



---

**Documentation Set**

---

**Links**

---

© Copyright (C) 1997-2011 INRIA/University of Nice-Sophia Antipolis/ActiveEon [Terms of Use](#)



# Getting Started

---

[What Is Matlab Connector Toolbox?](#)

An overview of the features, functions, and uses of Matlab Connector Toolbox.

[Installation](#)

Instructions on how to deploy the ProActive Scheduler for Matlab usage.

[Starting And Connecting](#)

Instructions on how to connect to ProActive Scheduler.

[Monitoring](#)

Instructions on how to use ProActive Scheduler monitoring tools.

[Running Matlab functions remotely](#)

Instruction on the use of Matlab Connector Toolbox to run matlab code remotely.

[Disconnected mode](#)


Description of Matlab Connector Toolbox's disconnected mode.

[Configuring PAsolve behavior and Debugging](#)

How to configure Matlab Connector Toolbox behavior.

[Using Matlab Connector Toolbox Simulink GUI](#)

A detailed description of the use of ProActive Scheduler's Simulink GUI.

What Is Matlab Connector Toolbox? 



# What Is Matlab Connector Toolbox?

On this page...

[Introduction](#)

[Can I Use Matlab Connector Toolbox?](#)

[Which versions of Matlab and Operating Systems are compatible?](#)

[What are the current limitations of Matlab Connector Toolbox?](#)

## Introduction

*Matlab Connector Toolbox* is a matlab toolbox designed to provide interaction with the *ProActive Scheduler*, part of *ProActive Parallel Suite*.

*ProActive Parallel Suite* is an innovative Open Source solution (OW2) for parallel, distributed, multi-core computing.

ProActive features Java Parallel Programming seamlessly integrated with Scheduling and Resource Management. ProActive simplifies the programming and execution of parallel applications on Linux, Windows and Mac, together with the management of resources such as Desktop, Servers, Clusters, Enterprise GRIDs and Clouds.

The objectives of *Matlab Connector Toolbox* are to provide you with tools that:

- Allow you to run Matlab functions on remote computers.
- Do not block the local Matlab session while remote results are being produced.
- Allow you to seamlessly retrieve results when you need them, just as if the functions were run locally.
- Provide you detailed remote log/output information, altogether with errors if any occurred.
- Allow automatic source transfer, data file transfer, transfer of local matlab workspace and other configurable options.
- Allow fault-tolerant/disconnected sessions where intermediate results are persistent.

In comparison with the *Matlab Parallel Computing Toolbox*, the ProActive Connector :

- don't offer SPMD features or distributed matrices.
- don't offer a p-mode feature where interactive matlab sessions remain alive between executions.
- allows the deployment of matlab instances on more than one machine and is in that sense like the Parallel Computing Toolbox combined with the Distributed Computing Server.
- works well with multiple matlab installations and heterogeneous operating systems.
- add some extra features like fault-tolerant/disconnected sessions and resource selection.

[Back to Top](#)

## Can I Use Matlab Connector Toolbox?

Matlab Connector Toolbox is designed for both beginners and advanced users. There are basically two usage roles:

- The **scheduler administrator**: responsible for ProActive Scheduler installation on every machines, and the administration of ProActive Scheduler.
- The **toolbox user**: who will simply use the installed toolbox to run matlab code in parallel.

The basic requirements for the **administrator** is to have a standard Information Technology knowledge on windows or linux, for example:

- Start programs on windows or linux using the command line shell (sh or bat).
- Change environment variables on windows or linux.
- Be confident with XML syntax.
- Understand network protocols and principles.
- Have a basic knowledge of Security principles such as asymmetric key pairs, and accounts management frameworks such as Ldap.
- Have a basic knowledge of the Java Programming Language.
- Have a basic knowledge of scripting languages such as Javascript, Python or Ruby.

The basic requirement for the **toolbox user** is to be able to write standard Matlab code. A few aspects must be kept in mind while designing functions that will be run remotely :

- Any reference to a file must be done using relative paths, excluding any use of '..' from the current folder (any tree hierarchy higher than the current directory has no meaning remotely).
- Files to be transferred to or from a remote worker must be declared explicitly through the InputFiles and OutputFiles tasks information.
- Paths must be written with a portable syntax (usage of filesep() is recommended).
- Functions of any number of input parameters are accepted but number of output parameters must be 1.
- Functions must be independent from each others executions. It is not possible to write tightly coupled code with inter-dependant communications.

[Back to Top](#)

## Which versions of Matlab and Operating Systems are compatible?

Matlab Connector Toolbox has been tested and is compatible with the following Matlab Versions and Operating Systems :

- Matlab 2007b and above.
- Windows 32 and 64 bits (XP, Server, Seven).
- Linux 32 and 64bits.
- MacOSX

[Back to Top](#)

## **What are the current limitations of Matlab Connector Toolbox?**

Known bugs and limitations are reported on ProActive Matlab/Scilab's connector JIRA bug reporting tool:

<https://bugs.activeeon.com/browse/MSC>

[Back to Top](#)



Getting Started

Installation



© Copyright (C) 1997-2011 INRIA/University of Nice-Sophia Antipolis/ActiveEon [Terms of Use](#)

## Installation

On this page...

[Introduction](#)

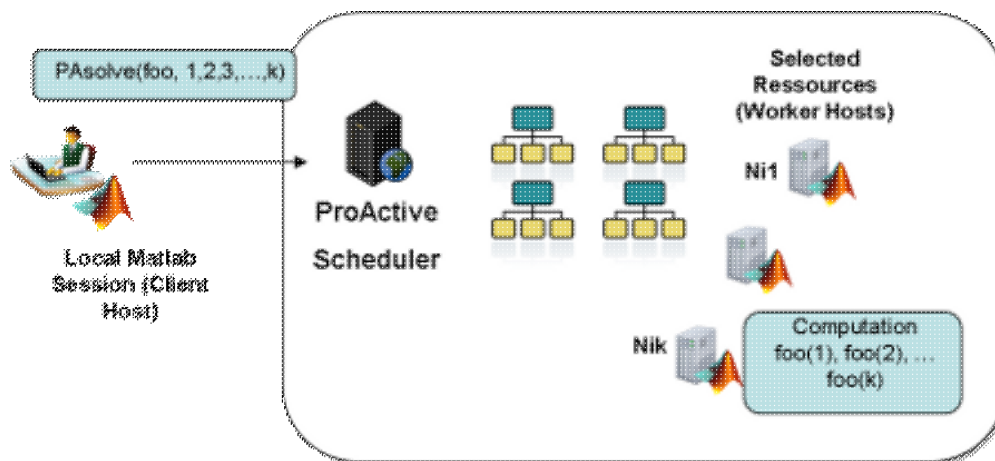
[Downloading and Installing ProActive Scheduler](#)

[Automatic Configuration of ProActive Scheduler for Matlab](#)

[Manual Configuration of ProActive Scheduler for Matlab](#)

## Introduction

A typical deployment of ProActive Scheduler with Matlab can be seen on the diagram below :



- The local Matlab session will connect to the ProActive Scheduler.
- The local user will submit a function **foo** to call with a set of parameters.
- The Scheduler will select among all its resources, those suited to the job.
- Eventually a License Server Proxy called *ProActive License Saver*, will ensure that desired license tokens are available on each machine.
- A Matlab engine will be started on each selected resource
- Each **foo(k)** will be executed on remote engines
- Results will be forwarded back to the user

*ProActive License Saver* is distributed separately from ProActive Scheduler, please contact <http://www.activeeon.com> for more information.

The License Saver address must be specified as an option while running remote tasks. See [Configuring PASolve behavior](#) for more details.

[Back to Top](#)

## Downloading and Installing ProActive Matlab Toolbox and ProActive Scheduler

Matlab Connector Toolbox can be downloaded from :

<http://www.activeeon.com/community-downloads>

The package which must be downloaded is called **Matlab\_Connector**

After downloading the package unzip it in a directory e.g. D:\Matlab\_Connector

ProActive Scheduler must also be downloaded : **ProActiveScheduling-3.3.1\_bin\_full.zip**

IMPORTANT: According to your deployment infrastructure, ProActive Scheduler must be installed on :

1. The host which will launch the **Scheduler**.
2. Each **worker host** used by the Scheduler to run matlab code.

The prerequisite for using the ProActive Scheduler is a **Java Virtual Machine**. On the above list, machines must have a Java Virtual Machine installed and available on the Path of either cmd.exe for Windows or sh for linux. For many ProActive features it is also necessary to define the environment variable JAVA\_HOME to the installation directory of the Java Virtual Machine.

Complete configuration and installation of the ProActive Scheduler is beyond the scope of this help, for more information, refer to:

[ProActive Scheduler Manual](#)

On each **client host**, on the matlab prompt add the path to Matlab Connector Toolbox (the folder MC/toolbox):

```
addpath('D:\Matlab_Connector\toolbox');
```

[Back to Top](#)

## Automatic Configuration of ProActive Scheduler for Matlab

### Automatic Find of Matlab Installations

In the typical utilisation of the toolbox, no configuration of the worker host will be necessary for Matlab. When a matlab task is waiting to be executed on a scheduler worker, a script will be executed to locate automatically matlab on the host. The script will search for matlab in typical locations in both windows and linux environments and will write the matlab installations found in a configuration file which will be stored on the host. If no matlab installation is found, then the configuration file will be empty and the host will be assumed not to contain any matlab. Subsequent executions of Matlab will reuse this configuration file to decide whether a matlab task can be executed on the host or not (though it is possible to force a matlab research via execution options, see option **ForceMatlabSearch** in [Configuring PAsolve behavior](#)).

### Automatic Transfer of ProActive Matlab Connector Libraries to the ProActive Scheduler

When the Matlab Connector creates jobs understandable by the ProActive Scheduler, it will transfer along the ProActive Matlab Connector libraries that will be used by the scheduler worker in order to create Matlab engines. Those libraries are not installed

in the ProActive Scheduler by default. This transfer is done via the option **UseJobClassPath**. If this option is set to false, then no transfer will be done, and the libraries will have to be copied manually on the scheduler server and workers installations.

[Back to Top](#)

## Manual Configuration of ProActive Scheduler for Matlab

### Manual Configuration of Matlab Installations

It may happen though that on some hosts, Matlab is not installed in a typical location and the automatic script doesn't manage to find it. In that case it will be necessary to write a Matlab configuration file manually.

A sample configuration file is located inside the **Matlab\_Connector/scheduler-plugin** folder under the name **MatlabWorkerConfiguration.xml**

This file must be edited and put inside the **addons** directory of ProActive Scheduler. This must be done on ProActive Scheduler Server and on every hosts that will be used as Matlab workers for which the automatic configuration didn't work.

If the scheduler is installed in a single directory shared by all workers (via NFS for example), then only one **MatlabWorkerConfiguration.xml** file needs to be edited. The syntax of the file allows to create a configuration for several machine groups via wildcards that will match a machine's hostname or ipaddress, and as well several matlab installations for the same machine.

Here is an example of **MatlabWorkerConfiguration.xml** :

```
<MatSciWorkerConfiguration>
  <MachineGroup hostname=".*">
    <matlab>
      <version>7.9</version>
      <home>C:\Program Files\MATLAB\R2009b</home>
      <bindir>bin\win64</bindir>
      <command>matlab.exe</command>
      <arch>64</command>
    </matlab>
  </MachineGroup>
</MatSciWorkerConfiguration>
```

Here is an explanation of tags used by this file:

- **MachineGroup**: this defines the machines which are targeted by this configuration. Machines can be identified by either **hostname** or **ip**. The syntax of hostname and ip attributes follow the Regular Expression syntax.
- **version**: The matlab version installed in numeric format.
- **home**: Full path to matlab home directory.
- **bindir**: relative path to the directory where to find the matlab executable. On windows, it is mandatory to use the matlab executable under bin/win32 (bin/win64) and not the executable directly in the bin directory. On linux systems on the other hand, it's mandatory to use the executable in the bin directory.
- **command**: matlab executable name.
- **arch**: matlab installation architecture (32 or 64).

The file supports multiple matlab configurations on the same Host (i.e. multiple matlab versions), and Matlab Connector Toolbox allows to select specific matlab versions for a given job dynamically.




## **Manual Installation of ProActive Matlab Connector Libraries to the ProActive Scheduler**

If the option **UseJobClassPath** is set to false, the content of the directory **Matlab\_Connector/scheduler-plugin** (not the directory itself, only the content) must be copied inside the **addons** directory of ProActive Scheduler and of every Matlab Worker which will be used as matlab workers.

Manually installing the librairies used by Matlab Connector inside the scheduler reduces the overhead of transferring them each time through the network.

[Back to Top](#)

 What is Matlab Connector Toolbox?

Starting And Connecting 

© Copyright (C) 1997-2011 INRIA/University of Nice-Sophia Antipolis/ActiveEon [Terms of Use](#)



## Starting and Connecting

On this page...

[Introduction](#)

[Starting ProActive Scheduler](#)

[Connecting to a running ProActive Scheduler from Matlab](#)

### Introduction

ProActive Scheduler is in fact composed of two programs working together:

- **ProActive Resource Manager:** the resource manager is the program in charge of a pool of resources called Nodes. Each node is a Java Virtual Machine running on a worker host. There can be several of these Nodes deployed on a single host, generally when a host has multiple processes or cores. It allows to maximize the processing power available on the network. The Resource Manager needs to be configured according to the network topology in order to deploy Nodes on available machines. Configuration of the Resource Manager for acquiring resources is beyond the scope of this document and is explained on the Resource Manager manual available at [http://proactive.inria.fr/release-doc/Resourcing/multiple\\_html/index.html](http://proactive.inria.fr/release-doc/Resourcing/multiple_html/index.html).
- **ProActive Scheduler:** the scheduler receives jobs to be executed. It will schedule their execution according to its policy and its workload. When a job is ready to be scheduled, the scheduler will contact the resource manager to find the maximum number of resources available to execute the job. Resources will be selected sometimes according to specific policies. In case of Matlab Connector Toolbox, only resources which have a valid Matlab configuration will be selected. A lot of extra selection will be done dynamically, for example specific Matlab versions needed by the job, or toolbox tokens availability .

[Back to Top](#)

### Starting ProActive Scheduler

The commands to start ProActive Scheduler are located in Scheduler\_Server/bin/unix or Scheduler\_Server/bin/windows, depending on your Operating System. The following commands are present:

- **rm-start(.bat):** To start an empty ProActive Resource Manager.
- **rm-start(.bat) -ln:** To start a ProActive Resource Manager with 4 local workers.
- **scheduler-start(.bat):** To start ProActive Scheduler.
- **scheduler-start-gui(.bat):** This starts the scheduler, the resource manager and all the web portals.

The following sequence of commands (executed from Scheduler\_Server/bin/unix) will start a Resource Manager with 4 local Nodes(JVM) and a Scheduler. Although this trivial deployment is not meant to be a practical case, it is still a good way to test the framework and become familiar with ProActive.

First the Resource Manager:

## Matlab Connector Toolbox

```
$ rm-start-clean -ln
Starting Resource Manager, Please wait...
Resource Manager successfully created on rmi://pendule.inria.fr:1099/
```

Then the Scheduler:

```
$ scheduler-start-clean
Starting Scheduler, Please wait...
Resource Manager URL was not specified, connection made to the local Resource Manager at rmi://pendule.inria.fr:1099/
Starting scheduler...
Scheduler successfully created on rmi://pendule.inria.fr:1099/
```

[Back to Top](#)

## Connecting to a running ProActive Scheduler from Matlab

[PAconnect](#) is the function used to connect to the scheduler from the toolbox.

From a matlab session, assuming that Matlab Connector Toolbox is already in Matlab path, run the following command (where PAconnect's argument is the url you received from the Scheduler starting command):

```
>> PAconnect('rmi://pendule.inria.fr:1099/');
```

you can also use the simplify syntax PAconnect(), it will automatically look for a scheduler deployed locally.

A popup window will appear asking for a username and password. This username/password refers to the username and password of your account on ProActive Scheduler. ProActive Scheduler features a full account management facility along with the possibility to synchronize to existing Windows or Linux accounts via LDAP. More information can be found at [Scheduler Manual:Configure users authentication](#).

Here for this example we will use the default account with login **demo** and password **demo**.

