



# Eric 4

as Python 2.x

Integrated Development Environment

---

## Technical Report

## Copyright Page

SP-M 110800

Current edition on PDF file, under "Creative Commons License"

Storage URL [Eric\_Deliver] <http://www.box.net/shared/k64yenrpey>



"Eric 4 – as Python 2.x Integrated Development Environment – Technical Report" by Pietro Moras (E-mail: Studio-PM@hotmail.com) is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License.

Based on a work at [Eric IDE] <http://eric-ide.python-projects.org/index.html>.

*No commercial uses, No modifications allowed; Jurisdiction International, Format Text*

*This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/3.0/> or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.*

- -

**Disclaimer** The information in this document is subject to change without notice. The author and publisher have made their best efforts about the contents of this book, nevertheless the author and the publisher make no representation or warranties of any kind with regard to the completeness or accuracy of the contents herein and accept no liability of any kind including but not limited to performance, merchantability, fitness for any particular purpose, or any loss or damages of any kind caused or alleged to be caused directly or indirectly from this book.

**Brand Names** Brand names such as Linux, Windows, Python are assumed to be universally known, and are here used in full respect of their respective owners.

- -

### *Planned edition on paper, under traditional copyright*

**All rights reserved** No part of this book may be reproduced or used in any form or by any means, such as: graphic, electronic, or mechanical, including photocopying, recording, videotaping, or information storage and retrieval systems, without written permission of the publisher.  
The media possibly accompanying this book contain software resources to be considered as an integral part of the same book, and therefore subject to the same rules.

**Published by** [not yet—just appointed] Town & Country Reprographics, Inc.  
230 N Main St Concord, NH 03301 (U.S.A.)

- = -

## Foreword

Dear Reader,

This **Report** is the result of a precise method of working, a precise vision of what a “Test & Documentation” process should be all about.

Both as a Designer, ordinary User, and Technical Writer of high-tech products (mainly s/w products) I've realized how crucial is a fair correspondence between technical documentation and actual performance. And how crucial is that the process of Test & Documentation be carried on authentically on behalf of nobody else but Mr. User — nobody else, Developers included. And this is exactly what I've tried do to here, that is:

- To operate as an ordinary Mr. User, exploring the whole process he should undergo to reach a satisfactory productivity, starting from scratch. Sort of quiet and accurate (Beta-)Testing, followed by a Technical Writing/Reporting process. Possibly in fair collaboration with the product's development team, but certainly in absolute independence from them.
- With the consequent **Report** made available to the benefit of whoever would bother to know.

Then, persuaded that 'Facts are sacred' but also that 'Comment is useful', I provided room to the latter under the dedicated header:

### Viewpoints

- Where I felt free to express questionable opinions and comments, hopefully to be taken as an opportunity for further discussions and new ideas.

Then, because of its very nature, this **Report** also requires an agile updating mechanism, here obtained through an Author-and-Date code marker, such as:

“S-PM yymmdd”

So that each **Report** segment, comprised between such Author-and-Date code and the subsequent “- -” End-of-Segment mark, is assumed to supersede and nullify all possibly preceding one.

- -

Plus, when for any reason—typically: lack of information, or evidence—I feel incapable of sound and reliable description of facts, I'll simply warn the reader with a mark such as: <?>[Hic sunt leones]<sup>2</sup>

As you see, that's rather an ambitious, challenging goal, certainly requiring not short a time, and a great deal of collaboration with all involved parties, so to be helped in minimizing my possible (or inevitable...) shortcomings.

And here you have the present result, hopefully to the common benefit, and about which I'd love to hear your criticisms, questions and remarks.

See you.

The Author

- = -

1 Actually, C.P. Scott's celebrated sentence was a bit different: 'Comment is free, but facts are sacred'.

2 Reminiscent of when, in ancient maps from the Roman era, African regions about which there was no information were identified by the Latin inscription HIC SUNT LEONES (here there are lions). The reason for the inscription was to warn the potential traveller of dangers that he may encounter in the unmapped areas.

## Instant Reference

SP-M 110600

Essential User Guide about most basic Eric 4 operations (simply “Eric”, now on).

**Prerequisites** As for chapter 5. Setup and General Management (see).

**Download** Follow the “eric Downloads” link on the Eric Python IDE web site:  
<http://eric-ide.python-projects.org/index.html>  
Select and download the package aimed at the target environment (here:  
Windows 32 bit, Python ver 2.x), that's currently: `eric4-4.4.15.zip`  
Extract and inspect the contents, particularly the “README” file.

**Install** Execute “`install.py`” procedure, located in the original Eric package. Inspect the result, to be found on the `\Python2x` hosting directory.

**Run** Execute the proper batch command file `Eric4.bat`, located in the main `\Python2x` hosting directory.  
For a complete technical reference see chapter 5. Setup and General Management

- -

**Uninstall** Execute “`uninstall.py`” procedure, located in the original Eric package.

- = -

# 1. Introduction & Generality

Eric is, in essence, an integrator of different, concurring tools, offering a centralized and unique cockpit for software development activity. More precisely, it is a multi-platform, multi-Integrated Development Environment (IDE), originally designed for the programming languages Python 2.x and Ruby 1.8.x; and, potentially, also for an extended set of other languages, as can be listed via right-click context menu command “Languages”, on the source edit text form (see).

In the execution of this Report, of all its capacities, Eric has been actually tested, used and, therefore, documented exclusively as a Python 2.x IDE, on a Windows platform; as specified in the hereafter *Version Reference* table (see). As a consequence, all aspects possibly related with different situations will be here ignored or, at most, merely cited and marked as: <~>[Off topic] (see also: Appendix > Typographical Conventions).

## *Version Reference*

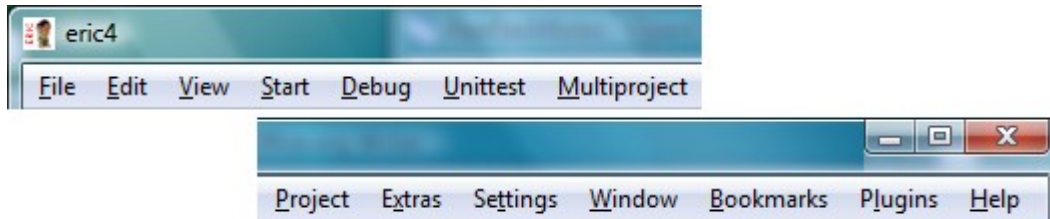
<i>Nr</i>	<i>Item</i>	<i>Version</i>	<i>Note</i>
1	Windows XP Pro Windows Vista Ultimate	5.1 (SP 3) 6.0 (6002:SP 2)	Operating System
2	\Python27	2.7.1	Hosting Directory
3	Eric 4	4.4.15 (r4034)	IDE Application

- = -

## 2. Menu Commands

S-PM 110600

Reference section of all Eric 4 menu commands available for user interaction<sup>3</sup>.



An ordered and detailed description of each and all Eric menu commands, with some *Remarks* where special considerations reveal useful to the user.

Not all interaction topics are here described the same way. Some, particularly relevant, will be here only cited, and then fully described in a dedicated section (see). Whereas some others will be, to some extent, overlooked, and for good reasons, as considered not requiring, or not deserving, too detailed description being classified, and marked as `<~>`[*AsFor*: Off scope] (see: *Typographical Conventions*).

### Remarks

- This Menu Bar is not the only way to interact with Eric. For instance, right-click context menus are here spread available all over (see), handy to offer the same Menu's functionalities in a different and possibly more convenient way.
- Functionalities possibly not reachable via this Menu Bar and, therefore, here not reported, will find place into the specific section dedicated to Eric Sub-Systems (see).

### Viewpoints

- Right-click context menus will be not detailed here but deliberately left to the personal experience of use, as we deem that it would be more confusing than useful.

--

---

<sup>3</sup> All menu commands will be here listed orderly by their full names, without making reference to their possible shortcut keys nor command bar buttons or context menus, all left to the direct interactive experience.

## Menu Commands

Detailed description of each and all menu commands as found available on the Eric menu bar.

File	Edit	View	Start	Debug	Unittest	Multiproject
Project	Extras	Settings	Window	Bookmarks	Plugins	Help

### Remarks

- Each menu command becomes automatically disabled, and “dimmed”, when incompatible with the current operating condition and, therefore, unusable.

- -

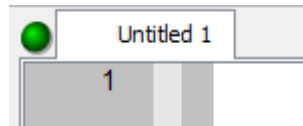
### Menu Command: **File**

New  
 Open...  
 Open Recent Files  
 Open Bookmarked Files  
 Close                      Close All  
 Search File...  
 Save  
 Save As...  
 Save all...  
 Save to Project  
 Export as  
 Print Preview  
 Print  
 Quit

- -

### File > **New**

To create a brand new Eric source form, with default name “Untitled *n*”, and where source text code can be interactively entered.



### Remarks

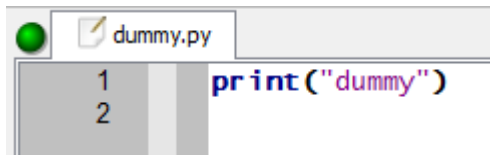
- This text form is meant to be saved as a source file, with the typical Python module extension “\*.py”

That's said to better understand the “spirit” of this command, and also in view of the subsequent `Save` command, and the related default extension (see).

- -

### File > **Open...**

To locate and open an existing file on a new Eric source edit form, named after the selected file.



Default directory and file extension are those of the last file selected during the current session, initially derived from the very `Eric4.cmd` start command file location.

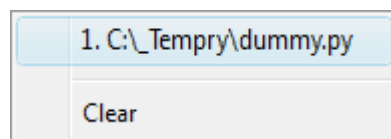
### Remarks

- This command is typically aimed at opening Python module source files, with the proper extension “\*.py” that can be defaulted via menu command `Settings > Preferences... > Editor > Filehandling > Default File Filters (see)`.

- -

### File > **Open Recent Files**

To display the history-list of the last files opened, handy to possibly re-open again one of them.



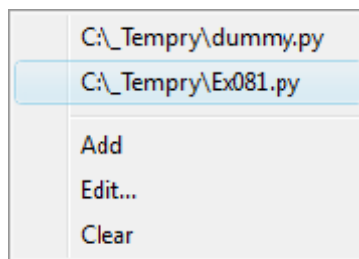
**Remarks**

- Length of this history list can be set with `Settings > Preferences... > Interface > Viewmanager > Number of recent files: (see)`.

- -

**File > Open Bookmarked Files**

To display list and controls of the “bookmarked” files, handy to possibly re-open again one of them at any given Eric session.



This sub-menu box is also aimed at managing the very set of the bookmarked files.

- -

**File > Close****File > Close All**

To close the source text form currently active, or all of them at once, with the current Eric session remaining active. In case of unsaved changes, a pop-up “File Modified” dialog box will ask the user what do to with them.

**Remarks**

- In case of modules belonging to an opened project (see `Project > Open`), the project in itself remains open.

- -

**File > Search File...**

To open a `Find File` dialog box aimed at finding out and then, possibly, at opening a file specified by means of usual wildcard characters and the directories where to search. The list of the matching file names is actually displayed when at least one of these three directory search modes has been checked:

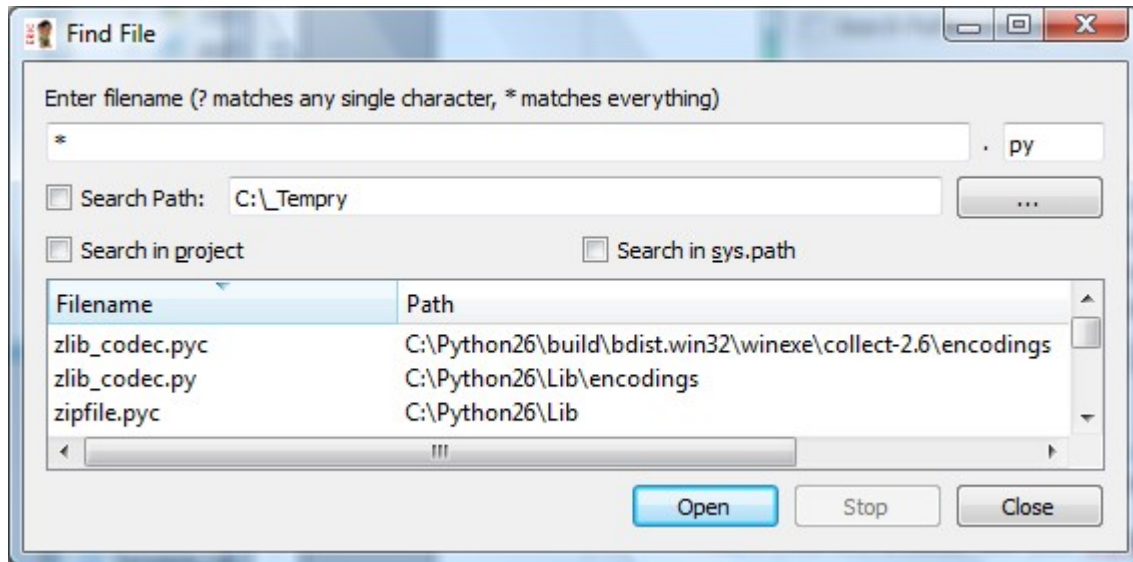
Search Path:

Search in project

Search in sys.path

## Viewpoints

- Here we have encountered what seems to be a tiny bug:



That is the permanence of the display of a not-empty name list even after all check boxes has been unchecked.

--

## File > **Save**

To save the source window's file currently active, with the opened file that remains open and active. This command results enabled only after some actual text change.

## Remarks

- Here the typical file extension is that of a Python source “\*.py”, which can be conveniently defaulted via menu command `Settings > Preferences... > Editor > Filehandling > Default File Filters` (see).

--

**File > Save As...**

To save a different copy of the source 's file currently active. Current file keeps being open and active.

This command results active also with unmodified files, therefore permitting their duplication.

*Viewpoints*

- An annoying trouble have we here encountered, the absence of the current file's extension offered as obvious default for the new one, which leads to some cumbersome fiddling when duplicating a file.

--

**File > Save All...**

To save all currently opened files, whose active status remains unaltered. This command results always available, irrespective of actual changes on opened files.

*Viewpoints*

- The ellipsis “...” after a menu command, as a rule, are used to say that that command in itself will not perform any actual action, but will simply activate a new situation—such as the display of a window or a sub-menu—where the user would make a subsequent choice about what to actually do. Whereas, in this case, it's not so, and an actual action will take place right away.  
Therefore we deem such ellipses as wrong and misleading.

--

**File > Save to Project**

To save a copy of the source form file currently active and assign it, with the desired name, to the currently opened Eric project. Original file is left unchanged.

*Remarks*

- <![AsFor: Importance] The *Eric Project* is rather a powerful feature, hereafter conveniently described (see menu command `Project`).
- With this command both files not belonging to a project can be copied into it and files already belonging to a project can be duplicated under a different name (by the way, that's why no default name is here offered).

--

## File > **Export as**

To export, that is to save, the source text file currently active into a new file of different format, at choice among: HTML, PDF, RTF, TeX.

### Remarks

- The currently available set of formats has a historical origin. TeX file type is associated with the LaTeX typesetting system.
- We have been told that a convenient default mechanism for directory and file name is here under development.

- -

## File > **Print Preview**

### File > **Print**

To show the Eric specific “Print Preview:” or “Print” dialog box for the edit text file currently active, so to examine precisely how Eric would possibly print it, or to actually print it.

### Remarks

- `<?>`[AsFor: Malfunction] A menu command `Settings > Preferences... > Printer > Configure printer settings (see)` is conveniently provided, but, alas, seems ineffective in changing the default page margins, rather too narrow.
- Another related `Setting > Preferences..., Editor > Style`, is provided to `Configure editor styles > Fonts`, where:
  - `Default Text Font` `<?>`[AsFor: Missing info] The info we miss here is not of generic nature, but that specifically related to this very command or parameter.
  - `Monospaced Font [+]` Use monospaced as default
    - Will set the font actually used for the source text, such as the “DejaVu Sans Mono”, for a neat, readable printing.

- -

## File > **Quit**

To close current Eric session. In case of unsaved changes, a pop-up “File Modified” dialog box will ask the user what do to with them.

- -

Menu Command: **Edit**

- Undo
- Cut
- Clear
- Indent
- Smart indent
- Comment
- Stream Comment
- Autocomplete
- Search
- Goto Line...
- Goto Brace
- Select to brace
- Select all
- Shorten empty lines
- Convert Line End Characters
- Redo
- Copy
- Unindent
- Uncomment
- Box Comment
- Deselect all
- Revert to last saved state
- Paste

--

Edit > **Undo**

Edit > **Redo**

Edit > **Revert to Last Saved State**

Usual commands to let you change your mind about last editing actions carried on the source text form, in LIFO sequence, up to the last state saved on disc.

--

Edit > **Cut**

Edit > **Copy**

Edit > **Paste**

Usual commands of basic edit actions on the currently active source text form.

--

Edit > **Clear**

To reset to blank the currently active source text form.

--

Edit > **Indent**

Edit > **Unindent**

To indent (shift right) or un-indent (shift left) a single line, or a selected set of lines, by a fixed amount of blanks. Commands useful to align blocks of source text lines in accordance with the Python syntactic rules.

### Remarks

- Same action can be carried on simply making use of the keyboard `Tab` key.
- Default indent value is four blanks, value changeable via menu `Settings > Preferences...`  
> `Editor > General > Tabs & Indentation`

--

Edit > **Smart Indent**

Same as for `Edit > Indent` (see), but with some automatic interpretation of the Python syntactic rules on the selected block of lines.

### Viewpoints

- `<~>`[AsFor: Perplexity] We are not sure of having really grasped the logic behind this command.

--

**Edit > Comment****Edit > Uncomment**

To insert/cancel a leading “#” character on a selected block of lines, as for Python comment lines, so to quickly exclude or include their coding effects without having to erase or retype them.

- -

**Edit > Stream Comment****Edit > Box Comment**

<~>[AsFor: Off scope] Topic just hinted at, being off the main scope of this Report.

Feature related with programming languages other than Python (such as: Ruby or C++), therefore out of the main scope of this work.

**Remarks**

- Eric is potentially a multi-platform, multi-language IDE which, besides Python, could handle other languages such as Ruby 1.8.x or many other languages as listed with the right-click context menu command “Languages”, on the source text form (see).

Anyhow, as already said, this aspect is currently out of the main scope of this Test & Documentation work, uniquely centered on the Python language.

- -

**Edit > Autocomplete**

To show a box of sub-menu commands, where to call one of the auto-completion tools available to support the current text editing entry. A <CR> or <Tab> is then required to possibly confirm the text auto-completion action.

Autocomplete	Ctrl+Space
Autocomplete from Document	Ctrl+Shift+Space
Autocomplete from APIs	Ctrl+Alt+Space
Autocomplete from Document and APIs	Alt+Shift+Space
Calltip	Alt+Space

To auto-complete texts according to:

Autocomplete Standard Python syntax [Ctrl+Space]

Autocomplete from Document  
Text patterns searched on the same document

Autocomplete from APIs  
<?>[AsFor: *Hic sunt leones*] Unexplored territory where, for lack of info, we don't even dare to venture. The info here missed is not of generic nature, but that specifically related to this very command or parameter.

Subject presumably related to internal database, populated from API files delivered with PyQt4, QScintilla and other original Eric add-ons.

### Viewpoints

- But we have no idea what all this really means.

Autocomplete from Document and APIs  
A combination of the last two cases

And where:

Calltip <?>[AsFor: Not tested] Tried, but never seen working.

### Remarks

- Use of keyboard shortcuts (e.g.: Ctrl+Space for standard Python Autocomplete) is here particularly convenient.
- Actual execution here depends upon Settings > Preferences... > Editor (see):  
Autocompletion → “Configure Autocompletion”  
Autocompletion > QScintilla → “Configure QScintilla Autocompletion”  
and:  
Calltips → “Configure Calltips”  
Calltips > QScintilla → “Configure QScintilla Calltips” (see)

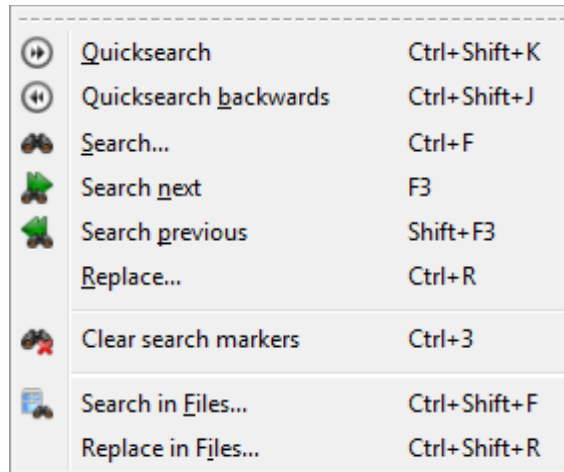
### Viewpoints

- But we must confess that we are far from having really mastered this whole tool, with its various configurations.

- -

## Edit > Search

To show the sub-menu of text search commands.



Where:

**Quicksearch** To search the currently active source text form for the first occurrence of the pattern entered into the “QuickSearch” field, case insensitive, starting from the current text insertion point, forward.



Auxiliary control buttons: Forwards, Backwards, Quicksearch extend

### Remarks

- A pattern not found in the search text is immediately perceived, as it cannot be entered in this QuickSearch field. Initially it's rather baffling, then you might love–or hate–it.

**Quicksearch backwards**

The same as Quicksearch, but backwards.

**Search**

To search the currently active source form for the first occurrence of the pattern

entered into the “Find:” field, in the different selectable search modes:

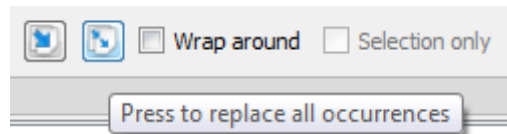
previous, next, Match case, Whole word, Regular expression, Wrap around, Selection only

Search next / previous

To repeat the same Search on and on. It's callable also with the usual F3 short-cut.

Replace

To add a “Replace:” text pattern action to the Find: command field. It requires anyhow explicit pressing of the “replace” or “replace all” buttons.

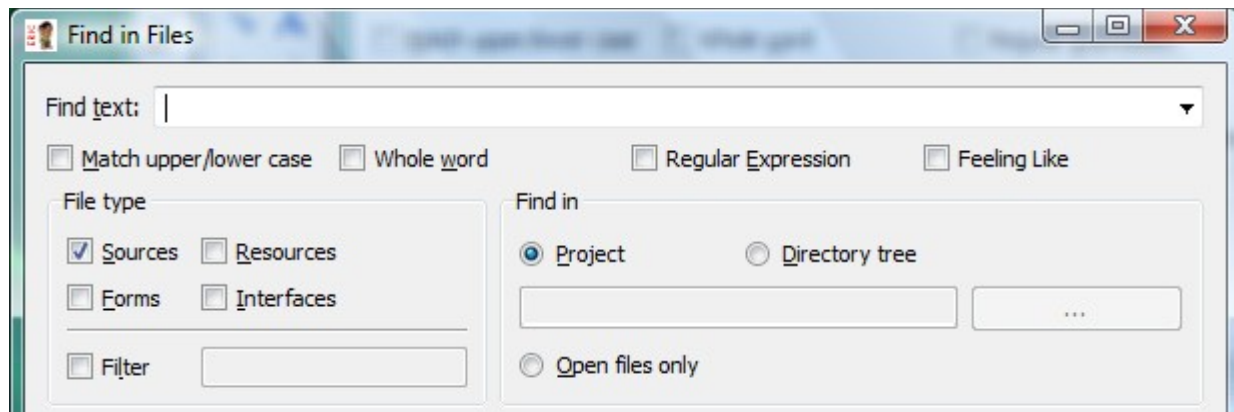


Clear search markers

To clear such “boxing” graphic markers: marker possibly left on the text by search operations.

