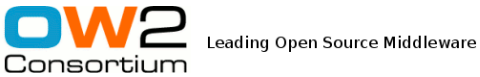
An Open Source Middleware For Parallel, Distributed, Multicore Computing

ProActive Windows Agent

Version 2.3

The OASIS Research Team and ActiveEon Company



Generated on 2011-01-12

ProActive Windows Agent v2.3 Documentation

Legal Notice

Contact: proactive@ow2.org or contact@activeeon.com

Mailing List

proactive@ow2.org

Mailing List Archive

http://www.objectweb.org/wws/arc/proactive

Bug-Traking System

http://bugs.activeeon.com/browse/PROACTIVE

Contributors and Contact Information

Team Leader

Denis Caromel
INRIA 2004, Route des Lucioles, BP 93
06902 Sophia Antipolis Cedex
France
phone: +33 492 387 631
fax: +33 492 387 971
e-mail: Denis.Caromel@inria.fr

Contributors from OASIS Team	Contributors from ActiveEon Company
Brian Amedro	Vladimir Bodnartchouk
Francoise Baude	Arnaud Contes
Francesco Bongiovanni	Cédric Dalmasso
Florin-Alexandru Bratu	Christian Delbé
Viet Dung Doan	Arnaud Gastinel
Yu Feng	Jean-Michel Guillaume
Imen Filali	Olivier Helin
Fabrice Fontenoy	Clément Mathieu
Ludovic Henrio	Maxime Menant
Fabrice Huet	Emil Salageanu
Elaine Isnard	Jean-Luc Scheefer
Vasile Jureschi	Mathieu Schnoor
Muhammad Khan	
Virginie Legrand Contes	
Eric Madelaine	
Elton Mathias	
Paul Naoumenko	
Laurent Pellegrino	
Guilherme Peretti-Pezzi	
Franca Perrina	
Marcela Rivera	
Christian Ruz	
Bastien Sauvan	
Oleg Smirnov	
Marc Valdener	
Fabien Viale	

Former Important Contributors

Laurent Baduel (Group Communications)
Vincent Cave (Legacy Wrapping)
Alexandre di Costanzo (P2P, B&B)
Abhijeet Gaikwad (Option Pricing)
Mario Leyton (Skeleton)
Matthieu Morel (Initial Component Work)
Romain Quilici
Germain Sigety (Scheduling)
Julien Vayssiere (MOP, Active Objects)

Table of Contents

List of Figures

Chapter 1. ProActive Windows Agent

1.1. Context

In distributed systems, desktop computers can be an important source of computational power. Moreover, one of the definitions of grid stands for a type of parallel and distributed system that enables the sharing, selection, and aggregation of resources distributed across multiple administrative domains based on theirs (resources) criteria: availability, capacity and performance. In such a context the main purpose of the ProActive Windows Agent is to make the configuration of these criteria achievable (schedule working plan and limit the shared amount of RAM and CPU).

1.2. Functionalities

The ProActive Windows Agent is a Windows Service: a long-running executable that performs specific functions and which is designed not to require user intervention. The agent is able to create a ProActive computational resource on the local machine. This resource will be provided to ProActive applications (such as Resource Manager) according to a user defined schedule. A tray icon shows the state of the agent and allows the user to start/stop it, or modify its schedule. The ProActive Windows Agent does not interfere with the day-to-day usage of the desktop Windows machine.

Understanding of ProActive basic concepts is required for the comprehension of the following description.

The core client is a process which:

- Loads the user's configuration.
- Creates schedules according to the working plan specified in the configuration.
- Spawns a JVM process that will run a specified java class depending on the selected connection type. 3 types of connections are available:
  - **Local Registration** - The specified java class will create a ProActive local node as a computational resource and register it locally.
  - **Resource Manager Registration** - The specified java class will create a ProActive local node as a computational resource and register it in the specified Resource Manager, thus being able to execute java or native tasks received from the Scheduler.

    It is is important to note that a JVM process running tasks can potentially spawn child processes.
  - **Custom** - The user can specify its own java class.
- Watches the spawned JVM process in order to comply to the following limitations:
  - **RAM limitation** - The user can specify a maximum amount of memory allowed for a JVM process and its children. If the limit is reached, then all processes are automatically killed.
  - **CPU limitation** - The user can specify a maximum CPU usage allowed for a JVM process and its children. If the limit is exceeded by the sum of CPU usages of all processes, they are automatically throttled to reach the given limit.
- Restarts the spawned JVM process in case of failures with a timeout policy.