Here is what is displayed when the connection worked successfully:

```
> PAconnect('rmi://192.168.1.187:1099/')
log file in use : C:\Users\fviale\AppData\Local\Temp\MatlabJVMSpawnHelper.log
[2012-12-03 14:20:007 precision][MIDDLEMAN]Logging to org.slf4j.impl.Log4jLoggerAdapter(org.mortbay.log)
[2012-12-03 14:20:007 precision][MIDDLEMAN]jetty-6.1.x
[2012-12-03 14:20:007 precision][MIDDLEMAN]Started SelectChannelConnector@0.0.0.0:32092
[2012-12-03 14:20:007 precision][MIDDLEMAN]Remote Object Factory provider found
[2012-12-03 14:20:007 precision][MIDDLEMAN]Remote Object Factory provider found
[2012-12-03 14:20:007 precision][MIDDLEMAN]Remote Object Factory provider found
[2012-12-03 14:20:007 precision][MIDDLEMAN]Remote Object Factory provider found
[2012-12-03 14:20:007 precision][MIDDLEMAN]Remote Object Factory provider found
[2012-12-03 14:20:007 precision][MIDDLEMAN]Remote Object Factory provider found
[2012-12-03 14:20:007 precision][MIDDLEMAN][MiddlemanDeployer] Starting Middleman JVM
[2012-12-03 14:20:008 precision][MIDDLEMAN]Created a new registry on port 1110
[2012-12-03 14:20:009 precision][MIDDLEMAN][MiddlemanDeployer] Middleman JVM started

Connection to JVM successful

Connection successful to rmi://192.168.1.187:1099/

Please enter login/password
```

```
Login successful
ans =

[]

]]>
```

If the scheduler is unreachable, here is what is displayed :

```
>> PAconnect('rmi://pendule:1099/')
??? Java exception occurred:
org.ow2.proactive.scheduler.common.exception.ConnectionException: java.io.IOException: The url
rmi://pendule:1099/SCHEDULER is not bound to any known object
    at org.ow2.proactive.scheduler.common.SchedulerConnection.join(SchedulerConnection.java:102)
    at
    org.ow2.proactive.scheduler.ext.matsci.client.AOMatSciEnvironment.join(AOMatSciEnvironment.java:2
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
    at java.lang.reflect.Method.invoke(Unknown Source)
    at org.objectweb.proactive.core.mop.MethodCall.execute(MethodCall.java:395)
    at org.objectweb.proactive.core.body.request.RequestImpl.serveInternal(RequestImpl.java:253)
    at org.objectweb.proactive.core.body.request.RequestImpl.serve(RequestImpl.java:197)
    at
    org.objectweb.proactive.core.body.BodyImpl$ActiveLocalBodyStrategy.serveInternal(BodyImpl.java:61
        at
    org.objectweb.proactive.core.body.BodyImpl$ActiveLocalBodyStrategy.serve(BodyImpl.java:577)
    at org.objectweb.proactive.core.body.AbstractBody.serve(AbstractBody.java:944)
    at org.objectweb.proactive.Service.serveOldest(Service.java:214)
    at
    org.ow2.proactive.scheduler.ext.matsci.client.AOMatSciEnvironment.runActivity(AOMatSciEnvironment
        at org.objectweb.proactive.core.body.ActiveBody.run(ActiveBody.java:198)
    at java.lang.Thread.run(Unknown Source)
Caused by: java.io.IOException: The url rmi://pendule:1099/SCHEDULER is not bound to any known object
    at org.objectweb.proactive.api.PAActiveObject.lookupActive(PAActiveObject.java:1524)
    at org.ow2.proactive.authentication.Connection.lookupAuthentication(Connection.java:94)
    at org.ow2.proactive.authentication.Connection.connect(Connection.java:105)
    at org.ow2.proactive.scheduler.common.SchedulerConnection.join(SchedulerConnection.java:100)
    ... 15 more

Error in ==> PAconnect at 83
    ok = solver.join(url);
```

[Back to Top](#)

 Installation

Monitoring 

© Copyright (C) 1997-2011 INRIA/University of Nice-Sophia Antipolis/ActiveEon [Terms of Use](#)



## Monitoring

On this page...

[Introduction](#)

[Scheduler and Resource Manager command line controllers](#)

[ProActive Web Portals](#)

[Matlab Connector Toolbox monitoring functions](#)

## Introduction

The present chapter is a quick start guide to **ProActive Scheduler** and **Resource Manager** controllers and Graphical User Interfaces. These tools allow to execute a collection of operations on the Scheduler or Resource Manager, but in the context of **Matlab Connector Toolbox**, we will be interested mostly in the monitoring part.

[Back to Top](#)

## Scheduler and Resource Manager command line controllers

The command line controllers of **ProActive Scheduler** and **Resource Manager** are called respectively **scheduler-client** and **rm-client**. They are located in SCHEDULING/bin/unix or SCHEDULING/bin/windows depending on your operating system.

- **scheduler-client** allows to submit jobs to **ProActive Scheduler** in XML format, monitor job executions, preview results, view job logs, etc.
- **rm-client** allows to deploy or monitor resources in **Resource Manager**.

Here is an example use of scheduler-client, default login and password is **demo/demo**:

```
fviale@pendule unix $ scheduler-client
Trying to connect Scheduler on rmi://localhost/
-> Connection established on rmi://localhost/

Connecting client to the Scheduler
login: demo
password: ****
Retrieved public key from Scheduler at rmi://localhost/
-> Client 'demo' successfully connected
```

Type command here (type '?' or help() to see the list of commands)

```
> listjobs();
```

| ID | NAME                     | OWNER | PRIORITY | PROJECT      | STATUS   | START AT |
|----|--------------------------|-------|----------|--------------|----------|----------|
| 1  | Matlab Environment Job 0 | demo  | Normal   | Not Assigned | Finished | 18:33:53 |

## Matlab Connector Toolbox

|    |                    |        |      |        |              |          |          |
|----|--------------------|--------|------|--------|--------------|----------|----------|
| 2  | Matlab Environment | Job 1  | demo | Normal | Not Assigned | Finished | 20:50:33 |
| 3  | Matlab Environment | Job 2  | demo | Normal | Not Assigned | Finished | 20:51:39 |
| 4  | Matlab Environment | Job 3  | demo | Normal | Not Assigned | Finished | 20:52:25 |
| 5  | Matlab Environment | Job 4  | demo | Normal | Not Assigned | Finished | 20:53:10 |
| 6  | Matlab Environment | Job 5  | demo | Normal | Not Assigned | Finished | 20:54:38 |
| 7  | Matlab Environment | Job 6  | demo | Normal | Not Assigned | Finished | 20:57:46 |
| 8  | Matlab Environment | Job 7  | demo | Normal | Not Assigned | Finished | 23:07:09 |
| 9  | Matlab Environment | Job 8  | demo | Normal | Not Assigned | Finished | 01:05:02 |
| 10 | Matlab Environment | Job 9  | demo | Normal | Not Assigned | Finished | 01:08:09 |
| 11 | Matlab Environment | Job 10 | demo | Normal | Not Assigned | Finished | 01:11:35 |
| 12 | Matlab Environment | Job 11 | demo | Normal | Not Assigned | Finished | 01:20:11 |
| 13 | Matlab Environment | Job 12 | demo | Normal | Not Assigned | Finished | 01:56:33 |
| 14 | Matlab Environment | Job 13 | demo | Normal | Not Assigned | Finished | 02:20:19 |
| 15 | Matlab Environment | Job 14 | demo | Normal | Not Assigned | Finished | 02:21:03 |
| 16 | Matlab Environment | Job 0  | demo | Normal | Not Assigned | Finished | 02:40:00 |

> jobstate(15)

Job '15'      name:Matlab Environment Job 14      owner:demo      status:Finished      #tasks:20

| ID   | NAME | ITER | DUP | STATUS   | HOSTNAME                      | EXEC DURATION | TOT DURATION |
|--|------|------|-----|----------|-------------------------------|---------------|--------------|
| 150001   | 9_0  |      |     | Finished | pendule.inria.fr (PA_JVM41... | 733ms         | 2s 943ms     |
| 150002   | 5_0  |      |     | Finished | pendule.inria.fr (PA_JVM15... | 753ms         | 3s 601ms     |
| 150003   | 8_0  |      |     | Finished | pendule.inria.fr (PA_JVM19... | 637ms         | 2s 531ms     |
| 150004   | 10_0 |      |     | Finished | pendule.inria.fr (PA_JVM15... | 723ms         | 1s 734ms     |
| 150005   | 1_0  |      |     | Finished | pendule.inria.fr (PA_JVM41... | 635ms         | 2s 493ms     |
| 150006   | 4_0  |      |     | Finished | pendule.inria.fr (PA_JVM14... | 823ms         | 3s 708ms     |
| 150007   | 16_0 |      |     | Finished | pendule.inria.fr (PA_JVM15... | 803ms         | 2s 405ms     |
| 150008   | 7_0  |      |     | Finished | pendule.inria.fr (PA_JVM19... | 733ms         | 2s 953ms     |
| 150009   | 6_0  |      |     | Finished | pendule.inria.fr (PA_JVM14... | 626ms         | 3s 741ms     |
| 150010   | 2_0  |      |     | Finished | pendule.inria.fr (PA_JVM41... | 724ms         | 798ms        |
| 150011   | 0_0  |      |     | Finished | pendule.inria.fr (PA_JVM14... | 612ms         | 3s 553ms     |
| 150012   | 3_0  |      |     | Finished | pendule.inria.fr (PA_JVM14... | 832ms         | 3s 575ms     |
| 150013   | 17_0 |      |     | Finished | pendule.inria.fr (PA_JVM15... | 614ms         | 3s 631ms     |
| 150014   | 15_0 |      |     | Finished | pendule.inria.fr (PA_JVM19... | 637ms         | 2s 490ms     |
| 150015   | 19_0 |      |     | Finished | pendule.inria.fr (PA_JVM41... | 716ms         | 2s 532ms     |
| 150016   | 11_0 |      |     | Finished | pendule.inria.fr (PA_JVM15... | 621ms         | 3s 675ms     |
| - more - (q : abort   a : display all   any : next page) |      |      |     |          |                               |               |              |
| 150017   | 12_0 |      |     | Finished | pendule.inria.fr (PA_JVM19... | 615ms         | 687ms        |
| 150018   | 13_0 |      |     | Finished | pendule.inria.fr (PA_JVM14... | 617ms         | 596ms        |
| 150019   | 18_0 |      |     | Finished | pendule.inria.fr (PA_JVM19... | 625ms         | 2s 487ms     |
| 150020   | 14_0 |      |     | Finished | pendule.inria.fr (PA_JVM41... | 720ms         | 2s 502ms     |

> task

```
taskoutput( taskresult(
> taskresult(15,'12_0')
Task 12_0 result =>
true
```

> taskoutput(15,'12\_0')

```
12_0 output :
[ Thu Jan 13 02:21:24 CET 2011 ][pendule OUT]
[ Thu Jan 13 02:21:24 CET 2011 ][pendule OUT]
[ Thu Jan 13 02:21:24 CET 2011 ][pendule OUT]
[ Thu Jan 13 02:21:24 CET 2011 ][pendule OUT]
[ Thu Jan 13 02:21:24 CET 2011 ][pendule OUT]
[ Thu Jan 13 02:21:24 CET 2011 ][pendule OUT]
[ Thu Jan 13 02:21:24 CET 2011 ][pendule OUT]
[ Thu Jan 13 02:21:24 CET 2011 ][pendule OUT]
```

```
[ Thu Jan 13 02:21:24 CET 2011 ][pendule OUT]
[ Thu Jan 13 02:21:24 CET 2011 ][pendule OUT]
[ Thu Jan 13 02:21:24 CET 2011 ][pendule OUT]
[ Thu Jan 13 02:21:24 CET 2011 ][pendule OUT]
[ Thu Jan 13 02:21:24 CET 2011 ][pendule OUT]
[ Thu Jan 13 02:21:24 CET 2011 ][pendule OUT]
```

```
> exit();
Exiting controller.
```

The **help()** function gives a description of all available commands. More information about the scheduler controller can be found at [Using the Scheduler controller](#).

Here is an example use of rm-client:

```
fviale@pendule unix $ rm-client
Trying to connect RM on rmi://localhost:1099/
-> Connection established on rmi://localhost:1099/

Connecting to the RM
login: demo
password: ****
Retrieved public key from Resource Manager at rmi://localhost:1099/
-> Client 'demo' successfully connected

Type command here (type '?' or help() to see the list of commands)
```

```
> listnodes();
```

| SOURCE        | NAME | HOSTNAME         | STATE | SINCE          | URL                                      |
|---------------|------|------------------|-------|----------------|--|
| GCMLocalNodes |      | pendule.inria.fr | Free  | 13/01/11 02:40 | rmi://pendule.inria.fr:6608/PA_JVM148585 |
| GCMLocalNodes |      | pendule.inria.fr | Free  | 13/01/11 02:40 | rmi://pendule.inria.fr:6608/PA_JVM157471 |
| GCMLocalNodes |      | pendule.inria.fr | Free  | 13/01/11 02:40 | rmi://pendule.inria.fr:6608/PA_JVM191614 |
| GCMLocalNodes |      | pendule.inria.fr | Free  | 13/01/11 02:40 | rmi://pendule.inria.fr:6608/PA_JVM416643 |

```
>
```

Here we see that the Resource Manager has been deployed with 4 local nodes. The STATE column displays if the node is currently executing a job, free or there was a node failure. More information about the Resource Manager controller can be found at [Interacting with the resource manager](#).

[Back to Top](#)

## Scheduler and Resource Manager Web Portals

The ProActive Web Portals are web graphical user interfaces for both Scheduler and Resource Manager. They provide a more convenient way of monitoring job execution than the command line interface. The ProActive Web Portals are embedded into the package **ProActive Scheduling Server Full** from the [ProActive downloads page](#). They are started automatically when the **scheduler-start-gui** command is used to start the scheduler see [Starting and Connecting](#)

Here is a view of the **Scheduler Web Portal**:

[←](#)
[→](#)
[↻](#)
<https://portal.cloud.sophia.inria.fr/scheduler/>

Apple
 Yahoo!
 Google Maps
 YouTube
 Wikipedia
 News
 Popular
 Imported From Safari

Portal ▾ Admin ▾ Help ▾
 [Submit job](#)
[Logout fviale](#)

Jobs list

| Id ▾ | State    | User     | Progress | Priority | Duration         | Na   |
|------|----------|----------|----------|----------|------------------|------|
| 9307 | Running  | yroux    | 0 / 1    | Normal   |                  | H2   |
| 9306 | Running  | yroux    | 0 / 1    | Normal   |                  | H2   |
| 9305 | Finished | ambonyin | 1 / 1    | Normal   | 1h 10m 11s 843ms | Test |
| 9304 | Finished | ambonyin | 5 / 5    | Normal   | 2m 29s 317ms     | Test |
| 9303 | Finished | yroux    | 1 / 1    | Normal   | 16h 14m 27s 54ms | H2   |
| 9302 | Finished | yroux    | 1 / 1    | Normal   | 25h 47m 37s 5ms  | H2   |
| 9301 | Killed   | mbenguig | 1 / 2    | Normal   | 1m 22s 323ms     | No   |
| 9300 | Killed   | mbenguig | 1 / 2    | Normal   | 19s 15ms         | No   |
| 9299 | Killed   | mbenguig | 1 / 2    | Normal   | 20s 205ms        | No   |
| 9298 | Killed   | mbenguig | 1 / 2    | Normal   | 2m 48s 60ms      | No   |
| 9297 | Killed   | mbenguig | 1 / 2    | Normal   | 2m 54s 729ms     | No   |
| 9296 | Killed   | mbenguig | 0 / 2    | Normal   | 36s 728ms        | No   |
| 9295 | Killed   | mbenguig | 1 / 2    | Normal   | 3m 12s 285ms     | No   |
| 9294 | Killed   | mbenguig | 1 / 2    | Normal   | 31s 211ms        | No   |
| 9293 | Killed   | mbenguig | 1 / 2    | Normal   | 21s 501ms        | No   |
| 9292 | Killed   | mbenguig | 1 / 2    | Normal   | 1m 3s 241ms      | No   |
| 9291 | Killed   | mbenguig | 1 / 2    | Normal   | 21s 529ms        | No   |
| 9290 | Killed   | mbenguig | 1 / 2    | Normal   | 3m 36s 738ms     | No   |
| 9289 | Finished | bamedro  | 9 / 9    | Normal   | 14s 742ms        | job  |

Details

Tasks

Visualization

Users

Statistics

|   | Id       | Status   | Name    | Duration     | Nodes | Executions | Failures |
|---|----------|----------|---------|--------------|-------|------------|----------|
| ▶ | 93040004 | Finished | task2_1 | 1m 30s 271ms | 300   | 1 / 1      | 0 / 2    |
| ▶ | 93040002 | Finished | task4   | 6s 159ms     | 1     | 1 / 1      | 0 / 2    |
| ▶ | 93040003 | Finished | task3   | 11s 208ms    | 1     | 1 / 1      | 0 / 2    |
| ▶ | 93040000 | Finished | task2_2 | 21s 413ms    | 100   | 1 / 1      | 0 / 2    |
| ▶ | 93040001 | Finished | task2_3 | 29s 143ms    | 1     | 1 / 1      | 0 / 2    |

Job In

Pending

Running

Finished

Total

Submitted

Finished

Pending



Here is a view of the **Resource Manager Web Portal**:

← → ↻ <https://portal.cloud.sophia.inria.fr/rm/>

🍏 Apple 📄 Yahoo! 🗺️ Google Maps 📺 YouTube 📄 Wikipedia 📁 News 📁 Popular 📁 Imported From Safari

📄 Cette page est en anglais Voulez-vous la traduire ? Traduire Non

Portal ▾ Help ▾ | 🌐 Add Nodes 🚪 Logout fviale

### Nodes

- ⊕ 🌐 CNSlice1 Infrastructure: SSH Infrastructure, Policy: Static Policy user access type [ALL], provider access type [ME
- ⊕ 🌐 CNSlice2 Infrastructure: SSH Infrastructure, Policy: Static Policy user access type [ALL], provider access type [ME
- ⊖ 🌐 CNSlice3 Infrastructure: SSH Infrastructure, Policy: Static Policy user access type [ALL], provider access type [ME
  - ⊕ 🖨️ node100.cloud.sophia.inria.fr
  - ⊕ 🖨️ node101.cloud.sophia.inria.fr
  - ⊕ 🖨️ node102.cloud.sophia.inria.fr
  - ⊕ 🖨️ node103.cloud.sophia.inria.fr
  - ⊕ 🖨️ node104.cloud.sophia.inria.fr
  - ⊕ 🖨️ node105.cloud.sophia.inria.fr
  - ⊕ 🖨️ node60.cloud.sophia.inria.fr
  - ⊕ 🖨️ node61.cloud.sophia.inria.fr
  - ⊕ 🖨️ node62.cloud.sophia.inria.fr
  - ⊕ 🖨️ node63.cloud.sophia.inria.fr
  - ⊕ 🖨️ node64.cloud.sophia.inria.fr
  - ⊕ 🖨️ node65.cloud.sophia.inria.fr
  - ⊕ 🖨️ node66.cloud.sophia.inria.fr

### Details

Selection Nodes **Monitoring**

🖨️ node105.cloud.sophia.inria.fr

Overview CPU Memory File System Network Processes

|          |                           |
|----------|---------------------------|
| Name:    | Linux                     |
| Arch:    | amd64                     |
| Version: | 2.6.32-220.4.2.el6.x86_64 |

Cpu Usage

Memory

100%

75%

50%

25%

1 Mb

1 Mb

1 Mb

0 Mb

Us...