Search in Files...

To extend the search scope beyond the currently active source text form to an entire class of files, as defined with the following Find in Files form.



Command particularly useful when working with projects (see Project menu). No further description of this form will be given, being it so clearly self-explanatory.

Replace in Files...

To add a “Replace:” text pattern action to the “Find in Files” dialog box.

### Remarks

- Be aware that source files possibly used in an Eric Project as “Imported” modules may not be considered part of the Project in the Eric sense and, as such, are ignored by `Search in Files...` command. Whereas they can be execution flow-managed with the usual `Debug` commands, breakpoints included. So that, to possibly search their text, you must explicitly have them `Open`, and searched separately from the other Project files.

### Viewpoints

- `<!>` Be aware that, as with Menu Command: `Debug` (see), any text editing action, such as to delete or add source lines, will disrupt the functionality of all subsequent search actions, and with no preventive warning. As a work-around to this flaw<sup>4</sup>, we'd suggest such quick action:  
`Save + Start > Restart, or > Debug`  
 on the very same source file / project, so to re-synchronize source text with search action<sup>5</sup>.

- -

### Edit > **Goto Line...**

To jump on a given  $n$ -th line on the currently active source form.

### Remarks

- Default “Line Number:” is always 1. Of course it should be more convenient to have it initialized to the currently pending line, shouldn't it?

- -

### Edit > **Goto Brace**

To shift the text insertion pointer to the next forward, or backward, companion bracket—round `(...)`, squared `[...]`, angular `<...>`, or brace `{...}`—matching that one currently pointed at. No action if no bracket is pointed, or if within a comment line.

### Remarks

- Matching braces can be highlighted according to command `Settings > Preferences > Editor > Style > Braces`
- In such not so infrequent a case: `_vobjmap[_stype](_ent, _data, _bitpos)`, to reach the contiguous brace at right of “] (”, you must start from right, instead of from left, as more usual.

---

<sup>4</sup> Because, of course, *it is* a flaw, and serious too.

<sup>5</sup> We dare to suggest that it shouldn't be that difficult to add such an action conveniently performed automatically, or semi-automatically; that is: upon user acceptance...

- The keyboard shortcut “`Ctrl+L`” to this command is here particularly handy<sup>6</sup>.

- -

### Edit > **Select to Brace**

Same as for “`Goto Brace`” (see), plus the selection of the included text.

- -

### Edit > **Select All**

### Edit > **Deselect All**

To select/de-select all source form currently active.

- -

### Edit > **Shorten Empty Lines**

To clean up all trailing blank characters possibly present in blank—i.e.: empty—lines on the current source form.

- -

### Edit > **Convert Line End Characters**

To convert all `End-Of-Line` characters according to the type currently set (see: menu `Settings > Preferences... > Editor > Filehandling > End of Line Characters`).

- -

---

<sup>6</sup> Just be careful not to misinterpret it as “`Ctrl+Shift+L`”, which is to delete lines.

**Menu Command: View**

Zoom in	Zoom Out	Zoom
Toggle All Folds	Toggle All Folds (Including Children)	Toggle Current Fold
Remove All Highlights		
Split View	Remove Split	
Arrange Horizontally		
Next Split	Previous Split	

- -

View > **Zoom In**View > **Zoom Out**View > **Zoom**

To zoom (in, out, or parametric) on either the source text or the interactive shell forms. Possibly useful to watch a larger amount of long text lines on a crammed display.

- -

View > **Toggle All Folds**View > **Toggle All Folds (Including Children)**View > **Toggle Current Fold**

Defining a *fold* as a source text section positioned at an equally indented level<sup>7</sup>, these commands are aimed at collapsing/expanding—that is: hiding/showing—each and all of them, so to get a synthetic, or detailed, view of the Python source code.

Collapsed folds are indicated by a horizontal line on the source edit form, and marked by “+” sign on the vertical ruler bar on the left margin; and the expanded folds by a “-” sign<sup>8</sup>. A click on these “+” / “-” symbols has the same toggling effect of these menu commands.

---

<sup>7</sup> Actually we've been told that this definition is determined by the QScintilla lexical analyzer, or “lexer”. We'd like to know better what does it mean.

<sup>8</sup> We've been also told that these “+” / “-” symbols are configuration dependent. Fine, it's a good occasion to observe that we'll never annoy the reader with such irrelevant distinctions, to be fairly left to the user's experience and insight.

By the way, here we have also no idea where/how such negligible configuration could be made and, frankly, we don't even care that much.

*Viewpoints*

- We'd like to be confirmed that this is a correct interpretations of this command's spirit.

--

**View > Remove All Highlights**

To eliminate special font-effects, such as red-highlight for exception rising lines, on the source text form.

--

**View > Split View****View > Remove Split**

To split—or un-split—the source edit form into distinct panes, aimed at displaying together all the different source modules normally displayed into overlapping tagged panes.

Split views can be arranged horizontally or vertically via “`Arrange horizontally`” command. The currently active pane is marked with a green head-pin, all the others with a red one.

--

**View > Arrange Horizontally**

To set the possible split view mode horizontal, otherwise vertical (see command: `split view`). A left check-mark appears when set.

--

**View > Next Split****View > Previous Split**

To move the active focus among the possibly split source forms (see command: `split view`).

--

## Menu Command: **Start**

Restart Script	Stop Script
Run Script...	Run Project...
Debug Script...	Debug Project...
Profile Script...	Profile Project...
Coverage run of Script...	Coverage run of Project...

A fundamental aspect of this set of fundamental menu commands is here the useful distinction between Python *Scripts*, that is “\*.py” files (see: menu `File > Open...`), and *Projects*, that is the Eric 4 specific “\*.e4p” files (see: menu `Project > Open...`).

Of course, generically speaking, any s/w work can be regarded as a project, but here, if what you're going to do is comprised in just one Python module, it's properly called a Script<sup>9</sup>. Whereas if what you're doing is articulated into a set of distinct and coordinated Scripts, it's here conveniently regarded unitary as a Project.

--

### Start > **Restart Script**

### Start > **Stop Script**

To stop and re-start, that is re-initialize and run, the same current execution, whatever it was, either a script or a project, and in run or debug execution mode (see: menu `Start > Run...`).

### *Viewpoints*

- In all next `Start` menu commands hereafter the term `Script` is used in alternative to `Program`. Not here, where this same term `Script` is to be intended as referring both to `Scripts` and `Programs`. Of course that's a bit misleading, it should be better to say “`Restart / Stop Execution`” or, simply “`Restart / Stop`”; shouldn't it?

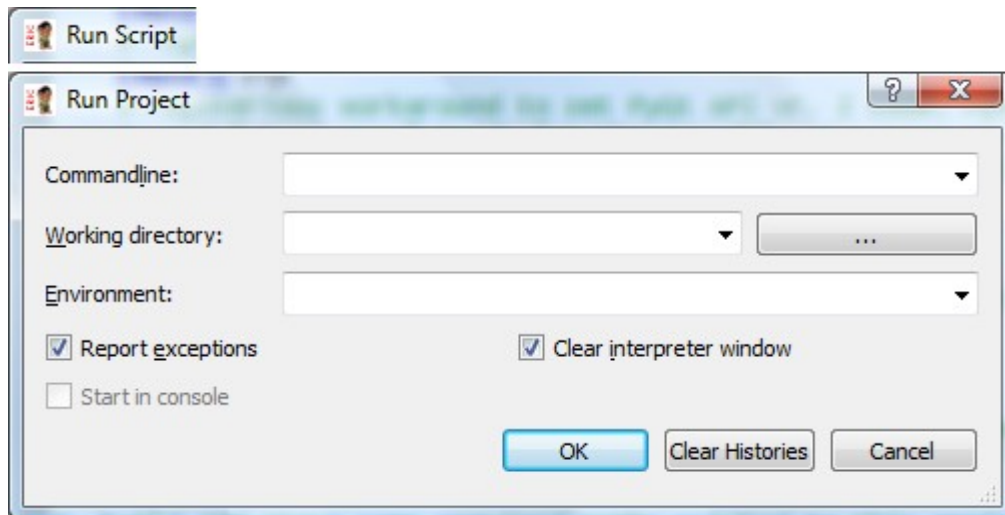
--

---

<sup>9</sup> We find another reason of interest, almost sympathy, for this term *Script*. It's that it suggests, better: implies, that Python is NOT precisely, or not only, a programming language, but rather a scripting language. Not at all to be intended as a diminishing distinction, but merely as a fact.

**Start > Run Script...****Start > Run Project...**

To start execution of the currently active script, possibly belonging to a project, or of the current project, with initial conditions that can be set on the specific “Run Script / Run Project” initializing form (see).

**Where:**

**Commandline:** List of arguments to be possibly entered on the run command-line, retrievable via the standard Python “`sys.argv`” call. First argument is automatically assigned to the executing *Main Script*<sup>10</sup> file path (see also: Working directory).

**Working directory:** To set the initial working directory, as for “`os.path.abspath(os.curdir)`”. Default value being the directory of the executing main script file (see also: Commandline).

**Environment:** To set the value of possibly new environment variables, on the standard dictionary “`os.environ`”. E.g.: `USERNAME=myName NEWVAR=aValue`

**Report exceptions**  
Check box ignored, in the sense that, differently with the “Start > Debug” case (see), here handled exceptions are never signaled; whereas un-handled exceptions are signaled anyway.

<sup>10</sup> About the Main Script concept and role, see: Menu Command: Project

Clear interpreter window

To reset what is here usually called the *Interactive Shell Form* (see section: Eric Specific Concepts and Terms).

Start in console

<?>[AsFor: Not tested] Always found disabled, unusable.

Clear Histories

Button to clear the set of previous values here entered on the initialization fields.

### Remarks

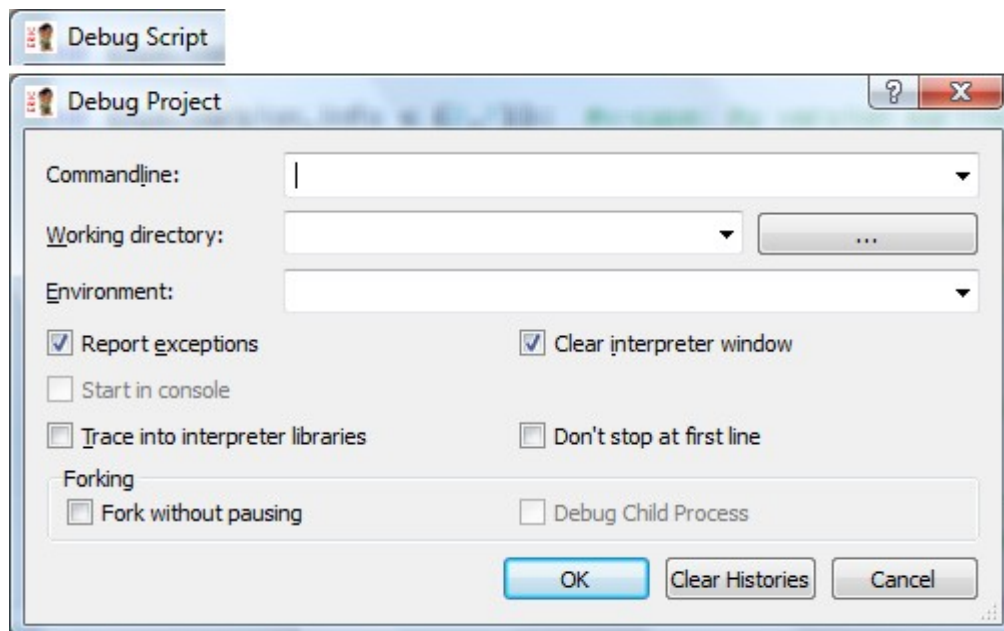
- It may be useful here to recall that a script, possibly belonging to a project, can also be run and tested as a self-standing script, with such executing condition that can be detected via the standard Python test: `if (__name__ == '__main__')`

--

Start > **Debug Script...**

Start > **Debug Project...**

To start execution of the currently active script, possibly belonging to a project, or of the current project, in “debug” mode. That is activating all tools as for menu `Debug` (see), and with initial conditions that can be set on the specific `Debug Script / Debug Project` initializing form.



Where, beyond what already seen with command `Run` (see):

`Report exceptions`

To get an Eric dialog box (see also command: `Debug > Exceptions`) related to possibly both un-handled and also handled exceptions. That is exceptions occurring within a “`try: ...`” block. When not checked, exceptions are effective, but no report box will be shown.

`Trace into interpreter libraries`

To activate tracing debug execution also within standard library modules, otherwise stepped over, as usual.

`Don't stop at first line`

To begin execution with a stop at the first executable statement<sup>11</sup>, waiting for user's intervention.

`Fork without pausing`

In case of a child process fork (see process management function “`os.fork()`”), the user is not asked which path of execution to follow (see next: `Debug Child Process`).

`Debug Child Process`

To instruct the debugger whether to follow the parent or child process fork (see former: `Fork without pausing`).

- -

**Start > Profile Script...**

**Start > Profile Project...**

To start a script or a project execution exactly as for command `Run` (see), but also with the activation of the *profiling* back-end<sup>12</sup>, so to collect and record the execution profiling data on disc.

Profiling data that can be inspected later on with:

`Project > Show > Profile Data...`

menu command (see), for a whole project;

`Show > Profile data...`

right click context menu for a particular module on a Project-Viewer (see);

---

<sup>11</sup> Consider that here a possibly initial “`pass`” statement is ignored, and not taken as an executable instruction. We do not agree.

<sup>12</sup> About which we'd like to know something more.

Show > Profile data...

right click context menu on the source edit form, for a single module (see).

- -

**Start > Coverage Run of Script...**

**Start > Coverage Run of Project...**

To start a script or a project execution exactly as for command `Run` (see), but also with the activation of the *coverage analyzer* back-end<sup>13</sup>, so to collect and record the coverage data on disc. That is the data telling which part of the source code has been actually used and, therefore, tested during the executing session.

Coverage data can be inspected later on with:

Project > Show > Code Coverage...

menu command (see), for a whole project;

Show > Code coverage...

right click context menu for a particular module on a Project-Viewer (see);

Show > Code coverage...

right click context menu on the source edit form, for a single module (see).

### Remarks

- As for command `Unittest` (see), this is a tool for software quality assurance.

### Viewpoints

- We have been told that, besides the cited `Show > Code Coverage...` commands, possibly un-executed code lines are marked with a red flag on the source form. Not confirmed by our tests, though.

- -

---

<sup>13</sup> About which we'd like to know something more.

## Menu Command: **Debug**

Continue	Continue To Cursor	
Single Step	Step Over	Step Out
Stop		
Evaluate...	Execute...	
Toggle Breakpoint		
Edit Breakpoint...		
Next Breakpoint	Previous Breakpoint	
Clear Breakpoints		
Breakpoints		
Variables Type Filter...		
Exceptions Filter...		
Ignored Exceptions...		

Rich set of debugging tools (see: `Start > Debug Script... / Project...`), precious companion for the s/w developers.

### Viewpoints

- `<!>` Be aware that, as with menu command `Edit > Search` (see), any text editing action, such as deleting or adding source lines, will disrupt the functionality of all subsequent debugging actions, with unpredictable results, and no preventive warning. As a work-around to this flaw<sup>14</sup>, we'd suggest such quick action:  
`Save + (re-)Open`  
 on the very same source file or project, so to re-synchronize source text with debugging<sup>15</sup> execution, breakpoints included.
- We have noticed that in a debug session, with the condition "stop at first line" on (see), a possibly initial "pass" statement is ignored, with execution starting afterwards. Likewise we've noticed that also breakpoints possibly set on "pass" statements are ignored. Well, of course we can manage it<sup>16</sup>, but we don't agree.

- -

<sup>14</sup> Because, of course, *it is* a flaw, and serious too.

<sup>15</sup> Plus we dare to suggest that it shouldn't be that difficult to add such an action conveniently performed automatically, or semi-automatically; that is: upon user prompted request...

<sup>16</sup> As work-around for a perfect "dummy" statement, instead of "pass", you could be using such an assignment statement: "null=NULL"

**Debug > Continue****Debug > Continue To Cursor**

To resume execution after a breakpoint, and possibly up to the next one, or up to the currently pointed statement, excluded.

--

**Debug > Single Step****Debug > Step Over****Debug > Step Out**

To run debug in a single statement-by-statement execution mode, possibly stepping, or not stepping, into embedded functions code.

--

**Debug > Stop**

To terminate current execution.

--

**Debug > Evaluate...****Debug > Execute...**

To call a dialog box where to possibly enter a Python expression, or execute a Python statement, whose possible result will be shown on the interactive shell form. Same action can be carried on operating directly into the very interactive shell form<sup>17</sup>.

**Remarks**

- Here one must be well aware of the so called “mangling” mechanism used by Python to handle private members of a class, conventionally named this way: “`__<privateMember>`”.

Indeed, such an identifier, when used outside its class and outside its Eric editing form, should be written as: “`__<className>__<privateMember>`”, otherwise will result unknown. The same happens within the “Debug-Viewer” form (see).

--

---

<sup>17</sup> Where no operative difference have we found, but the fact that working on the interactive form is easier.

## Debug > Toggle Breakpoint

To insert/cancel an execution breakpoint on the currently pointed source statement. Same action can be carried on clicking just at right of the line number on the vertical ruler bar, at the left margin.

Breakpoint line is signaled by a red white-crossed head-pin mark, “dimmed” when disabled (see: `Edit Breakpoint...`).



## Viewpoints

- `<?>`[AsFor: Inaccuracy]

Well, we've found that this fundamental debug feature, of setting a breakpoints, is severely inaccurate, almost unreliable; at least in these situations:

- Breakpoint at a “`def Function(arg)`” statement is apparently accepted, but then, at execution time, it is ignored. Of course, once you know this, you'd simply position a breakpoint elsewhere.
- Same thing happens with “`pass`” statement. There too a breakpoint apparently can be set, but is ineffective. We think it would be fairer simply not to permit at all the positioning of such a breakpoint there, as it were a comment line. Anyhow, for the same reason why zero is considered a number, we think that a “`pass`” should be considered a statement.
- Any editing on the source text, resulting in an alteration of the number of lines, will invalidate (“shift”) the actual positioning of breakpoints, with external markers not affected, that is: *with no warning for the poor developer* (that's the point). Once you know it, the remedy is to close and re-open project, but first you have to know it.
- As we've incredulously verified, the same text editing possibly altering the number of lines (such as the insertion or deletion of comments) will invalidate (“shift”) also the search-text feature. Here too the remedy is a close and re-open sequence, but first you have to know it. Anyhow, it's rather annoying.

--

## Debug > Edit Breakpoint...

To show an `Edit Breakpoint` dialog box about the the currently pointed breakpoint-line. With control fields:

Condition: A Python conditional statement, enabling the breakpoint when `True`

Ignore Count: For a breakpoint to be ignored that many times, typically useful in debugging execution loops

Temporary Breakpoint

For a breakpoint to be executed just one time, then disabled

Enabled            To disable/enable a breakpoint

- -

### Debug > **Next Breakpoint**

### Debug > **Previous Breakpoint**

To shift cursor to the next/previous breakpoint on the current source form.

- -

### Debug > **Clear Breakpoints**

To clear all currently defined breakpoints.

- -

### Debug > **Breakpoints**

To display the position—that is: Python module file and line number—of all currently defined breakpoints.

- -

### Debug > **Variables Type Filter...**

To show a window where to select which type of variables, either local or global, should *not* be displayed on the Debug-Viewer form, tags: Global/Local Variables (see).

The purpose obviously being that of not being distracted by variables possibly considered inessential.

- -

### Debug > **Exceptions Filter...**

To show a dialog box where to enter codes of the only Python handled<sup>18</sup> exceptions (such as: `TypeError`, `ZeroDivisionError`, ...) to be considered during a debugging session, so that all the others will be ignored. This exception list is unique on Eric, and saved upon program exit. Possible unhandled exceptions remain unaffected.

Command aimed at focusing debug on selected exceptions (see also: `Ignored Exceptions...`).

- -

---

<sup>18</sup> That is, exceptions occurring within a “try-except” block.

## Debug > Ignored Exceptions...

To show a dialog box where to enter codes of the Python handled<sup>19</sup> exceptions (such as: `TypeError`, `ZeroDivisionError`, ...) to be ignored during a debugging session, so that all the others will be considered. This exception list is unique on Eric, and saved upon program exit. Possible un-handled exceptions remain unaffected.

Command aimed at focusing debug on selected exceptions (see also: `Exceptions Filter...`).

### Remarks

- Pressing the “Ignore” button on a “Break here?” Eric dialog box will automatically add the involved exception code to this list (see also: “Report exceptions” check box on the “Start > Debug” menu command).

- -

---

<sup>19</sup> That is, exceptions occurring within a “try – except” block.

## Menu Command: **Unittest**

unittest...

Restart unittest...

unittest Script...

unittest Project...

### *Remarks*

- Another tool of software quality assurance, as it is the “Coverage run” command, within the “Start” menu (see).

### *Viewpoints*

- So far we have been successful in creating a simple Python “unittest” working example<sup>20</sup>, based upon two modules: one containing a function to be tested, and another containing the definition of a test class as requested by the standard `unittest` Python module.  
Fine, but then we haven't been able to grasp the relation of such Python mechanism with this Eric menu command set...
- Anyhow as a first impression, for what a first impression is worth, this whole mechanisms looks rather clumsy and utterly impractical. We'd really glad to know how actually popular it is, in itself and in its Eric version...

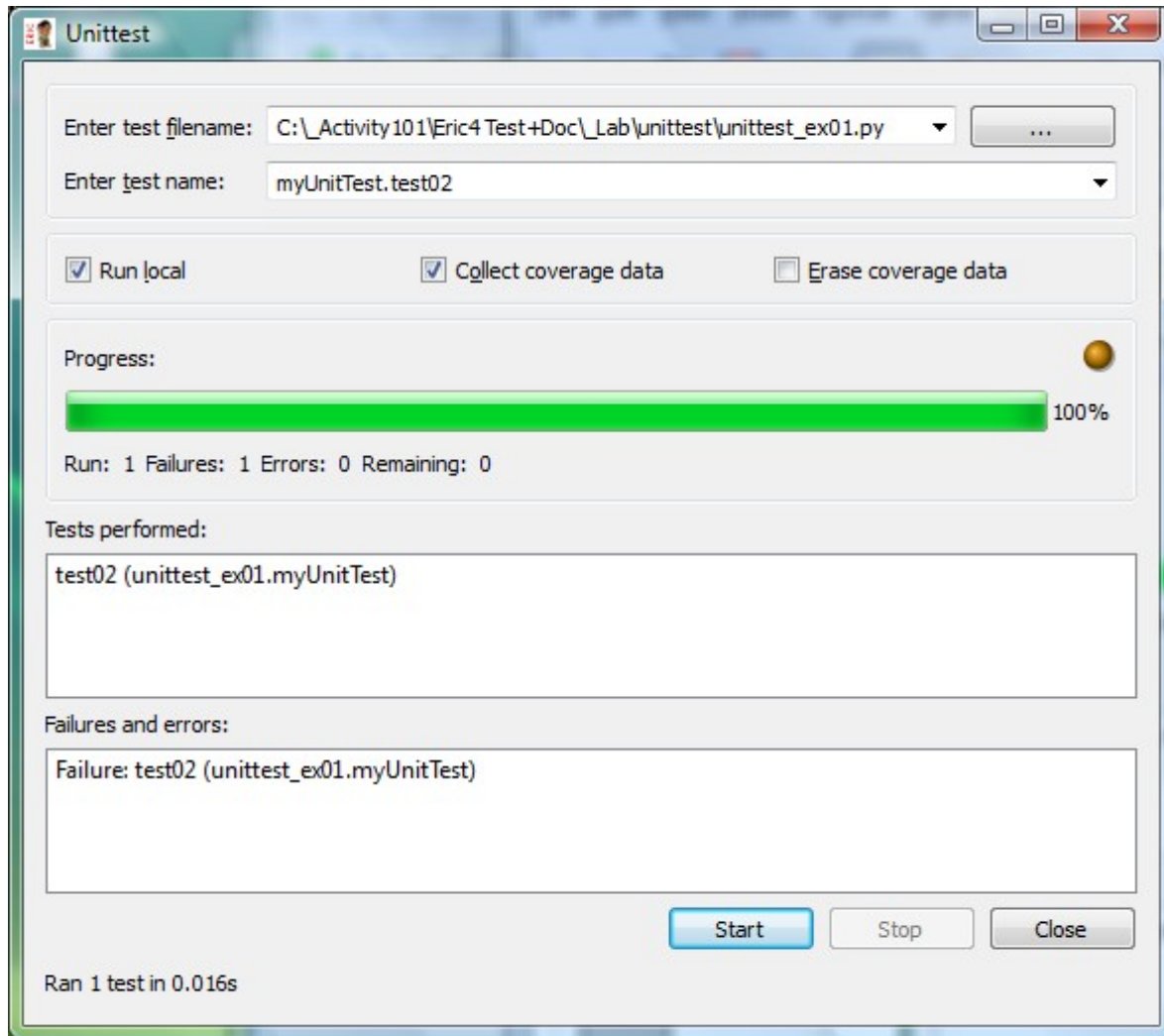
- -

---

<sup>20</sup> Following the instructions contained in chapter 11 of the “Python Essential Reference” by D. M. Beazley.

