and the system will reply with a message showing the operation result, such as:

```
> Create : examples.hello.HelloAglet from atp://linux:4434/
```

If you want to list all agents running in the platform, you have to use the `list` command; the system will show a list of all agents (in the following example only one is running):

```
> list
> aglet0 [examples.hello.HelloAglet]
```

The first word ("aglet0") represent the agent identity, useful for other commands. For example, if you want to dispose the above agent you have to specify the identity to the `dispose` command:

```
> aglet0 dispose
Removed : aglet0
> Dispose : examples.hello.HelloAglet
```

For a complete list of available commands, digit "help". To exit from the command line mode you have to shutdown Tahiti, that can be done with the `shutdown` command (without any option).

Chapter 4

Developing agents with Aglets

This chapter covers basic issues about developing agents with the Aglets library. In the following section it will be shown how to configure main development environments to support Aglets, how to compile and run your own agents and how to explore the library API.

An aglet (i.e., an agent able to run on the Aglets platform) is a simple Java class that must have as base class `com.ibm.aglet.Aglet`. To define the aglet behaviour you have to override methods of the base class, at least the `run()` one, and that is all you need to get a complete aglet.

4.1 Configuring your IDE to use the Aglets library

The Aglets library is composed by a single jar archive, `aglets-x.x.x.jar`, which is installed in `$AGLETS_HOME/lib`, where `x.x.x` means the library version number (e.g., 2.0.2). In order to compile your own agents, you must have the above jar in your classpath. The following subsections describes how to compile agents with different tools and IDEs.

4.1.1 Using command line tools

You can develop agents as you do with normal Java programs, that means you can write your Java file(s) with your favourite editor, compiling then them with the command line compiler (e.g., `javac`, `jikes`). Supposing you have created and saved in a file called `FirstAglet.java` the following agent:

```
import com.ibm.aglet.*;

public class FirstAglet extends Aglet
{
    public void run(){
        System.out.println("\n\tHello\n");
    }
}
```

you can compile it from the command line, after ensuring you have the Aglets library in your classpath. For example, supposing you have the Aglets platform installed in `/java/aglets`, you can do the following:

```
export CLASSPATH=$CLASSPATH:/java/aglets/lib/aglets-2.0.2.jar
```

in a Bash shell, or something like:

```
set classpath=%classpath%;c:\java\aglets\lib\aglets-2.0.2.jar
```

on a Microsoft Windows machine.

After that, just compile your agent from the command line:

```
javac FirstAglet.java
```

Even if aglets are like other Java classes, they cannot be run as stand-alone programs, thus you have to run agents into the platform. Before that, you must make agents reachable by the platform itself, that means you must have the agent (compiled class) in the server public root, that is by default the folder `public` of the Aglets platform installation. In other words, you have to copy the classes of your agents under the above folders, thus you can specify the aglet class name in the creation dialog (see figure 3.2).

Please note that adding the directory where your classes resides to the CLASSPATH variable will not work. It seems that Tahiti has a few bugs in the management of classes and classpath.

4.1.2 Using IBM Eclipse

You have to instrument Eclipse accepting the Aglets library in the project you are working on. Supposing you have already defined a project, steps required to use the Aglets library are the following:

1. select *Import* from the *File* menu
2. choose the library:
   in the opened dialog window (see figure 4.1), chose *Zip file* and click on *Next* button. After that, you have to browse the local filesystem in order to find the Aglets library jar (see figure 4.2), then you have to click on the *Finish* button.

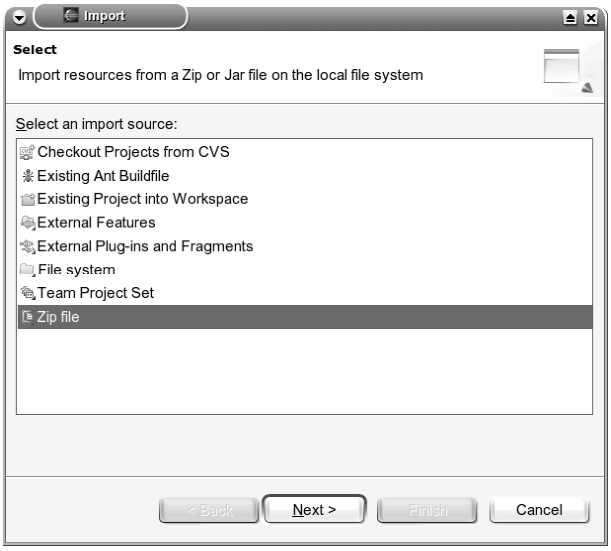

Figure 4.1: Importing a package in Eclipse, step 1.

3. test the library:
   you should now see the library packages in the project folder. Importing the package should give you no errors.

4.1.3 Using Netbeans IDE

The Netbeans IDE uses an approach similar to the IBM Eclipse one: you have to define in each project jars that must be included. To import the Aglets library, supposing you have already created a project, do the following:

1. mount the library:
   right click on the project name in the file system view (usually on your left), and select *mount* from the contextual menu. In the submenu, select *Archive*, as shown in figure 4.3.

2. choose the library:
   browse the local filesystem to find the library (see figure 4.4, then click on the *Finish* button.

3. test the library:
   you should now see the library packages in the project folder. Importing the package should give you no errors.

4.1.4 Using JBuilder

To quickly enables the Aglets library in the Borland JBuilder, you have to follow the steps below:

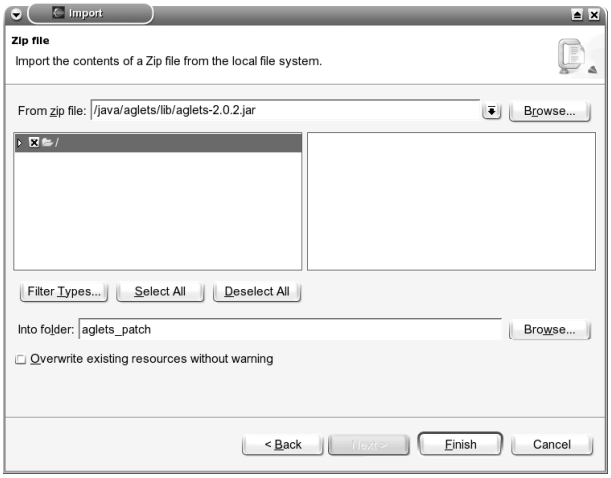

Figure 4.2: Importing a package in Eclipse, step 2.

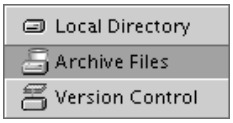

Figure 4.3: Importing a package in Netbeans, step 1.

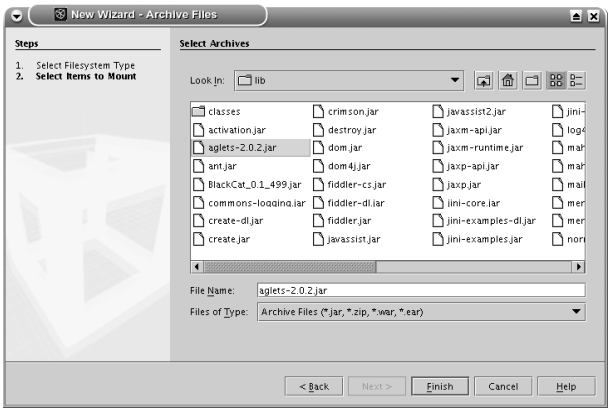

Figure 4.4: Importing a package in Netbeans, step 2.

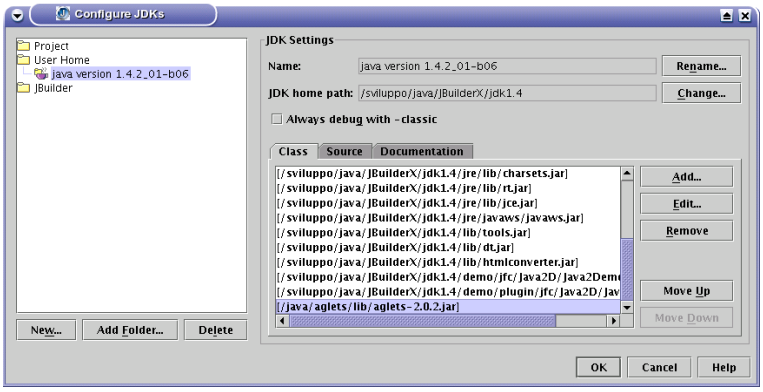

Figure 4.5: The JBuilder JDKs configuration window.

1. select *Configure JDKs* from the *Tools* menu.
2. choose the library:
   push the *Add* button and browse your local filesystem in order to find out the Aglets library jar, and then select it.
3. check the importation of the library:
   once you have selected the library, you should see it at the bottom of the jar list in the dialog window (see figure 4.5), then click on *OK*.

Please note that, doing the above, you will find the Aglets library in all projects you will create.

4.2 Base knowledge

This section provides basic information about the development of an aglet. In the following you will find which main methods you have to override, how to manage incoming messages and how to catch events. The ASDK (Aglets Software Development Kit) provides the infrastructure to develop and run aglets, which are software agents that

- Run in a very well defined environment/server or *context*.
- Have a unique identifier.
- Can be moved or *decide* to move from one context to another, i.e. these agents are mobile agents, with a set of well defined restrictions about
  - When an aglet can be migrated.
  - What state an aglet maintains in a migration.
  - How the aglet resumes execution after being migrated.

- Can be cloned, an identical aglet can be created from an existing one.
- Are able to communicate to each other by sending/receiving *messages*.
- Can handle (or *react to*) a set of predefined events.
- Can be made *persistent*, i.e. stored on disk.

And all of these characteristics will be explained in the next sections, starting wiht the most basic ones. Other issues such as those related to security are considered advanced and are not included in this chapter. Some of the previous material in this manual is related to one o more of the items above. However, most of the manual at this point has been devoted to describe the current *provider*/implementation of the Aglets context: Tahiti, the Aglets server. Tahiti and agletsd will be used as synonymous in the rest of the manual.