Warning: Do not confuse agent and node deployment!

The role of the agent is to create a node (locally) and to register it to the resource manager (in case of a resource manager connection) which can be distant. The deployment can be considered as the opposite process: the resource manager creates nodes on remote machine and register them to itself. So **when you use the ProActive Windows Agent, you do not need (and do not have) to use a node deployment**.

1.3. Example

This simple example illustrates how to create an active object on a ProActive node created with the ProActive Windows Agent. We suppose that the following conditions are verified:

- The agent is installed (see Section 1.4, “Installation” for installation) on a host with IP address equal to 192.168.1.62

- It is configured with the **RMI registration** as selected connection type and "toto" as node name.

Once the agent is started, the following java code gets a reference on a remote node and creates an active object on the remote node identified by the url "rmi://192.168.1.62:1099/toto"

Note

This piece of code should be executed on a seperate JVM. Moreover, this test is more useful for the RMI Registration connection type since for the Resource Manager one, you can directly see whether the node has been started.

```
import org.objectweb.proactive.api.PAActiveObject;
import org.objectweb.proactive.core.node.Node;
import org.objectweb.proactive.core.node.NodeFactory;


public class Test {

   public static void main(String[] args) {

      try {

         // Note that the port is 1100 and not 1099 like those used by the
         // resource manager. The initial port is incremented to ensure that
         // each runtime uses a unique port.
         // Thus, this port corresponds to the port of the first runtime.
         // If a second runtime has been launched, it would use the port 1101,
         // and so on and so forth...

         final Node n = NodeFactory.getNode("rmi://192.168.1.62:1100/toto");

         System.out.println("Nb of active objects on the remote node: " + n.getNumberOfActiveObjects() +
            " local runtime hashcode " + Runtime.getRuntime().hashCode());

         final Test stubOnTest = (Test) PAActiveObject.newActive(Test.class.getName(), null, n);

         final String receivedMessage = stubOnTest.getMessage();

         System.out.println("Nb of active objects on the remote node: " + n.getNumberOfActiveObjects() +
            " received message: " + receivedMessage);
      } catch (Throwable t) {
         t.printStackTrace();
      }
   }

   public Test() {
   }

   public String getMessage() {
      return "A message from " + Runtime.getRuntime().hashCode();
   }

}
```

The output is:

```
--> This ClassFileServer is listening on port 2027
Detected an existing RMI Registry on port 1099
Nb of active objects on the remote node: 0 local runtime hashcode 6588476
Generating class : pa.stub.org.ow2.proactive.resourcemanager.utils._StubTest
Nb of active objects on the remote node: 1 received message: A message from 26670261
```

1.4. Installation

The ProActive Windows Agent installation pack is available on the official ProActive website[1]. Follow the links and get the latest version.

Note

- .Net framework v3.5 or later should be installed on your system. If not, the installer will ask you to install it. Go to http://www.microsoft.com/downloads and download Microsoft .NET Framework Version 3.5 (or later) Redistributable Package.
- Visual C++ 2008 (or later) Redistributable Package is also needed. The installer will install it for you, if not found.

- Run the **setup.exe** file.
- Accept the licence agreement.
- Select the components you want to install.
- Choose the installation folder of the ProActive agent.
- Then, the following windows will appear:

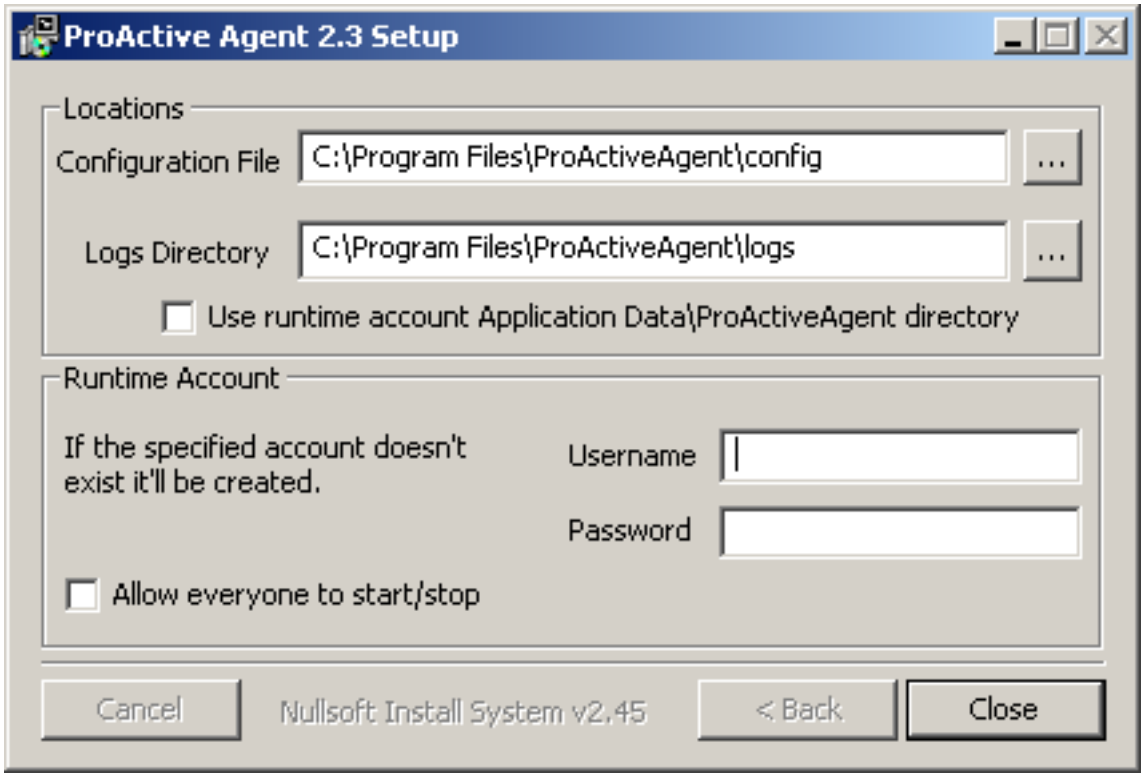


Figure 1.1. Configuration during installation

[1] http://proactive.inria.fr/

- Specify the directory that will contain the configuration file named PAAgent-config.xml, note that if this file already exists in the specified directory it will be re-used.
- Specify the directory that will contain the log files of the ProActive Agent and the spawned runtimes.
- Specify an existing, local account under which the ProActive Runtime(s) will be spawned. It is highly recommended to specify an account that is not part of the Administrators group to isolate the ProActive Runtime and reduce security risks.

  If the specified account does not exist the installation program will prompt the user to create a non-admin account with the required privileges.

  Note that the ProActive Agent service is installed under LocalSystem[2] account it cannot access remote UNC (Universal Naming Convention) path, the service account can be changed using the "services.msc" utility. ("Control Panel->Administrative Tools->Services")
- If you want that any non-admin user (except guest accounts) to be able to start/stop the ProActive Agent service check the "Allow everyone to start/stop" box. If this option is checked the installer will use the SubInACL[3] tool. If the tool is not installed in the "Program Files\Windows Resource Kits\Tools" directory the installer will try to download its installer from the official Microsoft page.

- The installer will check whether the selected user account has the required privileges. If not follow the steps to add these privileges:
  - In the 'Administrative Tools' of the 'Control Panel', open the 'Local Security Policy'.
  - In 'Security Settings', select 'Local Policies' then select 'User Rights Assignments'.
  - Finally, in the list of policies, open the properties of 'Replace a process-level token' policy and add the needed user. Do the same for 'Adjust memory quotas for a process'. For more information about these privileges refer to the official Microsoft page[4].

At the end of the installation, the ProActive Agent Control utility should be started. This next section explains how to configure it.

To uninstall the ProActive Windows Agent, simply run "Start->Programs->ProActiveAgent->Uninstall ProActive Agent".