**Unittest > unittest...**

To show a “Unittest” dialog box where to set and execute a standard Python unittest procedure.



Where:

Enter test filename

“\*.py” module file to perform the desired test, according to the Python “unittest” standard technique.

Enter test name

Possible dot-identifier of a specific test-method within the testing class (ref.: `unittest method TestLoader.loadTestsFromName`). When not entered, all available tests will be executed.

Run local

To make a choice whether to execute the test directly, in the current process, or in a debugger backend, possibly in another computer in remote testing<sup>21</sup>.

Collect coverage data

Erase coverage data

To handle the “Python Code Coverage” coverage data, as for `Start > Coverage run of Script...` (see), related with these tests.

### Remarks

- This “`Unittest > unittest...`” menu command results always enabled, keeping the values entered on the fields, so to possibly perform several tests, also with no Python module or project opened.

--

Unittest > **Restart unittest...**

Unittest > **unittest Script...**

Unittest > **unittest Project...**

Same as for the `unittest...` command (see), but with some fields conveniently pre-filled with data derived from the current operative elements, such as the possibly opened module.

--

---

<sup>21</sup> Functionality that we haven't tested.

## Menu Command: **Multiproject**

New...

Open...      Open Recent Multiprojects

Close

Save          Save as...

Set of “Multiproject” commands precisely corresponding to those aimed at handling the Python module and project files (see Menu Command: File, and Menu Command: Project), to which you may refer for description.

--

Add Project...

Properties...

Only “Multiproject” commands specific of the Multi-Project structure, and therefore hereafter documented.

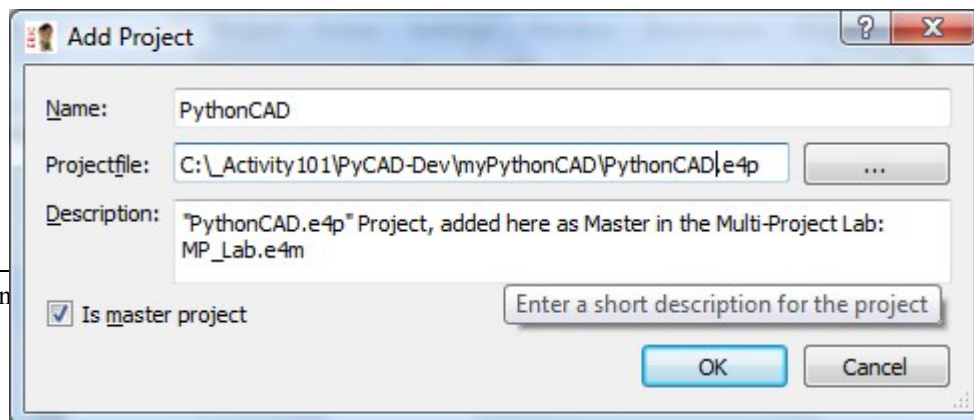
### Remarks

- As it's immediate to see a Project as a coherent collection of different Modules, it's likewise reasonable to see a Multi-Project as a collection of different Projects that, for some reason, it's convenient to regard as a single macro-entity.
- In a Multi-Project there is a “*Master Project*”, defined as the first to open. It's a concept corresponding to that of “*Main Script*”<sup>22</sup> defined in a Project (see: Menu Command: Project) as the first source module to open and execute.

--

### Multiproject > **Add Project...**

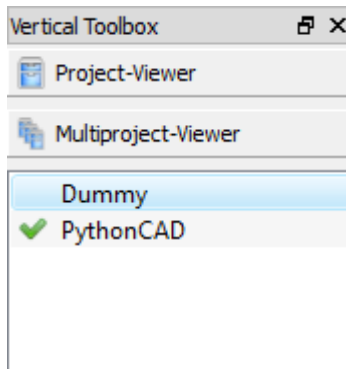
To open a dialog box aimed at adding an existing project to the currently opened multi-project.



<sup>22</sup> So correspond

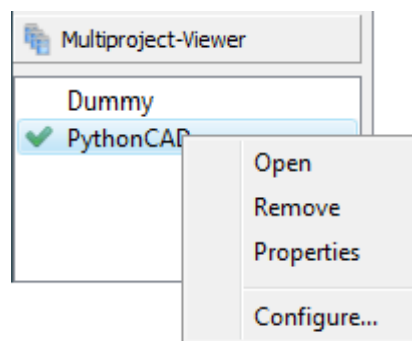
in Project”.

When a multi-project is opened, in the so called `Vertical Toolbox` (see `Window` menu command) the tagged form `Multiproject-Viewer` becomes significant.



And there you have:

- A list of the currently added projects, with ticked the name of the Master Project.
- Handy the `Project-Viewer` form, to inspect the currently selected project.
- Handy a right-click pop-up menu, to manage the selected project.

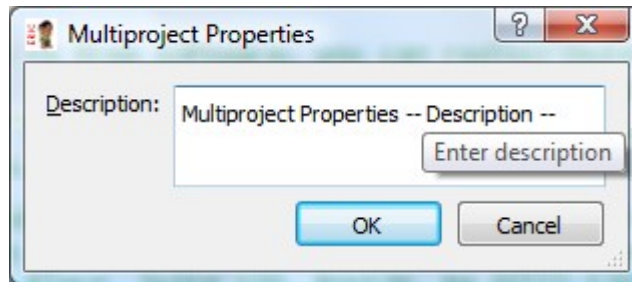


Here command `Configure...` corresponds to menu `Settings > Preference... - Projects > Multiproject - Configure multiproject settings (see)`.

--

**Multiproject > Properties...**

To display a dialog box where a “Description” can be first entered, then inspected<sup>23</sup>.



--

---

<sup>23</sup> A Description here to be considered as the initial of a set of properties, intended for future enhancements.

## Menu Command: **Project**

New...  
 Open...      Open Recent Projects  
 Close  
 Save          Save as...

Set of commands precisely corresponding to those aimed at handling Python module files, to which you may refer for description (see Menu Command: File).

- -

### Remarks

- An Eric project, differently from a simple Python module, has got specific properties that can be both assigned at its creation and also managed subsequently, via command menu `Project > Properties` (see). Some of these properties are worth to be recalled right away:

`Project Directory`

Project's host directory, holding, in particular:

`*.e4p`      An XML file to define all what a project is;

`_eric4project`

A project management sub-directory, automatically created to keep some complementary XML files (see section: [Eric Related Directory](#), in [Appendix](#));

`Main Script` Project's entry module, where execution will start.

It's a concept corresponding to that of the “Master Project”<sup>24</sup> in a Multi-Project (see).

`Version Control System`

<?>[AsFor: Not tested] Feature not tested.

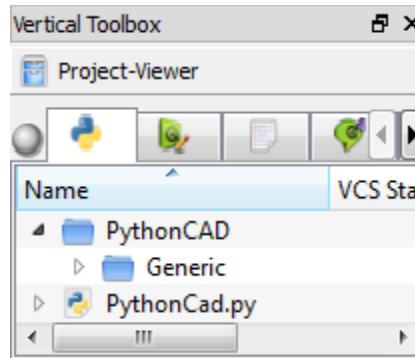
Property automatically asked at a new project inception about the possible adoption of a Version Control System (VCS)—that is: Revision Control System (RCS), or Source Code Manager (SCM)—, as with menu command `Project > Version Control` (see).

If no such system in use, a “None” answer will do.

---

<sup>24</sup> So corresponding that it would be reasonable to adopt a similar denomination, such as: “*Main Script*” and “*Main Project*”.

- When an Eric Project is opened, in the so called `Vertical Toolbox` (see also: `Window` menu command) the tagged form `Project-Viewer` becomes significant.



For a detailed description see section: `Vertical Toolbox`, chapter 3. `Menu Complements`.

- The closure of a project implies also the closure of its main script, but not always the closure of all other of its modules possibly open (see also: `File > Close`). That said also with reference to the close and re-open sequence needed to re-synchronize debug functions after the possible editing of a source script (see).

- -

Then, here below, about the specific and fundamental `Project's` command set.

Debugger

Session

Add Files...

Add Directory

Add Translation

Search New Files...

Diagrams

Check

Version Control

Show

Source Documentation

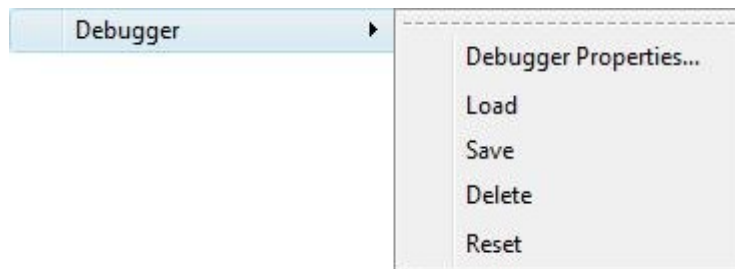
Packagers

Properties...  
 User Properties...  
 Filetype Associations...  
 Lexer Associations...

--

## Project > **Debugger**

To call a sub-menu aimed at managing the special “Project Debugger Properties”, possibly defined on a per-project basis.



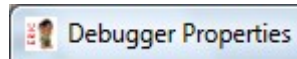
### Remarks

- `<?>`[AsFor: *Hic sunt leones*] Unexplored territory where, for lack of info, we don't even dare to venture. The info here missed is not of generic nature, but that specifically related to this very command or parameter.

--

## Project > Debugger > **Debugger Properties...**

To show the dialog box where special “Project Debugger Properties” can be set, to be stored into the current project management directory “\_eric4project”, on a XML “<myProject>.e4d” file.



**Remarks**

- `<?>`[AsFor: *Hic sunt leones*] Unexplored territory where, for lack of info, we don't even dare to venture. The info here missed is not of generic nature, but that specifically related to this very command or parameter.
- This matter is also related with menu command `Settings > Preferences..., Debugger > Python > Configure Python Debugger (see)`, in particular with the possible automatic loading of custom Debugger Properties as here defined.

- -

**Project > Debugger > Load**

To load the Project Debugger Properties file possibly defined for the current project (see: `Project > Debugger > Debugger Properties...`).

- -

**Project > Debugger > Save**

To save the Project Debugger Properties, as currently defined, on a XML “`<myProject>.e4d`” file of the project management directory “`_eric4project`”.

**Remarks**

- It's not clear which is the file here saved, nor what's the relation between this command and the `Project > Debugger > Debugger Properties...` (see).

- -

**Project > Debugger > Delete**

To delete the Project Debugger Properties file currently present on the project management directory (see: `Project > Debugger > Save`).

- -

**Project > Debugger > Reset**

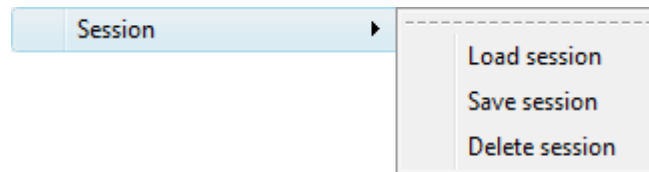
`<?>`[AsFor: *Hic sunt leones*] Unexplored territory where, for lack of info, we don't even dare to venture. The info here missed is not of generic nature, but that specifically related to this very command or parameter.

- -

## Project > **Session**

To call a sub-menu aimed at managing the file of data characterizing the current project's working session, that is:

- Open source files
- Breakpoints
- Command line arguments
- Working directory
- Exception reporting flags



Saving and re-loading such file is a quick way to conveniently restore possibly complex working conditions.

### *Remarks*

- The project's working session data are stored into a XML file named “<myProject>.e4s”, located into the project management directory “\_eric4project”.

- -

## Project > Session > **Load Session**

To load the current project's working conditions possibly saved (see: Project > Session).

- -

## Project > Session > **Save Session**

To save the current project's working conditions (see: Project > Session).

- -

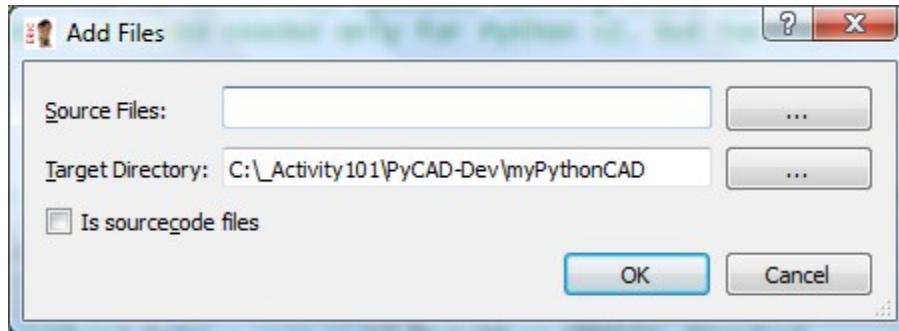
## Project > Session > **Delete Session**

To delete the current project's working conditions possibly saved (see: Project > Session).

- -

## Project > **Add Files...**

To show a dialog box aimed at adding new files to the current project.



Where:

Source files: Where to select the file(s) to be added

Target Directory:

Destination directory for the files to be added to the project

Is sourcecode files

<?>[AsFor: Missing info] The info we miss here is not of generic nature, but that specifically related to this very command or parameter.  
(see *Viewpoints* below)

The added files will result stored into a target directory assigned to the very project, and there copied if it is different from the original source directory. In that case, the original files will remain unaffected.

### Remarks

- This command can operates on one file or more than one at a time.

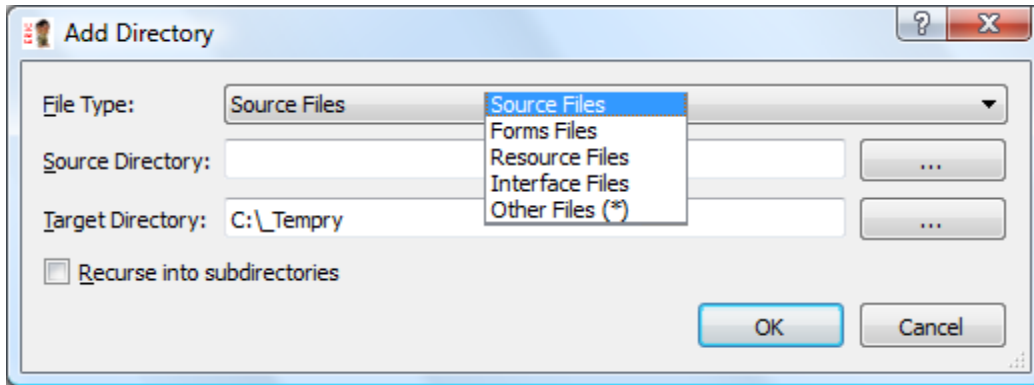
### Viewpoints

- <?> Assuming that the “Is sourcecode” Check Box is actually referred to Python source code, we find it somewhat baffling, indeed:
  - We'd like to know which other file-type could possibly be added to a project;
  - We've noticed that actual source code files are added correctly even with this check-box left unchecked, as it were ignored, or defaulted.
  - We wander if the very extension of the added file could be enough to imply what a file is all about, couldn't it?

- -

## Project > **Add Directory**

To show a dialog box aimed at adding a whole directory of new files to the current project.



Where:

**File Type:** File-type filter, for possibly selecting the type of files to be added to the project

**Source Directory:**  
Directory to scan for files to be added to the project

**Target Directory:**  
Directory to associate to the project, where current new files are to be added, and possibly copied if it's different from the source directory

**Recurse into subdirectories**  
Addition extended to the sub-directories' contents

### Viewpoints

- Here we don't see a "Is sourcecode" check-box, as in the "Add Files..." dialog box (see), so that we're encouraged thinking that it's useless there too.
- Could this drop-down list be an implicit answer to the afore-asked question about which the file possibly to be added to a project?
- We think it should be re-typed with trailing ellipsis "...", to show that this menu command is not immediately operative, but is to display a structure where an action will be possibly performed.

- -

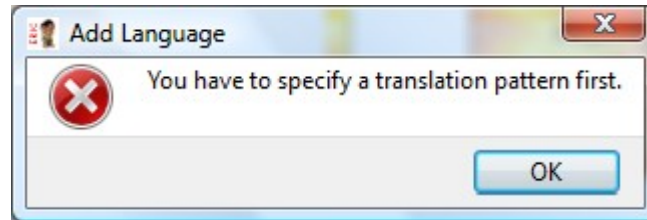
## Project > **Add Translation**

<~>[AsFor: Off scope] Topic just hinted at, being off the main scope of this Report.

See *Remarks* below.

### Remarks

- At first, all what you get is this “Add Language” error dialog box.



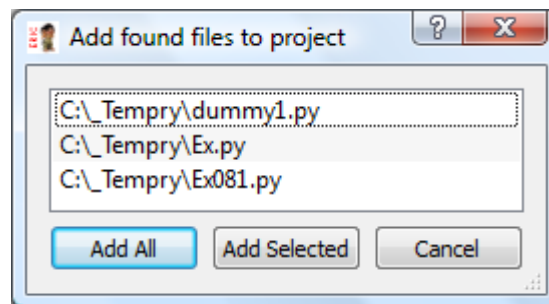
Well then, why not simply “dim”-disable this command in that circumstance?

- Anyhow, what's such “translation pattern”? To be set where/how?

- -

## Project > **Search New Files...**

To display a form aimed at possibly adding to the current project new files found in the current Project's directory, and not belonging to it.



### Viewpoints

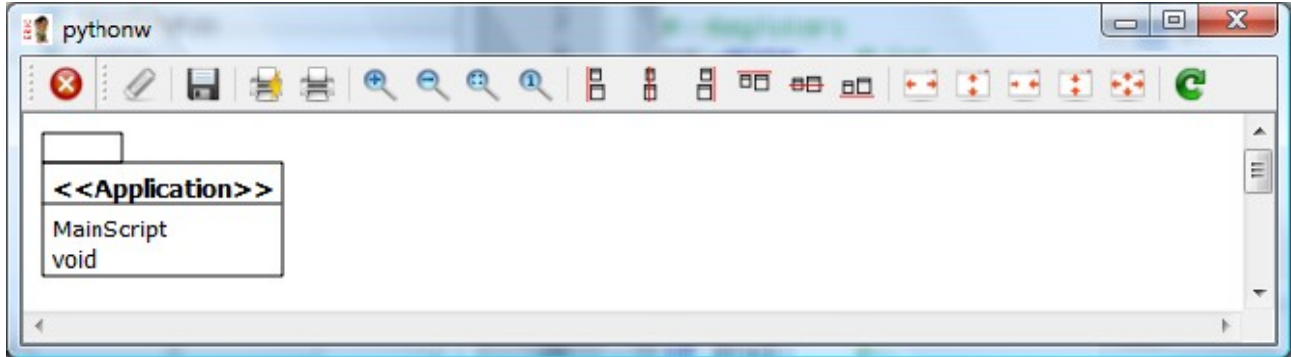
- <?> This command implies the relevant concept of a “Project Directory”, which we'd like to know exactly what it is, and in which relation it is with the possibly different “Target Directories” as for the “Add Files...” and “Add Directory” commands (see).

- -

### Project > **Diagrams**

<~>[AsFor: Off scope] Topic just hinted at, being off the main scope of this Report.

After the opening of a dialog box asking about the possible inclusion of “Module Names”, this command will display such an impressive “pythonw” window:



### Viewpoints

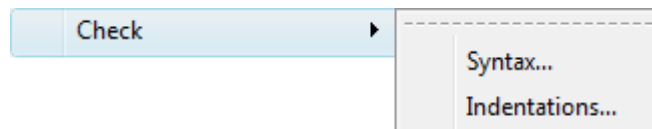
- Before daring to treat this “pythonw” topic, we'd like to be better informed about its general sense and purpose.
- Having seen this “Diagrams” command displaying such a one-item sub-menu, we are led to assume that it's meant for future enhancements, that is to host more than this unique “Application Diagram” choice.



--

### Project > **Check**

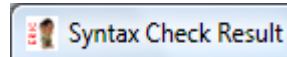
To activate a sub-menu for these two Python source check options.



--

## Project > Check > **Syntax...**

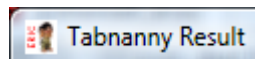
To quickly scan current project for Python syntax errors, possibly displayed on a “Syntax Check Result” window.



--

## Project > Check > **Indentations...**

To quickly scan current project for Python indentation errors, possibly displayed on a “Tabnanny Result” window.



## Remarks

- “*Tabnanny*” is a standard Python service module, aimed at the “*Detection of ambiguous indentation*”. But, having to do with semantics, be warned that this test is not, and couldn't be, a sure bet.

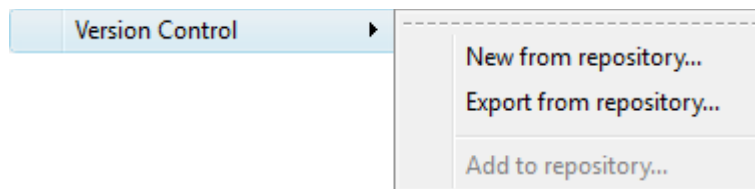
--

## Project > **Version Control**

<~>[AsFor: Off scope] Topic just hinted at, being off the main scope of this Report.

Feature not tested.

To show a sub-menu of commands aimed at managing the Version Control System (VCS)—that is: Revision Control System (RCS), or Source Code Manager (SCM)—possibly adopted for the project.

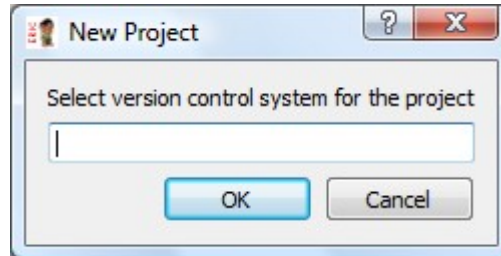


--

**Project > Version Control > New From Repository...**

<?>[AsFor: *Hic sunt leones*] Unexplored territory where, for lack of info, we don't even dare to venture. The info here missed is not of generic nature, but that specifically related to this very command or parameter.

With a given VCS installed (see chapter 5. **Setup and General Management**), here you'll presumably be guided through the steps to perform a project checkout, with a link then maintained with the repository.

**Remarks**

- With no VCS installed this dialog box should not appear.

- -

**Project > Version Control > Export From Repository...**

<~>[AsFor: Off scope] Topic just hinted at, being off the main scope of this Report.