4.2.1 Main methods of an aglet

Aglets are agents that follow an Applet-like development way, that means you have to override a few methods that will be called from an external entity (the Aglets run-time system) during the agent life. In Aglets, an agent can be considered as a Java object with a thread running on it. There is a base class called Aglet, which represents a simple agent for this platform, and it is necessary to inherit from the Aglet class to create a customized agent, as briefly shown in Section 4.1.1. The following piece of code shows the main methods you should override: `onCreation`, `run`, and `onDisposing`.

```
import com.ibm.aglet.*;

public class SecondAglet extends Aglet{

    public void onCreation(Object init){
        System.out.println("Agent created " + init);
    }

    public void run(){
        System.out.println("Agent running");
    }

    public void onDisposing(){
        System.out.println("Agent quitting");
    }
}
```

A short description of each method:

- `onCreation(Object init)`: Called once, when the aglet is created. It can accept an initialization parameter.
- `run()`: It is the life of the agent; it is called once per platform (i.e., for each platform the agent visits).
- `onDisposing()`: Called when the agent has been killed or is shutting down.

If you start this agent in Tahiti (e.g. using the create button), you will see two messages on the standard output (the console where you launched the `agletsd` command) almost immediately after creation:

```
Agent created null
Agent running
```

And, after using the Tahiti Dispose button, you will see one more message in the console

```
Agent quitting
```

4.2.2 Simple experiments

At this stage, you can experiment and see many of the Aglets features enumerated above. In particular, having created one SecondAglet in Tahiti, you can clone the aglet by using the button `Clone`. In the Tahiti console, this "new" aglet will print the message defined in the `run` method, i.e.

```
Agent running
```

Note that this aglet is not considered a newly *created* one, since it is not created from scratch. Instead, an aglet created as a clone has exactly the same state as the cloned one, i.e. the value of the cloned instance variables (aglet state) are copied in the new aglet. Thus, it does not make any sense that agletsd *calls* the method `onCreation` for the aglet which is a clone of an existing one. Cloning an aglet implies that the agletsd at least

- Copies the aglet state, just as in the Java `clone`.
- Calls the aglet `run` method.

And the new aglet clone of an existing aglet is running on its own thread.

If you start another Tahiti server, it will be possible to migrate aglets from one Tahiti server to another Tahiti server. Starting a new Tahiti server in a computer which already has a running Tahiti server is possible, just remember to define a listening port other than 4434 (the default Tahiti *communication* port), e.g.

```
agletsd -port 5000
```

The Tahiti server listening on port 4434 will be referred to as Tahiti1, and the Tahiti server listening on port 5000 will be referred to as Tahiti2 in order to avoid confusion. Now, in Tahiti1 an aglet and use the Dispatch button. In the *dispatch dialog* destination URL type

`atp://localhost:5000`

or

`atp://127.0.0.1:5000`

(the Tahiti2) and, also, click on the "Add to the AddressBook" button in order to be able to retract the aglet back later. Once dispatched the aglet, you will see

`Agent running`

on the Tahiti2 console, i.e. the aglet has been

- Serialized in terms of the Java object communication system.
- Transferred from the server Tahiti1 to the server Tahiti2.
- Started to run by the server Tahiti2, which calls the aglet `run` method at the aglet arrival.

Note that atp is referred to in the URL, since the Aglets Transport Protocol (atp) is used to transfer aglets among Tahiti servers. Once the server Tahiti2 is added to the AddressBook of another server Tahiti1, it is possible in the server Tahiti1 to retract aglets from the server Tahiti2. Retracting can be seen as a migration triggered from the destination (receiving) Tahiti server. Analogously, Dispatching can be defined as a migration triggered on the source (sending) Tahiti server.

The next aglet has an instance variable that can be used as an example of how the state is preserved when an aglet is migrated or cloned and, also, how the migrated or cloned aglet begins running.

```
import com.ibm.aglet.*;

public class PrintingAglet extends Aglet
{
  int i = 0;

  public void run()
  {
    int limit = i + 5;

    for (; i < limit; i++){
      System.out.println("\n i is " + i);
    }
  }
}
```

Starting this aglet in a Tahiti server will produce the output in the Tahiti console shown below.

```
i is 0
i is 1
i is 2
i is 3
i is 4
```

If you now clone this aglet, the next five lines will be added to the Tahiti console

```
i is 5
i is 6
i is 7
i is 8
i is 9
```

Thus, the value of the state variable is not initialized as in a new aglet, but takes the value of the cloned (*initial*) aglet. At this point, there are two independent aglets running in the same Tahiti server. If, for example, the original aglet is dispatched to another Tahiti server, you will see in the corresponding console, the next five lines

```
i is 5
i is 6
i is 7
i is 8
i is 9
```

And, also, the migrated aglet will no be shown in the initial Tahiti server, since it was dispatched to another context.

These simple experiments show how an aglet can be cloned, migrated and retracted. None of these activities has been programmed in the aglet, i.e. in the SecondAglet and in the PrintingAglet classes above. The Aglets platform, (Tahiti server/s and atp, more specifically), are ready to perform these operations on aglets.

4.2.3 Self cloning and migrating aglets

For the next examples, it will be useful to start Tahiti server/s with security "turned off". This is not a good idea in general, but will allow to run code without to be concerned on security risks. Tahiti server security can be turned off by using the file aglets.prop which is included with the Aglets distribution as follows:

- Make a copy of the aglets.prop file, usually found in Linux in the directory `$AGLETS_HOME/cnf`.

- Uncomment line (in the aglets.prop file)

  `#aglets.secure=false`

- Start agletsd with

  `agletsd -f aglets.prop`

If you need to start another Tahiti server in the same computer, you can do it with

`agletsd -f aglets.prop -port 5000`

which is useful to migrate aglets from a Tahiti servber to another Tahiti server in the same computer, for example.

An aglet can clone and migrate itself by using the methods `clone` and `dispatch` defined in the base class Aglet. The following code (SelfCloning class) shows a simple aglet that clones itself and uses a state variable to avoid the clone aglet to clone itself (and this sequence would continue endlessly).

```
import com.ibm.aglet.*;

public class SelfCloning extends Aglet{

  boolean theFirst = true;

  public void onCreation(Object init){
    // Only non clone aglet runs this method
    System.out.println("Agent created");
  }

  public void run(){
    if (theFirst){
        theFirst = false;
        System.out.println("Cloning");
        try {
          clone();
        } catch (Exception e){
          System.out.println("Failed to clone");
        }
    } else{
        System.out.println("Clone running");
    }
  }
}
```

Starting this aglet in a Tahiti server, the aglet will produce the output shown below (in the corresponding Tahiti console).

```
Agent created
Cloning
Clone running
```

and there will be two aglets running in the Tahiti server. A self migrating aglet can be programmed rather easily using the previous SelfCloning aglet. Start another Tahiti server and, in the SelfCloning aglet, change the line

```
clone();
```

by

```
dispatch(new java.net.URL("atp://127.0.0.1:5000"));
```

in order to migrate the aglet to a Tahiti server running on localhost and listening on port 5000. Take into account that in this case, the state variable `theFirst` is used to avoid migrating the aglet from the Tahiti server listening on port 5000 to the same Tahiti server (since the URL is hardcoded in the aglet).

The examples given in this section are not meant to be useful other than for learning. In fact, there are better ways to clone and migrate aglets, which will be analyzed in the next section/s.

4.3 Events

Aglets supports an *event/event listener* model, where an agent can register event listeners to particular kind of events, thus it can handle those events. There are mainly three kind of events, tied to different scenarios of the agent life cycle: cloning, mobility and persistency. Table 4.1 shows each kind of event, with types and event listener that can be associated to.

Event kind (class name)	*Time*	*Event listener*	*Method of the event listener*
`CloneEvent`	before cloning	`CloneListener`	`onCloning(..)`
`CloneEvent`	after cloning(1)	`CloneListener`	`onClone(..)`
`CloneEvent`	after cloning(2)	`CloneListener`	`onCloned(..)`
`MobilityEvent`	before migrating	`MobilityListener`	`onDispatching(..)`
`MobilityEvent`	when the agent arrives	`MobilityListener`	`onArrival(..)`
`MobilityEvent`	when the agent is being retracted	`MobilityListener`	`onReverting(..)`
`PersistencyEvent`	after the agent activation	`PersistencyListener`	`onActivation(..)`
`PersistencyEvent`	before the agent deactivation	`PersistencyListener`	`onDeactivating(..)`

(1) Delivered to the clone aglet

(2) Delivered to the *original* (cloned) aglet

Table 4.1: Available events in Aglets.

Events are defined as Java classes and event listeners are defined as Java interfaces in the package `com.ibm.aglet.event`. Furthermore, there is no other event than those defined in the table above. The next example shows a code enhancement of the previous SelfCloning (in the previous section) example by using events and event listeners.

```
import com.ibm.aglet.*;
import com.ibm.aglet.event.*;

public class SelfCloningWithEvents extends Aglet{

  boolean theFirst = true;

  // CloneListener implementation, "installed" later
  class MyCloneListener implements CloneListener {
     public void onCloning(CloneEvent ev) {
        System.out.println("Cloning");
     }
     public void onCloned(CloneEvent ev) {}
     public void onClone(CloneEvent ev) {
        theFirst = false;
     }
  }

  // The CloneListener is "installed" here
  public void onCreation(Object init){
    System.out.println("Agent created");
    CloneListener cloneLstnr = new MyCloneListener();
    addCloneListener(cloneLstnr);
  }

  public void run(){
    if (theFirst){
        try {
          clone();
        } catch (Exception e){
          System.out.println("Failed to clone");
        }
        System.out.println("Cloned running" + theFirst);
    } else{
        System.out.println("Clone running" + theFirst);
    }
  }
}
```

Setting aside the implementation of `CloneListener` and its usage as an event clone listener, the SelfCloningWithEvents aglet has subtle but important differences compared to the SelfCloning aglet:

- The SelfCloningWithEvents original aglet always has its state variable (theFirst) with value "true", which is the expected value for the *original* aglet.
- The SelfCloningWithEvents aglet can be programmed in order to react to clone events in a uniform way, independently of whether the clone event
  - is programmed (generated) in the code as in the run() method of the aglet, or
  - is generated by selecting clone in the Tahiti server, i.e, it is an *external* event.