1.5. Configuration

Launch "Start/Programs/ProActiveAgent/AgentControl" program or click on the notify icon if the "Automatic launch" is activated. Double click on the tray icon to open the ProActive Agent Control window. The following window will appear:

[2] http://msdn.microsoft.com/en-us/library/ms684190(VS.85).aspx
[3] http://www.microsoft.com/downloads/en/details.aspx?FamilyID=e8ba3e56-d8fe-4a91-93cf-ed6985e3927b
[4] http://msdn.microsoft.com/en-us/library/bb530716(VS.85).aspx

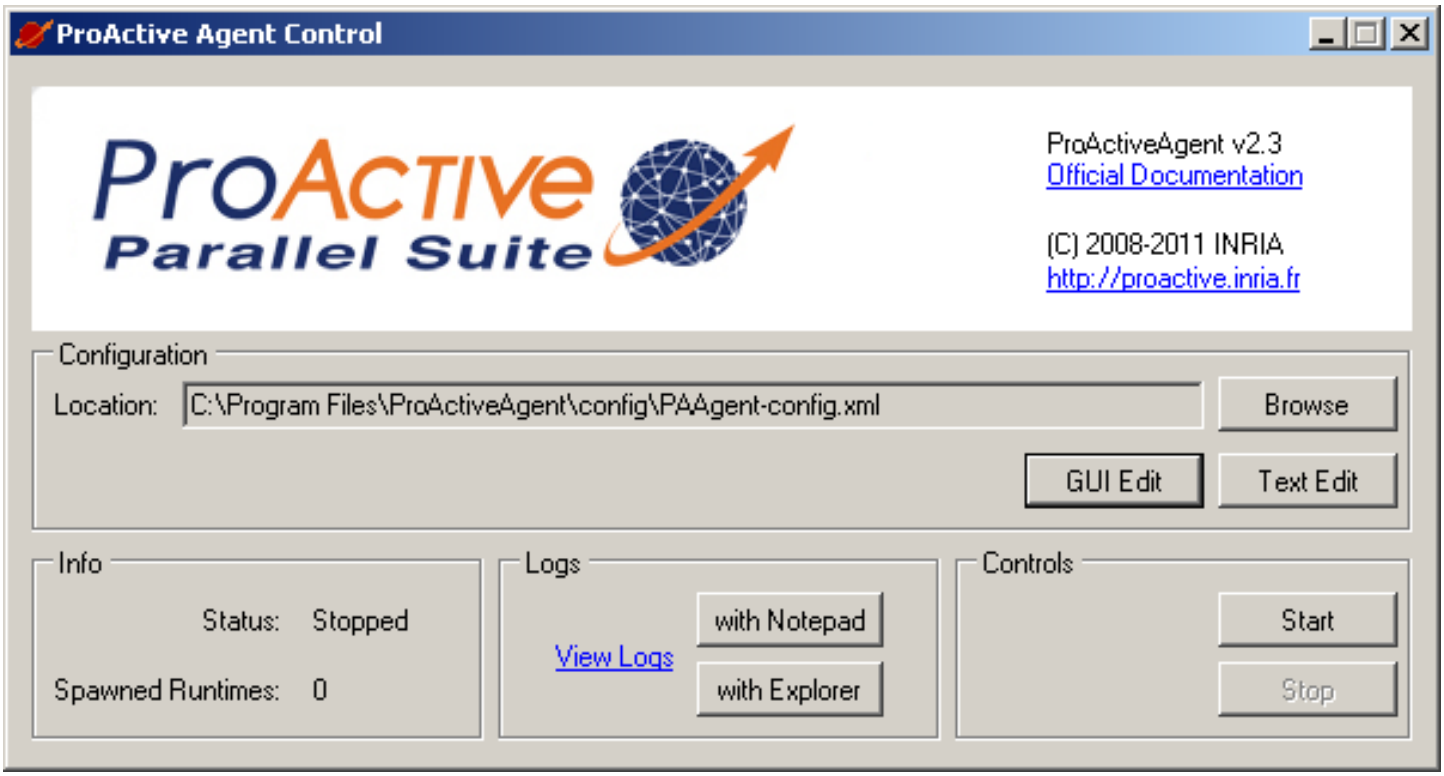


Figure 1.2. ProActive Agent Control window

From the ProActive Agent Control window, the user can load a configuration file, edit it, start/stop the service and view logs. A GUI for editing is provided (explained below). Even if it is not recommended, you can edit the configuration file by yourself with your favorite text editor.