Free

To...

### Nodes S

1 200

900

600

300

0

### Nodes H

1 000

750

1800

250

0

Scheduler and Resource Manager Web Portals

They can be accessed after the **scheduler-start-gui** command has been launched from a web browser at the addresses : **http://localhost:8080/sched** and **http://localhost:8080/rm**. *localhost* has to be replaced by the server hostname if it is not deployed locally.

Information on their usage can be found in the [Scheduler Manual](#).

[Back to Top](#)

## Matlab Connector Toolbox monitoring functions

Matlab Connector Toolbox provides a set of function for monitoring job execution, similarly as the scheduler-client command line interface. Here is the list of those functions :

|                              |  |
|------------------------------|--|
| <a href="#">PAstate</a>      | State of ProActive Scheduler             |
| <a href="#">PAjobState</a>   | State of the given job                   |
| <a href="#">PAjobRemove</a>  | Removes the given job from the scheduler |
| <a href="#">PAjobOutput</a>  | Textual log of the given job             |
| <a href="#">PAjobResult</a>  | Textual result of the given job          |
| <a href="#">PAkillJob</a>    | Kills the given job                      |
| <a href="#">PApauseJob</a>   | Pauses the given job                     |
| <a href="#">PAresumeJob</a>  | Resumes the given job                    |
| <a href="#">PAkillTask</a>   | Kills the given task                     |
| <a href="#">PAtaskOutput</a> | Textual log of the given task            |
| <a href="#">PAtaskResult</a> | Textual result of the given task         |

[Back to Top](#)

 Starting And Connecting

Running Matlab functions remotely 

© Copyright (C) 1997-2011 INRIA/University of Nice-Sophia Antipolis/ActiveEon [Terms of Use](#)



## Running Matlab functions remotely

On this page...

[Introduction](#)

[Simplest parametric sweep](#)

[Parametric sweep with user code](#)

[Receiving results](#)

[Complex Tasks definitions](#)

[Chaining Remote Tasks](#)

### Introduction

The three main actors of Matlab Connector Toolbox are the following functions and objects:

- **PAsolve**: It is the function used to run matlab code remotely. A call to PAsolve will create a Matlab job inside ProActive Scheduler. After the job is created, the PAsolve function will return, allowing the local Matlab session to continue while the results are being produced.
- **PAResult**: this object is returned by the PAsolve function. PAsolve returns a vector of PAResult objects, whose size matches the number of parameters to the PAsolve function. The PAResult object defines a collection of methods which can be used to wait (block the local Matlab session) for specific results.
- **PATask**: this object allows to define complex tasks that can be given as parameters to PAsolve.

[Back to Top](#)

### Simple parametric sweep

In this paragraph, we will explain how PAsolve should be called to evaluate remotely a matlab function. It is possible to do a single remote evaluation, but here we will use as example a multiple remote evaluation of the same function but with different parameters (parametric sweep). For this example, we will use the matlab function **factorial**, which has no practical use to be run remotely, but which will serve as a well-known example. After the matlab session is connected to the scheduler (explained in paragraph [Connecting to a running ProActive Scheduler from Matlab](#)), enter the following command on the matlab prompt:

```
>> res=PAsolve(@factorial,1,2,3,4,5)
Job submitted : 1
Awaited (J:1)
Awaited (J:1)
Awaited (J:1)
Awaited (J:1)
Awaited (J:1)
```

PAolve is being called with a function handle to the matlab function "factorial". A list of parameters from 1 to 5 is being given. This means that remotely factorial(1), factorial(2),..., factorial(5) will be executed. The PAolve call returns immediately, giving the ID of the job created (1) inside ProActive Scheduler and returning in the res variable an array of PAresult objects which display themselves as *Awaited* objects from job ID 1.

After a while, factorial results will be produced. It is possible to force Matlab to wait(block) for these results using a specific function call (explained in next chapter: [Receiving results](#)). But here, we will simply evaluate the variable **res** until it displays itself as received :

```
>> res

[ Tue Jan 11 14:27:32 CET 2011 ][pendule OUT]
[ Tue Jan 11 14:27:32 CET 2011 ][pendule OUT]

res =

    1

[ Tue Jan 11 14:27:33 CET 2011 ][pendule OUT]
[ Tue Jan 11 14:27:33 CET 2011 ][pendule OUT]

res =

    2

[ Tue Jan 11 14:27:35 CET 2011 ][pendule OUT]
[ Tue Jan 11 14:27:35 CET 2011 ][pendule OUT]

res =

    6

[ Tue Jan 11 14:27:33 CET 2011 ][pendule OUT]
[ Tue Jan 11 14:27:33 CET 2011 ][pendule OUT]

res =

   24

[ Tue Jan 11 14:27:33 CET 2011 ][pendule OUT]
[ Tue Jan 11 14:27:33 CET 2011 ][pendule OUT]

res =

   12
```

This output shows the remote outputs of the five executions factorial(1..5). Actually, this output is shown here for the example, in the real case, as the factorial function prints no output, there will be nothing displayed.

The remote outputs are displayed only once. They can be accessed later using the **logs** attribute of the PAResult task:

```
>> res

res =

     1

res =

     2

res =

     6

res =

    24

res =

   120

>> res.logs

ans =

[ Tue Jan 11 14:27:32 CET 2011 ] [pendule OUT]
[ Tue Jan 11 14:27:32 CET 2011 ] [pendule OUT]

ans =

[ Tue Jan 11 14:27:33 CET 2011 ] [pendule OUT]
[ Tue Jan 11 14:27:33 CET 2011 ] [pendule OUT]

ans =

[ Tue Jan 11 14:27:35 CET 2011 ] [pendule OUT]
[ Tue Jan 11 14:27:35 CET 2011 ] [pendule OUT]

ans =

[ Tue Jan 11 14:27:33 CET 2011 ] [pendule OUT]
[ Tue Jan 11 14:27:33 CET 2011 ] [pendule OUT]
```

```
ans =
```

```
[ Tue Jan 11 14:27:33 CET 2011 ][pendule OUT]
[ Tue Jan 11 14:27:33 CET 2011 ][pendule OUT]
```

Note that the PAsolve call PAsolve(function, param1, param2, ...) is not very practical when there are a lot of parameters. Matlab itself provides a way to assign parameters to a function call with cell arrays. For example, the factorial call used before:

```
>> res=PAsolve(@factorial,1,2,3,4,5)
```

Can be replaced by the following lines:

```
>> cl=num2cell(1:5) # cell creation (converted from a numeric vector)
>> res=PAsolve(@factorial,cl{:}) # cell expansion, converted to function parameters
```

The first line creates a cell containing the parameters 1,2,3,4,5 and the second line applies this cell to the PAsolve call.

We've been through a few examples of how to use PAsolve with the matlab function factorial, but in general, matlab functions are very fast and rarely need to be run remotely. Complex user-defined functions, on the other hand, can really benefit from parallel computing. We will explain in the next chapter [Parametric sweep with user code](#), how to run user functions remotely.

[Back to Top](#)

## Parametric sweep with user code

In this chapter, we will create user-defined matlab functions that we will try to run remotely. Here are the two functions that we will write:

- A function called **helloworld** which will not compute anything but will try to display remotely the message "Hello World!".
- A function called **makeerror** which will produce a matlab error remotely.

Here is how the **helloworld.m** file is defined :

```
function ok=helloworld()
disp('HelloWorld');
ok=true;
```

Here is the PAsolve execution of **helloworld**:

```
>> res=PAsolve(@helloworld,{}, {}, {})
Job submitted : 2
Awaited (J:2)
Awaited (J:2)
Awaited (J:2)
>> res

[ Tue Jan 11 16:30:03 CET 2011 ][pendule OUT]
[ Tue Jan 11 16:30:03 CET 2011 ][pendule OUT]>> HelloWorld
[ Tue Jan 11 16:30:03 CET 2011 ][pendule OUT]
```

```
res =

    1

[ Tue Jan 11 16:30:03 CET 2011 ][pendule OUT]
[ Tue Jan 11 16:30:03 CET 2011 ][pendule OUT]>> HelloWorld
[ Tue Jan 11 16:30:03 CET 2011 ][pendule OUT]
```

```
res =

    1

[ Tue Jan 11 16:30:02 CET 2011 ][pendule OUT]
[ Tue Jan 11 16:30:02 CET 2011 ][pendule OUT]>> HelloWorld
[ Tue Jan 11 16:30:02 CET 2011 ][pendule OUT]
```

```
res =

    1
```

You see in this example how the HelloWorld string is displayed on the remote engines and appears in the job logs. Please note that we didn't do anything particular to transfer the definition of the helloworld function to the remote matlab engine. In reality, this is done seamlessly by the ProActive toolbox: the toolbox automatically transfers the source code of the function used and all other dependant ones to the remote Matlab engine. Note as well the syntax of the call `PAsolve(@helloworld,{}, {}, {})`, which says that we call helloworld with no parameter (an empty cell array). If helloworld had more than one parameter, we would as well use a cell array to hold the parameters such as

`PAsolve(@multparam,{param_1_1,param_1_2,...,param_1_k},...,{param_n_1,param_n_2,...param_n_k})`. This would call the k-parameter function "multparam" n-times. In the rare case when the function requires a single parameter which is a cell, the following syntax should be used : `PAsolve(@onecellfunc,{cellparam_1}, ..., {cellparam_n} , where cellparam_1..n are cells.`

Here is how the **makeerror.m** file is defined :

```
function ok=makeerror()
% b doesn't exist!
disp(b)
ok=true;
```

Here is the `PAsolve` execution of **makeerror**:

```
>> res=PAsolve(@makeerror, {}, {}, {})
Job submitted : 4
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
>> res

[ Tue Jan 11 18:30:45 CET 2011 ][pendule OUT]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]??? Undefined function or variable 'b'.
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]Error in ==> makeerror at 3
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]disp(b)
```



## Matlab Connector Toolbox

```
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule OUT]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR] ??? Error using ==> save
[ Tue Jan 11 18:30:45 CET 2011 ][pendule OUT]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]Variable 'out' not found.
[ Tue Jan 11 18:30:45 CET 2011 ][pendule OUT]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule OUT]
```

Error using ==> PAREsult.PAwaitFor at 99  
Error during remote script execution

Error in ==> PAREsult.display at 49  
dp(PAwaitFor(RR), inputname(1))

```
[ Tue Jan 11 18:30:45 CET 2011 ][pendule OUT]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]??? Undefined function or variable 'b'.
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]Error in ==> makeerror at 3
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]disp(b)
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR] ??? Error using ==> save
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]Variable 'out' not found.
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule OUT]
```

Error using ==> PAREsult.PAwaitFor at 99  
Error during remote script execution

Error in ==> PAREsult.display at 49  
dp(PAwaitFor(RR), inputname(1))

```
[ Tue Jan 11 18:30:45 CET 2011 ][pendule OUT]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]??? Undefined function or variable 'b'.
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]Error in ==> makeerror at 3
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]disp(b)
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR] ??? Error using ==> save
[ Tue Jan 11 18:30:45 CET 2011 ][pendule OUT]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]Variable 'out' not found.
[ Tue Jan 11 18:30:45 CET 2011 ][pendule OUT]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule ERR]
[ Tue Jan 11 18:30:45 CET 2011 ][pendule OUT]
```

Error using ==> PAREsult.PAwaitFor at 99  
Error during remote script execution

Error in ==> PAREsult.display at 49  
dp(PAwaitFor(RR), inputname(1))

You see in this example how the matlab errors from the remote execution are forwarded and appear in the logs.

[Back to Top](#)

## Receiving results

As explained above, PAsolve calls are asynchronous and don't block Matlab session, a collection of functions are provided to wait(block) for results arrival or test their presence without blocking:

- **PAwaitFor**: given a vector of PAREsult, blocks the matlab session until all results are received.
- **PAwaitAny**: given a vector of PAREsult, blocks the matlab session until any one of those results are received. Successive calls to PAwaitAny, allows to retrieve the results one by one, in completion order.
- **PAisAwaited**: given a vector of PAREsult, this non-blocking call tells which results are available.
- **PAREsult.val attribute**: similarly to PAwaitFor, block the matlab session until all results are received. As it is an attribute, it provides full control on which particular results needs to be waited.

Here is an example of using **PAwaitFor**:

```
>> res=PAsolve(@factorial,1,2,3,4,5)
Job submitted : 2
Awaited (J:2)
Awaited (J:2)
Awaited (J:2)
Awaited (J:2)
Awaited (J:2)
>> val=PAwaitFor(res)

[ Tue Jan 11 20:19:09 CET 2011 ][pendule OUT]
[ Tue Jan 11 20:19:09 CET 2011 ][pendule OUT]

[ Tue Jan 11 20:19:07 CET 2011 ][pendule OUT]
[ Tue Jan 11 20:19:07 CET 2011 ][pendule OUT]

[ Tue Jan 11 20:19:07 CET 2011 ][pendule OUT]
[ Tue Jan 11 20:19:07 CET 2011 ][pendule OUT]

[ Tue Jan 11 20:19:07 CET 2011 ][pendule OUT]
[ Tue Jan 11 20:19:07 CET 2011 ][pendule OUT]

[ Tue Jan 11 20:19:07 CET 2011 ][pendule OUT]
[ Tue Jan 11 20:19:07 CET 2011 ][pendule OUT]

val =

      [1]      [2]      [6]      [24]      [120]

>> class(val)

ans =

cell

>> class(val{1})

ans =

double

>> class(res)
```

```
ans =
```

```
PAResult
```

Notice that **PAwaitFor** returns its result in a cell array, which contains the real result to the factorial call. The "res" variable above will always be of class **PAResult**, and thus is not usable directly.

Here is an example of using **PAwaitAny**:

```
>> res=PAsolve(@factorial,1,2,3,4,5)
Job submitted : 3
Awaited (J:3)
Awaited (J:3)
Awaited (J:3)
Awaited (J:3)
Awaited (J:3)
>> val1=PAwaitAny(res)
```

```
[ Tue Jan 11 20:24:08 CET 2011 ][pendule OUT]
[ Tue Jan 11 20:24:08 CET 2011 ][pendule OUT]
```

```
val1 =
```

```
1
```

```
>> val2=PAwaitAny(res)
```

```
[ Tue Jan 11 20:24:08 CET 2011 ][pendule OUT]
[ Tue Jan 11 20:24:08 CET 2011 ][pendule OUT]
```

```
val2 =
```

```
2
```

```
>> val3=PAwaitAny(res)
```

```
[ Tue Jan 11 20:24:08 CET 2011 ][pendule OUT]
[ Tue Jan 11 20:24:08 CET 2011 ][pendule OUT]
```

```
val3 =
```

```
6
```

```
>> val4=PAwaitAny(res)
```

```
[ Tue Jan 11 20:24:09 CET 2011 ][pendule OUT]
[ Tue Jan 11 20:24:09 CET 2011 ][pendule OUT]
```

```
val4 =
```

```
24
```

```
>> val5=PAwaitAny(res)
```

```
[ Tue Jan 11 20:24:08 CET 2011 ][pendule OUT]
```

[ Tue Jan 11 20:24:08 CET 2011 ][pendule OUT]

```
val5 =

    120

>> val6=PAwaitAny(res)
??? Error using ==> PAREsult.PAwaitAny at 54
All results have already been accessed

>>
```

Notice how successive calls to **PAwaitAny** returns different answers, and how the last call produces an error as all results have already been received. Please note as well, that the order in which results are received by a **PAwaitAny** call does NOT necessary match the order of the **PAsolve** call (i.e. first calls returns 1, second 2, etc...). It only depends on the order of completion and not submission!

