PyAALib-JyAALib - http://jyaalib.sourceforge.net
[20080827 – How To: Tag-DEBUG?]

by Dr. Romeu Andre' PIERITZ

How to: TAG-DEBUG?

NEW: A new parallel debug scheme called "Tag Debug" is proposed and implemented by the AALib-PyAALib-JyAALib Framework.
The "Tag debug" scheme allows the developer debug his own plug-in when running in parallel all other plug-ins in the application. The new scheme output only the information required by the user. The internal AALib is output only under user control.

Why?

No pollution in the output screen. Also, the log file contains only the debug information required by the user. It allows the user debug his code running in parallel with all other plug-ins and modules from other users. The user controls his own code only.

How it works?

The user doesn't need to change his code!!. The new scheme is applied by default.
The standard "ALVerbose.DEBUG" and " ALVerbose.DEBUGID" methods are the same.
The code used until now was:

```
>>>>>>>>
.... my code....

ALVerbose.DEBUGID( self.getID(), "my debug output" ) or
ALVerbose.DEBUG( "my debug output" )

...the rest of my code....
>>>>>>>>
```

where is used the default option "all". That means: only the user output is used in the console or log = The AALib internal is not output.

Reserved tags: "aalib" and "all"

To use "Tag Debug", the user must add for each plug-in or module its specific tag. For example: In my plug-in "MyPlugin", I want tag my all outputs using the tag "myTagPrivate". All debug information must add the same tag at the end, example:

```
>>>>>>>>
.... my code....

ALVerbose.DEBUGID( self.getID(), "my debug output", "myTagPrivate" ) or
ALVerbose.DEBUG( "my debug output", "myTagPrivate" )

...the rest of my code....
>>>>>>>>
```

To output only the DEBUG information tagged by "myTagPrivate", the user must execute the application using the command line option:

>> myaaplication --DEBUG-TAG myTagPrivate ……

Note: "myaaplication" represents a python or Java-Jython script to start the main user code based on PyAALib-JyAALib.

The next examples show the main usage and options:

>> myaaplication --DEBUG
It shows all user debug tags (by default: option "all") – NO AALib internal tags output – clean console.

>> myaaplication --DEBUG-FORCE
It shows all user debug tags (by default: option "all") and stop the application when a main error occurs NO AALib internal tags output – clean console.

>> myaaplication --DEBUG-TAG all
It shows all user debug tags.

>> myaaplication --DEBUG-TAG aalib
It shows all AALib internal debug tags.

>> myaaplication --DEBUG-TAG myTagPrivateBanane
It shows only the Debug with that specific tag.

>> myaaplication --DEBUG-TAG aalib –DEBUG-FORCE
It shows all AALib internal and stops when a main error occurs.

>> myaaplication --DEBUG-TAG myTagPrivate –DEBUG-FORCE
It shows only the debug tagged by "myTagPrivate" and stop when a main error occurs.

>> myaaplication --DEBUG-TAG all –DEBUG-FORCE
It shows all user debugs (not AALib internal) and stop when a main error occurs.

Standard Command Line Options?

The standard built-in options available for all code based on the PyAALib-JyAALib "ALApplication" class are:

--help or -h [option] :	to show help if available
--man or -m [option] :	to show manual if available
--help --all :	it shows the main command list help
--man --all :	it shows the main command list man
--verbose :	to iterative output
--quiet :	NO output (default option)
--no-log :	NO log file output
--profile :	profile output for trace actions
--version or -v :	executable version info
--plugin :	show the plug-in connected to the application
--plugin-group :	show the plug-in groups
--plugin-group-tree :	show the plug-in group tree
--DEBUG :	output debug info with no stops
--DEBUG-TAG [tag] :	output only user Tag debug - all, aalib or others
--DEBUG-FORCE :	output default Python-Java debug info and FORCE the application to stop
--DEBUG-FORCE-PLUGIN :	output default Python-Java debug only from a PLUGIN and FORCE the application to stop
--ASSERT :	output assert info for debug
--import-resource [filename] :	import a file application resource file

Compatible?

It was tested on: Windows Vista32, Ubuntu Linux and ESRF Linux. The MacOSX version is on test.

- On Windows Vista32 and Ubuntu Linux, it was tested using: Python-2.5.2 and Java/Jython-2.2.1.

- On ESRF Linux, the AALib was tested using: Python-2.4.4 and Jython-2.2.

Note: We recommend use the Firefox Web Browser to display the Profile GUI . Some users are experimenting problems with JavaScript on Windows Internet Explorer an KDE Web viewer.