Feature not tested.

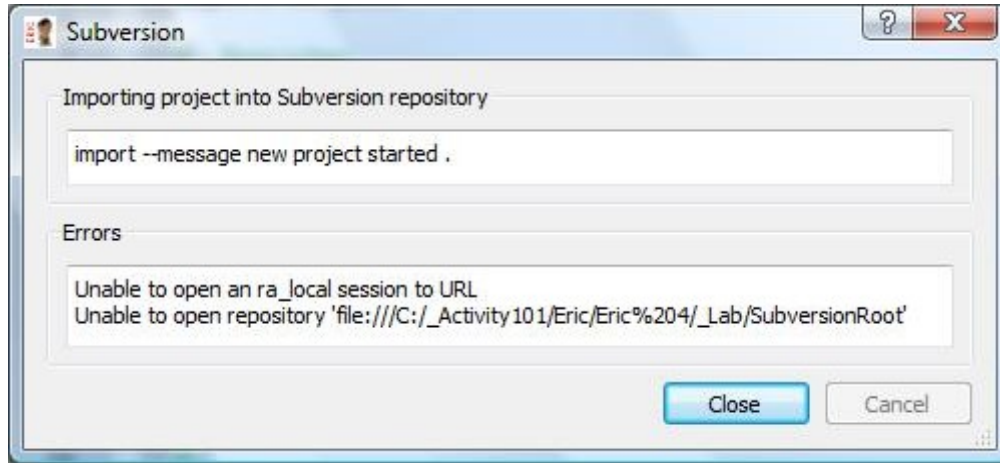
With a given VCS installed, here you'll be presumably guided through the steps to perform a project export, no further link then maintained with the repository.

- -

Project > Version Control > **Add to Repository...**

<~>[AsFor: Off scope] Topic just hinted at, being off the main scope of this Report.

To add a non-VCS project into a VCS source code repository.

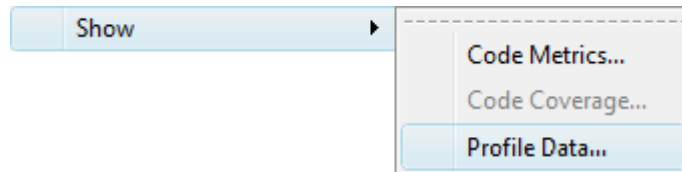


Feature not tested.

--

Project > **Show**

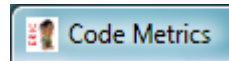
To activate a sub-menu for displaying Project statistic data.



--

**Project > Show > Code Metrics...**

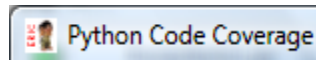
To collect and display a “Code Metrics” window of comprehensive “static” census data, such as: name and number of files, count of empty lines, ...



- -

**Project > Show > Code Coverage...**

To display the “Python Code Coverage” window of comprehensive “dynamic” census data, possibly collected and stored during a “Start > Coverage run” session (see).

*Viewpoints*

- <!> Rich, impressive set of data, that we'd like to interpret better than we currently do.

- -

**Project > Show > Profile Data...**

To display the “Profile Results” window of comprehensive “dynamic” census data, possibly collected and stored during a “Start > Profile Project...” session (see).

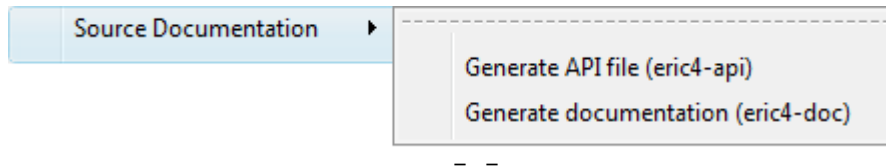
*Viewpoints*

- <!> Rich, impressive set of data, that we'd like to interpret better than we currently do.

- -

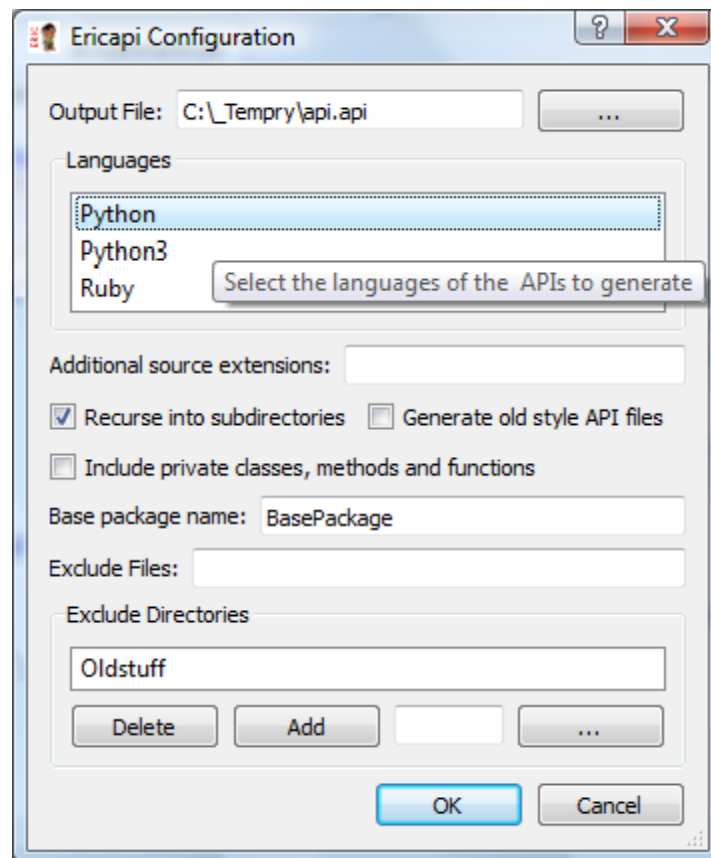
## Project > **Source Documentation**

To show a sub-menu aimed at generating “API” and “documentation” files derived from Eric source projects. About the role and meaning of such files, see related command.



## Project > Source Documentation > **Generate API file (Eric4-API)**

To show a control box aimed at generating a so called “\*.api” type file<sup>25</sup>, derived from the current source project.



<sup>25</sup> As it's well known, the acronym API usually stands for “Application Program Interface”.

A file listing such kind of fully qualified names:

```
BasePackage.Generic.Kernel.GeoUtil.geolib.Vector.rotate?4 (angle)
BasePackage.Generic.Kernel.GeoUtil.geolib.Vector.x?4 ()
BasePackage.Generic.Kernel.GeoUtil.geolib.Vector.y?4 ()
BasePackage.Generic.Kernel.GeoUtil.geolib.Vector?1 (p1, p2)
```

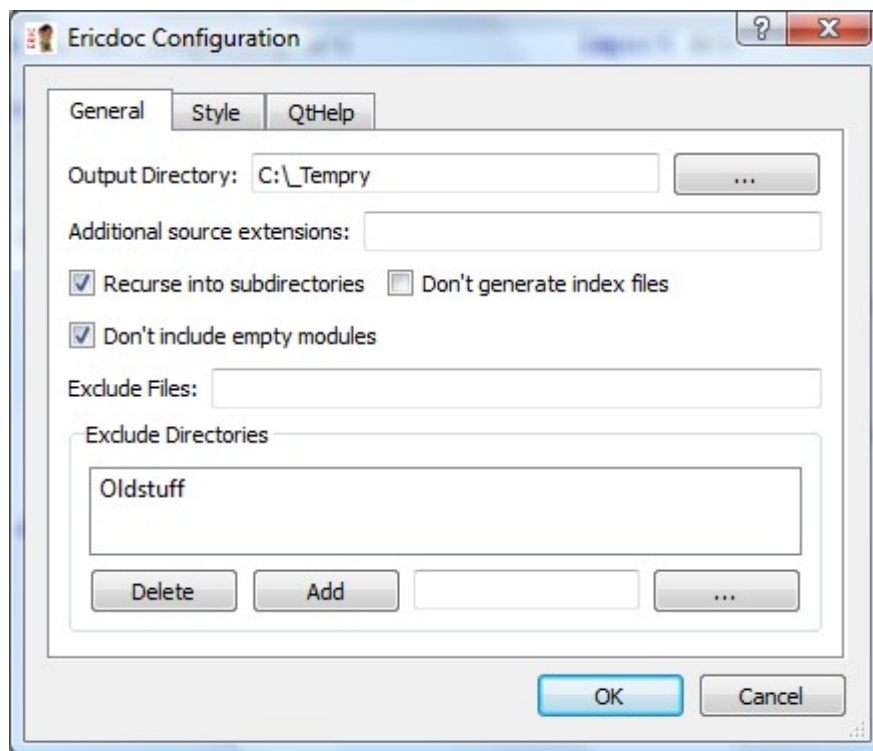
### Viewpoints

- Which exact role and use can be only guessed, being not precisely known.

- -

### Project > Source Documentation > **Generate Documentation (Eric4-Doc)**

To show a control box aimed at generating automatically a set of documentation files, derived from the current source project.



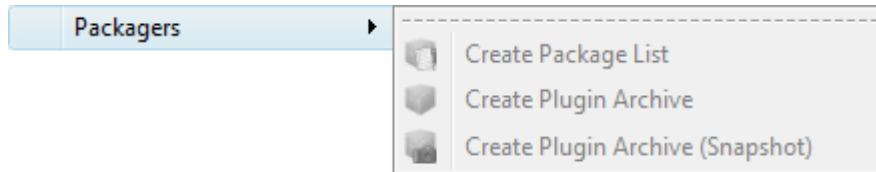
### Viewpoints

- Quality of the documentation generated this way reflects also the quality of the Python standard “Documentation Strings” actually inserted into the source project.
- In a real case of a rather large Python project we've realized that several actual entries were missing. To nobody's concern, though, being documentation issues rarely a concern. Alas.

- -

### Project > **Packagers**

To show a sub-menu aimed at the development of Eric Plugin (see).



### Project > Packagers > **Create Package List**

### Project > Packagers > **Plugin Archive**

### Project > Packagers > **Plugin Archive (Snapshot)**

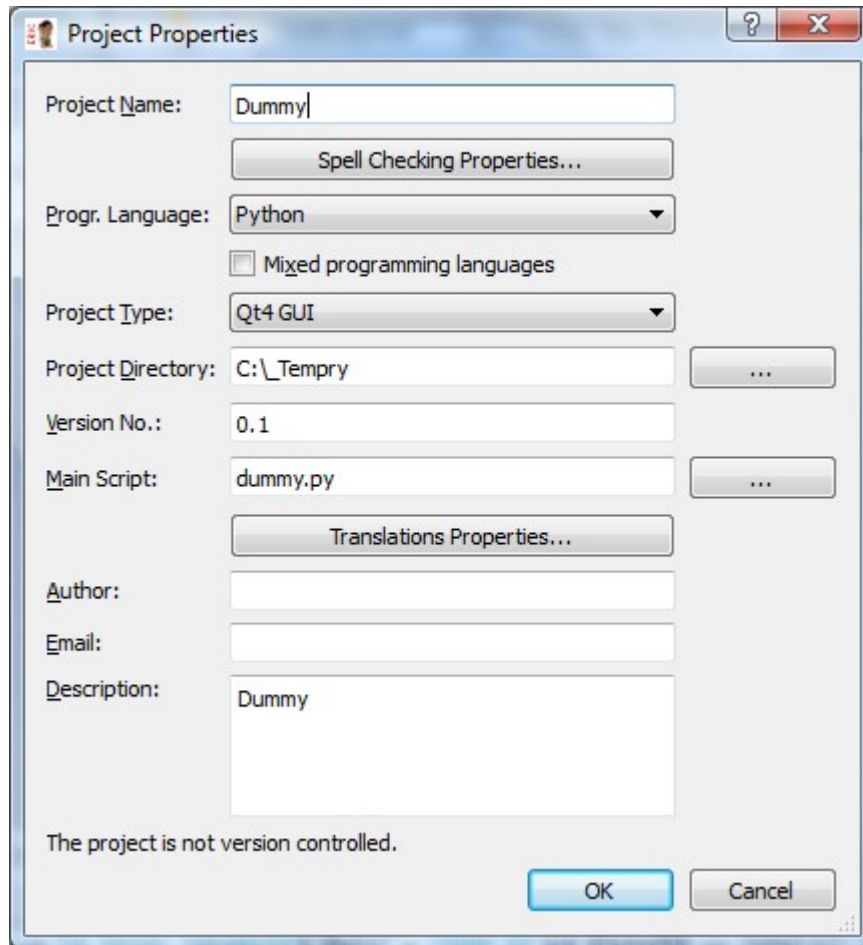
<~>*[AsFor: Off scope]* Topic just hinted at, being off the main scope of this Report.

This whole subject, being not related with the ordinary use of Eric, but with its development instead, is assumed out of the scope of this work and, therefore, here only mentioned and not treated.

- -

## Project > **Properties...**

To show a window where to set current project's properties.



With notable fields:

Spell Checking Properties... <?>[AsFor: Missing info] The info we miss here is not of generic nature, but that specifically related to this very command or parameter.

Prog. Language

A drop-down list offering an interesting choice amongst Python and Ruby.

Mixed programming languages

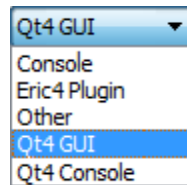
<?>[AsFor: Missing info] The info we miss here is not of generic nature, but that specifically related to this very command or parameter.

**Remarks**

- Well, does this mean that this is an IDE that could equally be used for anyone of these languages? Or even a mixture of them? Rather relevant a question, isn't it?

Main Script    Project's entry module, where execution will start

Project Type    A pre-established list of choices:

**Remarks**

- It's certainly a relevant choice, worth to be conveniently explained in each of its possibilities, and related consequences.

Main Script:    Rather relevant setting of the entry-module for the project

Translations Properties...

<?>[AsFor: Missing info] The info we miss here is not of generic nature, but that specifically related to this very command or parameter.

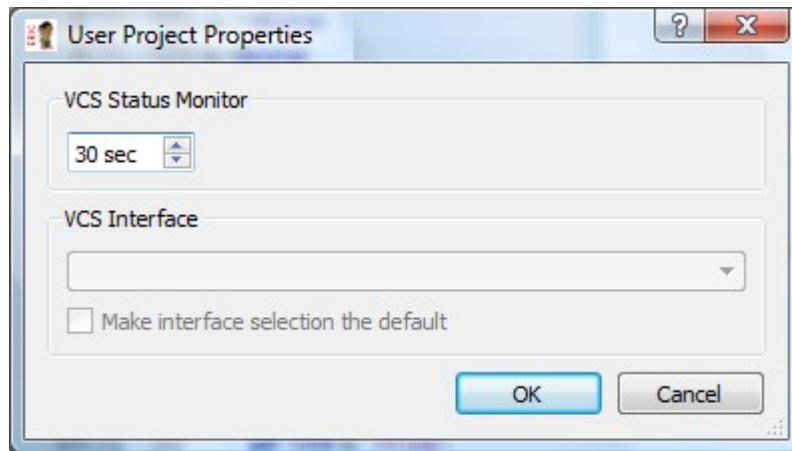
**Remarks**

- The end-caption “The project is not version controlled.” is related to the Version Control System (VCS)—that is: Revision Control System (RCS), or Source Code Manager (SCM)—possibly adopted, as with menu command `Project > Version Control` (see).

- -

### Project > **User Properties...**

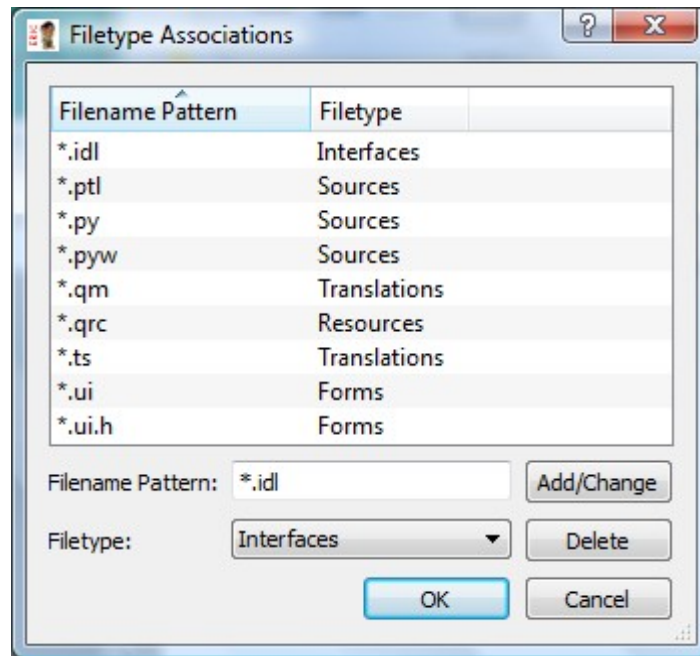
To show a box just to display some info about the Version Control System (VCS)—that is: Revision Control System (RCS), or Source Code Manager (SCM)—possibly adopted (see: Project > Version Control).



- - -

### Project > **Filetype Associations...**

To show a window where to see and set the extensions to be associated to the files normally used in a project.

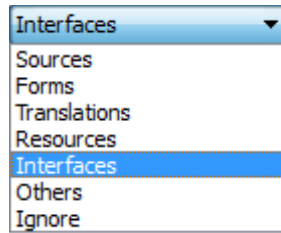


Remarks

- What here is called “Pattern” (Filename Pattern) is usually called “file extension”, or simply “extension”.

Viewpoints

- We deem such a “Filetype” drop-down list much more important for what it says than for what it asks.

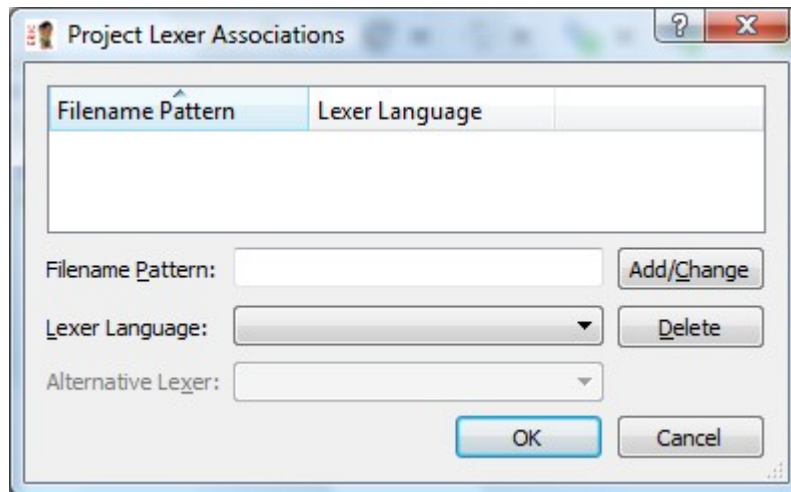


- In particular it is our intention to thoroughly commented this Filetype list item-by-item, as soon as we've all the related information we need.

--

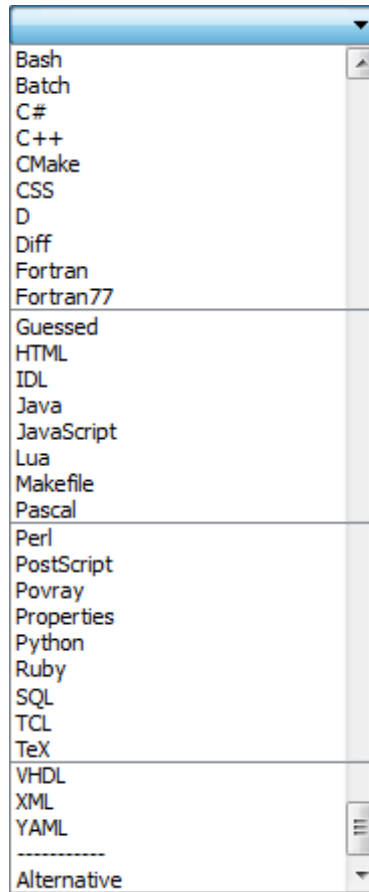
Project > **Lexer Associations...**

<?>[AsFor: Hic sunt leones] Unexplored territory where, for lack of info, we don't even dare to venture. The info here missed is not of generic nature, but that specifically related to this very command or parameter.



### Viewpoints

- “Lexer Language:” control drops down a rather interesting and intriguing list,



which we intend to comment thoroughly, as soon as we'd collected the related info.

--

## Menu Command: **Extras**

Spell Check...  
 Automatic Spell Checking  
 Wizards  
 Macros  
 Tools

### Extras > **Spell Check...**

To show a “`Check spelling`” window, where to check and manage the spelling of text written on the current source form, typically the comments.

#### *Remarks*

- This command requires the presence of the PyEnchant spell-checker toolkit, as described in chapter 5. **Setup and General Management**, section **Optional Packages** (see). With such toolkit not installed, this command results “dim”-disabled.
- Spell-check options—notably: the language—can be managed with menu command `Settings > Preferences... > Spell checking` (see). When PyEnchant not installed, you will be there informed that “`Spell checking with PyEnchant is not available`”.

#### *Viewpoints*

- `<!>` This is a good place where to recall the fundamental relevance of source text documentation.

--

### Extras > **Automatic Spell Checking**

To enable/disable automatic spell checking of the text on the currently active editing form. Possibly misspelt words are merely signaled with a red wavy underlining, otherwise unchanged.

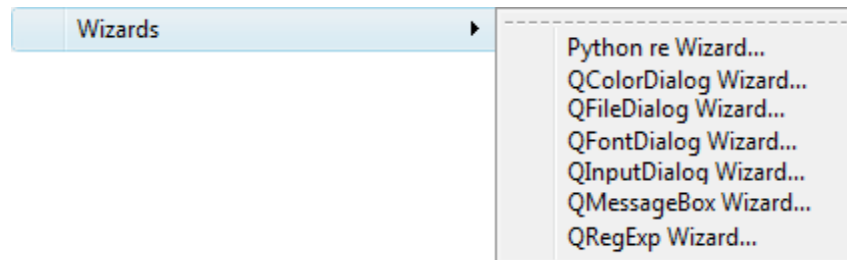
#### *Remarks*

- For actual spell editing see menu command `Extras > Spell Check...`

--

## Extras > **Wizards**

To show a sub-menu of Eric “wizards”.



### Remarks

- The commands in this `Extras > Wizards` sub-menu correspond to last seven “Wizard” items in the `Plugins > Plugin Infos...` list (see).