In fact, the SelfCloningWithEvents aglet is able to react to cloning events, while the SelfCloning aglet is not. The latter is just programmed to clone itself.

Handling mobility events is rather the same as handling cloning events: you have to implement the `MobilityListener` interface

```
class MyMobListener implements MobilityListener {
  public void onDispatching(MobilityEvent ev) {
  ...
  }
  public void onReverting(MobilityEvent ev) {
  ...
  }
  public void onArrival(MobilityEvent ev) {
  ...
  }
}
```

and, later, “install” the listener

```
MobilityListener mobilityLstnr = new MyMobListener();
addMobilityListener(mobilityLstnr);
```

Handling persistency events is straightforward taking into account the previous examples for cloning and mobility events. Summaryzing, handling events implies:

- Implement `<x>Listener` interface.
- Create an object of the interface implementation.

- "Install" the object as the listener with `add<x>Listener`.

and `<x>` should be replaced by one of `Clone`, `Mobility`, or `Persistency` in order to handle cloning, mobility, or persistency events.

The following code example shows how the interfaces `MobilityListener`, `CloneListener`, and `PersistencyListener` can be used. First of all, have a look to the listener class:

```
package examples.agletevents;

import com.ibm.aglet.*;
import com.ibm.aglet.event.*;

public class myListener
    implements MobilityListener, CloneListener, PersistencyListener
{
    ////////////////////
    // mobility listener methods
    ////////////////////

    public void onArrival(MobilityEvent event){
        System.out.println("Agent arrived "+event);
    }

    public void onDispatching(MobilityEvent event){
        System.out.println("Before moving..."+event);
    }

    public void onReverting(MobilityEvent event){
        System.out.println("Before coming back home..."+event);
    }

    ////////////////////
    // clone listener methods
    ////////////////////

    public void onClone(CloneEvent event){
        System.out.println("I'm the clone "+event);
    }

    public void onCloned(CloneEvent event){
        System.out.println("A clone of myself created "+event);
    }

    public void onCloning(CloneEvent event){
        System.out.println("Someone is cloning myself "+event);
    }
```

```
    ////////////////////
    // persistency listener methods
    ////////////////////

    public void onActivation(PersistencyEvent event){
        System.out.println("Activating "+event);
    }

    public void onDeactivating(PersistencyEvent event){
        System.out.println("Deactivating "+event);
    }
}
```

The above listener defines methods to catch events at different time, as explained by the code itself. The following agent registers the above listener and use it to handle events:

```
package examples.agletevents;

import com.ibm.aglet.*;
import com.ibm.aglet.event.*;

public class agletC extends Aglet{

    public boolean move=true;

    public void onCreation(Object init){
        // create a listener object
        myListener listener = new myListener();

        // register a mobility listener
        this.addMobilityListener((MobilityListener)listener);

        // register a clone listener
        this.addCloneListener((CloneListener)listener);

        // register a persistency listener
        this.addPersistencyListener((PersistencyListener)listener);
    }

    public void run(){
        System.out.println("HELLO!");
    }
}
```

For example, after a dispatching, mobility events are triggered and the agent prints, in the source console, the following statements:

```
Before moving...MobilityEvent[DISPATCHING]
```

while in the destination console prints:

```
Agent arrived MobilityEvent[ARRIVAL]
HELLO!
```

To better understand how event listeners work, execute the above agent and try cloning, dispatching and deactivating it; have a look at what messages are printed out in all the consoles.

Examples and explanations given so far show that Aglets basic programming requires using the base class `Aglet`, and handling events requires class/es and interface/s given in the package `com.ibm.aglet.event`. However, there are other tasks that require other classes and/or interfaces. In fact, many tasks require direct communication with the aglets environment, such as the Tahiti server. Creation of other aglets is an example of an aglet interacting with its context, as will be shown in the next section.

4.4 Creating other aglets and `AgletContext`

The Aglet `clone` method produces a new aglet with identical state as the cloned one, the clone aglet is not a *created* aglet, but is a copy (clone) of an existing one. An existing aglet can generate a new (created) aglet by calling the method

```
AgletContext.createAglet(URL codebase, String name, Object init)
```

i.e. the aglet which is going to create another aglet needs to

- Get an object of and `AgletContext` implementation class.
- Call the `createAglet(...)` method of the `AgletContext` object.

This means that creation of aglets is a task directly related to the aglets container, the context on which an aglet resides, the aglet sever. `AgletContext` provides an interface to the runtime environment that occupies the aglet. Any aglet can obtain a reference to its

```
AgletContext
```

class implementation object via the

```
Aglet.getAgletContext()
```

primitive, and use it to obtain local information such as the address of the hosting context, or to create a new aglet in the context. Once the aglet has been dispatched, the context object currently occupied is detached and the new context is attached. The runtime library is responsible for providing the implementation of this interface; thus, aglet programmers do hot have to implement this interface. The following two aglet classes are defined such that

- CreatedAglet is an aglet class defined only for creating aglets (CreatedAglet instances) from another aglet.

- Each CreatingAglet instance creates a CreatedAglet instance by calling `createAglet` to its `AgletContext`.

Note that while `CreatedAglet` aglets do not interact with the environment (context), `CreateingAglet` aglets make explicit requests to the environment in order to create other aglets.

```
////////////////////
// CreatedAglet (in CreatedAglets.java)
////////////////////
import com.ibm.aglet.*;

public class CreatedAglet extends Aglet{

     public void onCreation(Object init){
          System.out.println("CreatedAglet agent created");
     }

     public void run(){
          System.out.println("CreatedAglet running");
     }
}

////////////////////
// CreatingAglet (in CreatingAglets.java)
////////////////////

import com.ibm.aglet.*;

public class CreatingAglet extends Aglet{

  public void onCreation(Object init){
        // Only non clone aglet runs this method
        System.out.println("Agent created");
  }

  public void run(){
        try{
            // get the aglet context
            AgletContext context = this.getAgletContext();

            // create another CreatingAglet instance
            context.createAglet(null, "CreatedAglet", null);
        }catch(Exception e){
            System.out.println("Exception "+e);
        }
    }
}
```

After compiling and copying .class files in the corresponding Tahiti server

binaries directory, you will be able to create a CreatingAglet aglet. Then, you will see in the Tahiti console

```
Agent created
CreatedAglet agent created
CreatedAglet running
```

where the first line in the console corresponds to the `CreatingAglet` aglet and the other two correspond to the `CreatedAglet` aglet, i.e. the aglet which is explicitly created by the instance of `CreatingAglet`. Note that being able to explicitly create aglets allow

- New aglets to be running, independently of the creating aglet, which is not possible with the method `clone`, for example. Cloning generates new aglets with exactly the same behavior as the cloned aglet.
- New aglets with new state, i.e. aglets which execute the `onCreation` method, and which can be initialized *from scratch*.
- An explicit interaction with the aglet context, since the aglet explicitly requires the creation of a new aglet to its container/aglet server.

This section has shown that each aglet is able to require specific actions to its environment/context, and one of those actions has been explained: aglet creation. Also, each aglet is is able to interact almost directly with each other, by means of messages, topic covered in the next section.

4.5 Message handling

Aglets exploit a communication system based on *message passing*: two agents that want to communicate each other have to exchange a message. Messages are instances of the `Message` class, and their kind is specified by a string. An agent that wants to explicitly manage messages has to override the `handleMessage(..)` method, returning `true` in the case the message is managed by the agent, and `false` otherwise. The

`handleMessage(..)`

method, is also defined in the Aglet base class, just like

`onCreation(..)`
`run()`
`onDisposing()`

The following piece of code (HandlingMessages aglet class) shows an agent that handle all messages; you can launch it and send dialogs messages through the Tahiti GUI. This simple aglet shows interesting message and message handling features:

- An aglet is not necessarily aware of message source/s, it just receives and handles messages. Message source should be included in some way in the message itself, since there is no other way to identify it.
- Message handling is as complex as the `handleMessage(...)` implementation.
- The method `handleMessage(Message msg)` has one parameter, the message received by the aglet, which can be (and usually is) used to define the aglet message handling behavior.
- The message type, that is its classification, is stored as a String object in the message itself. Keeping the message type as a string attribute makes simpler and faster the use of new messages, since you don't have to create classes for them. The `getKind()` method is used to get the string which determines the message type.

```
import com.ibm.aglet.*;

public class HandlingMessages extends Aglet{

    public void run(){
        System.out.println("Agent running");
    }

    public boolean handleMessage(Message msg){
        System.out.println("Received a message " + msg.getKind());
        return true; // if the message is used
    }
}
```

4.5.1 Aglets, messages, and threads

There are a few concepts that must be clear when working with message handling. First of all, each aglet executes within a thread, but threads are managed by the platform and can be shared among agents for efficiency. The efficiency of this approach can be understood thinking at a *buyer-seller* example: image a couple of agents, with one playing as a *seller* and one playing as a *buyer*. In this situation, it is not needed that the buyer agent is active before the seller has put a good on sale, thus there is no reason to use a thread-per-agent approach. Furthermore, the seller can simply send a message to the buyer specifying the good on sale, and then should wait the answer of the buyer (i.e., should deactivate or suspend until an answer comes). Following this example, it should be clear that the number of active threads (i.e., agents) could be reduced at one per time. Aglets exploits this condition in its whole design: if agents can share the same thread, no

additional threads will be created. In other words, the number of agents and threads are not strictly related.

