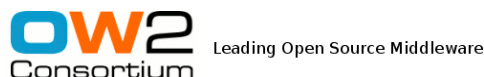# ProActive Parallel Suite

An Open Source Middleware For Parallel, Distributed, Multicore Computing

ProActive Windows Agent

The OASIS Research Team and ActiveEon Company

INRIA

Université Nice SOPHIA ANTIPOLIS

CNRS CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

OW2 Consortium Leading Open Source Middleware

ProActive v2009-10-13 Documentation
An Open Source Middleware For Parallel, Distributed, Multicore Computing: ProActive Windows Agent

The OASIS Research Team and ActiveEon Company

Legal Notice

This document is extracted from the official ProActive documentation, for further information please refer to the complete ProActive documentation.

ProActive is a GRID Java middleware, distributed under the GPL2, or any later version, license.

Table of Contents

List of Figures

Part I. ProActive Windows Agent

Table of Contents

Chapter 1. ProActive Windows Agent

1.1. Context

In distributed systems, desktop computers can be a 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 their (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:
  - **RMI Registration** - The specified java class will create a ProActive local node as a computational resource and register it in a local RMI registry.
  - **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's 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 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.

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 agent is already:

- Installed (see next chapter for installation) on a host with ip address 192.168.1.62
- 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"

```java
public class Test {

public static void main(String [] args){
```

```java
 try{
  final Node n = NodeFactory.getNode("rmi://192.168.1.62:1099/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. Follow the links and get the latest version.

Note the following requirements:

- .Net framework v3.5 or later should be installed on your system. If it is 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 needed. The installer will install the packages if not found.