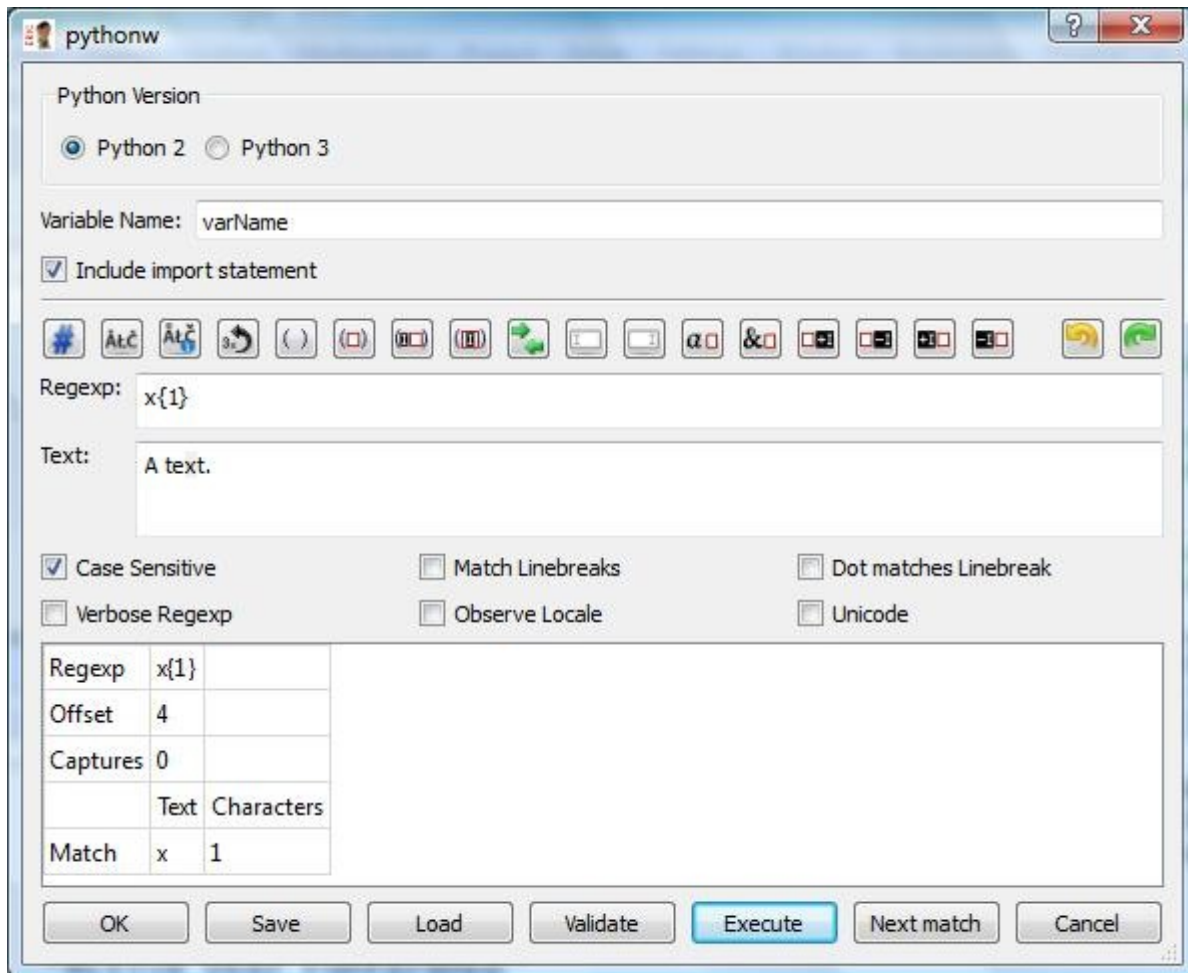
### Viewpoints

- In this set of sub-menu commands we suggest to drop the repetition of the term “Wizard”, for manifest redundancy.

--

## Extras > Wizards > Python re Wizard...

A s/w productivity tool concerning the Python “re” regular expressions module.  
A tool offering this control box:



and generating this kind of source fragments:

```
import re
varName = re.compile(r""x{1}""")
```

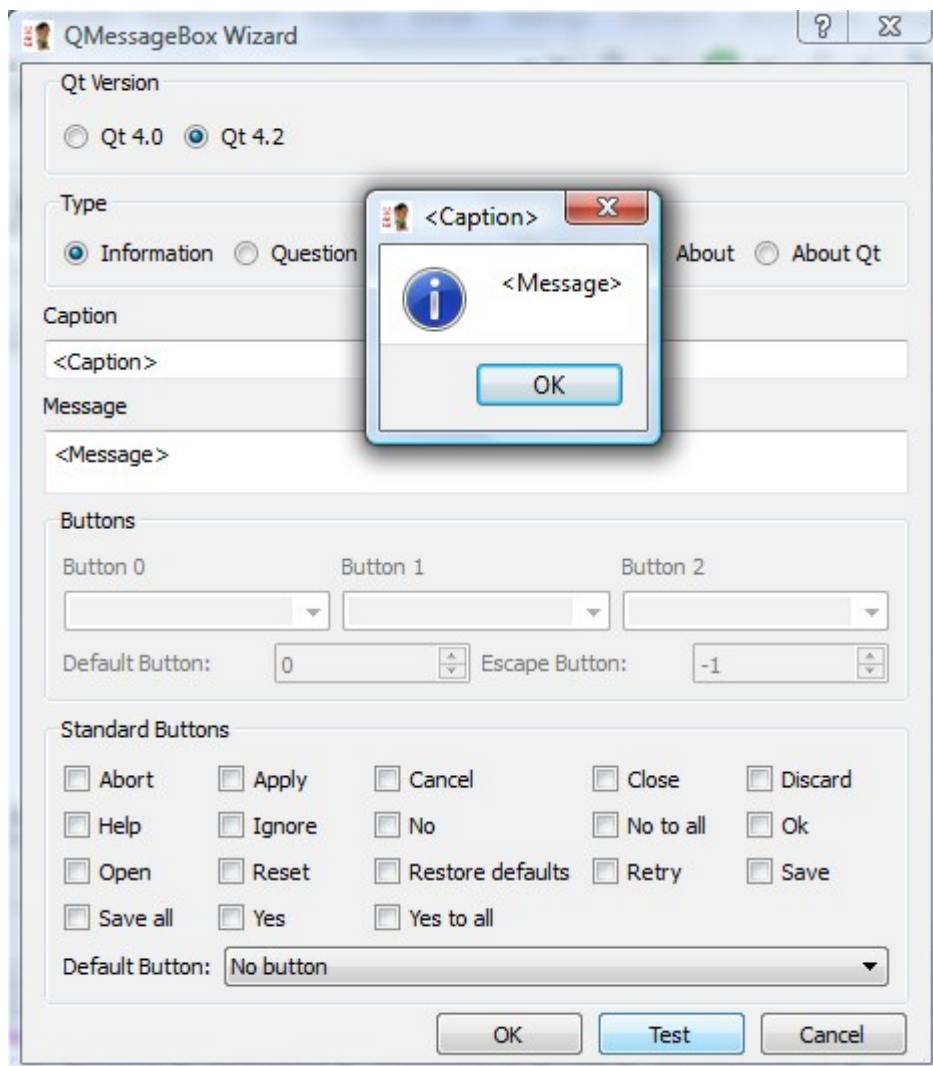
### Viewpoints

- Further investigation on this matter is deliberately left to the personal experience of use.

--

- Extras > Wizards > **QColorDialog Wizard...**
- QFileDialog Wizard...**
- QFontDialog Wizard...**
- QInputDialog Wizard...**
- QMessageBox Wizard...**
- QRegExp Wizard...**

A set of s/w productivity tools, offering such kind of control box:



Software productivity tools for PyQt<sup>26</sup> library users, concerning QtGui module (classes: QColorDialog, QFileDialog, QFontDialog, QDialog, QMessageBox) and QtCore module (class: QRegExp). So to conveniently test and generate such kind of source fragments:

```
QMessageBox.information(None, self.trUtf8("<Caption>"), self.trUtf8(""<Message>
    """), QMessageBox.StandardButtons(\QMessageBox.Ok))
```

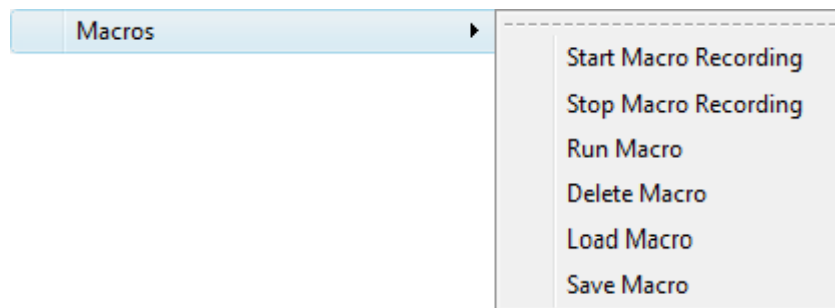
### Viewpoints

- Further investigation on this matter is deliberately left to the personal experience of use.

--

### Extras > Macros

To show a sub-menu of commands aimed at managing not so well defined “Macros”.



### Viewpoints

- We have been told that this functionality has been derived from the QScintilla toolkit<sup>27</sup>, and then, subsequently, not much looked after. We can confirm it looks all rusty and abandoned.
- We have tried to use them to record and save some frequently used source code fragments, ready to be recalled and inserted when needed. Such as a "Shebang" command:
 

```
#!/usr/bin/env python
```

 inserted as the first source line for Unix-like systems. Not very successfully, though.
- Here we dare suggest to drop the term “Macro”, for manifest redundancy. Or, better, to eliminate this whole set of commands altogether, in wait for a possible recovery, if worth doing.

--

<sup>26</sup> About PyQt see chapter 5. Setup and General Management, section Prerequisites.

<sup>27</sup> Which doesn't bring that much clarification to us.

**Extras > Macros > Start Macro Recording****Extras > Macros > Stop Macro Recording**

To start/stop the recording of a text editing session on the source form. Recorded macro then can be Run and Saved (see).

- -

**Extras > Macros > Run Macro**

To execute a just Recorded or a possibly Loaded macro (see).

- -

**Extras > Macros > Delete Macro**

To delete a just Recorded or a possibly Loaded macro (see). Related file remains unaffected.

- -

**Extras > Macros > Save Macro****Extras > Macros > Load Macro**

To save/load a possibly Recorded (see) macro on a disc file<sup>28</sup>. To be Run, a Saved macro must be Loaded first.

- -

**Extras > Tools**

Here is where the Eric's end user can create his own custom menu commands, aimed at running external tools. Custom menu commands organized into so called *Tool Groups*, and possibly added besides the two initial standard ones: Builtin Tools and Plugin Tools (see).

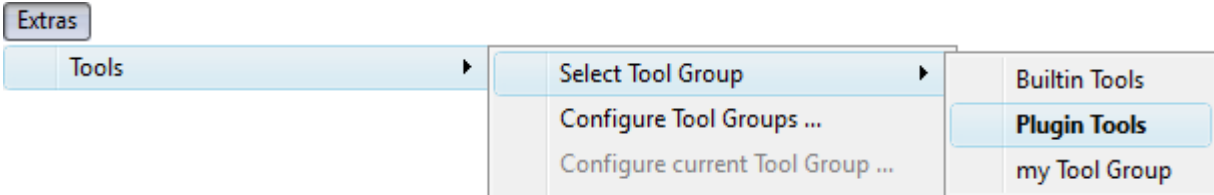
- -

---

<sup>28</sup> Default directory: <User>\\_eric4, File extension: “\*.macro”.

## Extras > Tools > **Select Tool Group**

To select a Tool Group among the two standard “Plugin” and “Builtin” and, possibly, the custom ones subsequently added by the end user (see: `Configure Tool Groups...`).



Effects on the selected Tool Group:

- Name bold-faced;
- Related tools listed after the bottom menu separator, available to be used;
- Command `Configure current Tool Group...` enabled, for a custom tool group (see).

### Remarks

- `<?>`[AsFor: Missing info] We'd like to know something more about the origin and definition of the “Builtin Tools” set.
- The “Plugin Tools” set can be defined with command “`Plugins > Install`” (see). As an example, here we've added the “`Character Tables...`” (see).

- -

Extras > Tools > **Configure Tool Groups...**

To manage—create, delete, ...—the custom Tool Groups, that is the set of menu commands aimed at calling external toolkits that the end user can add besides to the two standard “Builtin Tools” and “Plugin Tools” (see).

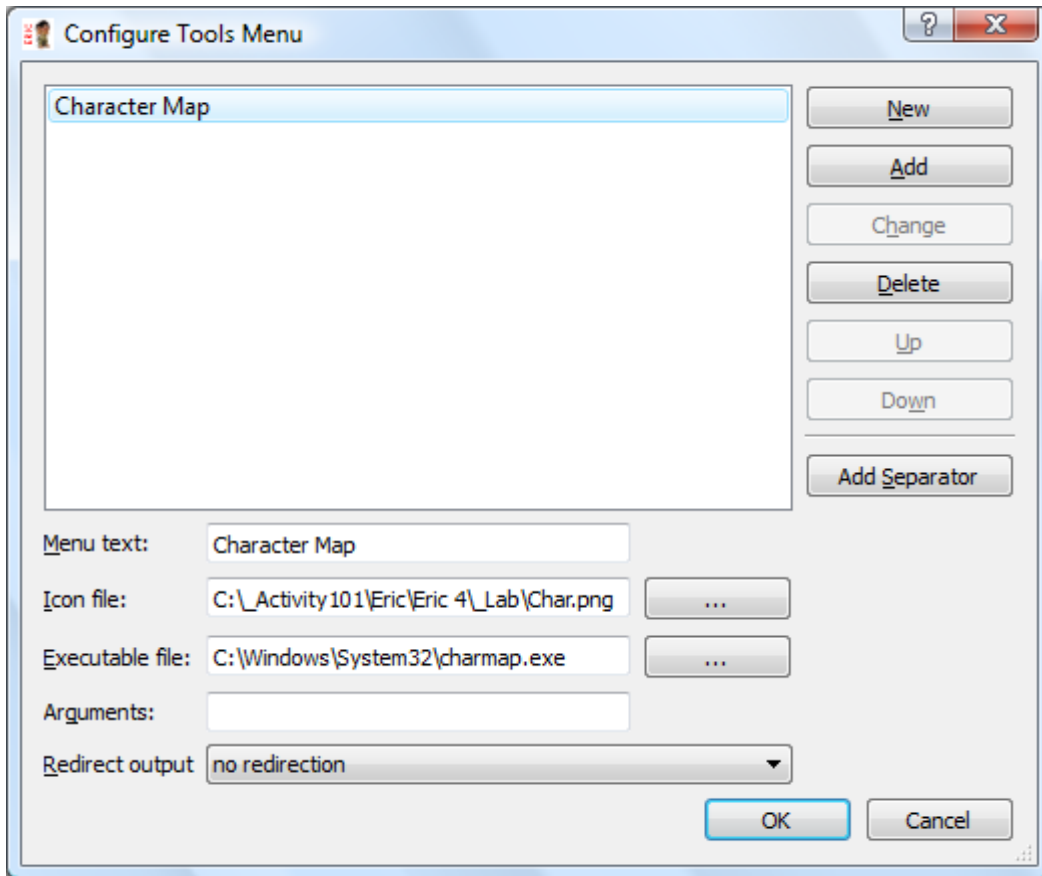


“my Tool Group” here seen in figure is just an example (see also: menu command Extras > Tools > Configure Current Tool Group...).

- -

## Extras > Tools > **Configure Current Tool Group...**

To show a control window where to configure the currently selected custom tool group (see: Extras > Tools > Select Tool Group, with the example “my Tool Group” in bold). That is, primarily, to define which is the external “Executable file” to call, and how (see figure).



### Remarks

- Standard “Builtin” and “Plugin” tool groups are here not configurable.

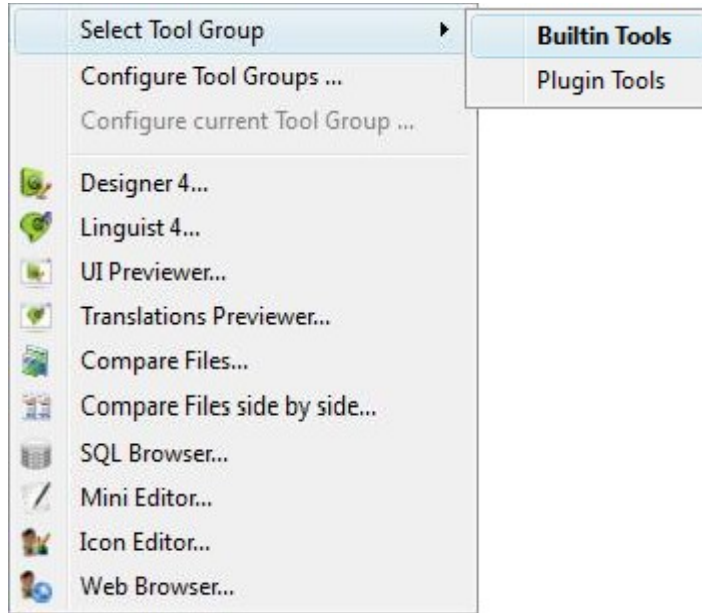
### Viewpoints

- Here, as an example, we've chosen to call the `charmap.exe` system tool (see), both for its simplicity and also because it may be handy to overcome this tiny Eric flaw:  
The entering of a special character, that is a character that cannot be directly found on your keyboard, such as: Alt 126, for: “~”, doesn't work well on the Eric text edit form.
- This command should be re-typed with no blank before the “Group . . .” ellipsis.

--

Extras > Tools > Select Tool Group > **Builtin Tools**

To show the list of extra tools made available by selecting the standard Built-in Tools (see: Extras > Tools > Select Tool Group).



The set of extra tools here in figure are what we have found as automatically assigned to such Built-in Tools group (see also: menu command Extras > Tools > Select Tool Group).

--

Extras > Tools > **Designer 4...**

<?> <...>

--

Extras > Tools > **Linguist 4...**

<?> <...>

--

Extras > Tools > **UI Previewer...**

<?> <...>

--

Extras > Tools > **Translations Previewer...**

<?> <...>

--

Extras > Tools > **Compare Files...**

<?> <...>

--

Extras > Tools > **Compare Files Side by Side...**

<?> <...>

--

Extras > Tools > **SQL Browser...**

<?> <...>

--

Extras > Tools > **Mini Editor...**

<?> <...>

--

Extras > Tools > **Icon Editor...**

<?> <...>

--

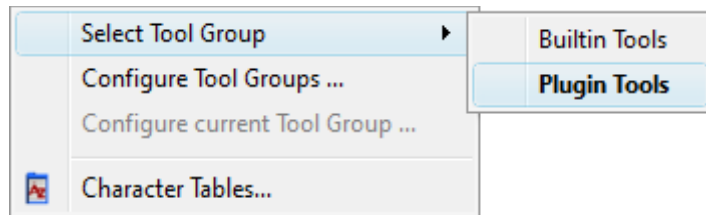
Extras > Tools > **Web Browser...**

<?> <...>

--

### Extras > Tools > Select Tool Group > **Plugin Tools**

To show the list of extra tools made available by selecting the standard Plugin Tools (see: Extras > Tools > Select Tool Group).

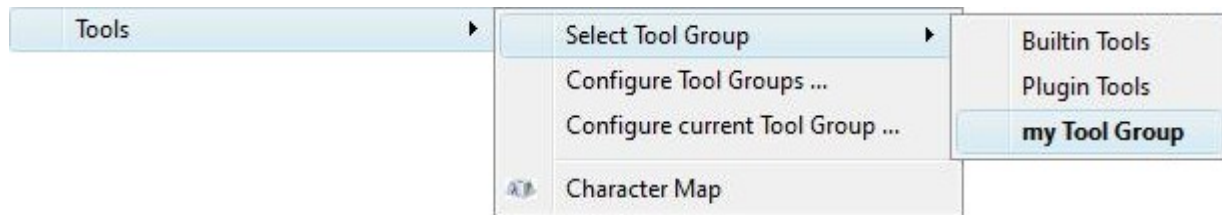


The “Character Tables...” extra tool here in figure is just an example of a tool that we have manually assigned to such Plugin Tools group (see also: menu command Plugins > Install Plugins...).

--

### Extras > Tools > Select Tool Group > **my Tool Group**

To show the list of extra tools made available by selecting the custom “my Tool Group”, here created just as an example (see: Extras > Tools > Select Tool Group).



The “Character Map” extra tool here in figure is just an example of a tool assigned to such custom tool group (see: Extras > Tools > Configure Current Tool Group...).

--

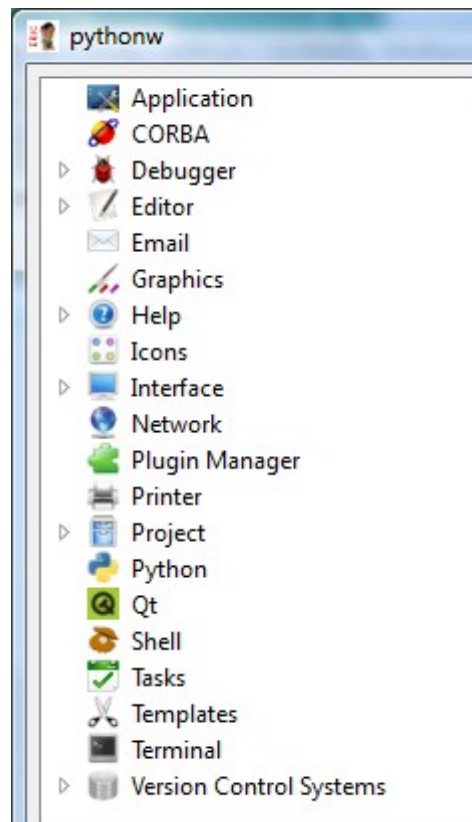
### Menu Command: **Settings**

- Preferences...
- Export Preferences... Import Preferences...
- Reload APIs
- View Profiles...
- Toolbars...
- Keyboard Shortcuts... Export Keyboard Shortcuts... Import Keyboard Shortcuts...
- Show external tools

--

### Settings > **Preferences...**

To show a “pythonw” window aimed at managing the comprehensive set of Eric operative preferences.



### Viewpoints

- We assume that most items in this `Preferences` tree are enough self-explanatory so that, instead of a dedicated, huge and impractical documentation, we'd rather offer specific description where needed, on a case-by-case basis.  
     <?>[AsFor: Questionable] Different opinion are welcome.
- For such rich `Preferences` tree, instead of a persistent full-expanded initial status, as it is now, we'd suggest a memorizing expand/collapse mechanism, so to permit the user to conveniently concentrate his attention only where more useful.

- -

### Settings > **Export Preferences...**

### Settings > **Import Preferences...**

To export/import a file containing all Eric preference settings.

### Viewpoints

- Oddly enough, no default file name nor extension is offered, only this crowded default directory:  
     `\Python2x\Lib\site-packages\eric4`  
     Then, giving a look at a preference file just created, it appears to be a traditional Windows “\*.ini”-like file. Therefore we suggest at least to assign to it a convenient extension `Notepad-compatible`.
- A preference file appears to be organized into sections and keywords, rather easy to recognize and inspect as Eric preferences.

- -

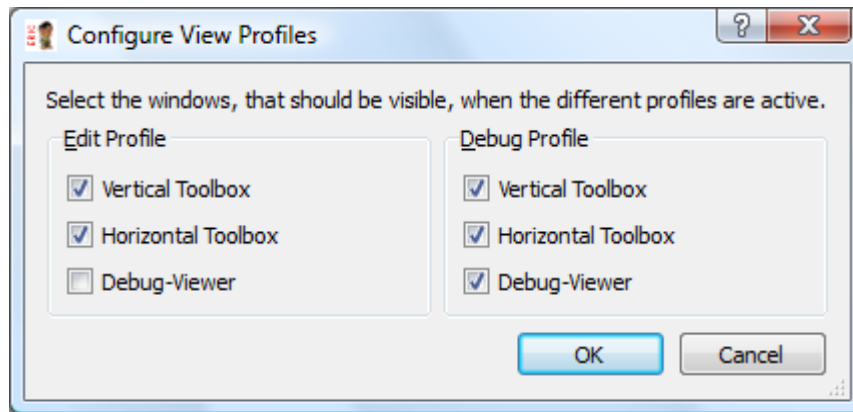
### Settings > **Reload APIs**

<?>[AsFor: *Hic sunt leones*] Unexplored territory where, for lack of info, we don't even dare to venture. The info here missed is not of generic nature, but that specifically related to this very command or parameter.

- -

## Settings > **View Profiles...**

To show a control window where to check which one of the available view forms are to be displayed on the main Eric window, when editing or when debugging a source text module. That is when the “Edit Profile” or the “Debug Profile” is active (see also: menu Window).