Due to the Aglets thread model, it is important to understand that each message is delivered by a thread, that can be different from the one the agent runs (or have run). Since aglets are implicitly synchronized, a message can be delivered if the agent is active and running, that means while your agent is in the `run()` method (or another method). In fact, while the agent is in the `run()` method (or another one), there is a thread active in the agent itself, and another thread (the message deliver thread) cannot deliver the message because of the Java synchronization.

From the above considerations, it is possible to see how an agent performing an (in)finite loop will be unable to receive and manage any incoming message. Nevertheless, there is a way to force an agent to release the lock, and this can be done with the `exitMonitor()` method, that causes all waiting messages to be dequeued and, at the same time, all threads locked on a `waitMessage()` to be resumed. Please be aware that forcing an agent to release locks could produce race conditions.

4.5.2 Sending messages and `AgletProxy`

An aglet is not allowed to interact with another aglet directly, and this includes messaging. Instead, every interaction among aglets, as sending a message is made through `AgletProxy`, which is an interface acting as a handle of an aglet and provides a common way of accessing the aglet behind it. Since an aglet class has several public methods that should not be accessed directly from other aglets for security reasons, any aglet that wants to communicate with other aglets has to first obtain the proxy object, and then interact through this interface. In other words, the aglet proxy acts as a shield object that protects an agent from malicious agents.

When invoked, the proxy object consults the SecurityManager to determine whether the current execution context is permitted to perform the method. Another important role of the AgletProxy interface is to provide the aglet with location transparency. If the actual aglet resides at a remote host, it forwards the requests to the remote host and and returns the result to the local host. This is why the `createAglet(...)` method seen in the previous section does not return an aglet but an `AgletProxy` instead. As in the case of `AgletContext`, the runtime library is responsible for providing the implementation of the `AgletProxy` interface; thus, aglet programmers do not have to implement this interface.

From the specific point of view of sending a message from an aglet, the message is sent to an `AgletProxy` of an aglet, by using the method `sendMessage(Message msg)` as shown schematically in figure 4.6. The continuous line for the arrow in figure 4.6 denotes that agletA has to explicitly identify the agletB proxy, and the dashed line for the arrow denotes that

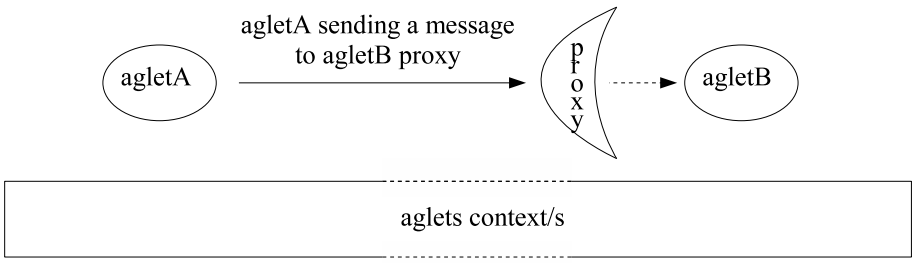

Figure 4.6: Sending a message to an aglet.

message delivering is dependent on the runtime implementation, i.e. not known to the aglet programmer. Also, sending and receiving aglets are not required to be in the same aglet container/context.

The following code shows how an AgletSndr creates an aglet and sends a message to an AgletRcvr. This code is not different from that in the section devoted to agent creation, adding the send message in one aglet class and the message handling in the other one.

```
////////////////////
// AgletSndr (in AgletSndr.java)
////////////////////

package messaging;

import com.ibm.aglet.*;
import java.net.*;

public class AgletSndr extends Aglet{

    public void run(){
        try{
            // get the aglet context
            AgletContext context = this.getAgletContext();

            // create an AgletRcvr instance
            AgletProxy rcvr = context.createAglet(null,
                                  "messaging.AgletRcvr",
                                  null);

            // send a message to the agent
            rcvr.sendMessage(new Message("HELLO"));
        }catch(Exception e){
            System.out.println("Exception " + e);
        }
    }
}
```

```
////////////////////
// AgletRcvr (in AgletRcvr.java)
////////////////////

package messaging;

import com.ibm.aglet.*;

public class AgletRcvr extends Aglet{

    public void run(){
        try{
            // get my ID
            AgletID myID = this.getAgletID();
            System.out.println("\nMy ID is "+myID);
        }catch(Exception e){
            System.out.println("Exception " + e);
        }
    }

    // handle the "HELLO" message
    public boolean handleMessage(Message msg){
        if(msg.sameKind("HELLO")){
            System.out.println("HELLO msg received");
            return true;
        }
        return false;
    }
}
```

The next messaging example combines several features of Aglets: creation, migration, and remote messaging. The receiving/handling messages aglet does not necessarilly has to change regarding the previous example, i.e. the AgletRcvr aglet in the Java package messaging above. Specifically on messaging and the `AgletProxy` needed to be used for sending a message:

- The `AgletProxy` returned by `createAglet(...)` is useful only in the local context, i.e. it should not be used when the aglet has migrated (is in another context).
- The `agletID` of the remote agent is needed in order to communicate with it. More specifically, the `agletID` is used to get the remote `AgletProxy` from the remote context in which the receiving aglet is running.
- Even when the `getAgletProxy(..)` method called in code is deprecated, a call to the no-deprecated method such as:

  ```
  AgletProxy remoteProxy = context.getAgletProxy(remoteID);
  ```

will not work, since it can work only with the local agents. The use of the MAF (Mobile Agent Finder) will work better, but at the moment there is not a lot of documentation about how to use it in Aglets.

```
////////////////////
// AgletSndr2 (in AgletSndr2.java)
////////////////////

package messaging;

import com.ibm.aglet.*;
import java.net.*;

public class AgletSndr2 extends Aglet{

    public void run(){
        try{
            // get the aglet context
            AgletContext context = this.getAgletContext();

            // create an AgletRcvr instance
            AgletProxy rcvr = context.createAglet(null,
                                  "messaging.AgletRcvr",
                                  null);

            // save the new aglet ID
            AgletID remoteID = rcvr.getAgletID();
            System.out.println("The new agent has ID = " + remoteID);

            // migrate the new agent
            String remoteContext = "atp://127.0.0.1:5000";
            URL url = new URL(remoteContext);
            rcvr.dispatch(url);

            // get the remote proxy
            AgletProxy remoteProxy = context.getAgletProxy(url,
                                       remoteID);

            // send a message to the remote agent
            remoteProxy.sendMessage(new Message("HELLO"));

        }catch(Exception e){
            System.out.println("Exception " + e);
        }
    }
}
```

4.6 A sleeping aglet

In general, it is not possible to make an aglet sleeping, since threads should not be managed directly from the developer. You can use something similar, but more expensive, to simulate sleeping, that is deactivation: you can deactivate and reactivate an agent, but you must be careful since deactivation means that the agent is serialized and its execution restarts from the beginning. This means that the following code will run indefinitely:

```
import com.ibm.aglet.*;

public class agletF2 extends Aglet{
    public void run(){
        for(int i = 10; i > 0; i--){
            System.out.println(i + " seconds left!");
            try{
                this.deactivate(1000);
            }catch(IOException e){
                System.out.println("Ops!");
            }
        }
    }
}
```

Since, after a reactivation, the execution of the agent restarts from the beginning of the `run()` method, the agent will restart the `for` loop from the same point (i.e., `i = 10`). Instead, the following code is correct and works:

```
import com.ibm.aglet.*;

public class agletF extends Aglet{
    int i = 10;

    public void run(){
        for(; i > 0; i--){
            System.out.println(i+" seconds left!");
            try{
                this.deactivate(1000);
            }catch(IOException e){
                System.out.println("Ops!");
            }
        }
    }
}
```

Nevertheless, it should be clear how deactivation cannot substitute the sleeping mechanism, and that making a thread to sleep will produce strange effects on the whole platform (such as locking the message passing mechanism). This is due to the fact that the Aglets platform uses a set of threads (i.e., a *pool*) to work efficiently, thus the same thread can be shared by different agents. There is a trial feature, done with the method `suspend(..)` of the class `AgletProxy` that can be used as a sort of sleeping command. Unfortunately, it does not work yet, and in fact the following agent causes the exception `IllegalThreadStateException` to be thrown:

```
import com.ibm.aglet.*;

public class agletZZ extends Aglet{
    public void run(){
        try{
            for(int i = 10; i > 0; i--){
                System.out.println(i+" seconds remaining");
                this.suspend(1000);
            }
        }catch(AgletException e){
            System.out.println("Ops!");
        }
    }
}
```

Warning on this code example Please note that running the agent `agletF2` will have tremendous effects on your system. Due to the Aglets thread management, you will not be able to destroy (either by killing or deactivating or disposing) the running agent, while the latter will run forever. Even stopping an restarting the Tahiti server will not solve the problem, thus the only things you can do is to perform a *clear start* (see the FAQ section) or to manually remove the spool file in the file

$HOME/.aglets/spool/*hostname@runningPort/agletID*

and then restart the Tahiti server.

```
import com.ibm.aglet.*;

public class agletZZ extends Aglet{
    public void run(){
        try{
            for(int i = 10; i > 0; i--){
                System.out.println(i+" seconds remaining");
                this.suspend(1000);
            }
```

```
        }catch(AgletException e){
            System.out.println("Ops!");
        }
    }
}
```

Appendix A

FAQ & Configuration Files

This chapter presents a list of Frequently Asked Questions (FAQ) update to the current Aglets SDK available releases. Please note that there are other available sources on the Internet, but they could be out of date or could not include information about the latest releases. Please be sure to refer to this chapter before any other information source, and in case of problems or troubles, refer to the official site and to the mailing lists to post requests.

A.1 FAQ

This section presents a list of Frequently Asked Questions about Aglets. The list has been changed from the old FAQ, since a lot of questions were related to earlier versions.

- **What does AWB stands for?**
  AWB means Aglets WorkBench, and it was the original name given by IBM to the Aglets platform and library. Today's common trend is to use simply Aglets (with the capital 'A') to indicate both the library and the platform, and to explicitly specify the latter when required.