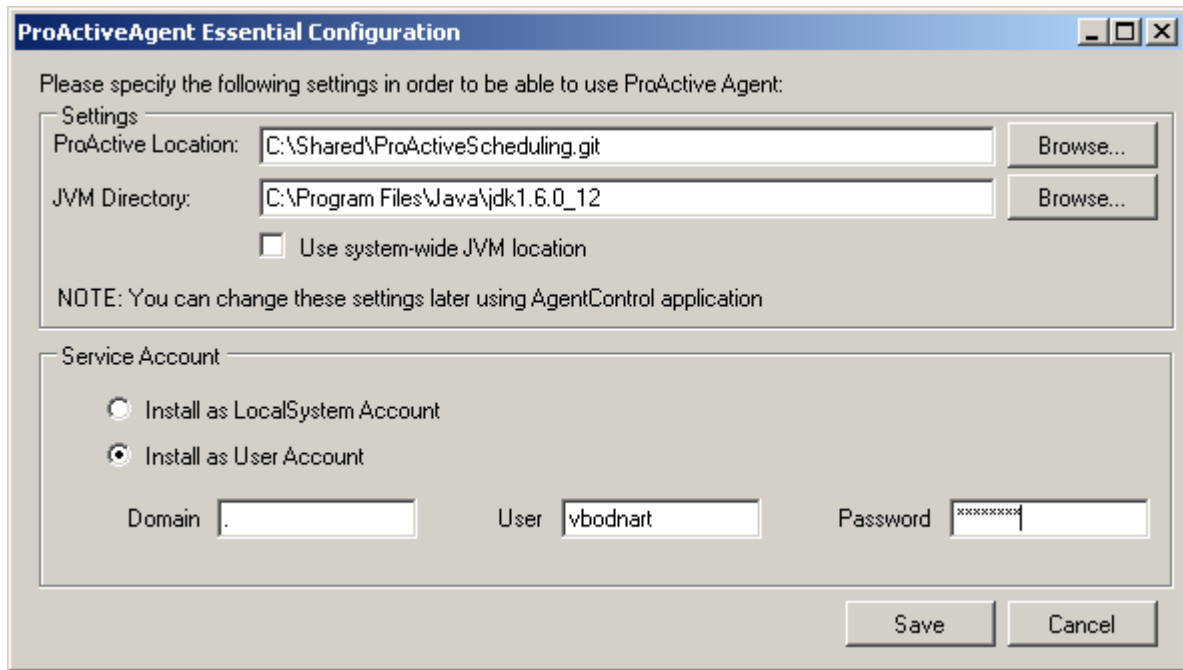
Figure 1.1. Configuration during installation

- Run the **setup.exe** file.
- Enter the Java location, usually **C:\Program Files\Java\jdk1.6.0_12** .
- Enter the ProActive location, if you are using the ProActive Scheduler just provide the Scheduler location.
- Select the service account, if you want to install the agent under a specific user account enter the domain ("." or hostname is the local domain), user and location.

    Note that you can change these settings at anytime in the "services.msc" utility. ("Control Panel->Administrative Tools->Services")
- The installer will check if the selected user account have the log on service right assignment.

    In not follow the following steps to add the log on service right assignment:
    - 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 'Log on as a service' policy and add the needed user.