Finally, an example of using **PAisAwaited**:

```
>> res=PAsolve(@factorial,1,2,3,4,5)
Job submitted : 5
Awaited (J:5)
Awaited (J:5)
Awaited (J:5)
Awaited (J:5)
Awaited (J:5)
>> val=PAisAwaited(res)

val =

     1     1     1     1     1

>> val=PAisAwaited(res)

val =

     1     1     1     1     1

>> val=PAisAwaited(res)

val =

     1     1     1     0     1

>> val=PAisAwaited(res)

val =

     0     0     0     0     0

>>
```

We see in this example how results are progressively received. Finally the **val** attribute of the **PAResult** class can be used like **PAwaitFor**:

```
>> res=PAsolve(@factorial,1,2,3,4,5)
Job submitted : 1
```

```
Awaited (J:1)
Awaited (J:1)
Awaited (J:1)
Awaited (J:1)
Awaited (J:1)
>> res(1:2).val

[ Wed Jan 12 18:33:59 CET 2011 ][pendule OUT]
[ Wed Jan 12 18:33:59 CET 2011 ][pendule OUT]

[ Wed Jan 12 18:33:59 CET 2011 ][pendule OUT]
[ Wed Jan 12 18:33:59 CET 2011 ][pendule OUT]

ans =

      [1]      [2]

>> res(3).val

[ Wed Jan 12 18:34:01 CET 2011 ][pendule OUT]
[ Wed Jan 12 18:34:01 CET 2011 ][pendule OUT]

ans =

      6

>> res(4:5).val

[ Wed Jan 12 18:34:00 CET 2011 ][pendule OUT]
[ Wed Jan 12 18:34:00 CET 2011 ][pendule OUT]

[ Wed Jan 12 18:33:59 CET 2011 ][pendule OUT]
[ Wed Jan 12 18:33:59 CET 2011 ][pendule OUT]

ans =

      [24]      [120]
```

You see how the **val** attribute has been used here to wait for the first two results, the third one and then the last two. You see as well how the variable type returned in the ans variable varies. Here is the convention : if more than one results is retrieved, the **val** attribute packs all results retrieved in a cell array. If only one result is retrieved, the **val** attribute returns it without packing.

We've seen above that we can as well call the **logs** attribute to retrieve task logs. Similarly a call to the **logs** attribute will block the matlab session until the result is available, and will behave the same as the **val** attribute concerning packing.

[Back to Top](#)

## Complex Tasks definitions

Running single matlab functions remotely is the standard way of using **PSolve**, but sometimes Matlab functions will also need to read input files and/or write output files. In this chapter, we will describe how to do that using the object class **PATask**.

The **PATask** object works exactly like a structure. It contains, among others the following fields (see [PATask](#) for the full list):

- **Func**: the function to be executed in this task, i.e. @factorial.
- **Params**: the parameters to the function, which can either be a single parameter or a cell array containing the function multiple parameters.
- **InputFiles**: the input files, given as a PAFFile array. Paths to the input files must be given relatively to the current directory and it is not allowed to refer to a file which is outside the current directory hierarchy (i.e. no '..').
- **OutputFiles**: the output files, given as a PAFFile array. Similarly the paths must be relative to the base directory of the remote matlab engine (which will be a subdirectory of the TEMP directory). The matlab function will have to create those files and all subdirectories relative to the base one.
- **Description**: a char array representing the task.
- **SelectionScript**: a char array which defines an additional Selection Script to select resources (see ProActive Scheduler Manual).
- **Static**: a boolean which defines if the SelectionScript used should be static or dynamic, default to dynamic (see ProActive Scheduler Manual).
- **ScriptParams**: a string containing the parameters of the custom script delimited by spaces.
- **NbNodes**: an integer value which indicates the number of Scheduler Nodes necessary to run this task. This parameter is interesting in case the Matlab engine is running in multithreaded mode to specify the number of processors necessary or if an external program is called from the task, and this programs requires multiple processors and/or machines.
- **Topology**: a string parameter which affects how nodes will be chosen to meet the NbNodes required. The values possible are : 'arbitrary', 'bestProximity', 'thresholdProximity', 'singleHost', 'singleHostExclusive', 'multipleHostsExclusive', 'differentHostsExclusive' Please refer to ProActive Scheduler manual for more information on these values.
- **ThresholdProximity**: an integer value which indicates the threshold in case the 'thresholdProximity' is set on the Topology attribute. Please refer to ProActive Scheduler manual for more information on this setting.
- **Compose**: true or false. When composing tasks, is the result of the previous task given as first parameter of this one (see next chapter [Chaining Remote Tasks](#)) ?

To illustrate the usage of **PATask**, we will define a simple Matlab function **factfile** which will read an integer from a mat file and will output the factorial of this integer to a second mat file

Here is the content of **factfile.m**:

```
function ok=factfile()
load('-mat','factfile_in.mat')
b=factorial(a);
save('factfile_out.mat','b');
ok=true;
```

Below is the PAsolve execution of **factfile** with the creation of the PATask and input/output files. We execute only one task, as multiple task would involve handling multiple files and would be too complicate for this example. The PAsolve function takes the PATask as parameter, which contains all the necessary information:

```
>> t=PATask(1,1);
>> t.Func = @factfile;
>> t.Params = {};
>> t.InputFiles='factfile_in.mat';
>> t.OutputFiles='factfile_out.mat';
>> t
```

```
t(1,1) =
```

```
Func:      @factfile
Params:
```

```

InputFiles: 'factfile_in.mat'
OutputFiles: 'factfile_out.mat'
Compose:      false
>> a = 5;
>> save('factfile_in.mat', 'a');
>> r=PAsolve(t)
Job submitted : 7
Awaited (J:7)
>> r.val

[ Wed Jan 12 20:57:47 CET 2011 ][pendule OUT]
[ Wed Jan 12 20:57:47 CET 2011 ][pendule OUT]

ans =

     1

>> load('-mat','factfile_out.m')
>> disp(b)

    120

```

[Back to Top](#)

## Chaining Remote Tasks

In the previous chapter, we learnt how to create a **PATask** with InputFiles and OutputFiles parameters. In this chapter, we will learn how to chain PATasks that will be run successively, i.e. the output of the first PATask will be given as the input of the second PATask, etc.

For the computation, we will use the matlab function sqrt which computes the square root of its argument. We won't use input or output files in this example, but of course it's possible to use this feature while chaining tasks:

```

>> t=PATask(3,1);
>> t(1:3,1).Func = @sqrt;
>> t(1,1).Params = 2;
>> t(2,1)=t(1,1);
>> t(2,1).Params={};
>> t(2,1).Compose=true;
>> t(3,1)=t(2,1);
>> t

t(1,1) =

    Func:      @sqrt
    Params: [2]
    Compose:      false

t(2,1) =

    Func:      @sqrt
    Params:
    Compose:      true

t(3,1) =

```

```

Func:      @sqrt
Params:
Compose:   true
>> r=PAsolve(t)
Job submitted : 8
Awaited (J:8)
>> r.val
[ Wed Jan 12 23:07:14 CET 2011 ][pendule OUT]
[ Wed Jan 12 23:07:14 CET 2011 ][pendule OUT]

ans =

    1.0905

>> sqrt(sqrt(sqrt(2)))

ans =

    1.0905

```

**Explanation:** the first 3 lines create a **PATask** with function **sqrt** and parameter **2**. The fourth line replicates this task to the second line. On the fifth and sixth command, we set the parameters to the second task to the empty list and we activate the **Compose** flag. This means that the first argument of the function **Func** will be taken from the result of the previous task instead of the task **Params** attribute. As **sqrt** has only one parameter, we set the **Params** attribute to the empty list. Finally, on the seventh line we replicate the line two of the **PATask** matrix, without changing it. At the end we verify that the result received matches the same computation done locally.

In these two examples we saw how to submit a **PATask** matrix, each time with a number of columns equal to 1. If we add more columns to the **PATask** matrix, it will mean that more parallel tasks will be submitted. To resume:

- A **line** vector of **PATask** of length **k** means that **k** **PATask** will be run in parallel
- A **column** vector of **PATask** of length **m** means that **m** **PATask** will be chained
- A **matrix** of **PATask** of size **m,k** means that **k** parallel series of **m** **PATask** chained together will be run

For convenience, it is also possible to call **PAsolve** with the following syntax:

```
r=PAsolve(c1,c2,...,ck)
```

where **ci** are column **PATask** vectors.

[Back to Top](#)

 Monitoring

Disconnected Mode 

© Copyright (C) 1997-2011 INRIA/University of Nice-Sophia Antipolis/ActiveEon [Terms of Use](#)





## Disconnected/Fault Tolerant mode

On this page...

[Introduction](#)

[Exiting Matlab before a job completion](#)

[Reconnecting and Retrieving previous job results](#)

[Using fault-tolerant sessions](#)

### Introduction

In the previous chapter we saw how we can run complex matlab code remotely, and wait for the results produced. Sometimes, when tasks are really long and matlab licences scarce, it can be very convenient to close the local matlab session to avoid consuming a token or local processing resources.

It is as well important, when submitting a long serie of distributed computations, that every intermediate result is kept and not lost if a Matlab crash occurs before the end.

In this chapter, we will learn how to use the **Disconnected/Fault Tolerant** mode of **Matlab Connector Toolbox**.

[Back to Top](#)

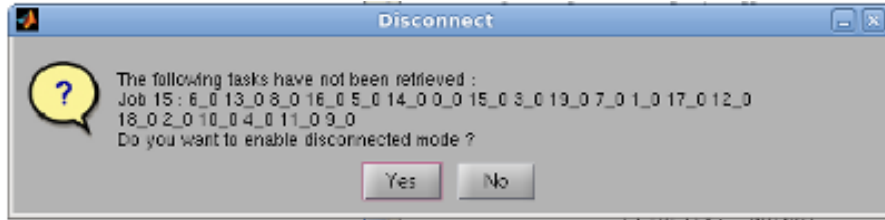
### Exiting Matlab before a job completion

In this example we will run a long list of factorial tasks and will close the matlab session:

```
>> res=PAsolve(@factorial,1,2,3,4,5,6,7,8,9,10,1,2,3,4,5,6,7,8,9,10)
Job submitted : 4
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
```

```
Awaited (J:4)
Awaited (J:4)
Awaited (J:4)
>>
>> exit
```

At this point a popup window will appear, asking if we want to enable disconnected mode, with the list of jobs unfinished.



We can focus on other works while the job is computed, but with a limitation: we cannot turn off the computer as the workers will try to transfer output data to the local computer when they terminate.

An alternative exists though, we can use a **Shared Dataspace** as explained in chapter [The PAoptions.ini file](#) to allow a completely disconnected mode. In the Shared Dataspace mode, all data will be kept in a cached space on the scheduler and it will not be necessary to leave the local computer on. It is also safer to use the Shared Dataspace mode when in an environment where network disconnections can occur.

[Back to Top](#)

## Reconnecting and Retrieving previous job results

When we restart the matlab session, we will need to reconnect to ProActive Scheduler in order to receive the job results. The **PAconnect** function will display the list of jobs that were unfinished at last session. We will then use the function **PAgetResults** to receive the results :

```
>> PAconnect('rmi://192.168.1.187:1099/')
log file in use : C:\Users\fviale\AppData\Local\Temp\MatlabJVMSpawnHelper.log
The following jobs were uncomplete before last matlab shutdown : 4
Connection successful, please enter login/password
Login successful

ans =

    '4'

>> res=PAgetResults(4)
Retrieving results of job 4
Job 4: Task 0_0
[2012-11-16 18:35:016 precision][MatlabExecutable]
[2012-11-16 18:35:016 precision][MatlabExecutable]

res =

    1

Job 4: Task 1_0
[2012-11-16 18:34:031 precision][MatlabExecutable]
```

```
[2012-11-16 18:34:031 precision] [MatlabExecutable]
```

```
res =
```

```
2
```

```
Job 4: Task 2_0
```

```
[2012-11-16 18:35:030 precision] [MatlabExecutable]
```

```
[2012-11-16 18:35:030 precision] [MatlabExecutable]
```

```
res =
```

```
6
```

```
...
```

The previous output has been stripped for clarity. Please note that only the jobs from the last matlab session will be kept this way and be accessible via `PAgetResults`, jobs from previous sessions will be unavailable. Sometimes it can be complicated to remember to which computation is associated the `JobID`, but it is possible to read the **jobid** attribute from the **PAresult** object right after submitting via `PAsolve` (non-blocking call) :

```
>> res2=PAsolve(@factorial,1,2,3,4,5)
```

```
Job submitted : 16
```

```
Awaited (J:16)
```

```
Awaited (J:16)
```

```
Awaited (J:16)
```

```
Awaited (J:16)
```

```
Awaited (J:16)
```

```
>> res2.jobid
```

```
ans =
```

```
16
```

```
ans =
```

```
16
```

```
ans =
```

```
16
```

```
ans =
```

```
16
```

```
ans =
```

```
16
```

This allows for example to store the job id into a mat file and automate the retrieval of results.

But it is also possible to fully automate the disconnected mode with Fault-tolerant sessions, as described in next chapter.

[Back to Top](#)

## Using fault-tolerant sessions

The idea behind fault-tolerant sessions is that when submitting a long serie of PAsolve computations (lasting several hours, days, ...), with post-processing of intermediate results and resubmission of additional PAsolve computations, any problem occurring (network disconnection, crash of the matlab program) can compromise the overall computation. It will be often necessary to restart the computation from zero.

A ProActive fault tolerant-session records automatically every step of the computation. Any intermediate result will be kept in a database. If Matlab crashes, or the user simply exits Matlab, all already retrieved results will be available at the next Matlab session.

The user will simply have to resubmit the same serie of PAsolve calls, no new ProActive Scheduler job will be resubmitted which have already been submitted in a previous session. Calls to PAwaitFor for results which have already been retrieved will be instantaneous.

It is thus important not to modify the parameters, the order, or the code of the functions used. Generally speaking, if the serie of computations is executed via a long Matlab script, then the same script must be rerun at the next session and it will very quickly reach the point it was before the crash.

Establishment of fault tolerant sessions is done via the function **PAbeginSession** and **PAendSession**:

**PAbeginSession** starts the initial fault-tolerant sesssion or reload an already started fault-tolerant session. Only one fault-tolerant session can be active at a time.

**PAendSession** stops the fault-tolerant session when the long computation is finished

### Example of use :

1) we start the session using PAbeginSession and do some computations, after a while matlab crashes:

```
>> PAbeginSession()
Started Recording Session at Fri Nov 16 19:26:20 CET 2012

ans =

     1

>> res=PAsolve(@factorial,1,2,3,4,5)
Job submitted : 5
Awaited (J:5)
Awaited (J:5)
Awaited (J:5)
Awaited (J:5)
Awaited (J:5)
>> val = PAwaitFor(res)
Job 5: Task 0_0
[2012-11-16 19:27:033 precision][MatlabExecutable]

Job 5: Task 1_0
[2012-11-16 19:27:033 precision][MatlabExecutable]
```

```
Job 5: Task 2_0
[2012-11-16 19:27:033 precision] [MatlabExecutable]
```

```
Job 5: Task 3_0
[2012-11-16 19:27:044 precision] [MatlabExecutable]
```

```
Job 5: Task 4_0
[2012-11-16 19:27:033 precision] [MatlabExecutable]
```

```
val =

      [1]      [2]      [6]     [24]    [120]
```

```
>> res2=PAsolve(@factorial,1,2,3,4,5)
Job submitted : 6
Awaited (J:6)
Awaited (J:6)
Awaited (J:6)
Awaited (J:6)
Awaited (J:6)
>> exit % simulate matlab crash
```

**2) After the crash, we start over Matlab and issue the same serie of commands (the PAwaitFor calls become instantaneous):**

```
>> PAconnect('rmi://192.168.1.187:1099/')
log file in use : C:\Users\fviale\AppData\Local\Temp\MatlabJVMSpawnHelper.log
The following jobs were uncomplete before last matlab shutdown : 6
Connection successful, please enter login/password
[2012-11-19 14:21:043 precision] [MIDDLEMAN]Logging to org.slf4j.impl.Log4jLoggerAdapter(org.mortbay.log)
[2012-11-19 14:21:043 precision] [MIDDLEMAN]jetty-6.1.x
[2012-11-19 14:21:044 precision] [MIDDLEMAN]Started SelectChannelConnector@0.0.0.0:3092
[2012-11-19 14:21:044 precision] [MIDDLEMAN]Remote Object Factory provider found
[2012-11-19 14:21:044 precision] [MIDDLEMAN]Remote Object Factory provider found
[2012-11-19 14:21:044 precision] [MIDDLEMAN]Remote Object Factory provider found
[2012-11-19 14:21:044 precision] [MIDDLEMAN]Remote Object Factory provider found
[2012-11-19 14:21:044 precision] [MIDDLEMAN]Remote Object Factory provider found
[2012-11-19 14:21:044 precision] [MIDDLEMAN]Remote Object Factory provider found
[2012-11-19 14:21:044 precision] [MIDDLEMAN]Remote Object Factory provider found
[2012-11-19 14:21:045 precision] [MIDDLEMAN]Detected an existing RMI Registry on port 1099
Login successful
```

```
ans =

      '6'
```

```
>> PAbeginSession()
Recalled Recorded Session at Fri Nov 16 19:26:20 CET 2012
```

```
ans =

      1
```

```
>> res=PAsolve(@factorial,1,2,3,4,5)
Job recalled : 5
Job 5: Task 0_0
[2012-11-16 19:27:033 precision] [MatlabExecutable]
```

```
res =
```

1

Job 5: Task 1\_0  
[2012-11-16 19:27:033 precision] [MatlabExecutable]

res =

2

Job 5: Task 2\_0  
[2012-11-16 19:27:033 precision] [MatlabExecutable]

res =

6

Job 5: Task 3\_0  
[2012-11-16 19:27:044 precision] [MatlabExecutable]

res =

24

Job 5: Task 4\_0  
[2012-11-16 19:27:033 precision] [MatlabExecutable]

res =

120

>> val = PAwaitFor(res)

val =

[1] [2] [6] [24] [120]

>> res2=PAsolve(@factorial,1,2,3,4,5)

Job recalled : 6

Job 6: Task 0\_0  
[2012-11-16 19:28:043 precision] [MatlabExecutable]

res2 =

1

Job 6: Task 1\_0  
[2012-11-16 19:28:044 precision] [MatlabExecutable]

res2 =

2


Job 6: Task 2\_0  
[2012-11-16 19:28:043 precision] [MatlabExecutable]


```
res2 =  
  
    6  
  
Job 6: Task 3_0  
[2012-11-16 19:28:055 precision] [MatlabExecutable]  
  
res2 =  
  
    24  
  
Job 6: Task 4_0  
[2012-11-16 19:28:043 precision] [MatlabExecutable]  
  
res2 =  
  
   120  
  
>> val2 = PAwaitFor(res2)  
  
val2 =  
  
    [1]    [2]    [6]   [24]   [120]  
  
>>
```

**3)** finally, we call `PAendSession()` to exit the fault-tolerant session :

```
>> PAendSession()  
Ended Recording Session started at Fri Nov 16 19:26:20 CET 2012  
  
ans =  
  
    1
```

[Back to Top](#)

 [Running Matlab functions remotely](#)

[Configuring PASolve behavior and Debugging](#) 

© Copyright (C) 1997-2011 INRIA/University of Nice-Sophia Antipolis/ActiveEon [Terms of Use](#)



## Configuring PAsolve behavior and Debugging

On this page...