Warning

The configuration file format has changed since version 2.2 which is backward compatible with previous version files. If the user edits such a configuration and saves the modifications the file will be overwritten with the new format.

Clicking on "GUI Edit", the following windows appears:

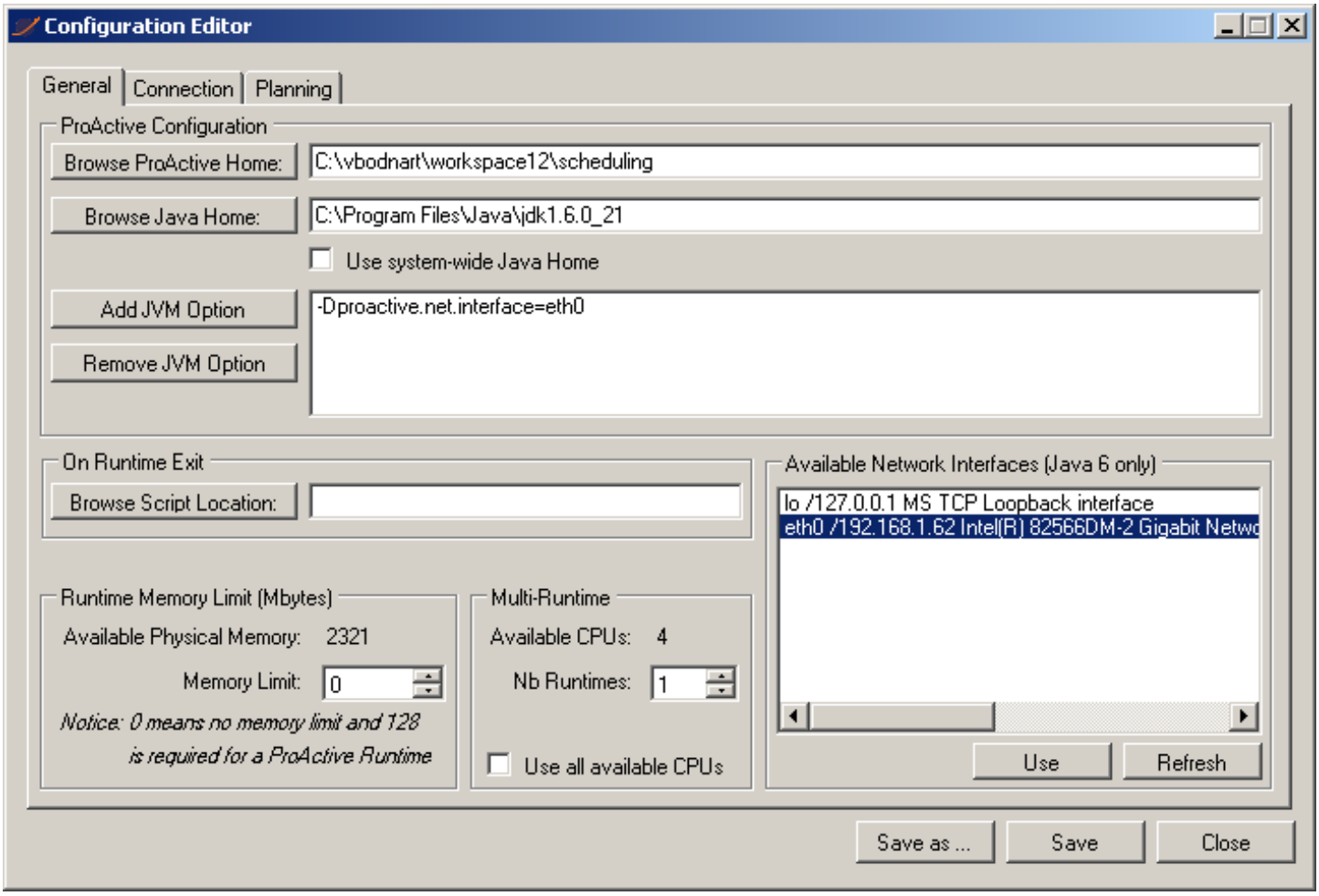


Figure 1.3. Configuration Editor window - General Tab

In the general tab, the user can specify:

- The ProActive (or Scheduler) location.
- The JVM location (usually something like C:\Program Files\Java\jdk1.6.0_12).
- The number of Runtimes (the number of spawned JVMs).
- The JVM options.