At the end of the installation the user can choose to run the ProActive Agent Control utility. It is explained in the next section.

To uninstall the ProActive Windows Agent simply run "Start/Programs/ProActiveAgent/uninstall".

1.5. Usage

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.
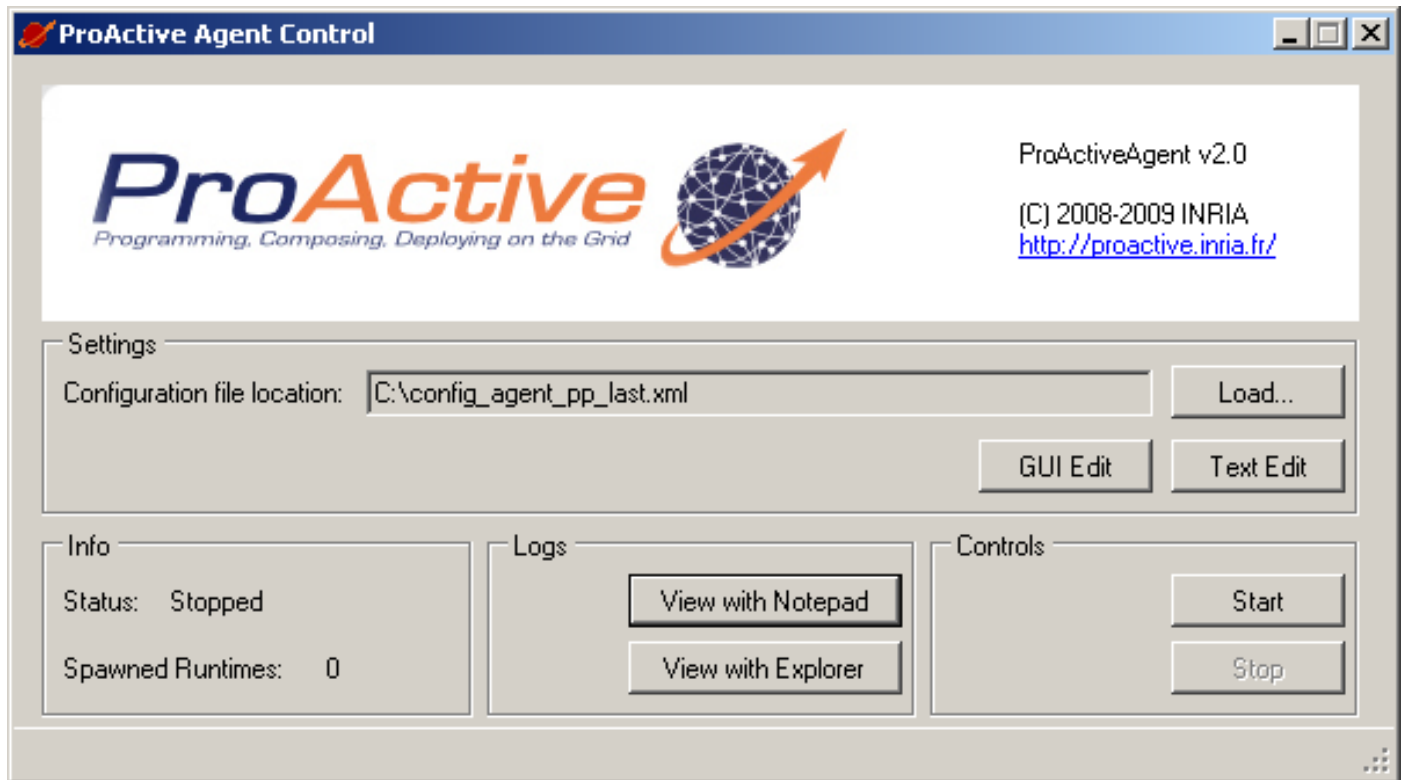
Figure 1.2. ProActive Agent Control window

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 your self with your favorite text editor.

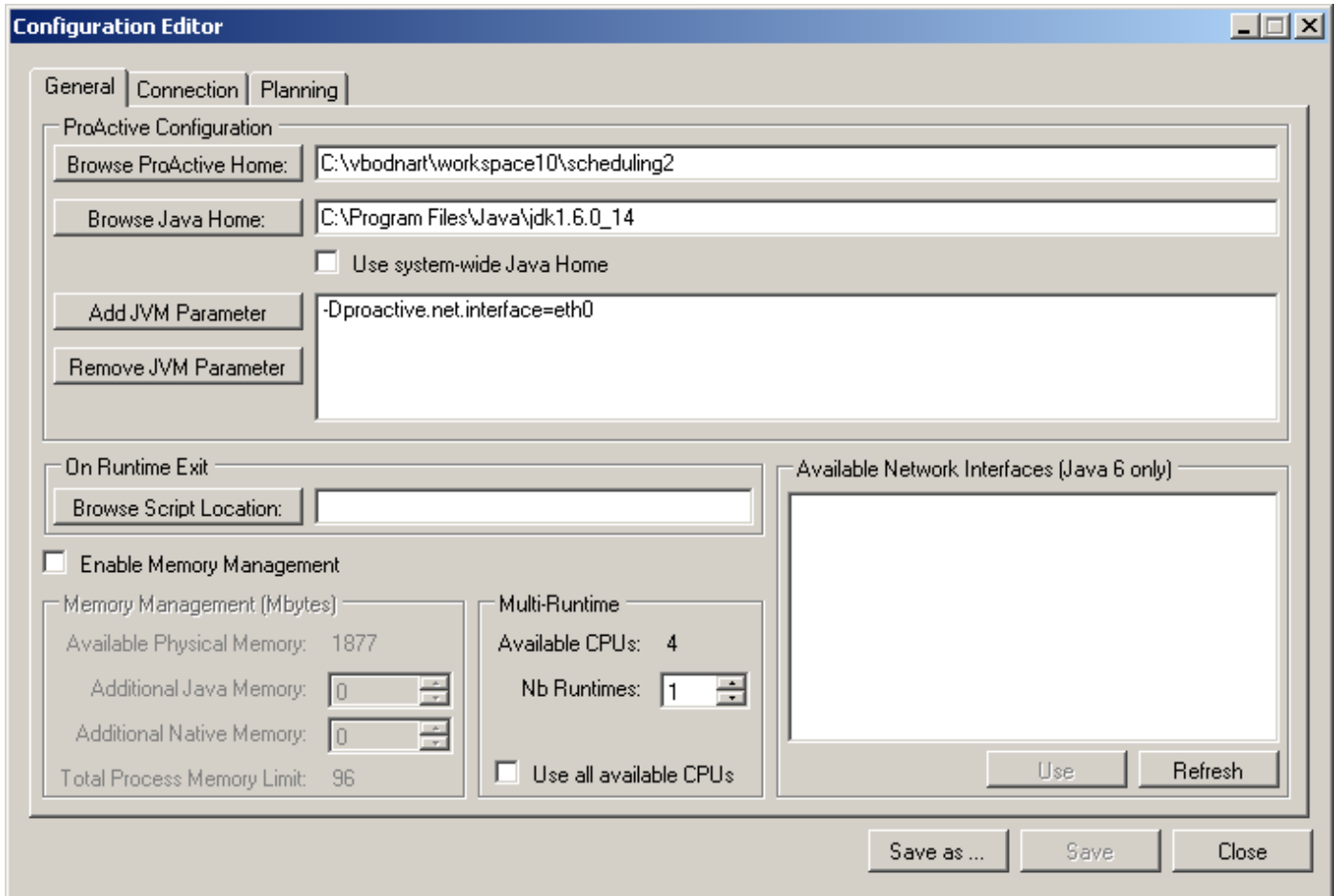Note that configuration files are not compatible between major releases.

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 parameters.

  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 enable the memory management (disabled by default) in order to limit the amount of RAM used by the jvm process.

  By default the limit is set to 96 mbytes, thus each time the jvm process asks more memory it will be killed as well as its child processes.