- **What does JAAPI stands for?**
  JAAPI means Java Aglet API.

- **What is Fiji?**
  Fiji was a project to enable Aglets capabilities in a web browser. So far, the project is no more maintained and available.

- **What are the differences between the IBM Aglets platform and the open source one?**
  IBM does not mantain Aglets anymore, thus the version you can download by their server is the 1.03 (or 1.1 beta), while the version available at SourceForge is greater than 2. Of course, the open source version is

more updated and functional than 1.03 version, thus you should use it. Furthermore, the version of IBM was thought as commercial software, that means you requires a license to run it.

- **Can I run Aglets on Java 2 (JDK1.2 +)?**
  Yes, of course, and in fact this manual is thought to run Aglets over the Java 2 platform. The first versions of Aglets were developed over Java 1 (JDK 1.3), and often this causes confusions to newbies. Running Aglets 2 over Java 2 is the best way to get the platform working.

- **Are there any archives of the Aglet Mailing list?**
  Yes, visit the Aglets web page at sourceforge.net to get information about mailing lists, archives and how to subscribe.

- **I need some help! What should I do?**
  Reading this manual is a good start. Unfortunately, at time of writing this manual there are not a lot of documentation sources available. There are still a few pages at the IBM Tokyo Research Laboratory web site, but they are quite old and no more maintained. See the Aglets web site at sourceforge.net to get on-line help.

  If you need to report some problems or bugs to the mailing lists, please include as much information as possible, in order to allow other participants to understand the problem. Information you should:

  1. the version of the platform;
  2. the operating system;
  3. the path where you have installed the platform (for example /home/luca/aglets);
  4. the exception stack trace (if an exception is thrown);
  5. in case of a `SecurityException`, the content of the `java.policy` file;
  6. the code that you believe is causing the misbehaving;
  7. a dump of the following environment variables (if set up): CLASSPATH, JDK_HOME, JAVA_HOME, AGLETS_HOME. You can obtain the variable values writing in a console:

     ```
     # Unix - Linux
     echo $CLASSPATH
     echo $JDK_HOME
     echo $JAVA_HOME
     echo $AGLETS_HOME

     # Windows
     echo %CLASSPATH%
     ```

```
echo %JDK_HOME%
echo %JAVA_HOME%
echo %AGLETS_HOME%
```

Be quite and polite when asking help, nobody wants to be bothered with others' problems. If you find the answer by yourself, post a message as a reply to indicate the answer; it will be interesting for other people and will let them to not waste their time. Finally, do not repeat the message if you do not get an immediate answer, let people the time to understand and to reply.

- **Are there any public Aglets servers I can send my aglets to?**
  There were a few trials about a public aglet server, but at the best of our knowledge, there is no one server running now.

- **When I launch Tahiti I get: "Please set HOME environment variable!"**
  This indicates an abnormal situation in your operating system: check that the HOME variable points to your home directory. Contact your system administrator to fix this problem, that is not related to Aglets.

- **When I try to access a local file with my aglet the server throws java.jang.SecurityException although all the necessary permissions are set.**
  There is a little known feature of Tahiti (OK, call it a bug ;) that when Tahiti is installed, it creates a security domain "file:///*/" and all permissions are given to this domain. It should mean that any aglet having its codebase on this machine can access any file. But it's not true. You have to create another security domain describing exactly the path to the codebase of the aglet, grant necessary permissions and then it works.

- **When I try to get my Aglet to access the file 'test.txt' I get a FileNotFound exception**
  When attempting to open a file, any path to a file is relative to the directory from which the Aglet was created. Therefore, the solution is that you should specify the file absolute name like this:

  ```
  FileInputStream inputStream = new FileInputStream("c:/test.java");
  ```

  You can use a single slash (/) in Unix like systems and Windows, or you can specify a double backslash in Windows.

- **Can I run Tahiti with no network connection on Microsoft Windows 95?**
  Yes, even if this system is quite old and I suggest you to run Aglets

over a newer version (have you ever thought passing to Linux?). If you have troubles running Aglets over Microsoft Windows 95, try the following:

1. edit the `hosts` text file that is under the `WINDOWS` folder and add the loopback address entry: `127.0.0.1 localhost`;
2. start Tahiti with the `-resolve` option.

- **I want to send my aglet around to lots of different hosts and to pop up a window at each host, but when I try nothing happens, or I get an exception.**
  A permission to create top level window is given to any aglet in default aglets security policy. If you want to modify security preferences, do the following instructions on every host where your aglet will visit.

  Add the following line into files "aglets.policy":

  ```
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
  ```

- **Why would not my stand-alone server take my environment variables into account?** This is a feature of the stand-alone servers, which is they do not take the Aglets related environment variables into account by default. You should explicitly put the AGLET_PATH and AGLET_EXPORT_PATH on the java command-line when launching a stand-alone server (as far as we know, at least). For example under *nix, if your server is called `StandAloneServer`:

  ```
  java -Daglets.class.path=\$AGLET_PATH \
  -Daglets.export.path=\$AGLET_EXPORT_PATH StandaloneServer
  ```

- **Can an aglet perform SNMP operations?** Yes - an aglet can perform SNMP operations by using a Java class that provides SNMP operations and that can be serialized. A good package to look at is jmgmt - it is small, straightforward, and has all source included. It is available from

  http://i31www.ira.uka.de/~sd/manager/jmgmt/.

  Good overviews of various packages can be found at:

  http://www.mindspring.com/~jlindsay/javamgmt.html

  http://wwwsnmp.cs.utwente.nl/software/pubdomain.html

  Some of the packages use threading- and you have to be careful when you serialize agents that use threads. The other big restriction on some of the packages is that they have commercial licenses, even if you are doing not commercial research activity.

- **How can an aglet be used as a HTTP Server?**
  First, take a look to the example agent `WebServerAglet.java` shipped with the Aglets platform examples. Please note that using an aglet as an HTTP server has a few drawbacks: developing an HTTP server is not a trivial task and it is subject to security risks, thus you should be sure of the quality of your server before making it available to third parties. Furthermore, even if the Aglets platform has a good thread system, it can handle threads in a way that does not fit very well your needs. Finally, running an HTTP server embedded in an aglet, means that your server will be available through the Aglets running port. For example, if you are running Tahiti at the host `myHost`, port 4434 (the default for Tahiti), you will be able to reach your HTTP server at the address `http://myHost:4434/`.

  You must also set the options in Tahiti as specified in the comments. If you are coding/adapting the sample - make sure to include the line:

  ```
  getAgletContext().setProperty("name.test", getAgletID());
  ```

  since this is how Tahiti identifies the aglet to forward the HTTP request to.

  Another option is to look at using Fiji. The disadvantage is that the status of Fiji is uncertain right now. Another option is to look at finding/coding Java classes to listen for incoming HTTP requests. There are quite a few HTTP servers written in Java out on the web. Depending on the requirements for the web server (i.e., response time, threading, etc.), there are a few different servers to use.

- **Can an aglet work with other Servers (HTTP, FTP, etc)?**
  The general answer to this question is "yes". Nevertheless, before start developing your aglet-based server/client, you should take care of a few things. First of all, you must know the protocol, or at least you must have a Java library to interface to the protocol. The Java classes used (in the case) by the aglet must be compatible with the JRE and the Aglets library version you are running, and most important, if they must travel with the aglet, they must be serializable. The latter requirements (serialization), can be overtaken if the aglet will be stationary on a specific host, thus it will never be serialized (of course, this is true if you will never deactivate the aglet!).

- **How does Aglets determine a hostname?**
  Using the configuration of your system, typically using the DNS (Domain Name System).

- **Can an aglet use SSL?**
  (For some good background information on the security model check

the aglets book in Chapter 10) It depends on the availability of SSL Java libraries, and how they are compatible with the Aglets library you are running.

- **How is memory used when a message is sent between two aglets?**
  You can monitor the memory activity using the Tahiti memory tool: open the *Tools* menu and then select the option *Memory Usage*.

- **How can the sleep operation be used in an aglet?**
  Please note that threads are hidden to aglets, thus you should not use normal Java thread operations in your aglets. You should use a timer or something similar to obtain the required feature. Using `sleep(..)` is dangerous, since the message passing mechanism will be locked until the sleeping thread wakes up. There is an experimental feature, called `suspend(..)` that could work. Take a look at Section 4.6.

- **How can local and remote Aglets discover each other and communicate?**
  There are a few options available to attempt to discover remote contexts, depending on what you need to do.

  If you want to create a local agent, dispatch it to a remote context and communicate with the now remote agent, you need to get the remote agent proxy. The proxy discovering can be done with the MAF architecture (at the moment there is no documentation on how to use the MAFFinder in your programs). Another option is to manually keep a track of where your remote agents are, and this can be useful to build critical mission systems, where the MAF architecture can fail down.

  See the code example sections.

- **I got a message similar to *java.util.MissingResourceException: Can't find bundle for base name tahiti, locale en_US* but I do not know how to fix it.**
  As you can see the problem is caused by the `ResourceBundle` class, which is used for localization. If you look in the lib directory of your Aglets installation, you can see a text file called `tahiti.properties`, which contains menu and button entries for the Tahiti window. You have to set your CLASSPATH to the `lib` directory, thus the above file can be read by the `ResourceBundle` class.

- **I want to add an agent to the agent list, thus when I click on the *Create* button I can choose it directly. How can I do this?**
  The first and common way of doing it is through the creation dialog

window. Otherwise you can write the agent in a text file, placed in the user's home directory, and in particular in

`$home/.aglets/users/username/aglets.properties`

and add the agent class name to the line that contains the property `aglets.agentList`. Class names must be add as separated by a blank space, without new line characters.

- **What is FIPA?**
  FIPA means Foundation for Intelligent and Physical Agents, and is a no-profit organization that defines agents' standards, such as communication languages (called ACLs), interoperability protocols, and so on.