Available view forms, and contents:

Vertical Toolbox

for Project-Viewer, Multiproject-Viewer, Template-Viewer

Horizontal Toolbox

for the interactive shell form

Debug-Viewer and Threads

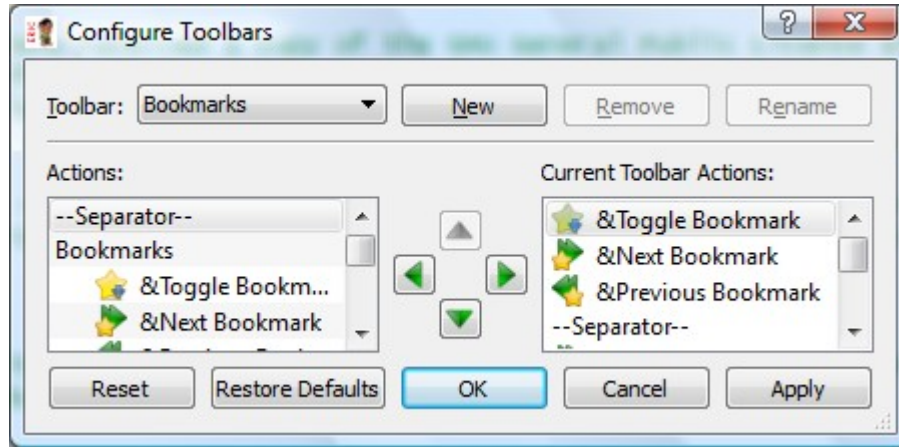
### Viewpoints

- We are not sure whether “Profile” is here an appropriate term.

- -

## Settings > **Toolbars...**

To show a rich and friendly configurator to personalize the tool-bar located just under the menu-bar, so to possibly modify the default setting and adapt it to specific user's needs and preferences.



--

## Settings > **Keyboard Shortcuts...**

### Settings > **Export Keyboard Shortcuts...**

### Settings > **Import Keyboard Shortcuts...**

To activate a configurator to possibly personalize the keyboard shortcuts associated by default to menu commands. And possibly save/restore the related keyboard shortcut configuration file.

#### *Remarks*

- A specific XML configuration file, with extension “\*.e4k”, or “\*.e4kz”.

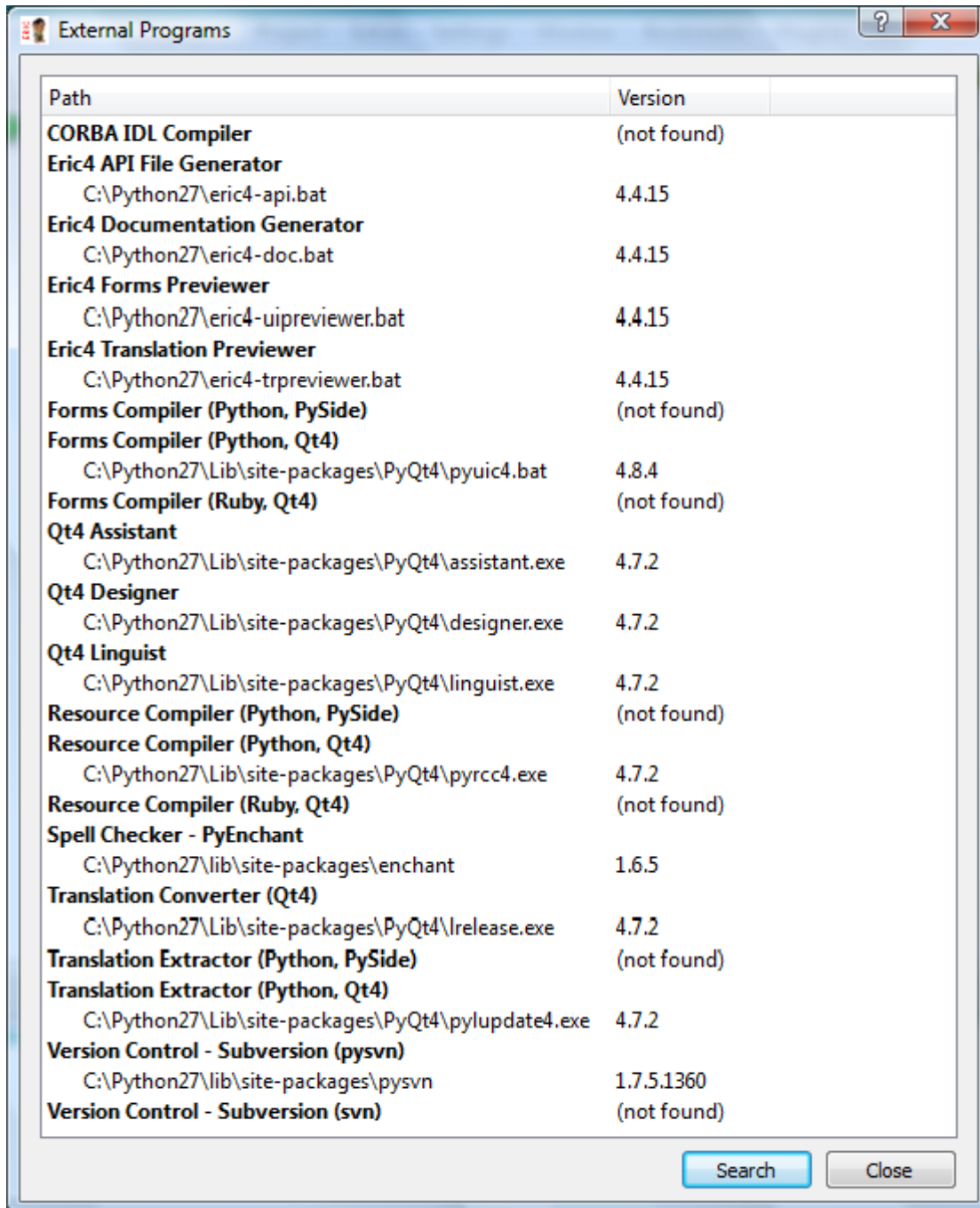
#### *Viewpoints*

- Frankly we'd like to be more convinced about the validity of such configuration mechanism, that we fear could reveal more dangerous than useful.

--

## Settings > Show External Tools

To show the whole set of programs and tools that Eric is designed to “Integrate” into one single “Developing Environment”, with specified actual parameters of path, presence and version.



### Remarks

- For a distinction between mandatory and optional items see chapter 5. **Setup and General Management**, and also: Help > Show Versions.

### Viewpoints

- Anyway it's an impressive list, that we think will deserve an detailed analysis and description.

- -

**Menu Command: Window**

Edit Profile	Debug Profile	
Vertical Toolbox	Horizontal Toolbox	Debug-Viewer
Toolbars		

- -

**Window > Edit Profile****Window > Debug Profile**

To switch between two modes of working on the current source module: as *Edit* or *Debug* mode. Different display forms will consequently be shown (see: menu `Settings > View Profiles...`).

*Viewpoints*

- We are not sure whether “Profile” is here an appropriate term.

- -

**Window > Vertical Toolbox****Window > Horizontal Toolbox****Window > Debug-Viewer**

To hide/show the referred view forms (see also: menu command `Settings > View Profiles...`):

Vertical Toolbox

for Project-Viewer, Multiproject-Viewer, Template-Viewer

Horizontal Toolbox

for the interactive form

Debug-Viewer and Threads (see also: section `Debug-Viewer`, on chapter 3. `Menu Complements`)

- -

**Window > Toolbars**

To set the hide/show mode for each and all tools (`Bookmarks, ..., View`) on the tool-bar.

- -

## Menu Command: **Bookmarks**

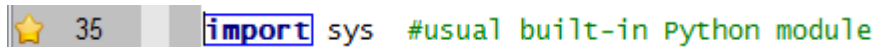
Toggle Bookmark  
 Next Bookmark            Previous Bookmark  
 Clear Bookmarks  
 Bookmarks  
 Goto Syntax Error  
 Clear Syntax Errors  
 Next uncovered line    Previous uncovered line  
 Next Task                Previous Task

To manage the bookmarks that can be set on the source modules.

- -

### Bookmarks > **Toggle Bookmark**

To set/unset a bookmark on the current pending line of the active source module. Bookmarked lines are marked with a corresponding yellow star icon on the ruler bar, on the left margin.



A mouse click in correspondence with such “star”-icon is equivalent to a `Toggle Bookmark` command.

#### *Remarks*

- Usefulness of this bookmark mechanism is severely limited by the fact that all bookmarks possibly defined during an Eric work-session are not saved at program exit.

- -

### Bookmarks > **Next Bookmark**

### Bookmarks > **Previous Bookmark**

To jump to the next/previous bookmark on the current module (only), in wrap-around mode (see also: menu command `Bookmarks > Bookmarks`).

- -

## Bookmarks > **Clear Bookmarks**

To clear all bookmarks currently defined.

### Remarks

- This action happens anyhow at program exit.

--

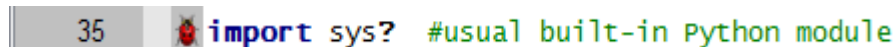
## Bookmarks > **Bookmarks**

To list all and possibly jump to one of the bookmarks possibly defined in any module during current Eric work-session (see also: menu command `Bookmarks > Next / Previous Bookmark`).

--

## Bookmarks > **Goto Syntax Error**

To jump to the next possible syntax error initially diagnosed on the current source module (only). This command found, as initially, “dim”-disabled, implies that no syntax error has been found on the current module.



```
35 import sys? #usual built-in python module
```

Offending lines are “bug”-marked on the vertical ruler bar, on the left margin.

### Remarks

- Usefulness of this—otherwise very interesting—command is severely limited by the fact that it works only at the work-session opening, before any possible subsequent source text modification (even at a `#comment`), after which it becomes anyhow disabled.
- A possible work-around for such functional limitation is to close and re-open current source work-session, so to enjoy initial condition again.
- But, alas, the re-opening of a project doesn't implies the re-opening of the same modules opened during last work-session, as a consequence possible “bugs” there located are not immediately brought to user's attention.

--

## Bookmarks > Clear Syntax Errors

To clear all possible syntax error marks initially detected, and so disabling the `Goto Syntax Error` command (see) for the rest of the current session.

--

## Bookmarks > Next Uncovered Line

## Bookmarks > Previous Uncovered Line

<?>[AsFor: *Hic sunt leones*] Unexplored territory where, for lack of info, we don't even dare to venture. The info here missed is not of generic nature, but that specifically related to this very command or parameter.

Always seen disabled.

--

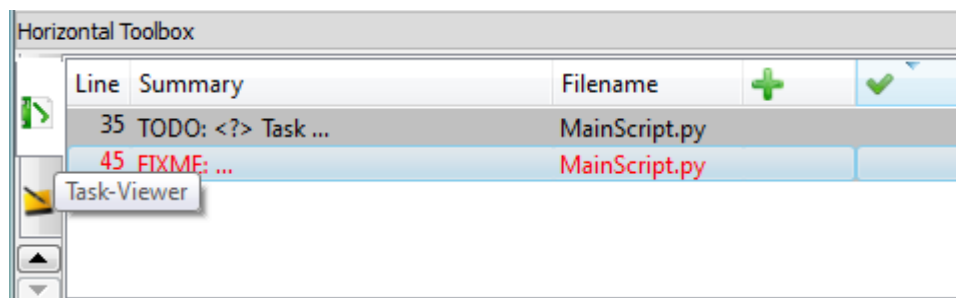
## Bookmarks > Next Task

## Bookmarks > Previous Task

To jump to the next/previous line on the current source module marked with the usual task comments “`TODO:`” and “`FIXME:`”. Such lines are detected when source module are saved or opened, and also signaled with such an icon on the vertical ruler bar:



Such tasks can also be inspected and also manually created in the “Task-Viewer” of the `Horizontal Toolbox` (see), where they are conveniently kept available to the user, also after source file closure.



**Remarks**

- Only markers typed exactly as in the related `Settings > Preferences... > Tasks > Configure Tasks` (see) will be detected.
- In case of Eric projects, such tasks are recorded into a dedicated “\*.e4t” file, located into the project management directory `_eric4project` (see section: **Eric Related Directory**, in **Appendix**).

- -

## Menu Command: **Plugins**

Plugin Infos...

Install Plugins...          Uninstall Plugin...

Plugin Repository...

Configure...

Plugin mechanism is provided as a flexible way to extend and complement Eric functionality.

### Remarks

- <~>[AsFor: Off scope] Topic just hinted at, being off the main scope of this Report.  
Test and documentation of each single Plugin tool is out of the main scope of this Report.

- -

## Plugins > **Plugin Infos...**

To show the “Loaded Plugins” comprehensive list and status window.

Loaded Plugins					
Double-Click an entry to show detailed info. Plugins with an error are shown in red.					
Module	Name	Vers	Auto	Act	Description
PluginAbout	About Plugin	4.4.0	Yes	Yes	Show the About dialogs.
PluginCharTables	CharTables Plugin	4.1.4	Yes	Yes	Dialog showing various character tables.
PluginEricapi	Ericapi Plugin	4.4.0	Yes	Yes	Show the Ericapi dialogs.
PluginEricdoc	Ericdoc Plugin	4.4.0	Yes	Yes	Show the Ericdoc dialogs.
PluginSyntaxChecker	Syntax Checker Plugin	4.4.0	Yes	Yes	Show the Syntax Checker dialog.
PluginTabnanny	Tabnanny Plugin	4.4.0	Yes	Yes	Show the Tabnanny dialog.
PluginVcsPySvn	PySvn Plugin	4.4.0	No	No	Implements the PySvn version control interface.
PluginVcsSubversion	Subversion Plugin	4.4.0	No	No	Implements the Subversion version control interface.
PluginVmListspace	Listspace Plugin	4.4.0	No	No	Implements the Listspace view manager.
PluginVmMdiArea	Workspace Plugin	4.4.0	No	No	Implements the MDI Area view manager.
PluginVmTabview	Tabview Plugin	4.4.0	No	Yes	Implements the Tabview view manager.
PluginVmWorkspace	Workspace Plugin	4.4.0	No	No	Implements the Workspace view manager.
PluginWizardPyRegExp	Python re Wizard Plugin	4.4.0	Yes	Yes	Show the Python re wizard.
PluginWizardQColorDialog	QColorDialog Wizard Plugin	4.4.0	Yes	Yes	Show the QColorDialog wizard.
PluginWizardQFileDialog	QFileDialog Wizard Plugin	4.4.0	Yes	Yes	Show the QFileDialog wizard.
PluginWizardQFontDialog	QFontDialog Wizard Plugin	4.4.0	Yes	Yes	Show the QFontDialog wizard.
PluginWizardQInputDialog	QInputDialog Wizard Plugin	4.4.0	Yes	Yes	Show the QInputDialog wizard.
PluginWizardQMessageBox	QMessageBox Wizard Plugin	4.4.0	Yes	Yes	Show the QMessageBox wizard.
PluginWizardQRegExp	QRegExp Wizard Plugin	4.4.0	Yes	Yes	Show the QRegExp wizard.

### Remarks

- Last seven “Wizard” Plugins in this list clearly correspond to the items in the Extras > Wizards sub-menu (see).

### Viewpoints

- All items here listed are present as the result of some automatic setup action, but for the “PluginCharTables” tool, that we've used to test the Plugins > Install Plugins... mechanism (see).
- So far, but for the “Wizard” Plugins, for all others in this list we haven't been able to find out where on Eric they could be seen and used.

- -

**Plugins > Install Plugins...****Plugins > Uninstall Plugin...**

To show control boxes where to `Install / Uninstall` plugins downloaded from the Eric Web Repository (see menu command: `Plugins > Plugin Repository...`).

Installed Plugins will appear available as new commands on the `Extras > Tools` sub-menu (see).

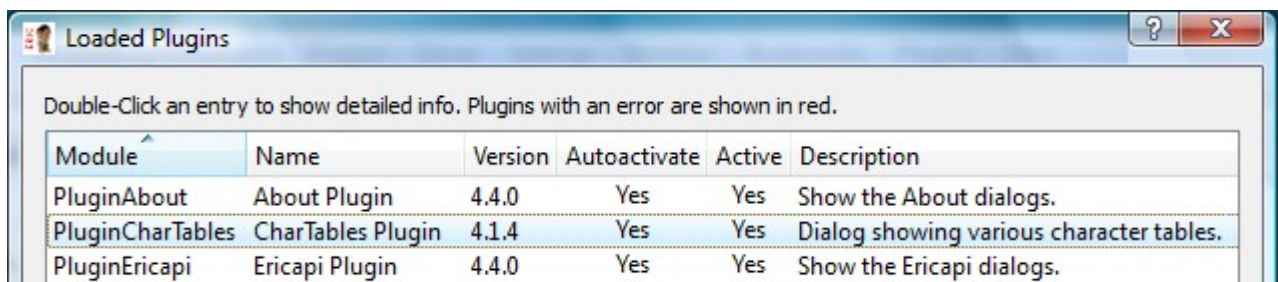
**Remarks**

- `<~>`[AsFor: Off scope] Topic just hinted at, being off the main scope of this Report.

Plugin tools, being independent from Eric, should also be independently tested and documented. As a consequence, here they are considered out of the main scope of this work and, therefore, not treated.

**Viewpoints**

- We have here tested this `Install` command with a “Tools, CharTables” plugin, whose “\*.zip” archive was previously download-ed into the directory “\\_eric\Downloads” (see).
  - Asked to select a destination between a “User” and a “Global directory”, we chose “User”, without actually knowing what's the difference.
  - After action `Install`, the named plugin could be spotted ok on the “Plugins > Plugin Infos...” list, where a right click will get a pop-up menu to: `Show Details`, `Activate`, `Deactivate` (see).

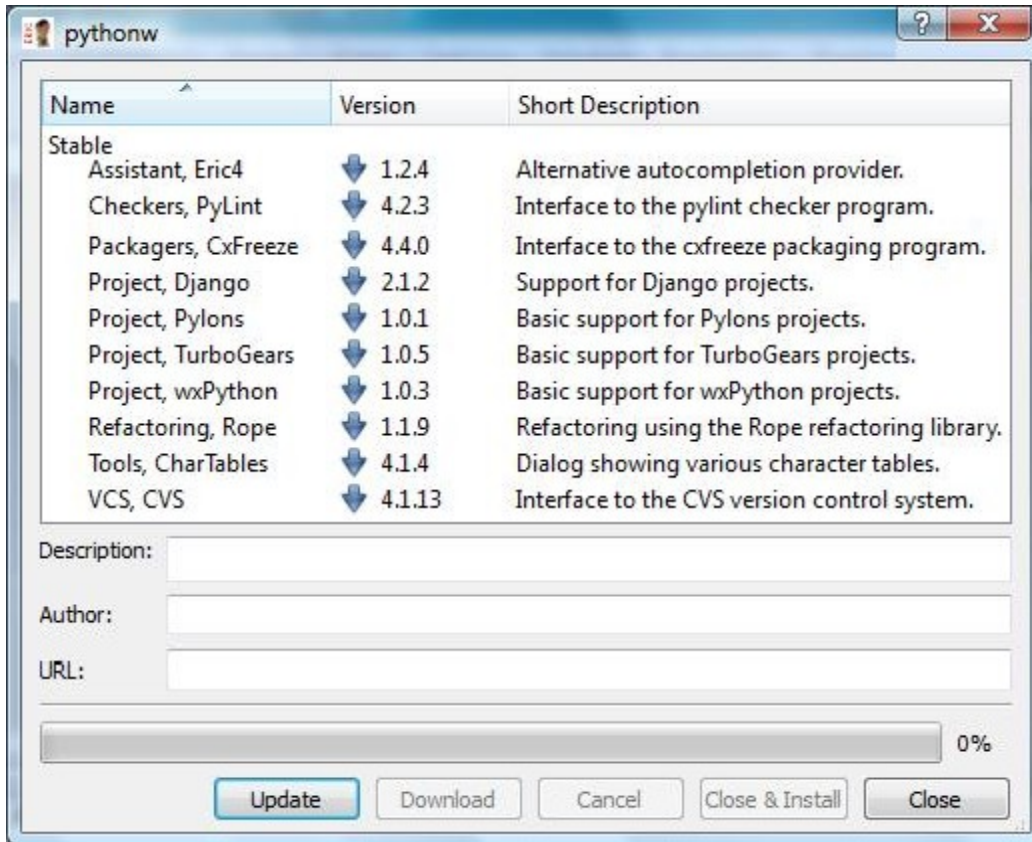


- Then a new “Character Tables...” command appeared at bottom of the `Extras > Tools`, `Plugin Tools` sub-menu, ready to be used (see).

- -

## Plugins > **Plugin Repository...**

To inspect, and possibly “Update”, the list of Eric plugins currently available on the dedicated Web repository.



Downward-arrow symbol on the list is to tell that the item is available for “Download”, arrow missing is for item already down-loaded. Click on an item to get related info.

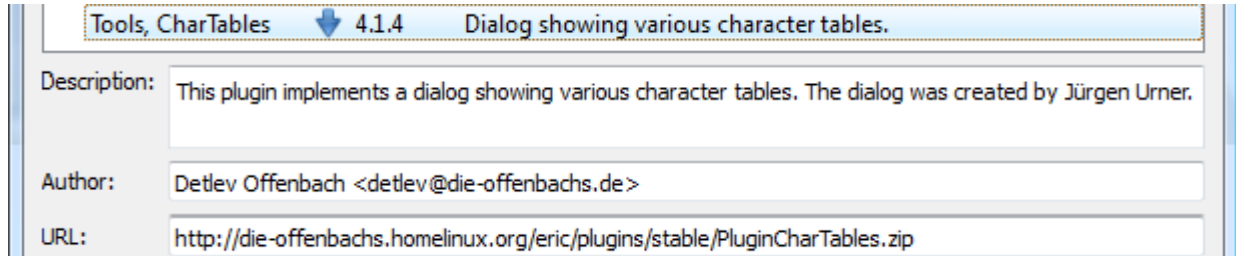
### Remarks

- This is the current Repository's URL:  
`http://die-offenbachs.homelinux.org/eric/plugins/repository.xml`
- Top-left label “Stable” (see) is here presumably to characterize a sub-Repository container, as opposed to “under development”.

### Viewpoints

- To better test and understand this whole mechanism we made use of the specific item: “Tools,

CharTables”, for which here we have:

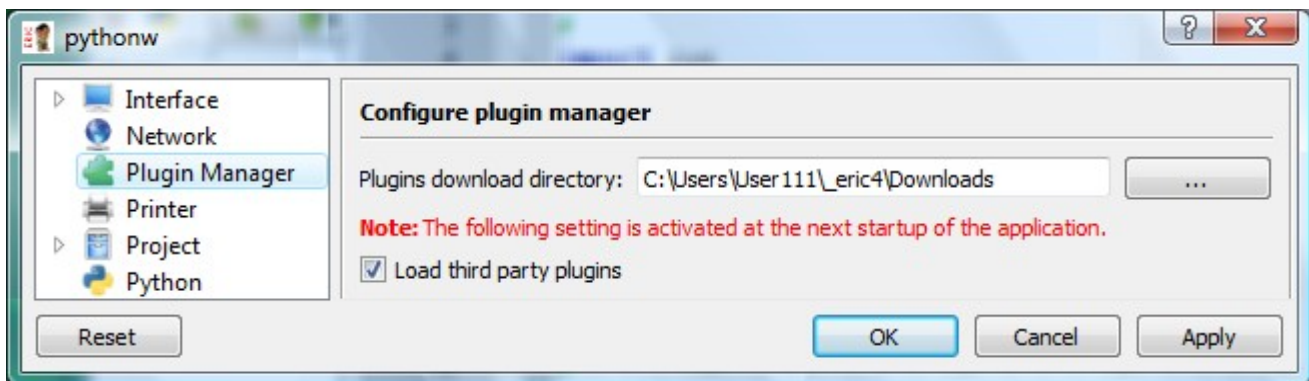


and on which we experimented Download, Infos, Install, Uninstall commands (see).

- -

## Plugins > **Configure...**

To show the same window as with command “Settings > Preferences...” (see), opened directly on the “Plugin Manager” section.