  The user can extend this limit by giving more memory for java tasks (executed by the jvm process) or for native tasks (child processes spawned by the jvm process).

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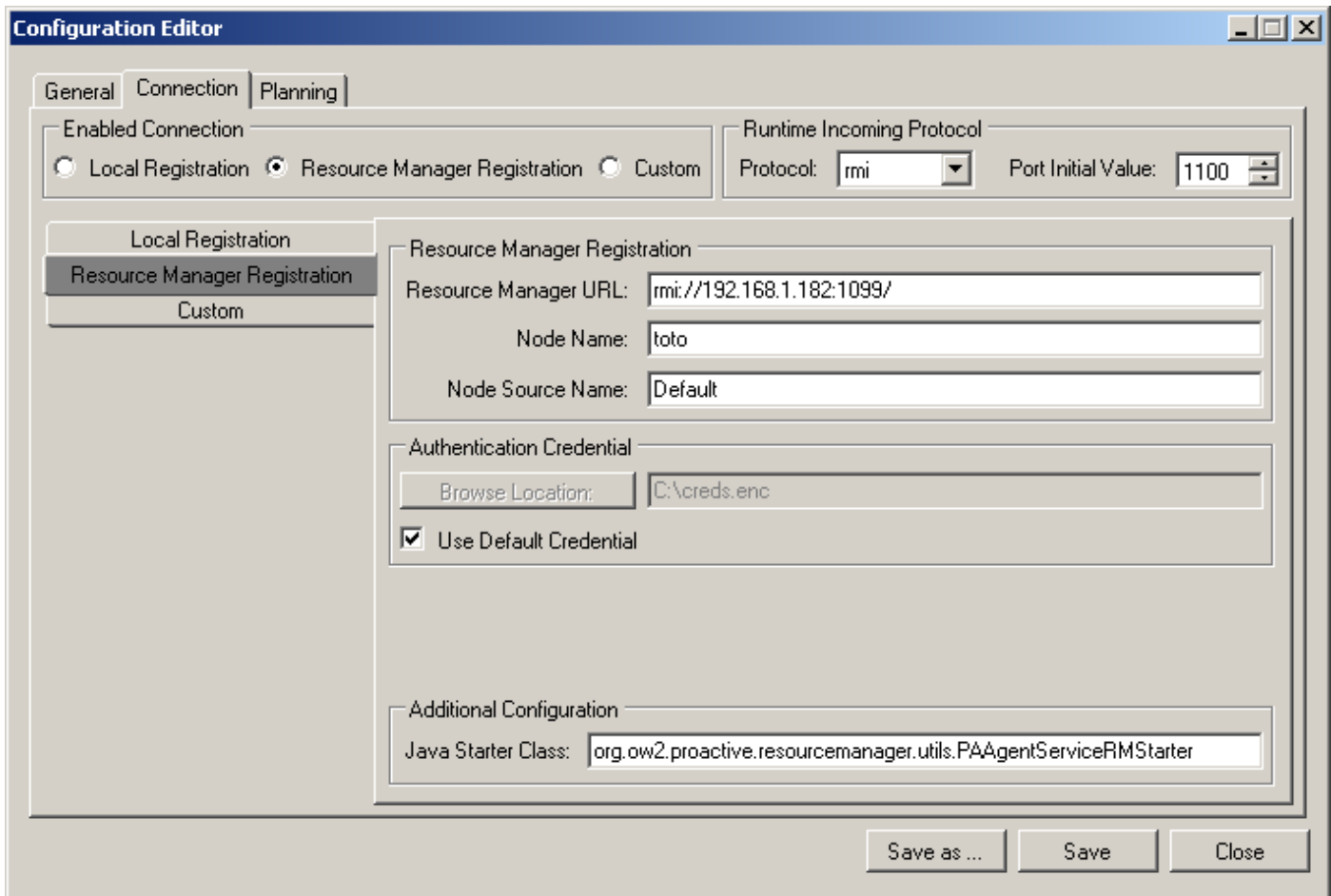
Figure 1.4. Configuration Editor window - Connection Tab (Resource Manager Registration)

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

- **RMI 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 specify the file that contains the credential or use the default one usually located in "%USERPROFILE%\.proactive\security" folder.
- **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 port initial value, if available, will be incremented for each Runtime and given as value of the "-Dproactive.SELECTED_PROTOCOL.port" jvm property.
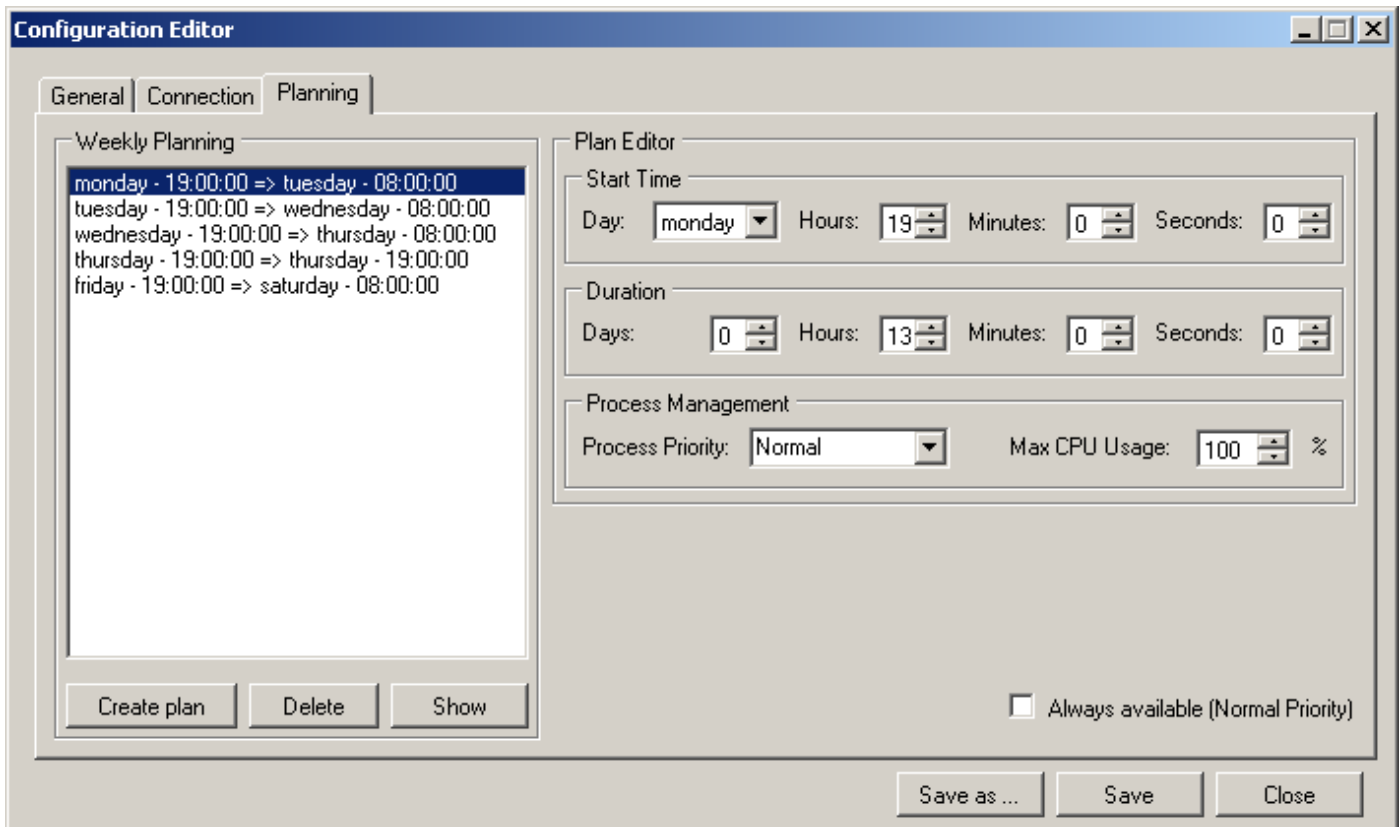
Figure 1.5. Configuration Editor window - Planning Tab

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. Other information

The agent was successfully tested on Windows XP, 2003, Windows Vista. 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