- **Is Aglets 2.0.2 FIPA compliant?**
  No. Aglets is not FIPA compliant, since it has been developed when FIPA was only a proposal. Furthermore, in those days, there was another standard: the *MASIF* (Mobile Agent Systems Intercommunication Facility). Due to this Aglets is MASIF compliant, even if there is not a lot of documentation (or better, there is no documentation) regarding MASIF in Aglets and how well it works. It must be noticed that Aglets is RMI compliant, thus you can use it in combination with the Java's RMI services. Please note that the fact that Aglets is not FIPA compliant does not mean that developers do not want that standard, it is simply a developing lack! Maybe one day Aglets will be FIPA compliant...

- **Is FIPA so important?**
  It is difficult to answer to this question, since it depends on a lot of opinions and point of views. FIPA is in general good, but as most of the standards, it could not reflect what developers really wants (usually simplicity and performance). It depends on what you are going to develop if Aglets can be the right choice: if you have to interoperate with a FIPA systems, please choose a FIPA compliant platform (such as JADE). However, please note that there are other platforms which do not adhere to the FIPA standard, such as DIET, while other platforms implements both FIPA and MASIF (such as Grasshopper).

- **Do I need to install Aglets on every machine I want to send an agent to?**
  Yes, or at least you have to write a program which can act as an agent server (i.e., a Tahiti substitute) by your own. The fastest way to get your aglets running is to install Aglets on every host you want to send agents to.

- **Is there another source of documentation? I often hear something about the Aglets book...**
  You can find a few web pages over the Internet that discuss several Aglets related arguments, but please take care that these pages could be out of date (i.e., too old regarding the Aglets version you are running). There is an Aglet book, Programming and deploying Java (TM) Mobile Agents with Aglets, by Danny B. Lange and Mitsuru Oshima, but it is old (it is related to the Aglets 1.0 version), and a lot of things have changed since it has been published. I don't believe you need the Aglets book to develop agents using Aglets.

- **Can Aglets run over a PDA or a smart device?**
  Smart devices usually have limited JVMs (except if you install Linux Familiar and Kaffe), thus it is difficult to install and run Aglets as it is. Actually, we are planning on the migration of Tahiti over PDAs, and maybe a FAQ about the use of Aglets and PDAs will be available soon. Here you can find a web project related to the Aglets 1.0.3 version:

  http://siul02.si.ehu.es/~jirgbdat/FACILITIES/PDAs/principal_Ingles.html

- **How many agents can run over the same instance of Tahiti?**
  It depends on how powerful is your run-time. Aglets exploits a good threading system, without mapping every agent in a separated thread, but using instead a single thread for multiple agents. This means that the number of agents you can create is not directly dependent on the number of threads your JRE support. Furthermore, due to the message architecture of Aglets, where a thread is assigned to each message to be processed, the number of supported agents (and their performance) depends on the use of messages that currently running agents are doing.

- **I have agents developed with the 1.x version of Aglets, can I run them with the 2.x version?**
  So far, there is no knowledge of incompatibility among agents developed with different major versions, even if it is possible that old agents do not run. The first thing to try is to recompile the old agent (if possible) with the new API. If you know or find some incompatibility, please send a message or write a bug report.

  The main difference between the 1.x and the 2.x series, is the use of the Java 2 security mechanism: the old 1.x version did not use it, while the 2.x version do, leading to a more Java 2 compliant application.

- **It seems as the ant file is corrupted, what can I do? (only for *nix operating systems)**
  Check if the `build.xml` contains any DOS carriage return characters, and clean the file with the `dos2unix` command. If it is still

not working, try downloading a newer version of Apache Ant from http://www.apache.org.

- **I try to start my Tahiti server using command `agletsd -f myAglets.props`, but I got error messages `Out of environment space`. What do I have to do? (only for MS Windows operating systems)**
  You have at least two possible ways of extending the memory space: (i) change the size of the argument of the /E: parameter for `command.com` in the `config.sys` shell setting. For instance, set of the size of environment variables to 512 bytes, specify:

  ```
  SHELL=C:\COMMAND.COM C:\ /P /E:512
  // Maximum is 4K:
  SHELL=C:\COMMAND.COM /E:4096 /P
  ```

  You can add this to your `config.sys` file. If this does not work, try changing the environment variables of the MS-DOS prompt accessing the *memory* section of the *Property* of the prompt icon.

- **I got an `AccessControlException`, what do I have to do?**
  This exception is thrown when your code is trying to execute an operation for which it has not enough rights (for example, it is trying to open a server-socket). Have a look to your policy file, and try using the sample shipped with Aglets, that can be installed running the `install-home` option of Ant.

- **Is there a way to directly log-in to Tahiti without inserting a username and a password?**
  Yes. You have to specify the username and the password to use in a properties file, and then you have to launch Tahiti specifying the properties file to use. First of all, place the username and the password in the properties file:

  ```
  aglets.owner.name=aglet_key
  aglets.owner.password=aglets
  ```

  where aglet_key and aglets are the username and the password existing in the keystore; substitute them with the couple you want to use. Launch Tahiti as in

  ```
  agletsd -f /path/to/the/properties/file
  ```

  thus Tahiti will not prompt you for the username and the password. Please note that this option is enabled with the default properties file, `cnf/aglets.props`, but you have to explicitly pass the file to the Tahiti command line.

Please take into account that storing a password in a plain text file is not a good security design, thus be carefull with permission of such file (i.e., nobody except you should have read/write access to the file).

- **Why the keystore contains [an `aglets_key` and `anonymous`] key pair?**
  Aglets requires that each agent has an owner. When you log in to Tahiti, you are implicitly saying that all agents created through the Tahiti user interface will have "you" as owner. Each aglet will have, as attachment, the keystore data to recognize its owner, and this is the reason why you must to log in to Tahiti, before you can create any agent.

  Now think at what happens when your platform is receiving an agent from an external source, that could be another agent platform. In this case, you do not have in your keystore credentials about the owner, since these credentials have been stored in the source platform. To solve this problem, the anonymous keypair is used, and this is the reason why the keystore comes with pre-set keys.

  Please note that, even if it is possible to assign permissions on the base of the owner rather than the simple code base, this feature does not seem to work very well.

- **Can I disable security checks in Tahiti?**
  First of all ask yourself if you really need to run Tahiti without security settings; this is strongly unrecommended for production machines. Nevertheless, if you are sure you want to do this, edit the `aglets.props` file and set the property `aglets.secure` to *false*.

- **Can I change the logging system?**
  Aglets is currently using Log4J; the logger class is determined by the value of the property `aglets.logger.class`, thus you can change the logger simply changing the above property in the `aglets.prop` file. The Jakarta Log4J class is *org.aglets.log.log4j.Log4jInitializer*, but you can change it to another, such as

  - *org.aglets.log.console.ConsoleInitializer*, that prints everything to the STDOUT (you should redirect or pipe to analyze output), or
  - *org.aglets.log.quiet.QuietInitializer* that suppresses most of the logging output.

- **Is there any debugging capability?**
  Actually no. I suggest you to use smart printing functions, to understand what is happening to the code. You can try also using the Java DeBugger (jdb), but it could be quite difficult.

- **Do I need any special library to compile the source version of Aglets?**
  In general no, but you could need a few libraries like log4j in your classpath. If you have any error, please report it to the mailing lists.

- **How can an aglet transport a file from one host to another?**
  This is an often asked question over all the aglets mailing lists, therefore please read this point before asking it by yourself. An aglet cannot transport anything that is not a Java serializable object, that means you have to transform your file into a Java serializable object. The kind of the object depends on your needs. For example, if you have to transport a text file, you can read all the file and place its content into a string (i.e., `java.lang.String`). If the file is a binary one, you have to translate it into a portable object, even a MIME one. Please do not try to migrate a `File` object, since it will not work! The most efficient way to transform a file into an object depends on what your application must do, and I suggest you to have a look even at the SOAP or any other XML based document form. Finally, please take care that if the file is available by a network filesystem (such as AFP, SMB, NFS), you do not need to migrate the file at all, but simply to adjust the file name on the destination.

- **Can I use HTTP messaging among aglets?**
  Please note that you can implement any kind of network messaging in Aglets, from standard sockets, to HTTP, SOAP, RMI, etc. But it is on your own to implement such way of communications; you can have a look at the code available at the aglets-net project (see `http://sourceforge.net/projects/agletsnet`).

- **Is there a way to exchange data among agents?**
  Yes, you have to send messages containing the data you want to exchange, but please take care of the serializability of your objects, since the messaging system allows only serializable messages.

- **Is it possible to run multiple context over the same server? How can I do that?**
  The general answer to this question is yes, even if Tahiti currently does not allow users to create multiple context. Please note that this does not mean that it cannot handle multiple contexts, and in fact you can develop an agent in charge of creating multiple contexts for you. To do this, use the `createContext(..)` method of the `AgletsRuntime` class. When working with multiple context, take care of the URL for dispatching agents to a specific context: place the context name after the machine address, such as `atp://machineAddress/contextName`.

- **How can I move an agent among different contexts?**
  You can use the ATP migration protocol, specifying the same address but changing the URL in order to reflect the destination context.