Where you can manage the destination directory where to possibly down-load the Eric plugins, as for menu command Plugins > Plugin Repository... > Update (see).

## Viewpoints

- We'd like to be better informed about check box “Load third party plugins”, whether “Load” actually stands for “Download”, and how to tell the difference between “third party” and not-“third party plugins”.

- -

## Menu Command: **Help**

- Helpviewer...
- Eric API Documentation
- Python Documentation
- Qt4 Documentation
- PyQt4 Documentation
- About eric4
- About Qt
- Show Versions
- Check for Updates...
- Show downloadable versions...
- Report Bug...
- Request Feature...
- What's This?

Central Eric's feature is that of offering a single, integrated environment for a comprehensive set of tools aimed at the development of Python projects. Central to this general feature is that of offering all technical information necessary to the developer here conveniently centralized into a single menu of commands.

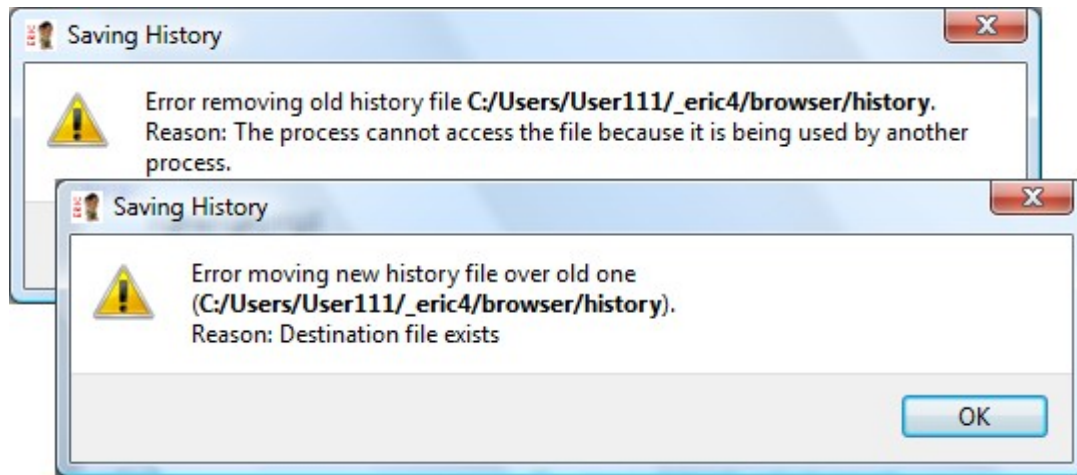
--

### Help > **Helpviewer...**

To show the dedicated viewer aimed at browsing all Eric help material. It's basically a full-fledged Web Browser, even if not comparable with the well known major browsers currently in use, and normally automatically used when inspecting the Help material here available (see also: menu command `Settings > Preferences...`, `Help Viewers > Eric Web Browser`).

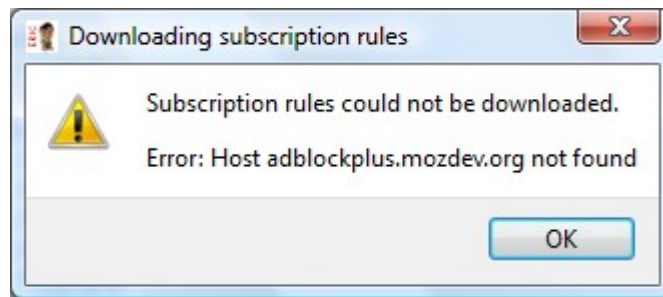
### *Viewpoints*

- At a certain point, presumably as a consequence of some Eric update fiddling, calling this command we got the following error messages:



Situation that we fixed simply getting rid of that “history” file. An experience here recalled just in case it would happen again...

- And this is another error message box that, to our surprise, we happened to get calling this command.



Then, to our further bewilderment, we discovered that:

- Once OK-accepted this message, calling again this menu command during the same Eric session, did not arise any more.
- Whereas, at a new Eric session, this same error condition occurred again.
- Then, assuming that an active Internet connection was here unexpectedly required, we got on-line, re-called this same command, and realized that no such error box appeared any more; not even at the following sessions.  
That's presumably a behavior caused by an Eric initial setup process not completed, sort of. And, to our opinion, to be fixed.
- The real point here is that Eric takes unexpected, unauthorized initiatives, like that of trying to go on-line without being so asked to. About this subject see also hereafter the *Viewpoints* at section 2/2] PyQt, chapter 5. Setup and General Management.

- -

## Help > Eric API Documentation

<?>[AsFor: Perplexity] Here we'd like to know what's exactly the spirit and purpose of this command, about which, so far, we are unfortunately able only to express some *Remarks*.

### Remarks

- Such Application Programming Interfaces (APIs) documentation, presumably related to the very Eric development, would be of interest only for an Eric's development team, not for the generality of its users, who here would obviously appreciate much more a documentation of use, rather than of development; wouldn't they?

### Viewpoints

- Instead of this API documentation—which to us, ordinary users, is totally useless, if not even confusing—we'd suggest to conveniently display the collection of all pop-up hints, as for menu command Help > What's This?, that obviously would be here much more appropriate and useful.

- -

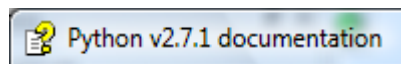
## Help > Python Documentation

To gain access to the usual Python documentation, directly within Eric.

That is, provided that the appropriated path, such as: C:\Python27\Doc\python271.chm had been given, with the menu command: Settings > Preferences..., Help > Help Documentation > Python Documentation (see also: [Eric Setup Completion](#)).

### Remarks

- Here we had somehow to force the actual “pythonxxx.chm” file name, being Eric defaulted to such a nonexistent “python2x.chm”, and with the related “[...]” search button of little use, being limited to searching directories, not files.
- Anyhow we notice also that here we got the usual Python help display:



That is, NOT the aforementioned Eric Web Browser, here again to our mild confusion.

- -

## Help > Qt4 Documentation

<?>[AsFor: Missing info/Perplexity] No idea what it is the documentation here intended.

### Viewpoints

- Here we haven't been that lucky as with the Python documentation (see). Indeed we haven't got a clue about which is, and where to find such “Qt4 Documentation”, as here intended. The underlying question being more serious than it may appear at first. Indeed:
  - <?> Why also a Qt4 documentation, when the corresponding package here in use is the PyQt4? What does it really mean, that the PyQt4 documentation is insufficient, or unpractical?
  - <?> In other words, would such other possible “Qt4 Documentation” be indispensable, or simply useful?

- -

## Help > PyQt4 Documentation

To gain access to the usual PyQt4 documentation, directly within Eric. More precisely, to open the Eric Web Browser on the “PyQt Reference Guide”. Where, in the Introduction page (see), you'll find also a useful link to the [PyQt Class Reference](#)<sup>29</sup>.

That is, provided the appropriated configuration has been performed as with next *Remarks* (see also: [Eric Setup Completion](#)).

### Remarks

- Relying upon the existence of a PyQt4 toolkit installed, the configuration here required is of this kind:
  - kind: C:\Python27\Lib\site-packages\PyQt4\doc\html\
  - set on: Settings > Preferences..., Help Documentation > PyQt4 Documentation

That is, this time with a directory path, not a file path as with the Python Documentation (see). Otherwise you'll get a: “The PyQt4 documentation starting point has not been configured” error message.

### Viewpoints

- <?> By the way, on this “Configure help documentation” form we read such a “Note: Leave empty to use the PYTHONDOCDIR environment variable, if set.”
  - Well, we have no idea what such a PYTHONDOCDIR parameter, or then the other QT4DOCDIR, or PYQT4DOCDIR, ... are. Nor who is supposed to manage each and all of them. Of course we can guess, but we'd better be informed.

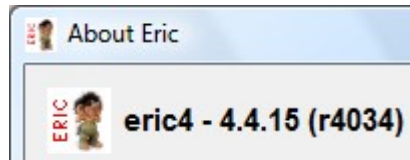
- -

---

<sup>29</sup> Currently an un-searchable documentation, therefore not so handy to use. We've been promised substantial improvements soon.

## Help > **About eric4**

To display this very product's identity card, comprising: official denomination, revision code, authors, acknowledgments and License (GNU – General Public License).



Plus two most appreciable e-mail references:

Please send bug reports to [eric4-bugs@eric-ide.python-projects.org](mailto:eric4-bugs@eric-ide.python-projects.org).

To request a new feature please send an email to [eric4-featurerequest@eric-ide.python-projects.org](mailto:eric4-featurerequest@eric-ide.python-projects.org).

Where users are invited to express their problems and wishes, to the Eric team.

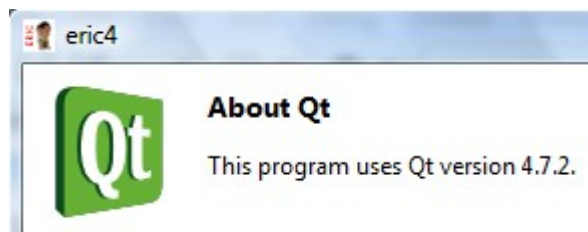
### *Viewpoints*

- Here, in a few nearby lines, we read: **About eric4, Eric API, ERIC, About Eric, eric4**  
Well, we dare suggesting to decide, once and for all, precisely how this kind of names should be written. Personally we are in favor of: **Eric**, with initial uppercase and final rev. number only when it's appropriate to be so specific. Anyhow we'd be happy to conform to any other style, provided coherently adopted.

- -

## Help > **About Qt**

To display a copyright acknowledgment box for such Qt GUI toolkit, as here utilized.



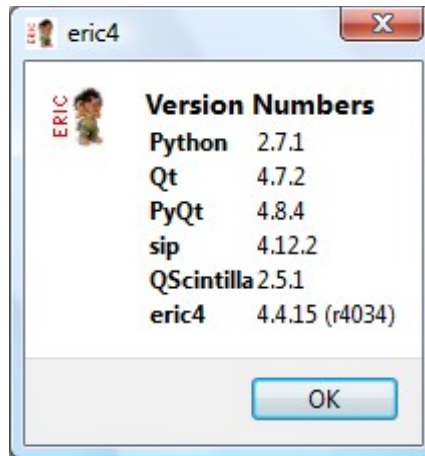
### *Viewpoints*

- More precisely, this product is here utilized through its PyQt Python binding (see section: Prerequisites).

- -

## Help > Show Versions

As already noted, the central Eric's feature is that of offering a single, integrated environment for the development of Python projects.



This menu command is to show a box listing all the toolkits currently integrated into this unique environment, along with their precise brand names and revision codes<sup>30</sup>.

- -

## Help > Check for Updates...

To activate an on-line comparison between the local version of the Eric product in use vs. the latest stable version (that is: not under development) currently available for down-loading, so to check whether there is any updating opportunity.



For more information about all available versions see: [Help > Show Downloadable Versions...](#)

<sup>30</sup> Impressive list. See also: “Prerequisites”, at chapter 5. Setup and General Management.

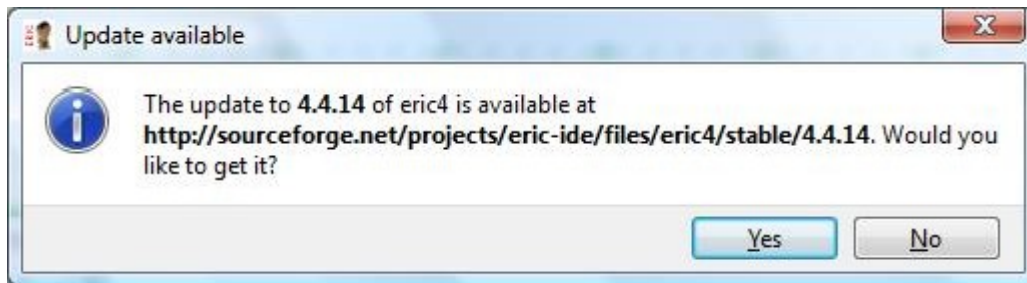
### Remarks

- The versions here considered are all relative to the same Eric 4 family, related to Python ver. 2.x, not to the Eric 5 which, being related to Python ver. 3.x, is altogether considered as a different line of product.
- By the way, we can testify that these two Eric 4 and 5 lines are compatible, in the sense that they can coexist within the same computer system<sup>31</sup>.

### Viewpoints

- Having heard that a new Eric 4.4.14 revision was available, we seized the occasion to test-verify this “Help > Check for Updates...” command. Here a report of that experience.

First we got this info-box:



Where we “Yes”-accepted the offered automatism, with these results:

- No choice about whereto, nor what, to download;
- No indication of what was actually going on, if anything. Nothing.

Therefore we decided to renounce, and follow the old way as in **Instant Reference** section (see). So that we went to the URL: <http://eric-ide.python-projects.org/index.html>

And then we followed the “eric Downloads” link, where we spotted a familiar “eric4-4.4.14.zip”, which we downloaded and happily used as described in the chapter 5. **Setup and General Management**, section **Update** (see).

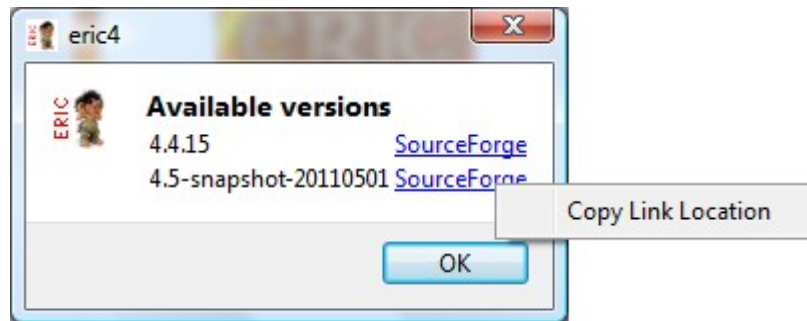
- -

---

<sup>31</sup> Well, a little incompatibility have we suffered. Once installed Eric 5, the `eric4.pyw` “no-console” run command doesn’t work any more, and we have to turn back to the primordial `eric4.bat` run command. That’s because of a known Windows issue, where a “\*.pyw” call works only for one Python version installed, the last.

## Help > Show Downloadable Versions...

To display some more info—besides what offered by Help > Check for Updates... (see)—about the possibly different Eric versions currently available for downloading.



Of the two links in display, the first one points to the so defined “stable” version, the second to the “unstable”, or “snapshot”; that is: the latest version currently under development.

### Remarks

- An actual test of these two links led to no result, presumably because they were meant to run the Eric Web Browser, which couldn't show being blocked up by this same dialog box, which operates as a “modal” window (that is: stuck firm on the screen). Anyway, given a look at these links, they revealed as:

`http://sourceforge.net/projects/eric-ide/files/eric4/stable/4.4.15`

`http://sourceforge.net/projects/eric-ide/files/eric4/unstable/4.5-20110501`

- -

## Help > Report Bug...

## Help > Request Feature...

Two commands aimed at granting the users a channel of dialog with the producer. A highly appreciable attitude.

### Viewpoints

- A highly appreciable attitude, confirmed also by the e-references of dialog offered in a even simpler<sup>32</sup> and direct way with the About eric4 command (see).

- -

---

<sup>32</sup> Simpler as it doesn't require such Configure Email settings as the “Outgoing mail server (SMTP)” and “Outgoing mail server port” into the Preferences Dialog window.

## Help > What's This?

To get a hint pop-up help, just then pointing at a given control item on the Eric window.

### Remarks

- Here the function key shortcut “Shift+F1” reveals particularly useful.

### Viewpoints

- Well, not always “particularly useful”, as in this real-case. Suppose we have no idea what a `Unittest` command is, therefore we ask: “Help > What's This?”



As a result we get a hint telling us that clicking the `Unittest...` command we “*Perform unit tests*”. Rather useful, isn't it? And, please, don't tell me that a Mr. User should already know what it is, 'cause it's not fair. Indeed, it's precisely when you don't know that it is particularly useful to ask, isn't it?

What I mean, for instance, is that in such a case it would suffice to clarify something like: “*the standard Python unittest module*”, to be of real help to the poor Mr. User who, this way, would be pointed to the right direction where to go, if he wished to.

- = -

## 3. Menu Complements

*S-PM 110600*

<...>[AsFor: Unfinished] Reference section to zoom on the special menu command features exceeding the scope of former chapter 2. Menu Commands (see), because of their nature, size or relevance.

- -

### Contour Forms:

#### Vertical / Horizontal Toolbox, and Debug-Viewer

Central, in all senses, to Eric workspace, lays the form where source text is edited. This form is usually surrounded (see menu command `Window`) by these three supporting forms, docked around:

Vertical Toolbox, Horizontal Toolbox, and Debug-Viewer; describe hereafter in some detail.

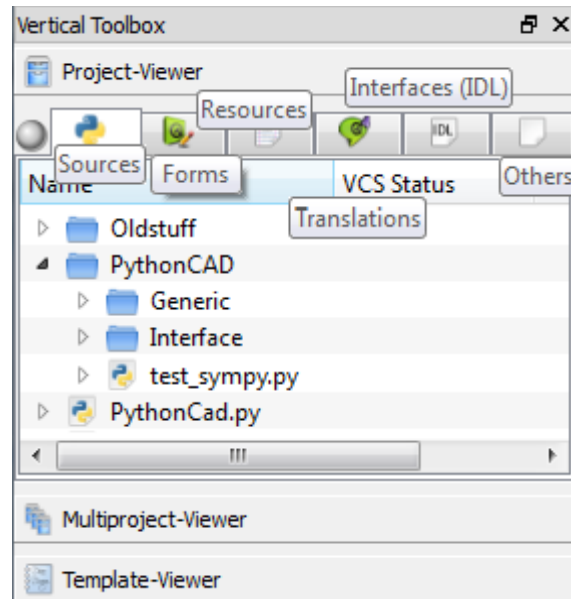
Then, just for completeness' sake, the other contour “sidekicks” will be here cited, without any further description, assuming they are of immediate comprehension and use:

Title bar, Menu bar, Tool bar, at top; Status bar, at bottom.

- -

## Vertical Toolbox

<...>[AsFor: Insufficient info] Left-docked form (hide/show command: Window > Vertical Toolbox), comprising a Project-Viewer, with its six tags (Sources, Forms, Resources, Translations, Interfaces (IDL), Others).



Plus, at the bottom of this form, other two Multiproject-Viewer and Template-Viewer. Three viewers that become enabled when a Project, Multiproject or Template<sup>33</sup> is open.

Where:

- |              |                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Sources      | File manager-like view on an Eric project (see: Project menu), with an useful context menu worth exploring (see)                                                                                                                                                                                                                                                                               |
| Forms        | File manager-like view on the graphic forms possibly belonging to the project. A powerful context menu is here provided, including the command “Open in Qt-Designer”, where to possibly create and edit such forms (see).<br><br>Description of such powerful tool is assumed as beyond the scope of this Report. References about Qt and PyQt can here be found at the Prerequisites section. |
| Resources    | <...>[AsFor: Insufficient info]                                                                                                                                                                                                                                                                                                                                                                |
| Translations | <...>[AsFor: Insufficient info]                                                                                                                                                                                                                                                                                                                                                                |

<sup>33</sup> <?> Presumably another type of project, about which, anyhow, we have no information.

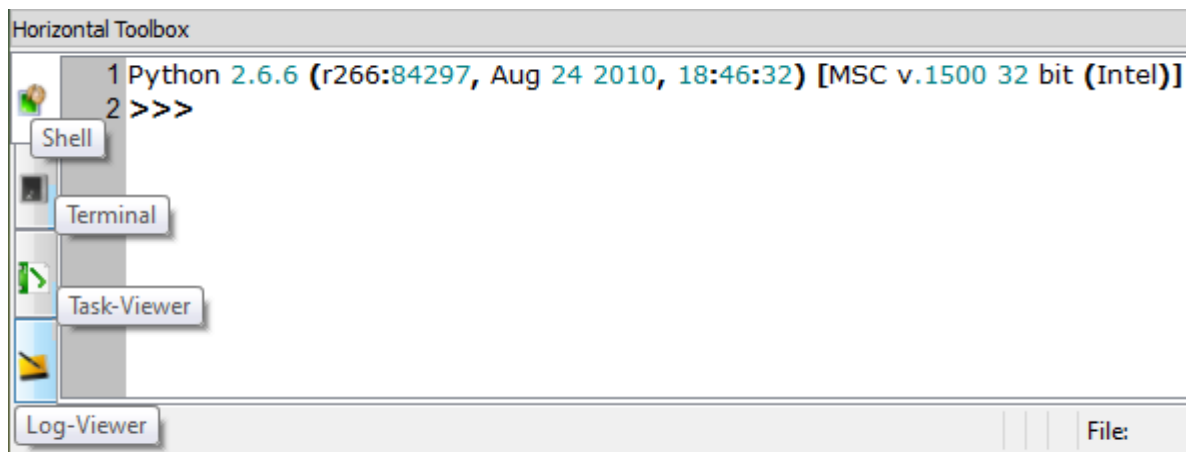
Interfaces (IDL)  
 < . . . > [AsFor: Insufficient info]

Others < . . . > [AsFor: Insufficient info]

- -

## Horizontal Toolbox

Bottom-docked form (hide/show command: Window > Horizontal Toolbox), comprising four tags (Shell, Terminal, Task-Viewer, Log-Viewer).



Where:

Shell Usual Python interactive shell

Terminal Command Prompt (DOS) system shell

Task-Viewer Where to manage “FIXME” and “TODO” tasks, as for menu command “Bookmarks > ... Task” (see). With a right-click context menu that offers a comprehensive set of related commands (see).

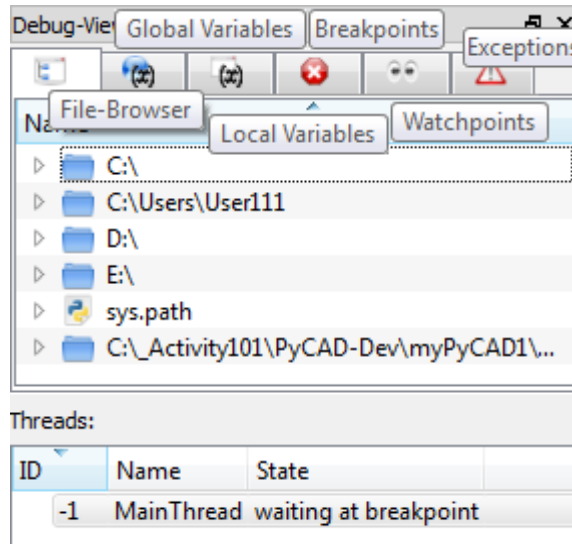
Log-Viewer < . . . > [AsFor: Insufficient info] No idea what's about ...

- -

## Debug-Viewer

< . . . > [AsFor: Unfinished] Right-docked form (hide/show command: Window > Debug-Viewer) with one Debug-Viewer, comprising six tags (File-Browser, Global Variables, Local Variables,

Breakpoints, Watchpoints, Exceptions), and a Threads sub-form. A viewer that becomes fully enabled when a Debug session is under way (see).



Where:

File-Browser General purpose file manager-like tool

Global Variables

<~>[AsFor: Insufficient info] To inspect the here so called Global Variables, with no precise definition about what they are, and with the “What's This?” hint offering no help<sup>34</sup>.

Local Variables

<?>[AsFor: No info] Always found empty. No idea about its supposed role and meaning, and with the “What's This?” hint offering no help<sup>35</sup>.

Breakpoints To inspect currently defined Debug breakpoints (see)

Watchpoints <?>[AsFor: No info] No idea what it is

Exceptions List of execution Exceptions occurred (see also: Debug > Exceptions)

Threads <?>[AsFor: No info] No clear idea about what it is

34 Alas, this is all the help you get: “This windows displays the global variables of the debugged program”

35 Alas, this is all the help you get: “This windows displays the local variables of the debugged program”

- -

### Remarks

- `<!>` About the identifiers here in display, one must be well aware of the so called “mangling” mechanism used by Python to handle private members of a class, conventionally named this way: “`__<privateMember>`”.  
Indeed, within this “Debug-Viewer” form, such an identifier will be reported as: “`__<className>__<privateMember>`”. The same as with the interactive shell, and the “Debug > Evaluate...” menu command (see).