[Introduction](#)

[The PAoptions.ini file](#)

[Using the PAoptions function](#)

[Debugging](#)

### Introduction

PAsolve can be configured using various options. It can be done either:

- **Statically** by editing the file **MatlabConnector/toolbox/config/PAoptions.ini** which will be loaded when the toolbox is loaded.
- **Dynamically** by calling the **PAoptions** function which will affect every subsequent PAsolve calls.

[Back to Top](#)

### The PAoptions.ini file

The **PAoptions.ini** file is located inside the ProActive Scheduler release at the following location :

`Matlab_Connector/toolbox/config/PAoptions.ini`

It contains many options which controls how PAsolve behave, we will detail a few important ones (see the PAoptions.ini file or the [PAoptions](#) function for the full list):

- **JobName**: Name of the job that will be submitted to the Scheduler
- **JobDescription**: Description of the job that will be submitted to the scheduler
- **Debug**: activate this option if any problem occurs and you want Matlab Connector Toolbox to run in debug mode (with verbose output)
- **TransferEnv**: this important option is by default set to off. If set to on, it allows the automatic transfer of the workspace where the PAsolve call is done to the remote matlab engines. Global variables are transferred but not local variables existing in workspaces higher than the current one. The options **EnvExcludeList** and **EnvExcludeTypeList** control the name and type of variables that should be excluded from the transfer.
- **LicenseSaverURL**: this very important option is the address of the proxy server which, if installed, acts as a proxy to the FlexNET server. The proxy server ensures that Matlab or toolboxes license tokens are available before any remote Matlab tasks starts. The scheduling of the task will be delayed until the tokens are available. The License Saver Proxy is not distributed with ProActive Toolbox and must be downloaded separately. See <http://www.activeeon.com> for availability and download.



- **CustomDataspaceURL** and **CustomDataspacePath**: by default these options are not set. Dataspaces are a ProActive internal mechanism allowing a shared file system between hosts. The mechanism allows the automatic transfer of files between the local matlab session and the remote engines. **Matlab Connector Toolbox** has an internal automatic Dataspace creation. But sometimes, one would like to rely on an external Dataspace such as a ftp server. This Dataspace MUST be accessible from the local file system in order to work with **Matlab Connector Toolbox**. The **CustomDataspaceURL** option gives the Universally accessible URL of this dataspace and the **CustomDataspacePath** gives the path to access this dataspace from the local file system.
- **SharedPushPublicUrl**, **SharedPullPublicUrl**, **SharedPushPrivateUrl**, **SharedPullPrivateUrl**: similarly to the CustomDataspaceURL option, these URLs define the addresses of the Shared DataSpaces (if the scheduler is configured to deploy them). Shared Dataspaces are used as "Cache" DataSpaces, the Matlab connector toolbox will push its input files to the SharedPushPublicUrl and pull the output files from the SharedPullPublicUrl. Private urls will be used by the workers to access the files.
- **VersionPref**, **VersionMin**, **VersionMax** and **VersionRej**: these options allow the user to define which version of Matlab should/should not be used. In an heterogeneous environment, several versions of matlab can be available simultaneously on the network. These options ensures that the submitted matlab code will meet the correct matlab version. It is the responsibility of the user to find which versions are/are not compatible with the code. For all these version use the version number of the form 7.9, 7.11, etc
- **VersionPref**: is the preferred version to use. It won't prevent another version to be used if the preferred version is not available. It is set automatically to the version of the local matlab session, but can be as well user defined via this option.
- **VersionMin** and **VersionMax**: is the minimum/maximum version to use. It is set by default to the minimum/maximum version supported.
- **VersionRej**: is a list of versions to be rejected.
- **VersionArch**: the matlab version architecture to be used. "Any" means any architecture can be used, "32" means only 32bit architectures and "64" means only 64-bit architectures..
- **Fork**: this option is set by default to off. It means the tasks will be executed (on remote hosts) in a separate Java Virtual Machine process.
- **RunAsMe**: this option is set by default to off. It means the tasks will be executed (on remote hosts) under the account of the currently logged user.
- **Priority**: this option is the priority of jobs created inside the ProActive Scheduler by PAsolve calls. Default to "Normal"
- **NbTaskExecution**: this option defines how many times maximum a task can be executed whenever it fails. This parameter is set by default to 2, which means that it will be restarted one time if an error occurs. This way, we can minimize the issues which can arise for example because of file transfer. On the other hand, a user task failure which would normally fail (syntax error, user code error) will also be rerun, and it is sometimes not desired (set this parameter to 1 to prevent that).
- **RemoveJobAfterRetrieve**: automatically removes the job from the scheduler after all results of the job have been retrieved.

[Back to Top](#)

## Using the PAoptions function

While options can be edited via the PAoptions.ini file, it is still possible to change options on the run via calls to the [PAoptions](#) function. The example in this chapter will detail the usage of the option **TransferEnv** which allows the local workspace to be transferred to remote engines.

We will define a function called **testenv** which will read a variable from the parent workspace and will compute the factorial of it. Here is how the **testenv** function is defined:

```
function out=testenv()  
a=evalin('caller', 'a');
```

```
out = factorial(a);
```

Here is how we set the TransferEnv option dynamically, and execute the call:

```
>> PAoptions('TransferEnv',true);
>> a=5

a =

     5

>> r=PAsolve(@testenv,{})
Job submitted : 4
Awaited (J:4)
>> r.val
Job 4: Task 0_0
[2012-11-19 18:54:058 precision][MatlabExecutable]

ans =

    120

>>
```

If now, we reset the TransferEnv option to false and try to execute testenv, we will receive the following error:

```
>> PAoptions('TransferEnv',false);
>> r=PAsolve(@testenv,{})
Job submitted : 5
Awaited (J:5)
>> r.val
Job 5: Task 0_0
[2012-11-19 18:57:000 precision][MatlabExecutable] Error occured in .
Error using ==> evalin
Undefined function or variable 'a'.

Error in ==> testenv at 2
a=evalin('caller','a');

Error in ==> PAMain at 9
out = testenv();

Warning: PABase:PAwaitFor Error during remote script execution
> In PABase.PAwaitFor at 141
   In PABase.subsref at 47

ans =

     []

>>
```

Indeed the remote execution of the testenv function couldn't find in its caller workspace the variable "a" !

[Back to Top](#)

## Debugging

This section is mainly for advanced users.

Debugging of the different Matlab Connector Toolbox components is done via all of the following:

- The **Debug** option.
- The **AutomaticFindMatlabXXX.log** and **ReserveMatlabXXX.log** files.
- The **MatlabExecutableXXX.log** and **MatlabStartXXX.log** files.
- The **MatSci\_Middleman\_JVM.log** and **MatlabJVMSpawnHelper.log** files.

Files with XXX in the above list are created with the proactive node name of the resource on which it is executed.

The **AutomaticFindMatlab**, **ReserveMatlab**, **MatlabExecutable** and **MatlabStart** log files are produced on the **worker hosts** *TEMP* directory. The TEMP directory is always */tmp* on linux systems. On Windows it can be known by opening a command line (cmd.exe) and typing `echo %TEMP%`:

```
C:\Users\Administrator>echo %TEMP%
C:\Users\Administrator\AppData\Local\Temp\2
```

The **AutomaticFindMatlab** log files logs the output of FindMatlab selection scripts which determines where to find matlab according to MatlabConfiguration.xml file or via automatic search(see [Automatic Configuration of ProActive Scheduler for Matlab](#) for more info). Read these files if you want to determine why the task couldn't execute on this worker host. Here is an example of those files:

```
AutomaticFindMatlablocal-LocalNodes-0:
Fri Nov 30 15:30:27 +0100 2012 : Finding Matlab on precision
Reading config in C:\Users\fviale\AppData\Local\Temp\MatlabWorkerConfiguration.xml
Found Matlab (7.11) : C:\Program Files\MATLAB\R2010b bin\win64 MATLAB.exe 64 bits
Deciding Matlab (7.11) : C:\Program Files\MATLAB\R2010b bin\win64 MATLAB.exe 64 bits
7.11(64) accepted
```

Similarly the **ReserveMatlab** log files are the output of ReserveMatlab selection scripts which reserve matlab tokens for computations when using the **LicenseSaverURL** option.

The **MatlabExecutable** log files are not created by default. They are the verbose output of the tasks submitted via PAsolve and are activated only if the PAoption Debug is set to on. Here is an example of those files:

```
MatlabExecutable_local-LocalNodes-0.log:
[2012-11-26 14:56:00 precision][MatlabExecutable] Unzipping source files from C:\Users\fviale\AppData\Local\Temp\PA_JVM6
[2012-11-26 14:56:00 precision][MatlabExecutable] Contents of C:\Users\fviale\AppData\Local\Temp\PA_JVM6
[2012-11-26 14:56:00 precision][MatlabExecutable] checktoolboxes_start_and_hide_desktop.m
[2012-11-26 14:56:00 precision][MatlabExecutable] keepalive_callback_fcn.m
[2012-11-26 14:56:00 precision][MatlabExecutable] MatlabPAsolveSrc_22_e37682bfe3442231097f57bfa66495983d
[2012-11-26 14:56:00 precision][MatlabExecutable] MatlabPAsolveVarIn_22_1_1.mat
[2012-11-26 14:56:00 precision][MatlabExecutable] myHello.m
[2012-11-26 14:56:00 precision][MatlabExecutable] Acquiring MATLAB connection using C:\Program Files\MATLAB\R2010b\bin\win64\MATLAB.exe
[2012-11-26 14:56:00 precision][MatlabExecutable] Adding to matlabpath sources from C:\Users\fviale\AppData\Local\Temp\PA_JVM6
[2012-11-26 14:56:00 precision][MatlabExecutable] checktoolboxes_start_and_hide_desktop( {'matlab'}, 'C:\Users\fviale\AppData\Local\Temp\PA_JVM6
[2012-11-26 14:56:00 precision][MatlabExecutable] Executing Keep-Alive timer
[2012-11-26 14:56:00 precision][MatlabExecutable] t = timer('Period', 300, 'ExecutionMode', 'fixedRate'); t.start();
[2012-11-26 14:56:00 precision][MatlabExecutable] Loading input variables from C:\Users\fviale\AppData\Local\Temp\PA_JVM6
[2012-11-26 14:56:00 precision][MatlabExecutable] Running MATLAB command: out = myHello(in1)
[2012-11-26 14:56:00 precision][MatlabExecutable] MATLAB command completed successfully, receiving output
[2012-11-26 14:56:00 precision][MatlabExecutable] Storing 'out' variable into C:\Users\fviale\AppData\Local\Temp\PA_JVM6
```

## Matlab Connector Toolbox

```
[2012-11-26 14:56:011 precision][MatlabExecutable] Testing output file : C:\Users\fviale\AppData\Local\Temp\
[2012-11-26 14:56:011 precision][MatlabExecutable] Closing MATLAB...
[2012-11-26 14:56:011 precision][MatlabExecutable]
  To get started, type one of these: helpwin, helpdesk, or demo.
  For product information, visit www.mathworks.com.
```

```
---- MATLAB START ----
```

```
Your variables are:
```

```
NODE_LIST      NODE_URL_LIST  in1          t
```

```
Number of nodes used : 2
```

```
Node nÂ°1: 192.168.1.187 rmi://192.168.1.187:1099/local-LocalNodes-1
```

```
Node nÂ°2: 192.168.1.187 rmi://192.168.1.187:1099/local-LocalNodes-0
```

```
Hello Dude1
```

```
out =
```

```
1
```

```
[2012-11-26 14:56:011 precision][MatlabExecutable] End of Task
```

The **MatlabStart** log file is the direct output of the Matlab process created on the worker. Unlike the **MatlabExecutable** log file, it is created each time.

```
/tmp/MatlabStart_local-LocalNodes-0.log:
```

```
  To get started, type one of these: helpwin, helpdesk, or demo.
  For product information, visit www.mathworks.com.
```

```
---- MATLAB START ----
```

```
Your variables are:
```

```
NODE_LIST      NODE_URL_LIST  in1          t
```

```
Number of nodes used : 2
```

```
Node nÂ°1: 192.168.1.187 rmi://192.168.1.187:1099/local-LocalNodes-1
```

```
Node nÂ°2: 192.168.1.187 rmi://192.168.1.187:1099/local-LocalNodes-0
```

```
Hello Dude1
```

```
out =
```

```
1
```

Finally the **MatSci\_Middleman\_JVM.log** file is created inside the temp directory of the **client host** and is logging everything concerning ProActive MiddleMan JVM. The **MatlabJVMSpawnHelper.log** is similar to the **MatSci\_Middleman\_JVM.log**. Here is an example:

```
/tmp/MatSci_Middleman_JVM.log:
```

```
[2012-04-25 19:29:016 pendule][AOMatlabEnvironment] Submitting job of 1 tasks...
[2012-04-25 19:29:016 pendule][AOMatlabEnvironment] Job 1 submitted.
[2012-04-25 19:29:016 pendule][AOMatlabEnvironment] waiting for request with no timeout
[2012-04-25 19:29:016 pendule][AOMatlabEnvironment] Request received : areAwaited
[2012-04-25 19:29:016 pendule][AOMatlabEnvironment] waiting for request with no timeout
[2012-04-25 19:29:022 pendule][AOMatlabEnvironment] Request received : taskStateUpdatedEvent
[2012-04-25 19:29:022 pendule][AOMatlabEnvironment] Received task 0_0 of Job 1 finished event...
```

## Matlab Connector Toolbox

```
[2012-04-25 19:29:022 pendule] [AOMatlabEnvironment] Looking for result of task 0_0 for job 1
[2012-04-25 19:29:022 pendule] [AOMatlabEnvironment] waiting for request with no timeout
[2012-04-25 19:29:022 pendule] [AOMatlabEnvironment] Request received : jobStateUpdatedEvent
[2012-04-25 19:29:022 pendule] [AOMatlabEnvironment] Received job 1 finished event...
[2012-04-25 19:29:023 pendule] [AOMatlabEnvironment] Updating results of job: 1(1) : Finished
[2012-04-25 19:29:023 pendule] [AOMatlabEnvironment] waiting for request with no timeout
[2012-04-25 19:29:024 pendule] [AOMatlabEnvironment] Request received : areAwaited
[2012-04-25 19:29:024 pendule] [AOMatlabEnvironment] waiting for request with no timeout
[2012-04-25 19:29:024 pendule] [AOMatlabEnvironment] Request received : waitAll
[2012-04-25 19:29:024 pendule] [AOMatlabEnvironment] Removed waitAll for job=1 request from the queue
[2012-04-25 19:29:024 pendule] [AOMatlabEnvironment] serving waitAll for job 1
[2012-04-25 19:29:024 pendule] [AOMatlabEnvironment] Sending the results of task 0_0 of job 1 back...
[2012-04-25 19:29:024 pendule] [AOMatlabEnvironment] waiting for request with no timeout
[2012-04-25 19:29:024 pendule] [AOMatlabEnvironment] Request received : isConnected
[2012-04-25 19:29:024 pendule] [AOMatlabEnvironment] waiting for request with no timeout
[2012-04-25 19:29:024 pendule] [AOMatlabEnvironment] Request received : isLoggedIn
[2012-04-25 19:29:024 pendule] [AOMatlabEnvironment] waiting for request with no timeout
[2012-04-25 19:29:024 pendule] [AOMatlabEnvironment] Request received : jobRemove
[2012-04-25 19:29:024 pendule] [AOMatlabEnvironment] waiting for request with no timeout
[2012-04-25 19:29:046 pendule] [AOMatlabEnvironment] Request received : isConnected
[2012-04-25 19:29:046 pendule] [AOMatlabEnvironment] waiting for request with no timeout
[2012-04-25 19:29:047 pendule] [AOMatlabEnvironment] Request received : solve
```

[Back to Top](#)



Disconnected mode

Using Matlab Connector Toolbox Simulink GUI



© Copyright (C) 1997-2011 INRIA/University of Nice-Sophia Antipolis/ActiveEon [Terms of Use](#)



## Using Matlab Connector Toolbox Simulink GUI

On this page...

[Introduction](#)

[Starting the GUI](#)

[Defining model and parameters](#)

[Running the model](#)

[Viewing results](#)

[Using Disconnected Mode](#)

### Introduction

The **Matlab Connector Toolbox Simulink GUI** is a simple GUI which allows to run a Simulink model on a remote engine. It allows to do batch simulations through the use of Matlab script initialization parameters. Thus, the same model can be run multiple times with parameters. The GUI is NOT an interface similar in its complexity to the Simulink interface, it doesn't allow real time display of the simulation. But results from the simulations can be displayed graphically.