  Note that if the parameter contains ${rank}, it will be dynamically replaced by the Runtime rank starting from 0.
- The "On Runtime Exit" script. A script executed after a Runtime exits. This can be useful to perform additional cleaning operation.

  Note that the script receives as parameter the PID of the Runtime.
- The user can set a memory limit that will prevent the spawned processes to exceed a specified amount of RAM. If a spawned process or its child process requires more memory, it will be killed as well as its child processes.

  Note that this limit is disabled by default (0 means no limit) and a ProActive Runtime will require at least 128 MBytes.
- It is possible to list all available network interfaces by clicking on the "Refresh" button and add the selected network interface name as a value of the "proactive.net.interface" property by clicking on "Use" button. See the ProActive documentation for further information.

Clicking on the "Connection" tab, the windows will look like this:

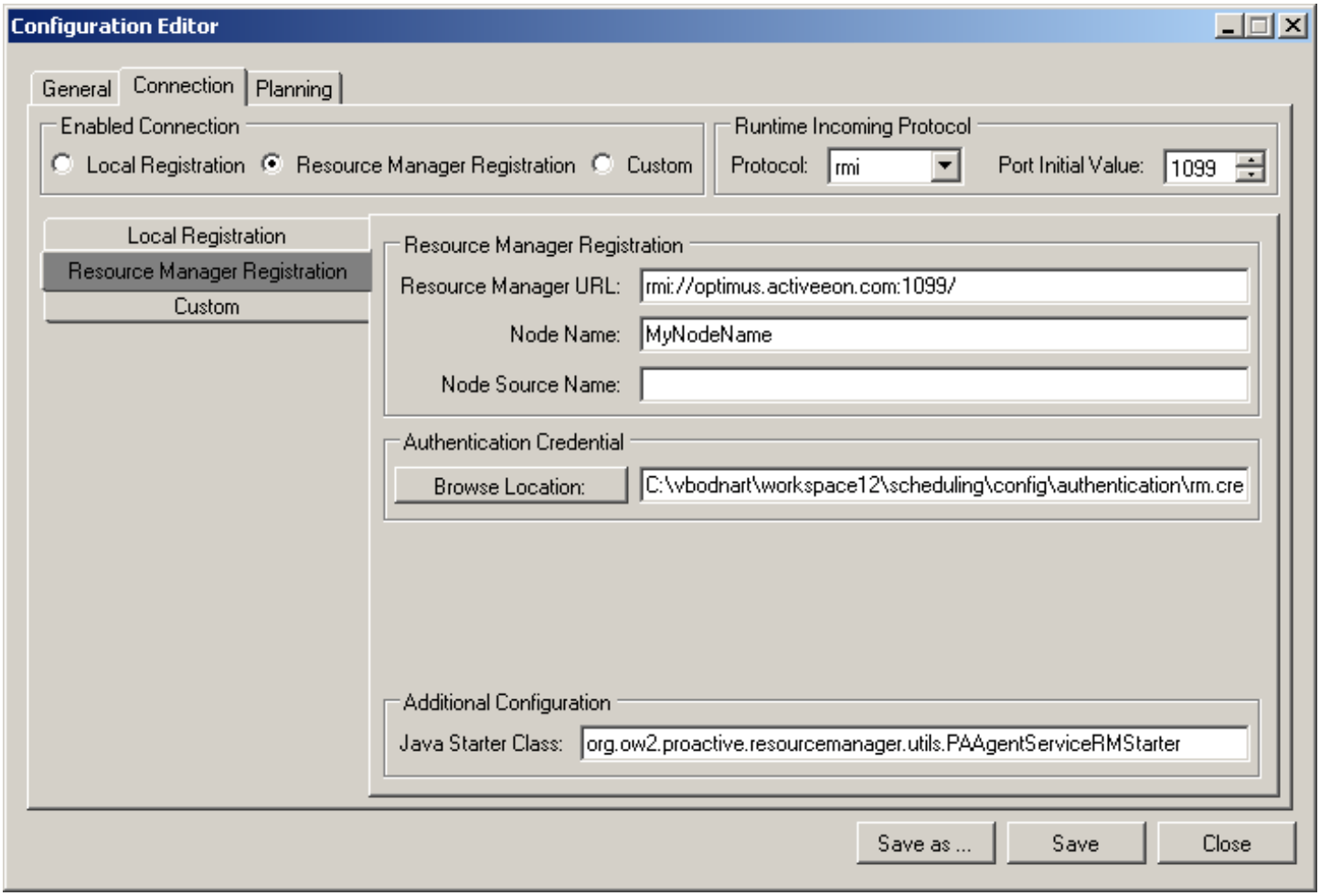


Figure 1.4. Configuration Editor window - Connection Tab (Resource Manager Registration)

In the "Connection" tab, the user can select between three types of connections:

- **Local Registration** - creates a local ProActive node and registers (advertises) it in a local RMI registry. The node name is optional.
- **Resource Manager Registration** - creates a local ProActive node and registers it in the specified Resource Manager. The mandatory Resource Manager's url must be like 'protocol://host:port/'. The node name and the node source name are optional. Since the Resource Manager requires an authentication, the user specifies the file that contains the credential. If no file is specified the default one usually located in "%USERPROFILE%\.proactive\security" folder is used.
- **Custom** - the user specifies its own java starter class and the arguments to be given to the main method. The java starter class must be in the classpath when the JVM process is started.

The user can specify the protocol (rmi or http) to be used by the Runtime for incoming communications.

To ensure that a unique port is used by a Runtime, the initial port value will be incremented for each Runtime and given as value of the "-Dproactive.SELECTED_PROTOCOL.port" JVM property. If the port chosen for a runtime is already used, it is incremented until an available port number is found.

Finally, clicking on the "Planning" tab, the windows will look like this:

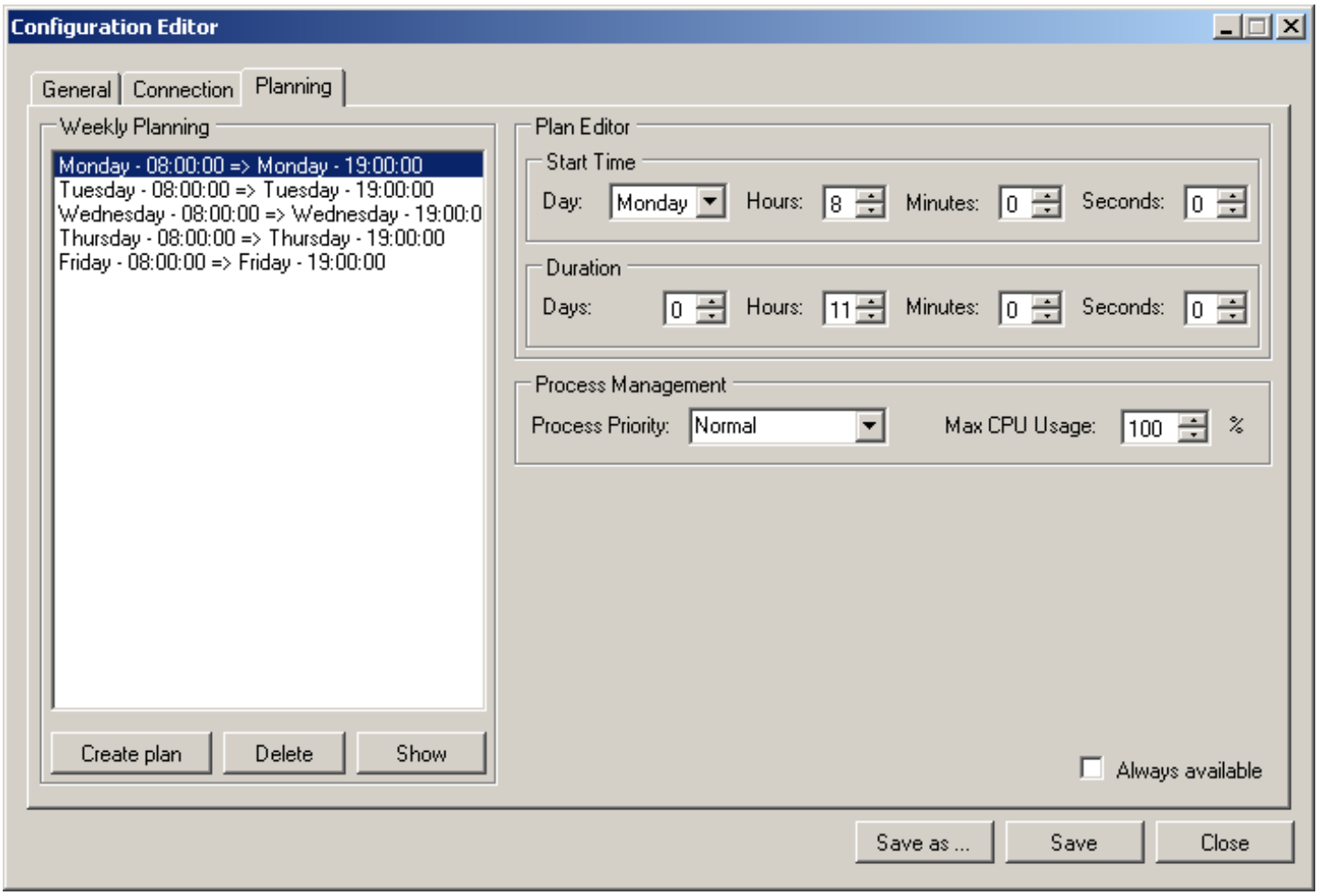


Figure 1.5. Configuration Editor window - Planning Tab

In the Planning Tab, depending on the selected connection type, the agent will initiate it according to a weekly planning where each plan specifies the connection start time as well as the working duration. The agent will end the connection as well as the ProActive Runtime process and its child processes when the plan duration has expired.

Moreover, it is possible to specify the JVM process Priority and it's CPU usage limit. The behavior of the CPU usage limit works as follows: if the JVM process spawns other processes, they will also be part of the limit so that if the sum of CPU% of all processes exceeds the user limit they will be throttled to reach the given limit. Note that if the Priority is set to RealTime the CPU % throttling will be disabled.

The "Always available" makes the agent to run permanently with a Normal Priority and Max CPU usage at 100%.

1.6. Start the agent

Once you have configured the agent, you can start it clicking on the "Start" button of the ProActive Agent Control window. However, before that, you have to ensure that a resource manager has been started on the address you specified in the agent configuration. **You do not need to start a node since it is exactly the job of the agent**.

Once started, you may face some problems. You can realise that an error occurred by first glancing at the color of the agent tray icon. If everything goes right, it should keep the blue color. If its color changes to yellow, it means that the agent has been stopped. To see exactly what happened, you can look at the runtime log file located into the agent installation directory and named **Executor<runtime number>Process-log.txt**.

The main troubles you may have to face are the following ones:

- You get an access denied error: this is probably due to your default java.security.policy file which cannot be found. If you want to specify another policy file, you have to add a JVM parameter in the agent configuration. A policy file is supplied in the scheduling directory. To use it, add the following line in the JVM parameter box of the agent configuration (Figure 1.3, “Configuration Editor window - General Tab ”):

  ```
  -Djava.security.poly=<scheduler directory>/config/security.java.policy-client
  ```

- You get an authentication error: this is probably due to your default credentials file which cannot be found. In the "Connection" tab of the Configuration Editor (Figure 1.4, “Configuration Editor window - Connection Tab (Resource Manager Registration)”), you can choose the credentials file you want. You can select, for instance, the credentials file located at <scheduler directory>/config/authentication/scheduler.cred or your own credentials file.
- The node seems to be well started but you cannot retrieve it in a Java code for example: in this case, make sure that the port number is the good one. Do not forget that the runtime port number is incremented from the initial resource manager port number. You can see exactly on which port your runtime has been started looking at the log file describing above.

1.7. Other information

The agent was successfully tested on Windows XP, 2003, Windows Vista and Windows 7. Some problems with service installation can occur on Windows NT.

The ProActive Windows Agent is written in C# and uses .Net Framework 3.5

Third-party libs used:

- C# JobObjectWrapper Api (JobManagement.dll) under Microsoft Permissive License (Ms-PL) v1.1
- C# Log4Net Api (log4net.dll - 1.2.10.0) under Apache License v2.0