- = -

## 4. Add-Ons and Accessories

*S-PM 110400*

< . . . > [AsFor: Future developments] Just as a memorandum for the subject here in title.

- = -

## 5. Setup and General Management

*S-PM 110600*

Reference chapter for setup and general management of Eric 4 Python IDE, onto the target environment here considered, that is: Windows 32 bit, Python 2.x (see: [Version Reference](#) table, at chapter 1. Introduction & Generality).

### Prerequisites

In essence, Eric prerequisites are the very tools which Eric is meant to integrate into its unique developing environment (IDE). With the Windows platform here of reference, all these tools are contained in packages that are not immediately available within the system—as it may be with other platforms, such as Linux-based, Ubuntu—, and that must be therefore collected from different sources, as hereafter described in detail.

From the Eric's point of view it's convenient to subdivide and classify all such packages into *Required* and *Optional*:

#### *List of Required Packages*

1/2] Python 2.x Programming Language  
 2/2] PyQt4 GUI Library  
       Eric Chief application

See following sections, ordered in the exact installation sequence as required by Eric.

#### *List of Optional Packages*

PyEnchant Spell-checking library for Python

Take this one simply as an uncommitted example, being this an area substantially left to the user's free initiative (see also: [Settings > Show External Tools](#)). Some related information is then hereafter offered at the end of this chapter (see).

--

*“Mutatis mutandis”*

This is to remind the reader that the following sections constitute a valid technical and practical reference, provided that in each actual situation all what should be changed—such as actual URLs, or revision codes—be adequately changed, and updated. Indeed, saying it in Latin: *“Mutatis mutandis”*.

--

## Required Packages

What to do, in detail and in this sequence, before installing Eric.

### 1/2] Python

S-PM 110600

Programming language Python, the main toolkit here needed first.

#### Where

Available for downloading from URL: <http://www.python.org/download>

#### What

Among the various items there available, it's the ver. 2.x that is required by the Eric ver. 4 here in use (whereas the ver. 3.x is that one required by Eric ver. 5), and for Windows platform, 32 bit. That is, currently, the Microsoft Installer file: `python-2.7.1.msi`

#### How

Setup-run, with Administrator permission, all defaults accepted, and you'll get such a: “\Python2x”<sup>36</sup> directory installed (see). Then a test-Start run of the “IDLE (Python GUI)” program, to verify that it's working all right.

#### Then

It's then convenient to assume possible subsequent upgradings<sup>37</sup> as an iteration of this same *Download->Setup* sequence, as here described, always as starting from scratch. That is, a setup possibly preceded by all due un-install sequences, executed in the reverse order.

To that purpose, see the related “Uninstall Python”, available under the system Start menu; Administrator permission.

- -

---

36 Actually a “\Python27”, in our current case.

37 Indeed here we've already experienced a—not painless—upgrading from `python-2.6.6.msi`.

## 2/2] PyQt

GUI (Graphic User Interface) toolkit PyQt, a Python binding of the cross-platform Qt GUI toolkit. A valid alternative to Tkinter, the GUI programming toolkit bundled with Python.

### Where

PyQt can be found at URL:

<http://www.riverbankcomputing.co.uk/software/pyqt/download>

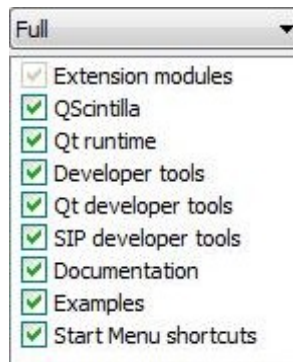
### What

Pick item: `PyQt-Py2.7-x86-gpl-4.8.4-1.exe`

As for: Python ver. 2.7, 32 bit, General Public License, ver. 4.8.4-1<sup>38</sup>. A few minutes of downloading.

### How

Setup-run, with Administrator permission, all defaults accepted, and you'll get a: “\PyQt4” into the same Python directory, assumed already installed, under the “\Lib\site-packages” (see). Hereafter the “Full” list of installed items.



After installation, on the Windows system Start menu you'll have the item:

`PyQt GPL v4.8.4 for Python v2.7 (x86)`

That's a folder rich with programs, Documentation, Examples and Links. So rich that, without a proper “primer” documentation<sup>39</sup>, it's easy to get lost. Plus, with a reassuring `Uninstall PyQt`.

This installation can be tested entering into the Python interactive shell such line of code:

```
>>> from PyQt4 import QtCore, QtGui
```

<sup>38</sup> Here too we experienced a—not painless—upgrading from a former PyQt ver. 4.8.3-1, aimed at Python 2.6.

<sup>39</sup> Which we haven't been able to find, so far. Alas.

If accepted, you're up and running.

### Then

As you'll see, the Eric menu command `Help > PyQt4 Documentation` is provided to browse the **PyQt Reference Guide**. To that aim just verify that the correct pointer:

```
C:\Python2x\Lib\site-package\PyQt4\doc\html
```

is set into the Eric menu command `Settings > Preferences...`,

```
Help Documentation > PyQt4 Documentation
```

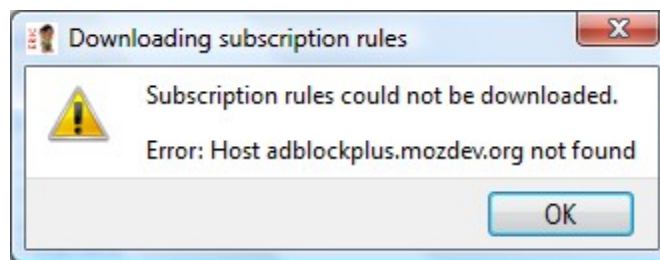
Browsing such `PyQt4 Documentation`, first you'll get the **PyQt Reference Guide** home page, where, in the **Introduction** page, you'll then find an useful link to the notable **PyQt Class Reference** home page (see).

### Remarks

- Here too, it's convenient to consider any possible subsequent upgrading as a fresh new setup, as already said with Python (see). To that purpose, see the related “`Uninstall PyQt`”, available under the system `Start menu`, `Administrator` permission.

### Viewpoints

- We have to confess that we've found such **PyQt** documentation<sup>40</sup> rather massive and un-friendly for beginners. Particularly unpractical is the search engine, a fundamental tool when you really need help. Well, here even the most common keywords are considered as unknown, so that you get sensible results only when you already know exactly what and where to search. In other words, here the search engine works when you don't need it.
- Besides, when browsing this documentation you may get, random, such an error box:



About which we'd like to express these serious perplexities:

- Here we simply asked to give a look at the local help file, therefore we don't accept that the help browser sought forcefully to go on-line.
- We'd like to be in control of such actions as whether to go on-line or not, and whether to download

---

<sup>40</sup> As author of this Report I'd be glad to discuss such an issue with PyQt people, would they care.

anything or not; “Downloading subscription rules” included<sup>41</sup>.

Do you remember the computer HAL in “A Space Odyssey”? In such a case, my attitude would be to get rid of such a computer well before it simply tried to bully me. Proactively.

- -

---

41 With the \*.org URL there found that refers to Adblock Plus (ABP), a forked version of Adblock, which is a content-filtering extension for the Mozilla Firefox and Google Chrome web browsers. That said to the benefit of whom has never heard of it.

## Eric

S-PM 110600

Here it's about the Eric toolkit, final and main subject of all this setup sequence. More precisely the Eric ver. 4, with future revision planned up to the ver. 4.5, as an IDE (Integrated Development Environment) for Python ver. 2.x; that is, currently: ver. 2.7<sup>42</sup>.

### Download

URL: <http://sourceforge.net/projects/eric-ide/files/eric4/stable/4.4.15/>

File: eric4-4.4.15.zip

### Remarks

- The set of Eric items available for downloading, such as:

```
eric4-i18n-de-4.4.13.zip
eric4-i18n-de-4.4.13.tar.gz
...
eric4-4.4.13.zip
eric4-4.4.13.tar.gz
changelog
```

can be easily interpreted, knowing that:

```
“*.zip”      is for platforms Windows
“*.tar.gz”   is for platforms Linux or Mac OS X
“i18n”       is a translation in some language, as “de” for German;
              otherwise, without such marker: default language (English)
“log”        <~>[AsFor: To be verified] Downloads history log
```

- -

---

42 Whereas the new corresponding product designed for Python ver. 3.x is Eric ver. 5.x.

## Install

Just expand the “.zip” package, then locate and run the “install.py” procedure inside<sup>43</sup>.

You'll get a directory “\eric4” into the same Python installation directory, within the “\Lib\site-packages” (see).

## Remarks

- In an actual install case-history, this was the log message displayed:



```
C:\Python27\python.exe
Checking dependencies
Python Version: 2.7.1
Found PyQt4
Found QtHelp
Found QScintilla2
Qt Version: 4.7.2
PyQt Version: 4.8.4
QScintilla Version: 2.5.1
All dependencies ok.
Compiling user interface files...
Compiling source files...
Installing eric4 ...
Installation complete.
```

Rather reassuring about the dependencies:

```
1/4] Python 2.7.1
2/4] Qt 4.7.2
3/4] PyQt 4.8.4
4/4] QScintilla 2.5.1
```

From which it can be inferred that the toolkit `PyQt4` brings along also the toolkits `Qt` and `QScintilla`, which clearly don't need to be handled individually.

--

## Uninstall

The Eric package offers a “uninstall.py” procedure. In case, you have to run it outside the very directory meant to be uninstalled, typically using the copy of this procedure contained within the original package.

---

<sup>43</sup> A “`python install.py -h`” is to display some help.

### Viewpoints

- No reassuring message is offered at uninstall procedure completion. We'd appreciate one.
- We are not sure that this uninstall procedure is really a complete eradication of what its install counterpart has done. Indeed, we vaguely remember that, at first setup, an Eric “configuration” has been performed, whereas, at all subsequent Eric setups, no such thing happened again. Probably because the original configuration had been kept and automatically used.

Certainly many good reasons can be declared for such a behavior, but we don't agree. In the sense that if it is my intention to uninstall, be uninstall. All exceptions, all what is left behind for whatever reason, either by purpose or by oversight, is pure unfair garbage, abandoned on somebody else's backyard without asking permission.

- -

### Upgrade

In case of a new Eric revision available, the process of updating is equivalent to a compound of the afore described: *Download + Uninstall + Install*. Plus, being this the last item in the install sequence, you're free to do it without affecting the preceding prerequisite installations.

### Viewpoints

- As we've actually verified, different main versions, that is: **Eric 4** and **5**, can coexist on the same system. Good news, in upgrading prospective.
- Updating is worth doing, as we've happily verified updating from **Eric 4.4.12 (r3937)** to **Eric 4.4.13 (r3989)**, then to **Eric 4.4.14 (r4014)**; and so overcoming some naughty bugs.
- We'd like to be better informed about the meaning and role of the menu command `Help > Check for Updates...`, and on the automatic `Settings > Preferences > Application > Configure the application > Check for updates` (see), particularly about the following points.
  - About what kind and what level of update this command, either activated manually or automatically, is supposed to inform. Indeed, it seems to us that this “Check” has not informed us of the existence of a new **ver. 4.13**, while we where using the **ver. 4.12**.  
Nor, certainly, it is meant to inform us of the existence of a main **ver. 5** which, evidently, is not considered an updating of the **ver. 4**, but another line of product altogether.
  - Then we'd like anyhow to be confirmed that this command, either activated manually or automatically, is only supposed to inform, and not to carry on any actual automatic updating procedure.

- -

## Eric Setup Completion

The Eric operative environment obtained with an initial successful installation is rich enough to deserve—or even require—some subsequent actions of setup completion with the menu command `Settings > Preferences...`, as those hereafter listed.

Default File Filters

Default file extensions can be conveniently set via menu command:

Editor > Filehandling > Default File Filters

For instance this way:

Open files: \*

Save File: \*.py

Being the initial setup values a not very convenient “blank”.

Help Viewers > Eric Web Browser

Help Documentation > Python Documentation

Help Documentation > PyQt4 Documentation

(see related menu commands)

## Viewpoints

- **This:** `C:\Python2x\Lib\site-packages\eric4\pixmaps` is the directory where you can recover `ericSplash.png`, `eric_small.png` and `eric.png` files to some aesthetic decency, as here shown.



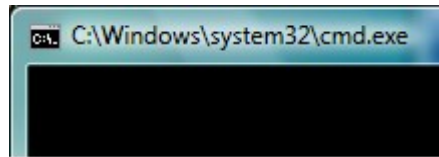
That said to the benefit of whom do not appreciate some extreme Gothic aesthetic expressions.

--

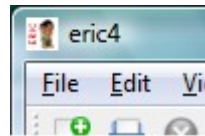
## Run

Instead of the usual Windows system `Start` menu command, Eric can be run using such<sup>44</sup> commands:

`Eric4.bat`      Batch command located in the very `\Python2x` standard installation directory. As a result you'll get first a DOS Command Prompt shell<sup>45</sup>,



and then also the proper Eric's working window.

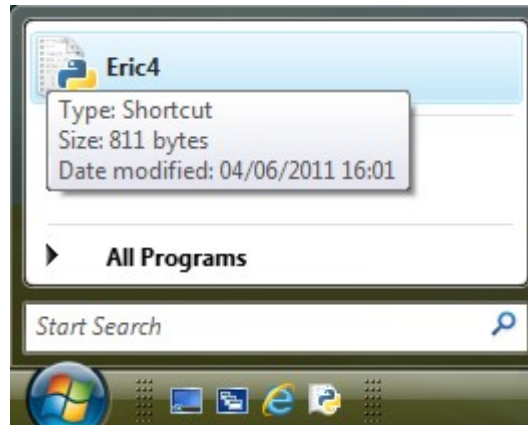


`eric4.pyw`      Python command file located into the standard Eric installation directory:  
                   `\Python27\Lib\site-packages\eric4\  
 which will open no other working window but the Eric's one.`

So that it's rather convenient to “Create a Shortcut” copy of this command, and store it where it's handy, as on top of the system `Start` menu:

<sup>44</sup> Rather awkward, for a Windows user.

<sup>45</sup> We've been told that this is not so unusual with Python programs. Maybe, but certainly any Windows user will look at it scornfully.



<!> Here is how to create such an useful shortcut:

- 1/ Locate `eric.pyw` on `C:\Python2x\Lib\site-packages\eric4\`;
- 2/ Right-click it to “Create Shortcut”, and keep this shortcut there;
- 3/ A drag-drop a copy of this shortcut can be set onto the system menu `Start`, possibly renaming it as “Eric4”, and/or also on the system `Taskbar`; keeping anyhow also the original shortcut as with former point.

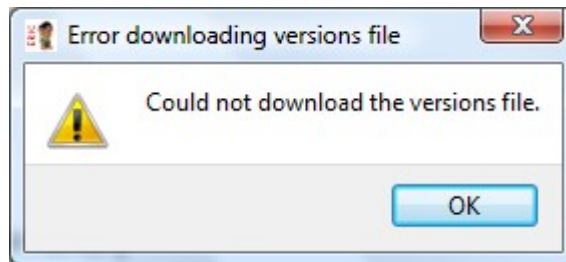
- -

### Remarks

- All the other similar “\*.bat” commands, such as “`eric4-api.bat`”, ..., which can be spotted into the same directory, along with “`Eric4.bat`”, are meant to run various Eric tools autonomously. For instance “`eric4-webbrowser.bat`” is to run the Eric Web Browser independently from Eric (see).

### Viewpoints

- Now (Apr. '11), all of a sudden, after weeks of ordinary activity, at start Eric began displaying such error dialog box.



We presume this pseudo-“Error” has to do with `Settings > Preferences > Application >`

Configure the application > Check for updates. Therefore we suggest to inform the user with another kind of a dialog box, more precise and less alarming .

- -

## Optional Packages

Optional packages can be installed at wish, in any moment, in no special order. What here follows is just a real-case, notable example.

### PyEnchant

*S-PM 110600*

Spell-checking library for Python.

#### *Where*

Available for downloading from URL:

<http://www.rfk.id.au/software/pyenchant/download.html>

#### *What*

Pre-built installation program for Windows platform (and Python 2.5–up):

`pyenchant-1.6.5.win32.exe`

#### *How*

Simply setup-run, it will store into: `\Python2x\Lib\site-package\enchant\`

#### *Then*

After such installation, the Eric menu command `Extras > Spell Check...` becomes enabled (see).

To possibly un-install, there is no special procedure, just delete the directory containing the PyEnchant code<sup>46</sup>.

- = -

---

<sup>46</sup> That's what we've been told by the author, Mr. Ryan Kelly, in a private communication; on the ground that this action is so simple that there is no need for a dedicated “Uninstall” procedure.

# Appendix

## Typographical Conventions

Prevalent typographical conventions here adopted to increase readability.

Times New Roman      Ordinary text font, throughout all this “Report”

Arial                    Titles and first occurrence or infrequently used brand names, usually with first letter Uppercase.

By the way, the titles in this Report, menu commands' included<sup>47</sup>, are written according the so called “nominal” style rule. That is: initial uppercase, then all words with initial uppercase but possibly articles, conjunctions, prepositions and case-significant standard identifiers, such as “unittest” (see).

Courier New          Standard technical elements and references, such as:  
 URL                    <http://sourceforge.net/projects/>  
 file path              C:\Program Files\  
 menu command        Settings > Preferences...

Georgia                Meta-text, as for header, footer and Author's notes

S-PM *yymmdd*        Begin-Of-Segment delimiter. An Author and Date-code mark, useful for text updating purposes (see *Foreword*)

- -                    End-Of-Segment delimiter

<!>                    [AsFor: Importance] Mark of particularly notable point;

<...>                    [AsFor: Unfinished] Place-holder mark, a section to be completed;

<~>                    [AsFor: Off scope] Editorial note, such as for a subject out of the main scope of this work, e.g. related to development, not to use;

<?>                    Questionable point, [AsFor: “Hic sunt leones”<sup>48</sup>]:  
 • Faulted / Unreliable / Still under investigation / Not tested  
 • Insufficient information available

- -

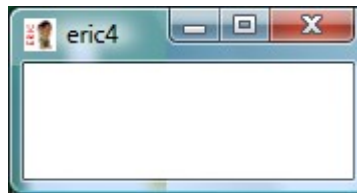
<sup>47</sup> This to be taken also as an implied suggestion for the Eric producer to conform its menu commands to this universally accepted rule. It's the same typographic rule adopted for titles in books, newspapers and magazines within the English printing tradition, where all words begins with a uppercase letter, possibly but conjunctions, prepositions and articles. Other linguistic traditions can be different, typically with only the first initial capitalized...

<sup>48</sup> In ancient maps from the Roman era, African regions about which there was no information were identified by the Latin inscription *HIC SUNT LEONES* (here there are lions). The reason for the inscription was to warn the potential traveller of dangers that he may encounter in the unmapped areas.

## Eric Specific Concepts and Terms

A few terms here illustrated as used in this Report, for clarity's sake.

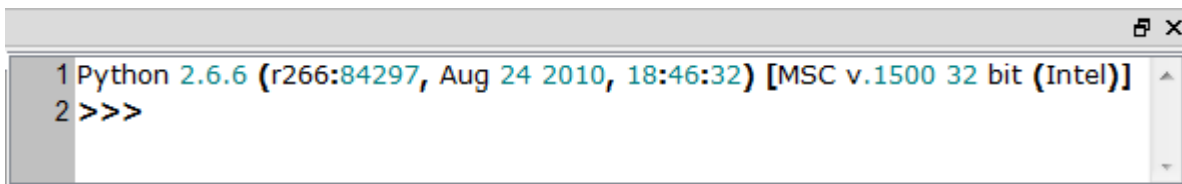
**Window** Term assumed as universally known, and here preferably used only where really appropriate. That is, with reference to such a top-workspace structure:



typically equipped with Title Bar, Borders, and Min-Max-Close Buttons.

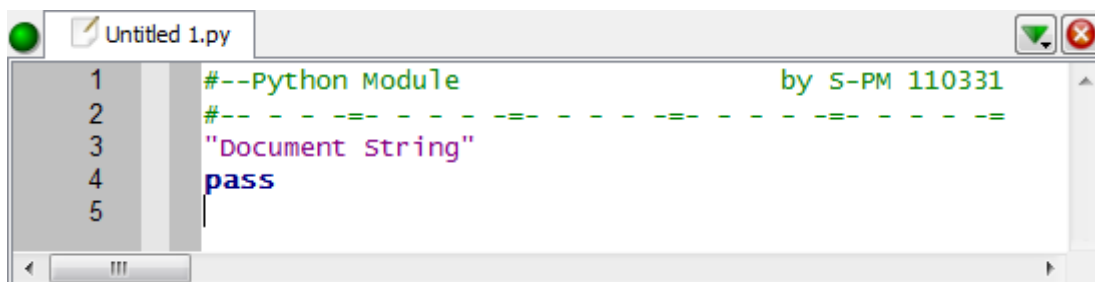
**Interactive Shell Form**

Also simply: “Shell”, for such workspace structure<sup>49</sup>:



**Source Edit Form**

Also “Source Text Edit Form”, or simply “Source Form”. It's the tagged workspace for the usual Python source editing and execution.



<sup>49</sup> As said, a structure here referred to with the term *Form*, not *Window* (see).

## Eric Related Directory

<User>\\_eric4 Utility directory, created and managed automatically, where you may find such sub-directory:

```
browser
Downloads
eric4plugins
spelling
```

And such files:

```
eric4_error.Log
*.macro
```

\_eric4project Eric project management directory, holding such XML <myProject>.xxx files:

```
*.e4q      <?>[AsFor: Unknown]
*.e4d      Debugger Properties
*.e4s      Working Session
*.e4t      TODO: / FIXME: Tasks
```

--

## Eric Related File Type

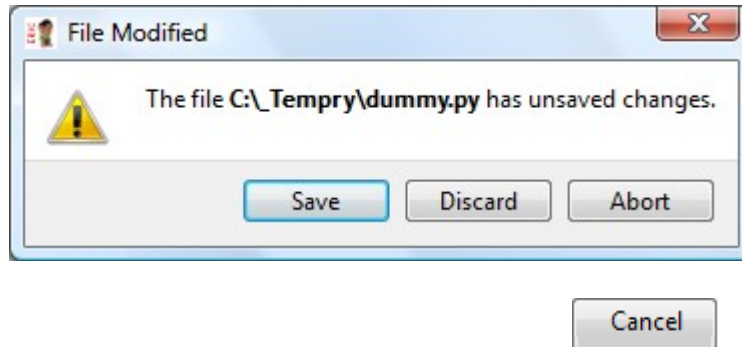
```
*.py      Standard Python source module
*.pyc     Standard Python source module's companion file, automatically created
*.e4p     Eric 4 Project file
*.e4m     Eric 4 Multi-project file
*.e4k, *.e4kz  Eric 4 keyboard shortcut configuration files
```

--

## Hot Issues

Introductory “billboard” section, dedicated to issues particularly relevant, urgent or annoying, according to the totally unquestionable and freakish Author's discretion.

- In such a circumstance:



The standard caption shouldn't contemplate a dramatic “Abort”, but a gentle “Cancel” instead.

- = -

## Contents

Copyright Page.....	2
Foreword.....	3
Instant Reference.....	4
1. Introduction & Generality.....	5
2. Menu Commands.....	6
3. Menu Complements.....	97
4. Add-Ons and Accessories .....	102
5. Setup and General Management.....	103
1/2] Python.....	104
2/2] PyQt.....	105
Eric.....	108
PyEnchant.....	115
Appendix.....	116

- = -