- **Can an Aglet communicate with a Servlet?**
  Yes, take a look at the code below (written by Angsuman Dutta):

```
public void run(){
  try{

    URL server=new URL("http://localhost:8100/servlet/" +
                       "FirstServlet");
    URLConnection con = server.openConnection();
    con.setDoOutput(true);
    con.setUseCaches(false);
    Calendar rightNow = Calendar.getInstance();

    ObjectOutputStream request = new ObjectOutputStream(new
              BufferedOutputStream(con.getOutputStream()));

    StringBuffer d=new StringBuffer("<?xNameVdaml version=" +
                   "\"1.0\" encoding=\"UTF-8\"?>");
    d.append("<Name>");
    d.append("dude");
    d.append("</Name>");
    String data=d.toString();
    String msgtype="xmlFile";
    String [] msg=new String[2];
    msg[0]=msgtype;
    msg[1]=data;
    request.writeObject(msg);
    request.flush();
    request.close();
    ObjectInputStream response = null;
    Object result = null;
    response = new ObjectInputStream(
    new BufferedInputStream(con.getInputStream()));

    // read response back from the server
    result = response.readObject();
  }
  catch(Exception e){
    System.out.println(e);
  }
}
```

- **Can I develop an agent server on my own? How can I embed the Aglets technology into my application?**
  You can develop an agent server by your own, and this will allow

you also to embed the Aglets technology in your applications. Before posting any question about how to write an agent server, you should carefully have a look at the `ServerApp.java` source code available with the source code package.

When developing your own server, you should take into account a few issues. There can be authentication problems, that means you could be unable to log in to the server as you are used to do with Tahiti. To overtake this problem, someone has suggested to hardcode the couple username/password in the server source file. Moreover, you can catch some exception due to the unavailability of fonts; if this happens remove the following lines from the `Tahiti.initializeGUI()` method:

```
try {
  Class.forName("sun.awt.PlatformFont");  // for 1.1
} catch (Exception ex) {
  ex.printStackTrace();
}
```

When developing your own server, you have to take care about properties and permissions, thus the new server can access all properties and can act as a real Tahiti substitute. Furthermore, remember that each agent must belong to one context, that means you have to create a context first, then you can create agents or other contexts. The following piece of code has been written as an example by Gustavo Nucci Franco:

```
import com.ibm.atp.daemon.*;
import com.ibm.aglet.system.*;
import com.ibm.aglet.*;
import java.net.*;

public class xAgletContext{
  public AgletContext context;
  public int portNumber;

  public xAgletContext(int pn){
    portNumber = pn;
    String [] arg = {"-port", String.valueOf(portNumber)};
    Daemon daemon = Daemon.init(arg);
    AgletRuntime runtime = AgletRuntime.init(arg);
    context = runtime.createAgletContext("");
    daemon.start("aglets");
    context.start();
```

```
    context.addContextListener(new CL());
  }

  class CL extends ContextAdapter{
    //You should implement listeners' events to monitor
    //the life cycle of your aglets here
  }
}
```

If you get a

```
java.lang.ExceptionInInitializerError:  java.lang.NullPointerException
```

related to the `LogInitializer.getCategory(Unknown Source)`, the logging system cannot be loaded and initialized statically. Try this:

```
String initName = System.getProperty("aglets.logger.class",
        "org.aglets.log.quiet.QuietInitializer" );
Class.forName(initName);
```

that will load the logger, making the exception disappear.

Another example of a server can be the following:

```
// usage: java SimpleServer <keystore> <policy>
//                          <username> <password> <port>
import java.net.*;
import java.util.*;
import com.ibm.aglet.*;
import com.ibm.aglets.*;
import com.ibm.aglets.tahiti.*;
import com.ibm.maf.*;

public class SimpleServer{
  com.ibm.aglet.system.AgletRuntime runtime;
  private String username;
  private String password;
  private String port;
  private String keystore;
  private String policy;
  AgletContext context;
  MAFAgentSystem maf_system;

  public SimpleServer(String args[]){
    try{
      // get all parameters
      keystore = args[0];
      policy = args[1];
      username =args[2];
```

```
    password = args[3];
    port = args[4];
  }

  catch (Exception ex){
    ex.printStackTrace();
  }
}

public void setup(){
  Properties props = System.getProperties();
  // Setup properties
  props.put("atp.resolve", "true");
  props.put("atp.useip", "true");
  props.put("maf.port", port);
  props.put("maf.protocol", "atp");
  props.put("java.policy", policy);
  props.put("aglets.keystore.file", keystore);
  props.put("maf.finder.port", "4435");
  props.put("maf.finder.host", "localhost");
  props.put("maf.finder.name", "MAFFinder");
  props.put("aglets.logger.class",
  "org.aglets.log.console.ConsoleInitializer");
  props.put("aglets.logfile", "aglets.log");
  String initName = System.getProperty("aglets.logger.class",
  "org.aglets.log.console.ConsoleInitializer");
  try{
    Class.forName(initName);
  }
  catch (ClassNotFoundException ex){
    ex.printStackTrace();
  }
}

public void start(){
  this.setup();
  runtime = runtime.init(null);
  runtime.authenticateOwner(username, password);
  maf_system = new MAFAgentSystem_AgletsImpl(runtime);

  String protocol = "atp";
  try{
    MAFAgentSystem.initMAFAgentSystem(maf_system, protocol);
    // use Tahiti classes to initialize
    Tahiti.installFactories();
    Tahiti.installSecurity();
    // create context
    context = runtime.createAgletContext("");
    MAFAgentSystem.startMAFAgentSystem(maf_system, protocol);
```

```
      //start context
      context.start();

    }
    catch (MAFExtendedException ex){
      ex.printStackTrace();
    }

  }

  public static void main(String[] args){
    SimpleServer simple = new SimpleServer(args);
    simple.start();
  }
}
```

- **Can I use IP addresses instead of DNS names for URLs?**
  Yes, and in some situations it is suggested you do that. For example, if you are working with machines not registered in a DNS; you should use IP addresses instead of host names.

- **I can send an agent to another machine but I cannot retract it back. Dispatching the agent other the same machine raises a `RequestRefusedException`. How can I solve this?**
  It is probably a problem of DNS. Try using IP addresses in the URLs or to add the machine name and address to each *hosts* file.

- **I need to fix the security policies of my server, but I don't know how to know the codebase agents are running from.**
  You can get the codebase developing a simple agent that executes:

```
try {
  URL codeBase =
  ((Aglet)this.getProxy().getAgletInfo().getCodeBase();
  System.out.println("codeBase: " + codeBase);
} catch (InvalidAgletException e) {
  System.out.println("InvalidAgletException");
}
```

  Please be aware that, if your hostname changes (e.g., change of the network, ISP, etc.), your permissions must be set up again, because your codebase changes accordingly to the hostname. Furthermore, consider that codebase are not interpreted but they are treated literally. This means that if your hostname is *myHost*, using *myHost* or *myHost.myDomain* is not the same, even if the letter is the full qualification of the former.

- **I checked permissions, but I still got an exception related to them.**
  Try placing permissions also in the security files into the `$HOME/.aglets` directory.

- **When I execute agents I got *No integrity Check because no security domain is authenticated.***
  This is a warning message, and you can ignore it. It simply reports that you haven't set up a security domain.

- **Sometimes, running particular agents or dispatching them, I got exception related to the class loading, e.g.,**

  `ClassNotFoundException`,

  `ClassFormatException`,

  etc. How can I fix it?

  Try adding your library jar to the classpath, even in the Tahiti property. If this does not work, try adding the unjared classes to the public root.

- **Can I avoid a few references to be serialized?**
  If your agent declares a few references as *transient*, then they will never be serialized during the aglet travel. This does not mean that they will be removed by the agent, but simply that they will be "reset" to a null value on the destination, that means you have to check (and in case, recreate) references in the `onCreation(..)` method (or in your mobility listener).

- **Can an Aglets wait for a specific message before continuing its execution?**
  Yes, but it is not very simple. You can deactivate the aglet, waiting for an incoming message and then restarting its execution.

- **When I try to launch my agent I got a `ClassNotFoundException`.**
  It means that the Aglets runtime cannot find your agent class. First of all, be sure that your classes and packages are stored in the public root, that is usually `$AGLETS_HOME/public`; if it is still not working, try manually setting your classpath to include your classes, and then restart the server. If it is still not working, try making a jar of your classes and place it in the `lib` directory of your Aglets installation.

- **If I print information about an aglet proxy, I get something like `Aglet [invalid]`, and the proxy is not working. How can I fix it?**
  You should use the proxy's id, instead of storing the proxy in a complex data (such as an hash map, a vector or an array), since using

data structures to store proxies can invalidate them, since the agent situation can change. The agent and proxy id is unique, thus you can always get the proxy back starting from the id.

- **Can I use static initializers in my agents?**
  You should avoid static initializers, since if your agent migrates, the initializers will not be re-executed. In particular, this can cause problems with transient variables, thus you should absolutely place your initializers in the `onCreation(..)` method.

- **Is it possible to avoid that deactivated agents are automatically reactivated when Tahiti restarts?**
  Yes, edit the properties file and set the `cleanstart` parameter to `true`.

A.2 Configuration Files

This chapter contains examples of configuration files, that can help you to check the set-up of your Aglets platform.

