TaskBrowser HELP

This document contains the same text as the `help` command in the TaskBrowser console.
See also the offical TaskBrowser Component Manual and the Component Builder's Manual.

[Task Browsing]
To switch to another task, type '`cd <path-to-taskname>`'
and type '`cd ..`' to go back to the previous task (History size is 20).
Pressing <tab> multiple times helps you to complete your command.
It is not mandatory to switch to a task to interact with it, you can type the
peer-path to the task (dot-separated) and then type command or expression :
`PeerTask.OtherTask.FinalTask.countTo(3)` [enter]
Where 'countTo' is a method of 'FinalTask'.
The TaskBrowser starts by default 'In' the current component. In order to watch
the TaskBrowser itself, type '`leave`' You will notice that it
has connected to the data ports of the visited component. Use '`enter`' to enter
the visited component again. The '`cd`' command works transparantly in both
modi.

[Task Context Info]
To see the contents of a task, type '`ls`'
For a detailed argument list (and helpful info) of the object's methods,
type the name of one of the listed task objects :
`this` [enter]

Command : bool factor(int number)
Factor a value into its primes.
number : The number to factor in primes.
Method : bool isRunning( )
Is this TaskContext started ?
Method : bool loadProgram(const& std::string Filename)
Load an Orocos Program Script from a file.
Filename : An ops file.
...

[Expressions]
You can evaluate any script expression by merely typing it :
`1+1` [enter]
= 2
or inspect the status of a program :
`myProgram.isRunning` [enter]
= false

and display the contents of complex data types (vector, array,...) :
`array(6)`
= {0, 0, 0, 0, 0, 0}

[Changing Attributes and Properties]
To change the value of a Task's attribute, type '`varname = <newvalue>`'
If you provided a correct assignment, the browser will inform you of the success
with '= true'.

[Commands]
A Command is 'sent' to a task, which will process it in its own context (thread).
A command consists of an object, followed by a dot ('.'), the command
name, followed by the parameters. An example could be :
`otherTask.bar.orderBeers("Palm", 5)` [enter]
The prompt will inform you about the status of the last command you entered.
It is allowed to enter a new command while the previous is still busy.

[Methods]
Methods 'look' the same as commands, but they are evaluated
immediately and print the result. An example could be :
`someTask.bar.getNumberOfBeers("Palm")` [enter]
= 99

[Events]
Events behave as methods, they are emitted immediately.
An example emitting an event :
`someTask.notifyUserState("Drunk")` [enter]
= (void)

[Program and StateMachine Scripts]
To load a program script from local disc, type '`.loadProgram <filename>`'
To load a state machine script from local disc, type '`.loadStateMachine <filename>`'
(notice the starting dot '.')
Likewise, '`.loadProgram <ProgramName>`' and '`.unloadStateMachine <StateMachineName>`'
are available (notice it is the program's name, not the filename).
You can use '`ls progname`'
to see the programs commands, methods and variables. You can manipulate each one of these,.
as if the program is a Task itself (see all items above).
To print a program or state machine listing, use '`list progname [linenumber]`'
to list the contents of the current program lines being executed,
or 10 lines before or after <linenumber>. When only '`list [n]`'
is typed, 20 lines of the last listed program are printed from line <n> on

(default : list next 20 lines after previous list).
To trace a program or state machine listing, use '`trace [progname]`' this will
cause the TaskBrowser to list the contents of a traced program,
each time the line number of the traced program changes.
Disable tracing with '`untrace [progname]`'
If no arguments are given to '`trace`' and '`untrace`', it applies to all programs.
A status character shows which line is being executed.
For programs : 'E':Error, 'S':Stopped, 'R':Running, 'P':Paused
For state machines : <the same as programs> + 'A':Active, 'I':Inactive

[Changing Colors]
You can inform the TaskBrowser of your background color by typing '`.dark`'
'`.light`', or '`.nocolors`' to increase readability.

[Macro Recording / Command line history]
You can browse the commandline history by using the up-arrow key or press 'Ctrl r'
and a search term. Hit enter to execute the current searched command.
Macros can be recorded using the '`.record 'macro-name'`' command.
You can cancel the recording by typing '`.cancel`' .
You can save and load the macro by typing '`.end`' . The macro becomes
available as a command with name 'macro-name' in the current TaskContext.
While you enter the macro, it is not executed, as you must use scripting syntax which
may use loop or conditional statements, variables etc.

[Connecting Ports]
You can instruct the TaskBrowser to connect to the ports of the current Peer by
typing '`.connect [port-name]`', which will temporarily create connections
to all ports if [port-name] is omitted or to the specified port otherwise.
The TaskBrowser disconnects these ports when it visits another component, but the
created connection objects remain in place (this is more or less a bug)!