In order to display the results the model MUST save remotely output signals to mat files via the **To File** block. Usages in the model of **Scope** or **To workspace** should be avoided as it would slow down the remote simulation and consume memory.

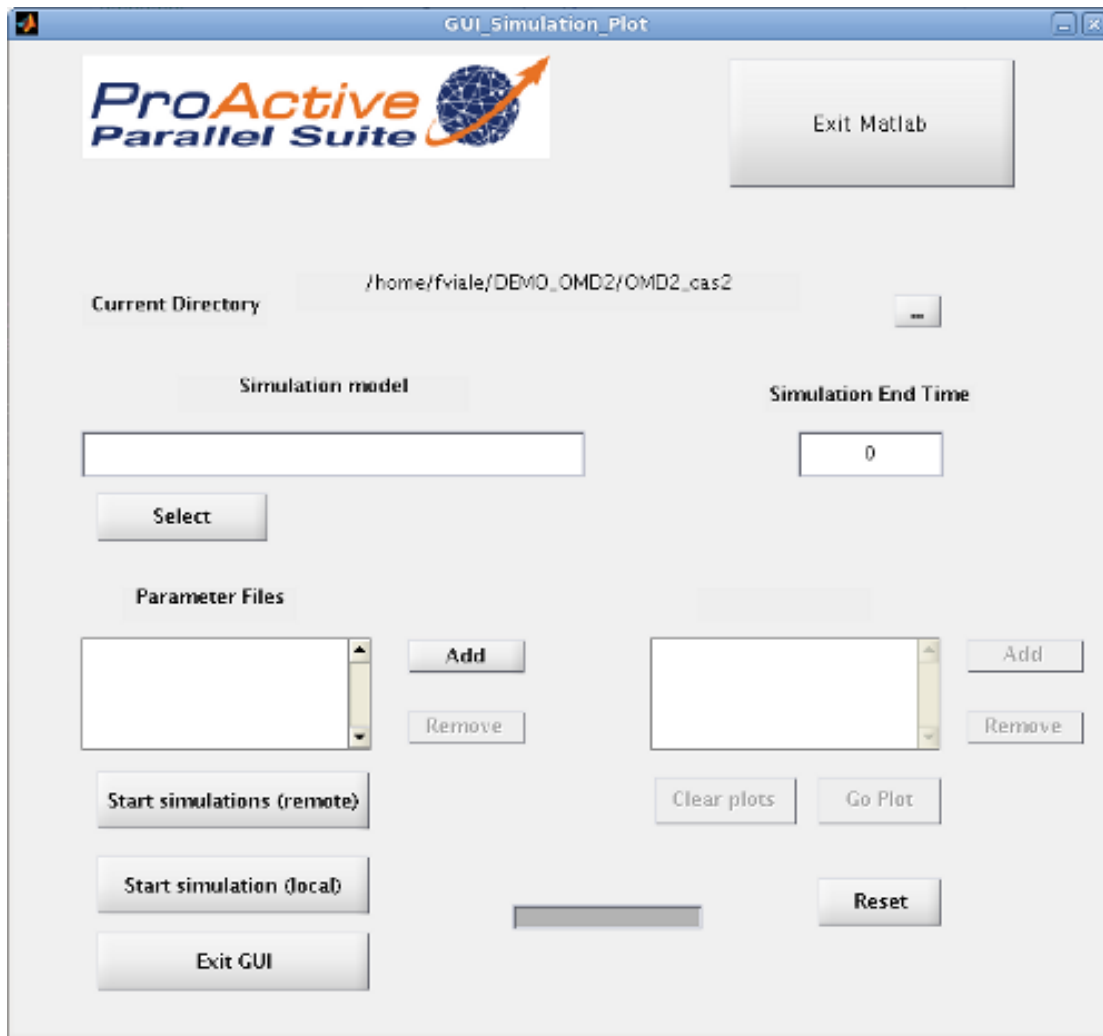
[Back to Top](#)

### Starting the GUI

The **Matlab Connector Toolbox Simulink GUI** is located at MATSCI/matlab/toolbox/GUI\_Simulink , before opening the GUI, make sure it's available in matlab path. Opening the GUI can be done either by running the function **GUI\_Simulation\_Plot**, or via Matlab start button at **Start > Toolboxes > ProActive Scheduler > Scheduler Simulink GUI :**

```
>> addpath('/home/fviale/eclipse_workspace/Matlab_Scilab_Connector/matlab/toolbox/GUI_Simulink/');
>> GUI_Simulation_Plot
```

At startup, a popup window appears asking if we want to load an existing model, we answer No for now. Here is a view of the GUI:



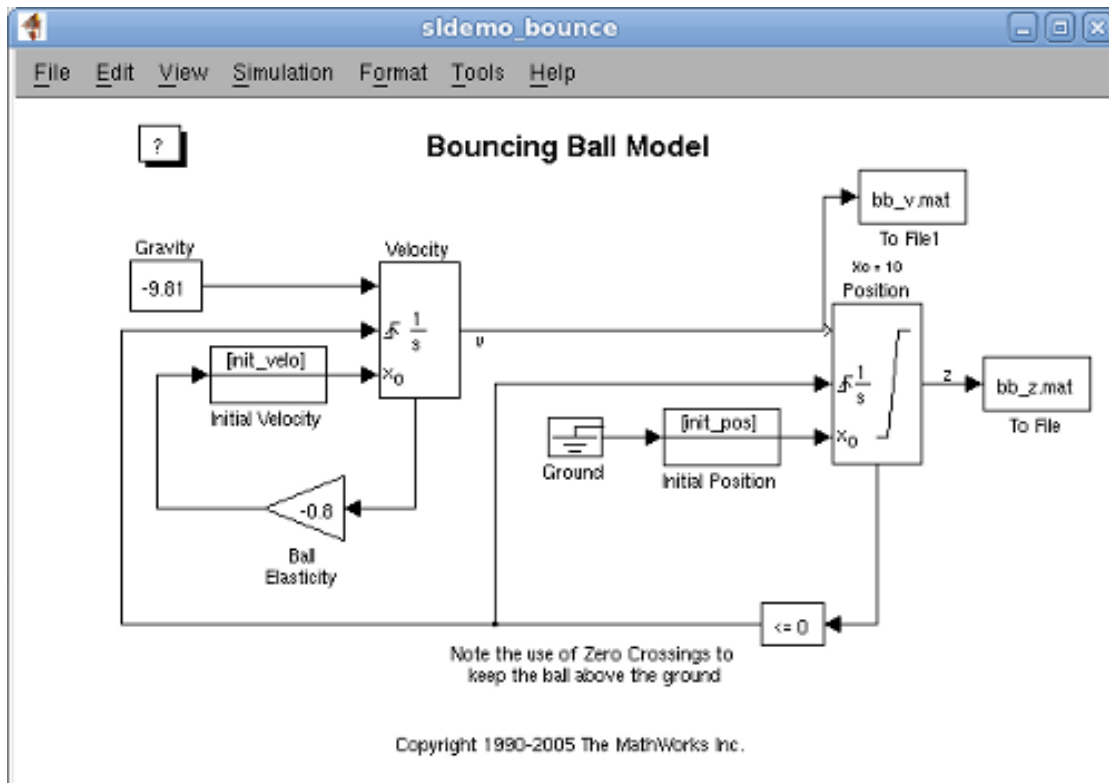
[Back to Top](#)

## Defining model and parameters

We will now create a model to use with the GUI and a set of parameters. We will reuse the Simulink Demo **Bouncing Ball Model** for this purpose. We edit the bouncing ball demo to fit to our use, doing the following:

- We remove the two **Scope** outputs present in the demo and replace it with **To file** blocks.
- We modify the **Initial Velocity** and **Initial Position** blocks by changing their initial value from a constant to a user defined variable.

Here is how the model looks like (available at MATSCI/matlab/toolbox/GUI\_Simulink/demo):



We will then write 3 matlab scripts which will assign the values of the **init\_velo** and **init\_pos** variables:

```
Param1.m:
init_velo=15;
init_pos=10;
```

```
Param2.m:
init_velo=10;
init_pos=30;
```

```
Param3.m:
init_velo=10;
init_pos=30;
```

[Back to Top](#)

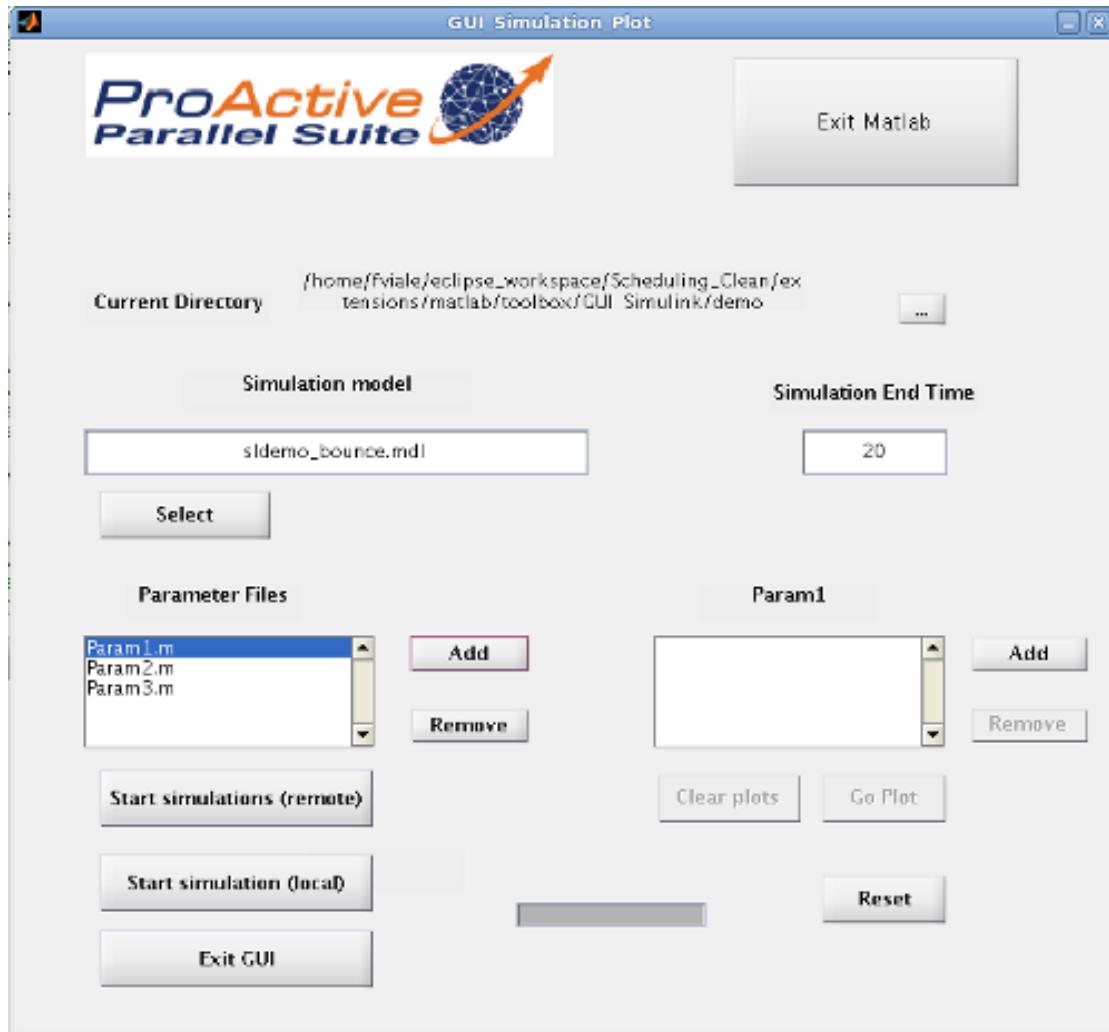
## Running the model

We are now ready to run the model. We must make sure beforehand that a connection to the scheduler is established. We will open the GUI and do the following actions:

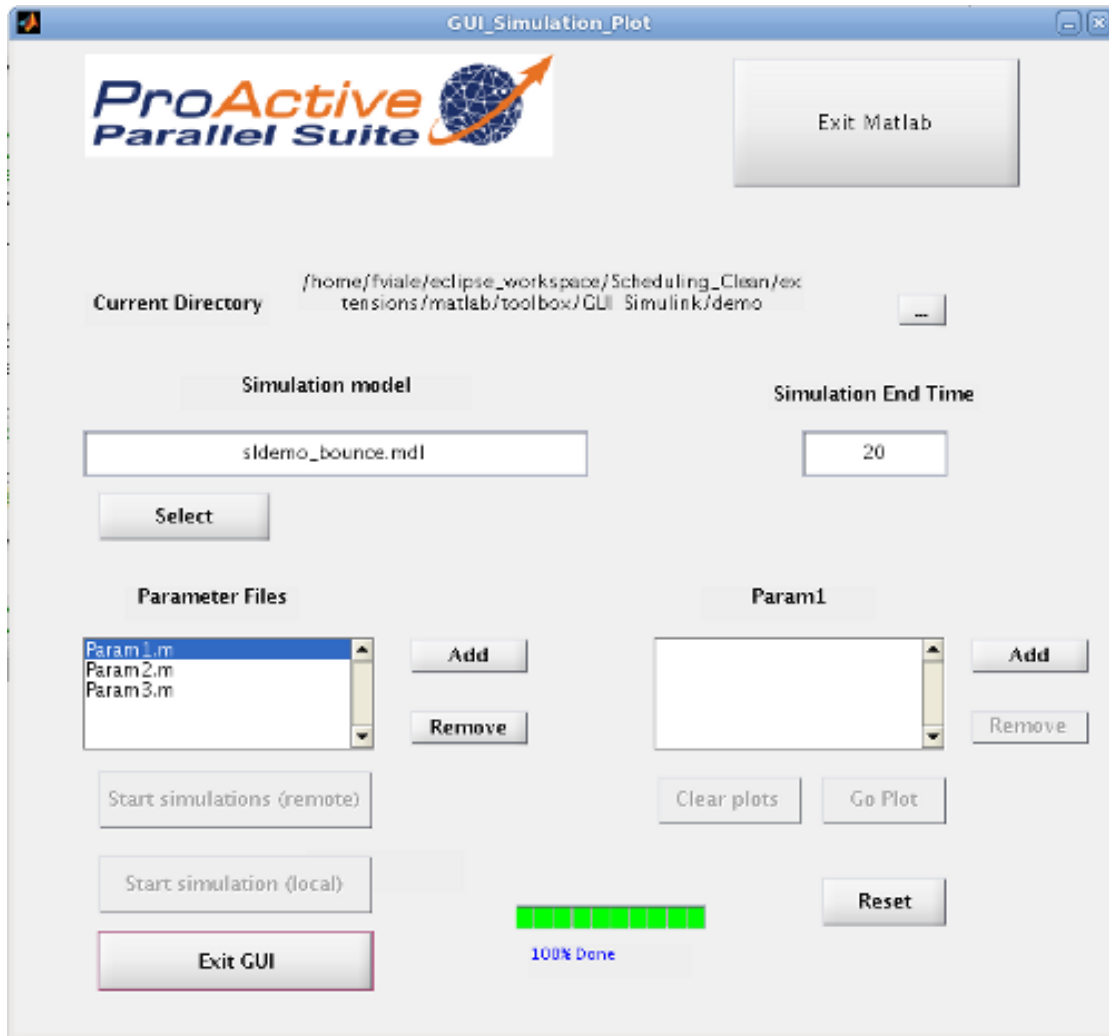
1. We change the **Current Directory** to the directory where the model and parameter files are (they must be in the same directory, here Matlab\_Scilab\_Connector/matlab/toolbox/GUI\_Simulink/demo).
2. We select the model in the **Simulation Model** field (here sldemo\_bounce.mdl).
3. We set the **Simulation End Time**, it will be the same for all batches run (here 20).
4. We add the 3 ParamX.m files that we created in the **Parameter Files** list.
5. Finally we press the **Start Simulations (remote)** button.