The `aglets.props` file

The following is the default `aglets.props` configuration file shipped with the Aglets platform

```
# (mandatory) A path under where aglets is installed.  Set on command
# line by agletsd but can be overridden here.
#aglets.home=d:\\aglets\\aglets1_2

# (optional) A path to the directory under where ".aglets"
# directory resides.  This is also where your KEYSTORE must be.
# default: $HOME (unix) or %HOME% (win32)
#user.home=

# (optional) Location of aglets.policy file,
# default: (user.home)/.aglets/security/aglets.policy
#java.policy=

# (optional) Which protocol to use(atp or rmi)
# default: atp
#maf.protocol=atp

# (optional) Port number used by agents server.
# default: 4434
#maf.port=4434

# (optional) Host name of Finder used to register/lookup
# the locations of agents.
# default: Not used
#maf.finder.host=artemis.trl.ibm.com
```

```
# (optional) Port number of Finder used to register/lookup
# the locations of agents.
# default: 4435
#maf.finder.port=4435

# (optional) Registry name of the Finder.
# default: MAFFinder
#maf.finder.name=MAFFinder

# (optional) verbose output
# default: false
#verbose=true

# (optional) Default search path for class files.
# Windows: ';' separated path list
# Unix:    ':' separated path list
# default: (aglets.home)/public
#aglets.class.path=

# (optional) Directory which are exported to other aglets servers.
# default: (aglets.home)/public
#aglets.public.root=C:\\Aglets\\public

# (optional) Aliases used for codebase of aglets.
#aglets.public.aliases=\
# ~tai=/home/tai,\
# ~mima=/home/mima

# (optional) If false, every activities of aglets in the server
# will be allowed.
# default: true
 aglets.secure=true

# (optional) Class name of an AgletContextListner (Viewer)
#      To run server with no UI, set null.
# i.e. "aglets.viewer="
# default: com.ibm.aglets.tahiti.Tahiti
#     (ALT: com.ibm.aglets.tahiti.CommandLine)
#aglets.viewer=com.ibm.aglets.tahiti.Tahiti

# (optional)
aglets.logfile=aglets.log

# (optional)
# default: false
#aglets.cleanstart=false

# (optional) Comma(,) separated list of URLs(or class names) of aglets
# which should be created just after the server starts.
#aglets.startup=
# examples.hello.HelloAglet,\
# atp://yourhost:434/examples.hello.HelloAglet

# (optional) Resolve the domain name of the host by querying DNS server.
```

```
# default: false
#atp.resolve=false

# (optional) TCP/IP domain name of the host
#atp.domain=calivera.com

# (optional) Set server's hostname to "localhost".  This is useful if
# the host does not have any network adapter.
# default: false
#atp.offline=true

# (optional) Authenticate other servers when the server try to communicate
# each other. Servers form security domains.
# default: false
#atp.authentication=false

# (optional) Use secure random seed generation which is provided by JDK.
# If this is set to false, aglet server uses a proprietary one,
# which is insecure but fast.
# default: true
#atp.secureseed=true

# (optional) User servers IP address in server URL instead of
# logical name.  This is useful if you don't have a DNS entry.
# default: false
#atp.userip=true

# User ID for authorization.  This key must exist in your keystore.
# See keytool documentation for info on creating entry. (genkey)
aglets.owner.name=aglet_key

# Password for above user ID.  Must be same as entered as the key password
# used with keytool.
aglets.owner.password=aglets

# Keystore password.  Same as used with keytool.
aglets.keystore.password=aglets

# Logger class for ASDK.
# For log4j - org.aglets.log.log4j.Log4jInitializer
# For output to standard out - org.aglets.log.console.ConsoleInitializer
# For quiet - org.aglets.log.quiet.QuietInitializer
# Default: org.aglets.log.quiet.QuietInitializer
aglets.logger.class=org.aglets.log.log4j.Log4jInitializer
#aglets.logger.class=org.aglets.log.quiet.QuietInitializer
#aglets.logger.class=org.aglets.log.console.ConsoleInitializer
```

The `agletslog.xml` file

The following is the default `agletslog.xml` configuration file shipped with the Aglets platform

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration>

<!-- Layout does not use location info and is faster. -->
<appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%d{ABSOLUTE} %-5p [%t] %c{2} - %m%n"/>
</layout>
</appender>


<appender name="FULLINFO" class="org.apache.log4j.ConsoleAppender">
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%d{ABSOLUTE} %-5p [%t] %C{2}:%c{1} (%F:%L) - %m%n"/>
</layout>
</appender>

<category name="org" additivity="false">
<priority value="debug" />
<appender-ref ref="CONSOLE" />
</category>

<category name="com" additivity="false">
<priority value="debug" />
<appender-ref ref="CONSOLE" />
</category>


<!-- Must be last element!! -->
<root>
<priority value ="debug" />
<appender-ref ref="CONSOLE" />
</root>

</log4j:configuration>
```

Appendix B

Managing login data

As already detailed in the installation sequence, the Aglets platform generates at installation time two identities that can be used for user authentication. Their username/password information is anonymous/aglets and aglet_key/aglets. You can create a new login (i.e., a new couple username/password) or modify the password of an existing username using the `keytool` command shipped with the Java 2 platform. This chapter will show a base use of the keytool command, in order to allow you to manage certificates and logins. For a better description of the keytool capabilities, refer to the official Java documentation.

To manage information stored in the keystore you need to know the keystore password, it will be asked for each operation. The keystore password protects the whole certificate database, and should not be confused with the user's password, required to access a single certificate. Furthermore, it is not a good idea to have the keystore password identical to a certificate password.

B.1 Creating a new account

To create a new account (i.e., a username/password) start the `keytool` command specifying the new username. To keep it simple, consider the creation of an account with *myAglet* as username and *buzzle* as password. Here there is the first step of the creation:

```
luca@linux:/java/aglets/bin> keytool -genkey -alias myAglet
Enter keystore password:  aglets
```

The command asks the keystore password, that for the default Aglets keystore (i.e., the keystore installed by Ant) is *aglets*. Please note that the keystore password is echoed as plain text on the terminal, and this means you should manage the keystore away from other people's eyes.

Once you have entered the correct keystore password, the command will ask for a few pieces of information, such as name, department, and so on. All

that information are required to generate a certificate that identifies the user; that certificate will be stored in the keystore. The following is an example set of values:

```
What is your first and last name?
  [Unknown]:  Luca Ferrari
What is the name of your organizational unit?
  [Unknown]:  AgentGroup
What is the name of your organization?
  [Unknown]:  University of Modena and Reggio Emilia
What is the name of your City or Locality?
  [Unknown]:  Modena
What is the name of your State or Province?
  [Unknown]:  Italy
What is the two-letter country code for this unit?
  [Unknown]:  it
Is CN=Luca Ferrari, OU=AgentGroup, O=University of
Modena and Reggio Emilia, L=Modena, ST=Italy, C=it
correct?
  [no]:  yes
```

Finally, keytool will ask you the password to use for the above new username. *Be careful when typing the password, since it will be asked once (not twice as many password programs do) and will be echoed as plain text on the terminal.*

```
Enter key password for <myAglet>
        (RETURN if same as keystore password):  buzzle
luca@linux:/java/aglets/bin>
```

When the keytool program finishes, the command prompt is displayed. Now you can use the new couple of username and password to login in the Aglets platform.

B.2 Changing the password of an existing account

To change the password of an existing username, use the `keypasswd` option of the keytool command. Suppose that you want to change the password of the username *myAglet*, the following is what you have to do:

```
luca@linux:/java/aglets/bin> keytool -keypasswd -alias myAglet
Enter keystore password:  aglets
```

First the command will ask you the password of the whole keystore, for the default installation it is *aglets*. After that, the old password is required:

```
Enter key password for <myAglet>buzzle
```

Finally, the new password is required. Please note that, even if here the password is asked twice to catch typing errors, the password value is also printed on the screen, and this requires that nobody is watching "over your shoulder".

```
New key password for <myAglet>: buzzle2
Re-enter new key password for <myAglet>: buzzle2
luca@linux:/java/aglets/bin>
```

Now you have changed the password of the specified username, and can use the new password to login in the Aglets platform.

B.3 Deleting an account

If you want to delete a whole account, you can use the `delete` option of the keytool command. For example, if you want to delete the *myAglet* account, do the following:

```
luca@linux:/java/aglets/bin> keytool -delete -alias myAglet
Enter keystore password:  aglets
luca@linux:/java/aglets/bin>
\end{center}
```

Be aware of what you are doing, since the command is very silent! As you can see, only the keystore password is required, after that the deletion happens without asking any user confirmation.

B.4 Listing the content of the keystore

To view which certificates are handled by the current keystore, simply do:

```
keytool -list
Enter keystore password:  aglets
```

that will print something like the following:

```
Keystore type: jks
Keystore provider: SUN

Your keystore contains 2 entries

anonymous, Sep 6, 2004, keyEntry,
Certificate fingerprint (MD5):
  78:13:74:36:92:F4:51:04:56:36:BB:41:CC:3E:96:94
aglet_key, Sep 6, 2004, keyEntry,
Certificate fingerprint (MD5):
  B6:8D:E5:6E:42:19:2F:AB:20:25:12:32:99:8B:77:09
```

The output above shows that only two certificates are present in my current keystore, and that the username to access those certificates are *anonymous* and *aglet_key*. The above certificates are created by the Ant installation.

B.5 User's Configuration Files

Aglets stores, for each users, a few configuration files in the user's home directory. In particular, Aglets will create a directory called `.aglets`, containing a few subdirectories as shown below:

- `cache` it will be used by a running platform to cache information about agents, and agents themselves (for example when they will be deactivated).
- `security` it contains a policy file and the secrets created with Tahiti.
- `spool` contains a directory for each combination host/port the platform has been bound to. In each directory, a few files used by the run-time system (such as platform properties) are stored.
- `users` contains a directory for each user (keystore alias) who has started Tahiti. Each directory stores preferences of the user, such as the Tahiti window size, agent lists, etc.

Appendix C

The IBM Public License - version 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS IBM PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

DEFINITIONS

"Contribution" means:

1. in the case of International Business Machines Corporation ("IBM"), the Original Program, and 2. in the case of each Contributor, 1. changes to the Program, and 2. additions to the Program; where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means IBM and any other entity that distributes the Program.

"Licensed Patents " mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Original Program" means the original version of the software accompanying this Agreement as released by IBM, including source code, object code and documentation, if any.

"Program" means the Original Program and Contributions.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

GRANT OF RIGHTS

1. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form. 2. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder. 3. Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program. 4. Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

1. it complies with the terms and conditions of this Agreement; and 2. its license agreement: 1. effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose; 2. effectively excludes on behalf of all Contributors all liability for damages, including

direct, indirect, special, incidental and consequential damages, such as lost profits; 3. states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and 4. states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

1. it must be made available under this Agreement; and 2. a copy of this Agreement must be included with each copy of the Program.

Each Contributor must include the following in a conspicuous location in the Program:

Copyright (C) 1996, 1999 International Business Machines Corporation and others. All Rights Reserved.

In addition, each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and war-

ranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in

a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

IBM may publish new versions (including revisions) of this Agreement from time to time. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. No one other than IBM has the right to modify this Agreement. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.