Here is how the GUI looks like before pressing start:



Here is how the GUI looks like after all simulations are executed:



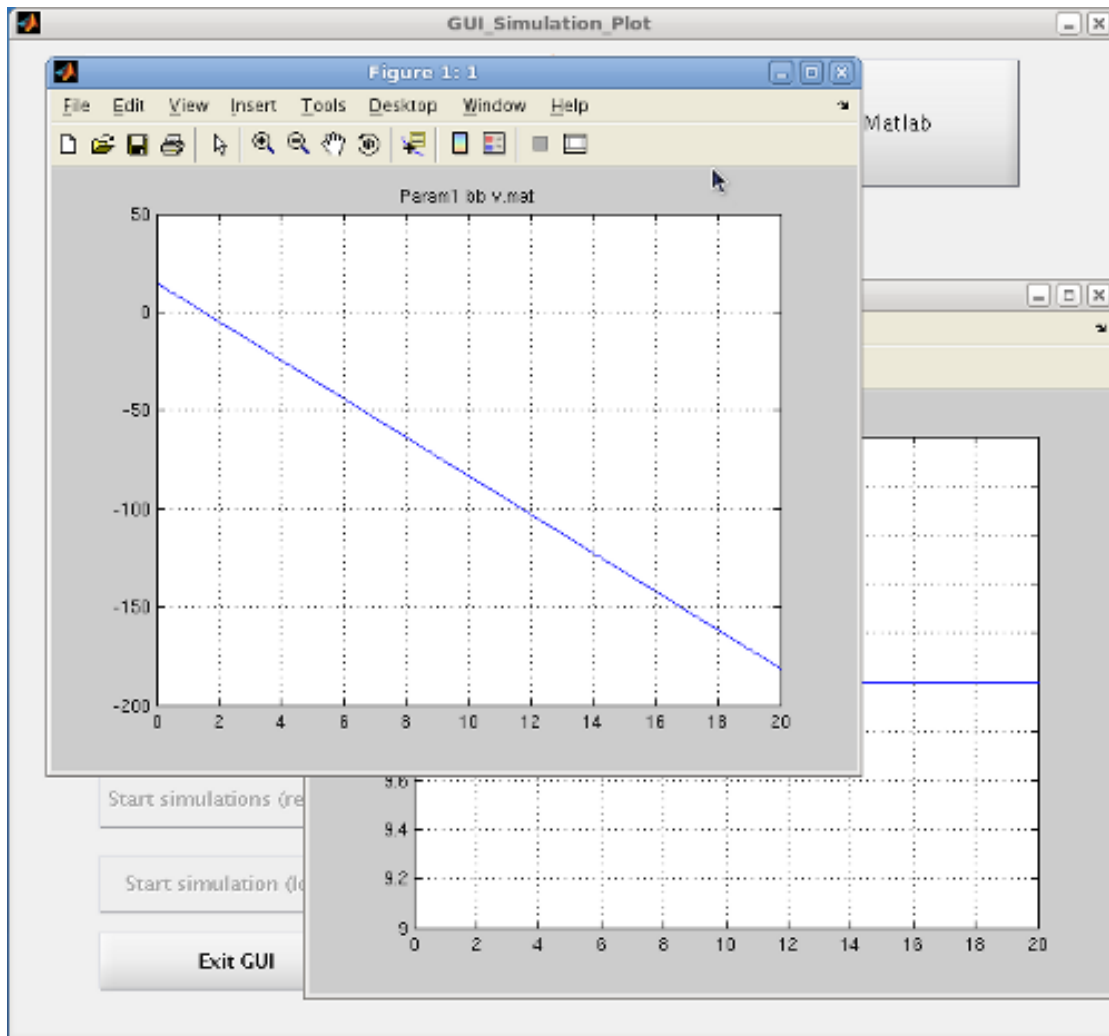
[Back to Top](#)

## Viewing results

Now that all simulations are executed, we want to visualize the results. This is done via the list box on the right side of the GUI.

1. We first need to select on the left list (**Parameter Files**) the parameter file from which we want to display results. Here we choose **Param1**.
2. We will then press the **Add** button to add signals to be visualized. Here we will choose the two output files **bb\_v.mat** and **bb\_z.mat**.
3. Finally we press the **Go Plot** button which will pop up two graphical plot windows displaying the results.

Here is the results of the simulation with Param1:



We can repeat the previous steps for the other parameter files Param2 nad Param3.


[Back to Top](#)

## Using Disconnected Mode

It is possible to use the GUI in combination with **Matlab Connector Toolbox Disconnected Mode**. In order to do that:

1. Submit a simulation.
2. Hit the button Exit Matlab. Accept the disconnected mode.
3. At next Matlab Startup reconnect to the scheduler and reopen the GUI.
4. A popup window will appear saying that an existing running model has been found. Accept to reload it.
5. The simulation will be updated to its current state and if the simulation is finished results will be accessible.

[Back to Top](#)

 [Configuring PAsolve behavior and Debugging](#)

[Function Reference](#) 

© Copyright (C) 1997-2011 INRIA/University of Nice-Sophia Antipolis/ActiveEon [Terms of Use](#)



# Function Reference

## [Connection and Status](#)

Connection to the Scheduler and other administrative commands

## [Submitting](#)

Submitting matlab computations to the Scheduler

## [Configuration](#)

Configures PAsolve options and behavior

## [Disconnected Mode](#)

Commands which are specific to the Disconnected Mode

[Back to Top](#)

## Connection and Status

### [PAconnect](#)

Connects to the ProActive Scheduler

### [PAisConnected](#)

tells if this Matlab session is connected to a ProActive Scheduler

### [PAensureConnected](#)

makes sure that this Matlab session is connected to a ProActive Scheduler

### [PAdisconnect](#)

disconnects Matlab from the ProActive Scheduler.

### [PAstate](#)

State of ProActive Scheduler

### [PAjobState](#)

State of the given job

### [PAjobRemove](#)

Removes the given job from the scheduler

### [PAjobOutput](#)

Textual log of the given job

### [PAjobResult](#)

Textual result of the given job

### [PAkillJob](#)

Kills the given job

### [PApauseJob](#)

Pauses the given job

### [PAresumeJob](#)

Resumes the given job

### [PAkillTask](#)

Kills the given task

### [PAtaskOutput](#)

Textual log of the given task

### [PAtaskResult](#)

Textual result of the given task

[Back to Top](#)

## Submitting

### [PAsolve](#)

Runs Matlab function remotely

|                             |  |
|-----------------------------|--|
| <a href="#">PATask</a>      | PATask objects are used to define complex computations with PAsolve  |
| <a href="#">PAFile</a>      | PAFile objects are used to define input and output files for PATasks |
| <a href="#">PAREsult</a>    | PAResult objects are returned by PAsolve calls                       |
| <a href="#">PAwaitFor</a>   | Waits for the computation of a set of PAREsult                       |
| <a href="#">PAwaitAny</a>   | Waits for the first computation available in a given set of PAREsult |
| <a href="#">PAisAwaited</a> | Tells, within a set of PAREsult, which results are available or not  |
| <a href="#">PAarrayfun</a>  | apply a remote function call to each element of an array             |

[Back to Top](#)

## Configuration

|                           |                             |
|---------------------------|-----------------------------|
| <a href="#">PAoptions</a> | Configures PAsolve options. |
|---------------------------|-----------------------------|

[Back to Top](#)

## Disconnected Mode

|                                |   |
|--------------------------------|---|
| <a href="#">PAgetResults</a>   | Gets the results of a job submitted during the previous Matlab session. |
| <a href="#">PAbeginSession</a> | Starts a fault-tolerant PAsolve session.                                |
| <a href="#">PAendSession</a>   | Ends a fault-tolerant PAsolve session.                                  |

 Using Matlab Connector Toolbox Simulink GUI

## PAconnect

PAconnect connects to the ProActive scheduler

### Syntax

```
PAconnect();  
PAconnect(url [,credpath]);  
jobs = PAconnect(url [,credpath]);
```

### Inputs

url - url of the scheduler  
credpath - path to the login credential file

### Outputs

jobs - id of jobs that were not terminated at matlab's previous shutdown

### Description

PAconnect connects to a running ProActive Scheduler by specifying its url. If the scheduler could not be reached, a Java window will appear, asking for login and password. An additional SSH key can also be provided when connecting to execute remote task under one's identity (RunAsMe option). ProActive Scheduler features a full authentication facility along with the possibility to synchronize to existing Windows or Linux accounts via LDAP. For more details, see ProActive Scheduler's manual chapter "Configure users authentication". If you haven't configured any account, the default account login "demo", password "demo".

You can as well encrypt credentials using the command line tool "create-cred" and provide the path to the credential file with the parameter credpath. Or you can simply reuse automatically the same credentials and use the last scheduler connection by using PAconnect() without parameter.

PAconnect() without parameter can also be used to connect to a local ProActive scheduler deployed on a machine using the RMI protocol.

In case jobs from the last Matlab session were not complete before Matlab exited. It is possible to retrieve the results using PAgetResults.

### Example

```
PAconnect('rmi://scheduler:1099')
```

### See also

[PASolve](#), [PAgetResults](#)

MATLAB File Help: PAisConnected

[View code for PAisConnected](#)

[Default Topics](#)

## PAisConnected

PAisConnected tells if this Matlab session is connected to a ProActive Scheduler

### Syntax

```
tf=PAisConnected();
```

### Outputs

tf - boolean value, true if this Matlab session is connected to a ProActive Scheduler

### Description

PAisConnected tells if this Matlab session is properly connected to a ProActive Scheduler.

### See also

[PAconnect](#), [PAdisconnect](#)



MATLAB File Help: PAensureConnected

[View code for PAensureConnected](#)

[Default Topics](#)

## PAensureConnected

PAensureConnected makes sure that we are connected to the ProActive Scheduler.

### Syntax

```
PAensureConnected();
```

### Description

PAensureConnected makes sure that we are connected to the ProActive Scheduler. If we are not connected, PAensureConnected will try to reconnect and will keep trying until the scheduler is reachable again, using the same credentials provided during the PAconnect call.

### See also

[PAconnect](#), [PAdisconnect](#), [PAisConnected](#)

MATLAB File Help: PAdisconnect

[View code for PAdisconnect](#)

[Default Topics](#)

## PAdisconnect

PAdisconnect disconnects Matlab from the ProActive Scheduler.

### Syntax

```
PAdisconnect();
```

### Description

PAdisconnect disconnects Matlab from the ProActive Scheduler.

See also

[PConnect](#), [PAisConnected](#)

MATLAB File Help: PAstate

[View code for PAstate](#)

[Default Topics](#)

## PAstate

PAstate prints the current state of the Scheduler

### Syntax

```
>> PAstate();  
>> out = PAstate()
```

### Outputs

out - a string containing the information printed (capture)

### Description

PAstate prints the current state of Scheduler. It prints which jobs are Pending, Running or Finished. To print the inner state of any given job, use PAjobState. Warning, this function can be time-consuming if the scheduler database

### See also

[PAjobState](#)

## PAjobState

PAjobState prints the current state of the given Scheduler job

### Syntax

```
>> PAjobState(jobid);  
>> out = PAjobState(jobid)
```

### Inputs

jobid - the id of the job (string or numeric)

### Outputs

out - a string containing the information printed (capture)

### Description

PAjobState prints the current state of the given ProActive Scheduler job. It prints the current state of each Task composing this job whether there are Running, Finished or Failed.

### See also

[PAstate](#)

MATLAB File Help: PAjobRemove

[View code for PAjobRemove](#)

[Default Topics](#)

## PAjobRemove

PAjobRemove removes the given job from the scheduler

### Syntax

```
>> PAjobRemove(jobid);
```

### Inputs

jobid - the id of the job (string or numeric)

### Description

PAjobRemove removes the given job from the scheduler. By default, the job is automatically removed from the scheduler when all results have been read. The option RemoveJobAfterRetrieve can be set to false to manually remove the job with the PAjobRemove function.

### See also

[PAjobResult](#), [PAtaskResult](#), [PAtaskOutput](#)

MATLAB File Help: PAjobOutput

[View code for PAjobOutput](#)

[Default Topics](#)

## PAjobOutput

PAjobOutput text output of the given Scheduler job

### Syntax

```
>> PAjobOutput(jobid);  
>> out = PAjobOutput(jobid)
```

### Inputs

jobid - the id of the job (string or numeric)

### Outputs

out - a string containing the output

### Description

PAjobOutput prints the textual output of the given ProActive Scheduler job.

### See also

[PAjobResult](#), [PAtaskResult](#), [PAtaskOutput](#)

## PAjobResult

PAjobResult returns or prints the textual representation of the result of the given Scheduler job

### Syntax

```
>> PAjobResult(jobid);  
>> out = PAjobResult(jobid)
```

### Inputs

jobid - the id of the job (string or numeric)

### Outputs

out - a string containing the result

### Description

PAjobResult returns or prints the textual representation of the result of the given Scheduler job. This function is for convenient-use only, it is mainly used to look at results of jobs handled by the scheduler which are not Matlab jobs as PAjobResult won't return the actual Matlab object result. To get the actual Matlab object result, use PAgetResults. To get the output log of the given job, use PAjobOutput.

### See also

[PAgetResults](#), [PAjobOutput](#), [PAtaskResult](#), [PAtaskOutput](#)

MATLAB File Help: PAkillJob

[View code for PAkillJob](#)

[Default Topics](#)

## PAkillJob

PAkillJob kills the given Scheduler job

### Syntax

```
>> PAkillJob(jobid);
```

### Inputs

jobid - the id of the job (string or numeric)

### Description

PAkillJob kills the given Scheduler job.

See also

[PApauseJob](#), [PAresumeJob](#), [PAkillTask](#)



MATLAB File Help: PApauseJob

[View code for PApauseJob](#)

[Default Topics](#)

## PApauseJob

PApauseJob pauses the given Scheduler job

### Syntax

```
>> PApauseJob(jobid);
```

### Inputs

jobid - the id of the job (string or numeric)

### Description

PApauseJob pauses the given Scheduler job. No new task will be scheduled until a call to PArsumeJob is made.

### See also

[PAkillJob](#), [PAkillTask](#), [PArsumeJob](#)

MATLAB File Help: PAresumeJob

[View code for PAresumeJob](#)

[Default Topics](#)

## PAresumeJob

PAresumeJob resumes the given Scheduler job

### Syntax

```
>> PAresumeJob(jobid);
```

### Inputs

jobid - the id of the job (string or numeric)

### Description

PAresumeJob resumes the given Scheduler job, previously paused by PApauseJob.

### See also

[PAkillJob](#), [PApauseJob](#), [PAkillTask](#)

MATLAB File Help: PAkillTask

[View code for PAkillTask](#)

[Default Topics](#)

## PAkillTask

PAkillJob kills the given Scheduler task

### Syntax

```
>> PAkillTask(jobid, tasmname);
```

### Inputs

jobid - the id of the job (string or numeric)  
taskname - the name of the task (string)

### Description

PAkillTask kills the given Scheduler task of the given Scheduler job.

### See also

[PAkillJob](#), [PApauseJob](#), [PAresumeJob](#)

## PAtaskOutput

PAtaskOutput text output of the given Scheduler task

### Syntax

```
>> PAtaskOutput(jobid, taskname);  
>> out = PAtaskOutput(jobid, taskname)
```

### Inputs

jobid - the id of the job (string or numeric)  
taskname - the name of the task (string)

### Outputs

out - a string containing the output

### Description

PAtaskOutput prints the textual output of the given Scheduler task of the given Scheduler job.

### See also

[PAtaskResult](#), [PAjobResult](#), [PAjobOutput](#)

## PAtaskResult

PAtaskResult returns or prints the textual representation of the result of the given Scheduler job

### Syntax

```
>> PAtaskResult(jobid);  
>> out = PAtaskResult(jobid)
```

### Inputs

jobid - the id of the job (string or numeric)  
taskname - the name of the task (string)

### Outputs

out - a string containing the result

### Description

PAtaskResult returns or prints the textual representation of the result of the given Scheduler task. PAtaskResult won't return the actual Matlab object result. To get the actual Matlab object result, use PAggetResults. To get the output log of the given task, use PAtaskOutput.

### See also

[PAgetResults](#), [PAtaskOutput](#), [PAjobResult](#), [PAjobOutput](#)

## PAsolve

PAsolve run matlab functions remotely

### Syntax

Basic:

```
>> results = PAsolve(func, arg_1, arg_2, ..., arg_n);
```

Advanced:

```
>> results = PAsolve(patask_1(1..k), patask_2(1..k), ... ,  
PATask_n(1..k));
```

```
>> results = PAsolve(patask(1..n,1..k));
```

### Inputs

func - a matlab function handle

arg\_k - a parameter to the "func" function if func takes only one parameter as input OR a cell-array containing the list of parameters to func.

patask\_k - a vector of PATask objects

patask - a matrix of PATask objects

### Description

The call to PAsolve is synchronous until the scheduler has received the information necessary to run the tasks. PAsolve returns right afterwards and doesn't block matlab until the tasks have been scheduled and completed.

PAsolve returns an array of objects of type PABaseResult. Its size matches the number of argk or pataskk given or the number of columns in the patask matrix.

Blocking wait functions can be called on this PABaseResult array or on a portion of this array (see PABaseWaitFor, PABaseWaitAny). Non-blocking functions can also be called to know if a result is available (PABaseIsAwaited)

PAsolve is based on the principle of parametric sweep, i.e. one task/many parameters (see Basic syntax).

PAsolve can either be called by giving a function handle and a list of parameters (Basic Syntax), or by providing arrays of PABaseTask objects which allows more advanced parametrization of the execution (see PABaseTask).

The semantic of execution for PABaseTask matrices is that each column will be executed separately, and within each column each line will be execute sequentially and thus will depend on the execution of the previous line.

PAsolve behaviour can be configured using the PABaseOptions function.

### See also

[PABaseConnect](#), [PABaseOptions](#), [PABaseGetResults](#), [PABaseTask](#), [PABaseResult](#), [PABaseResult/PABaseWaitFor](#), [PABaseResult/PABaseWaitAny](#), [PABaseResult/PABaseIsAwaited](#)

## PATask

PATask constructor of PATask objects

### Syntax

```
t = PATask(lin, col);
```

### Inputs

lin - the number of lines of the PATask matrix to create  
col - the number of columns of the PATask matrix to create

### Outputs

t - a PATask matrix

### Properties

Func - a matlab function handle (the function to execute remotely)

Params - a cell array containing the list of parameters for the "Func" function

Description - a textual description of this task (string)

InputFiles - a PAFfile array containing information about the desired input files used by this task.

OutputFiles - a PAFfile array containing information about output files that will be generated by this task remotely and copied back from the remote machine

SelectionScript - a string containing the pathnames of a user defined selection script (in languages such as Javascript, Ruby, Python), see ProActive Scheduler documentation for more information.

Static - a boolean, true if the SelectionScript is a static one (executed only once on a given machine), otherwise the SelectionScript will be dynamic (default)

ScriptParams - a string containing the parameters of the custom script delimited by spaces.

Compose - a boolean which defines if this task depends on the result of the previous task(previous line) inside this PATask column. If Compose is set to true, then the FIRST parameter of the Func function will be taken from the result of the previous task, subsequent parameters will be taken from the "Params" list.

NbNodes - an integer value which indicates the number of Scheduler Nodes necessary to run this task. This parameter is interesting in case the Matlab engine is running in multithreaded mode to specify the number of processors necessary or if an external program is called from the task, and this programs requires multiple processors and/or machines.

Topology - a string parameter which affects how nodes will be chosen to meet the NbNodes required. The values possible are : 'arbitrary', 'bestProximity', 'thresholdProximity', 'singleHost', 'singleHostExclusive', 'multipleHostsExclusive', 'differentHostsExclusive'  
Please refer to ProActive Scheduler manual for more information on

these values.

ThresholdProximity - an integer value which indicates the threshold in case the 'thresholdProximity' is set on the Topology attribute. Please refer to ProActive Scheduler manual for more information on this setting.

In case the topology parameters are used, the remote matlab engines can access the hostnames list of the Nodes used to run the task.

This hostnames list can then be used to run parallel code on those machines. As multiple ProActive Nodes can be deployed on the same host, the list may contain several times the same hostname. This list will be assigned to the variable `NODE_LIST` in the workspace calling the `Func` function. So, in order to retrieve this list, you'll need to add the following code to your Matlab function:

```
nl = evalin('caller','NODE_LIST');
```

"nl" will be affected a cell array of strings. The first element of this list is always the Node where the function `Func` is executed. The `TestTopology` function in the `Tests` folder provides an example of topology usage.

### Examples

```
>> t = PTask(2,1);
>> t.Func = @factorial;
>> t(1,1).Params = {3};
>> t(2,1).Compose = true;
>> r = PAsolve(t);
>> val = PAwaitFor(r)
val =
    720    % Result of factorial(factorial(3))
```

See also

[PAsolve](#), [PAFile](#)



## PAREsult

PAREsult constructor of PAREsult objects

### Properties

jobid - id of the Scheduler job submitted when calling PAsolve and receiving this PAREsult object.

val - contains the result of the computation, if the result is not yet available, calling the val property will result in blocking Matlab execution until the result is computed.

logs - the textual log associated with this PAREsult object (similarly to the val property, it can block matlab execution).

isError - a boolean value indicating whether this result has triggered an error in the remote execution or not.

### Methods

PAwaitFor - waits for the computation of the given PAREsult array

PAwaitAny - waits until any result has been computed in the given array of PAREsult objects.

PAisAwaited - tells which results among the given array of PAREsult objects is available.

### See also

[PAsolve](#), [PAREsult/PAwaitFor](#), [PAREsult/PAwaitAny](#), [PAREsult/PAisAwaited](#)

M-File Help: [PAREsult/PAwaitFor](#)

[View code for PAREsult/PAwaitFor](#)

[Default Topics](#)

## PAREsult/PAwaitFor

PAREsult/PAwaitFor blocks matlab execution until a set of results are available

### Syntax

```
>> val=PAwaitFor(r)
```

### Inputs

r - an array of PAREsult objects received by a call to PASolve

### Outputs

val - if r is a scalar, val contains the real result of the computation. If r is a vector, then val will contain a cell array containing the real results.

### Description

PAREsult/PAwaitFor will block matlab execution while waiting for a given set of results. PAREsult/PAwaitFor will wait until every results of the given set have been computed.

### Example

```
>> r=PASolve(@factorial, 1, 2, 3, 4);  
>> val = PAwaitFor(r) % Blocks Matlab execution until factorial(1), ..  
    , factorial(4) have been computed remotely and returns the results as  
val = {factorial(1), .. , factorial(4)}
```

### See Also

[PASolve](#), [PAREsult/PAwaitAny](#), [PAREsult/PAisAwaited](#)

## MATLAB File Help: PAwaitAny

[Default Topics](#)

### PAwaitAny

--- help for PAREsult/PAwaitAny ---

PAREsult/PAwaitAny blocks matlab execution until any result is available in a given array of PAREsult object

#### Syntax

```
>> val = PAwaitAny(r)
>> [val, index] = PAwaitAny(r)
```

#### Inputs

r - an array of PAREsult objects received by a call to PASolve

#### Outputs

val - contains the real result of the computation.  
index - contains the index of the result computed in the r array.

#### Description

PAREsult/PAwaitAny will block matlab execution while waiting for a given set of results. PAREsult/PAwaitAny differs from PAREsult/PAwaitFor in that it will wait for the first result available and returns this result, allowing post-processing treatments to be executed immediately without having to wait for a large set of results. Further calls to PAwaitAny will return the remaining results until all results have been computed.

#### Example

```
>> r=PASolve(@factorial, 1, 2, 3, 4);
>> val = PAwaitAny(r) % Blocks Matlab execution until either factorial(1), ..
, or factorial(4) has been computed remotely and returns the result as
val = factorial(i)
>> val2 = PAwaitAny(r) % Blocks for the second result, etc ...
```

```
r=PASolve(@factorial, 1, 2, 3, 4);
Job submitted : 24
[val, index] = PAwaitAny(r)
val =
    1
index =
    1    1
[val, index] = PAwaitAny(r)
val =
    2
index =
    1    2
[val, index] = PAwaitAny(r)
val =
    6
index =
    1    3
[val, index] = PAwaitAny(r)
val =
   24
index =
    1    4
```

See Also

[PAolve](#), [PAREsult/PAwaitFor](#), [PAREsult/PAisAwaited](#)

MATLAB File Help: PAisAwaited

[Default Topics](#)

PAisAwaited

--- help for PAREsult/PAisAwaited ---

PAREsult/PAisAwaited checks if results among an array of PAREsult objects are available or not

Syntax

```
>> tf=PAisAwaited(r)
```

Inputs

r - an array of PAREsult objects received by a call to PASolve

Outputs

tf - an array of boolean values, telling for each indice if the result is awaited (true) or available

Example

```
>> r=PASolve(@factorial, 1, 2, 3, 4);  
>> tf = PAisAwaited(r) % Non-blocking call
```

See Also

[PASolve](#), [PAREsult/PAwaitAny](#), [PAREsult/PAwaitFor](#)

## PAarrayfun

PAarrayfun apply a remote function call to each element of an array.

### Syntax

```
>> resultarray = PAarrayfun(func, blocks, array1, array2, ...);
>> [resultarray, indices] = PAarrayfun(func, blocks, array1, array2, ...);
```

### Inputs

func - a matlab function handle

blocks - a vector describing the size of blocks to create. Its length must correspond to the number of dimensions of arrays parameters. Example: [20 20] will create blocks of size 20x20.

arrayk - an array whose elements will be given to parameter k of function func, each arrayk must be the same size or a scalar. In case blocks are used, func will receive blocks of size specified by the blocks parameter, instead of a scalar value.

### Outputs

resultarray = an array of PAREsult objects, the size of the result array will be the same as arrayk parameters. In case blocks are used, the result array's size will be the parameters size divided by the blocks size.

indices = in case blocks are used, indices is a cell array which will contain for each block, the indices of the original block taken from the parameter arrayk. The indices use linear indexing. So is the indexing used to indentify the block in the indices cell array. Example {[1 2 4 5], [3 4 7 8]} means that block 1 correspond to the block [1 2 4 5] of arrayk and block 2 corresponds to the block [3 4 7 8] of arrayk.

### Description

PAarrayfun apply a remote function call to each element of an array. It realises inside a call to PASolve, so refer to the description of PASolve for behavior and remarks.

The parameter array can be divided into blocks of equal size. This should be done systematically when trying to process big matrices. Avoiding doing that can result in huge overheads and Out of Memory issues.

PATasks configuration such as Input/Output Files cannot be used with PAarrayfun.

PAwaitFor, PAisAwaited or PAwaitAny can be used on the result array similarly as with PASolve.

In the Example folder is a complete example of using PAarrayfun applied to a Mandelbrot fractal calculation.

### See also

[PASolve](#), [PAREsult](#), [PAREsult/PAwaitFor](#), [PAREsult/PAwaitAny](#), [PAREsult/PAisAwaited](#)

## PAoptions

PAoptions sets or returns the current options for the PAsolve execution

### Syntax

```
>> options = PAoptions();
>> PAoptions(param,value, ...);
```

### Inputs

param - a string containing the parameter name  
value - the new value of this parameter

### Outputs

options - a structure with fields corresponding to parameters names, containing all options values.

### Description

PAoptions sets options used by the next PAsolve call. A structure containing the current options can be retrieved by calling PAoptions with no parameter.

### Example

```
>> PAoptions('Debug', true);
>> r = PAsolve(@factorial, 1, 2, 3, 4) % Runs PAsolve in "Debug"
mode.
```

### Parameters

#### JobName

Name of the job that will be submitted to the Scheduler

#### JobDescription

Description of the job that will be submitted to the scheduler

#### Debug

true | false | 'on' | 'off'

Debug mode, default to 'off'

#### TransferEnv

true | false | 'on' | 'off'

Transfers the environment in which the PAsolve/PAeval function is called to every remote tasks. Variables transferred this way need to be accessed inside the subm via the evalin('caller', ...) syntax. Global variables are also transferred and can be accessed default to 'off'

#### EnvExcludeList

Comma separated list of variables which should be excluded from the workspace when transferring the environment (TransferEnv)

#### EnvExcludeTypeList

Comma separated list of object types which should be excluded from the workspace when transferring the environment (TransferEnv)

#### NbTaskExecution

integer >= 1

## Matlab Connector Toolbox

Defines how many times a task can be executed (in case of error),  
it defaults to 2, to limit accidental crash of the remote engine due to memory limitation.

|  |  |
|--|--|
| Fork   | true   false   'on'   'off'<br>Runs the tasks in a separate JVM process  |
| RunAsMe  | true   false   'on'   'off'<br>Runs the tasks under the account of the current user, default to 'off'  |
| RemoveJobAfterRetrieve   | true   false   'on'   'off'<br>Removes the job automatically after all results have been retrieved. If the options is "off", the job is removed at the end of the matlab session, or manually via PAjobRemove. default to 'on'   |
| LicenseSaverURL  | char<br>URL of the FlexNet LicenseSaver proxy. The LicenseSaver must be downloaded, installed and from the scheduler or from ProActive Matlab toolbox. Please send an email contact@activee The License Saver interacts with an existing FlexNet server to grab information about ava Matlab toolboxes tokens. A Matlab task running inside ProActive Scheduler and connected to will use tokens only when available, and will remain pending if no token is available. If this configuration option is empty, no license check will be done.  |
| CustomDataspaceURL   | char<br>URL of the dataspace (both input and output) to expose, if you don't want to rely on ProActive's automatic transfer protocol. The dataspace server must of course be started and configure manually in that case. e.g <a href="ftp://myserver/rootpath">ftp://myserver/rootpath</a>  |
| CustomDataspacePath  | char<br>Path to the root of the Custom Dataspace provided by CustomDataspaceURL. For example if <a href="ftp://myserver/rootpath">ftp://myserver/rootpath</a> is the URL of the Dataspace and the rootpath directory corresponds on the file system to /user/myserver/.../root then this path must be specified in CustomDataspacePath.  |
| SharedPushPublicUrl, SharedPullPublicUrl, SharedPushPrivateUrl, SharedPullPrivateUrl and SharedAutomaticTransfer | those url are those used by the Shared DataSpace on the scheduler (if it's activated on the scheduler). The Push urls define the spaces where task input files are pushed to. The Pull urls are the spaces from which task output files are pulled. The public urls are accessible from anywhere, the private urls are accessible only by worker nodes. Contact the scheduler administrator to know these values. They can be equal, but if the infrastructure allows it, it is more efficient to use a file url. If the computing nodes will then access the space directly via a shared file system like NFS. The option SharedAutomaticTransfer is an internal option and should not be modified. |
| TransferMatFileOptions   | char<br>If TransferEnv is set to on, tells which options are used to save the local environment. See the "save" command for more information. Default to '-v7'   |
| VersionPref  | char<br>Determines the matlab version preferred to use by the worker, e.g. 7.5   |
| VersionRej   | char<br>A string containing a list of matlab versions that must not be used, delimiters can be space or comma  |
| VersionMin   | char<br>A minimum matlab version that can be used  |
| VersionMax   | char<br>A maximum matlab version that can be used  |



## Matlab Connector Toolbox

A maximum matlab version that can be used

VersionArch 'any' | '32' | '64'

The matlab version architecture to be used. "Any" means any architecture can be used.

ForceMatlabSearch boolean

Do we force the automated search of Matlab ?

In the default behavior, this is set to false. The selection script will try to search Matlab only if a MatlabWorkerConfiguration.xml cannot be found on the host. If the script finds it it will create a new MatlabWorkerConfiguration.xml to speed up latter executions. If on the contrary ForceMatlabSearch is set to true, the selection script will always search for Matlab installations.

Priority 'Idle' | 'Lowest' | 'Low' | 'Normal' | 'High' | 'Highest'

Priority used by default for jobs submitted with PASolve, default to 'Normal'

UseJobClassPath

With this options set to true, the toolbox will use the jobClassPath feature of the scheduler. jar files necessary to the matlab workers will be copied at each task execution. It will copy jars in the addons directory, but it will introduce an overhead

WindowsStartupOptions char

Options given to matlab worker processes started on windows operating systems

LinuxStartupOptions char

Options given to matlab worker processes started on linux operating systems

CustomScript

url or path of a user-defined selection script used in addition to (before) FindMatlabScript and MatlabReservationScript

CustomScriptStatic

a boolean, true if the CustomScript is a static one (executed only once on a given machine), otherwise the CustomScript will be dynamic (default)

CustomScriptParams

a string containing the parameters of the custom script delimited by spaces.

FindMatlabScript and FindMatSciScriptStatic

url or path of selection script used to find matlab (internal)

MatlabReservationScript

url or path of selection script used to reserve matlab tokens (internal)

ProActiveJars, EmbeddedJars and WorkerJars

Comma separated list of jar files used by ProActive (internal)

PathJars

path to the proactive and embedded jars (internal)

ProActiveConfiguration

path to ProActive configuration file (internal)

Log4JConfiguration

path to log4j configuration file (internal)

## *Matlab Connector Toolbox*

SecurityFile  
path to java security configuration file (internal)

RmiPort  
default RMI port used when deploying the middleman JVM (internal)

JvmTimeout  
default timeout used when deploying the middleman JVM (internal)

JvmArguments  
Optional JVM arguments for the middleman JVM (internal)

UseMatlabControl  
do we use the MatlabControl framework ? (internal)

EnableDisconnectedPopup  
a popup will appear when the matlab session finishes and some jobs are uncomplete (internal)

WorkerTimeoutStart  
Timeout used to start the matlab engine (\*10ms) (internal)

## PAgetResults

PAgetResults results from a job

### Syntax

```
>> res = PAgetResults(jobid);
```

### Inputs

jobid - the id of the job (string or numeric)

### Outputs

res - an array of PResult objects

### Description

PAgetResults is used to retrieve results of a PAsolve call from the previous Matlab session (Disconnected mode).

The array of objects returned is the same returned by the initial call to PAsolve during the previous session. Calls to PAwaitFor or PAwaitAny can be done on this array if some results are not available yet.

### Example

```
>> res = PAgetResults('1')
```

### See also

[PAsolve](#), [PResult/PAwaitFor](#), [PResult/PAwaitAny](#)

**PAbeginSession**

PAbeginSession starts a fault-tolerant PAsolve session.

**Syntax**

```
>> PAbeginSession();
```

**Description**

By calling PAbeginSession, we start a new fault-tolerant session. Multiple session cannot be started (i.e. only one session can be active at a time).

A fault-tolerant session means that every PAsolve call will be recorded and results received via PAsolve will be remembered. If Matlab crashes in the middle of a session (or simply exits by the user's wish), after Matlab restart and resubmit the same computations. PAsolve calls which have already been successfully submitted again to the scheduler and will be linked to the previously submitted job. PAAwaitFor already completed in the previous session will return immediately with the correct results. Function code, Parameters, Input/Output Files used during the previous session should NOT be changed (otherwise will be ignored).

Multiple matlab crash/exits can occur in a row, simply reuse PAbeginSession() at each restart.

In order to finish a fault-tolerant session, PAendSession() must be called.

**Example**

```
>> PAbeginSession()
>> res1 = PAsolve(@longComputation1, param_1_1, param_1_2, param_1_3)
>> val1 = PAAwaitFor(res1);
(...)
>> res1 = PAsolve(@longComputation2, param_2_1, param_2_2, param_2_3)
>> xxxx CRASH xxxx
(... next matlab session ...)
>> PAbeginSession()
>> res1 = PAsolve(@longComputation1, param_1_1, param_1_2, param_1_3)
>> val1 = PAAwaitFor(res1); % Instantaneous !
>> res2 = PAsolve(@longComputation2, param_2_1, param_2_2, param_2_3)
>> val2 = PAAwaitFor(res2);
(... etc ...)
>> PAendSession();
```

**See also**

[PAendSession](#), [PAgetResults](#), [PAsolve](#), [PAResult/PAAwaitFor](#), [PAResult/PAAwaitAny](#)

MATLAB File Help: PAendSession

[View code for PAendSession](#)

[Default Topics](#)

## PAendSession

PAendSession ends a fault-tolerant PAsolve session.

### Syntax

```
>> PAendSession();
```

### Description

Ends a persistent PAsolve session, previously started using PAbeginSession.

See also

[PAbeginSession](#), [PAgetResults](#)